



LIVRO DE MINICURSOS

ercas2019.enucompi.com.br



Teresina-PI
2019

ORGANIZADORES

FLÁVIO HENRIQUE DUARTE DE ARAÚJO
FRANKLHES SANTOS CARVALHO
JACLASON MACHADO VERAS
RODRIGO AUGUSTO ROCHA SOUZA BALUZ
RODRIGO DE MELO SOUZA VERAS
ROMUERE RODRIGUES VELOSO E SILVA

TÍTULO

MINICURSOS ERCAS-PI E ENUCOMPI 2019

ORGANIZAÇÃO INSTITUCIONAL



Porto Alegre
Sociedade Brasileira de Computação – SBC
2019

COMITÊ DE PROGRAMA

Profa. MSc. Alcilene Dalília de Sousa
Universidade Federal do Piauí (UFPI)

Prof. Dr. Francisco Airton Pereira da Silva
Universidade Federal do Piauí (UFPI)

Prof. Dr. Antonio Oseas de Carvalho Filho
Universidade Federal do Piauí (UFPI)

Profa. Dra. Deborah Maria Vieira Magalhães
Universidade Federal do Piauí (UFPI)

Prof. Dr. Ricardo de Andrade Lira Rabêlo
Universidade Federal do Piauí (UFPI)

Prof. Dr. Ivan Saraiva Silva
Universidade Federal do Piauí (UFPI)

Prof. Dr. Thiago Carvalho de Sousa
Universidade Estadual do Piauí (UESPI)

Prof. Dr. Hermes Manoel Galvão Castelo Branco
Universidade Estadual do Piauí (UESPI)

Profa. Dra. Cornélia Janayna Pereira Passarinho
Universidade Estadual do Piauí (UESPI)

Prof. Dr. Ariel Soares Teles
Instituto Federal do Maranhão (IFMA)

Prof. Dr. Fabio de Jesus Lima Gomes
Instituto Federal do Piauí (IFPI)

Prof. Dr. Carlos Giovanni Nunes de Carvalho
Universidade Estadual do Piauí (UESPI)

Prof. Dr. Sérgio Barros de Sousa
Universidade Estadual do Piauí (UESPI)

Prof. Dr. Gustavo Augusto Lima de Campos
Universidade Estadual do Ceará (UECE)

Prof. Dr. Eduardo Takeo Ueda
Centro Universitário SENAC

Prof. Dr. Alcemir Rodrigues Santos
Universidade Estadual do Piauí (UESPI)

Prof. Dr. José de Ribamar Martins Bringel Filho
Universidade Estadual do Piauí (UESPI)

Prof. MSc. Danilo Borges da Silva
Universidade Estadual do Piauí (UESPI)

Prof. Dr. Dario Brito Calçada
Universidade Estadual do Piauí (UESPI)

Profa. Dra. Edna Yoshiko Senzako
Universidade Estadual do Piauí (UESPI)

Prof. Dr. Vinicius Ponte Machado
Universidade Federal do Piauí (UFPI)

Profa. Dra. Keylla Maria de Sá Urtiga Aita
Universidade Federal do Piauí (UFPI)

Prof. Dr. José Valdemir dos Reis Junior
Universidade Federal do Piauí (UFPI)

Prof. MSc. Marcos Aurélio Ayres da Silva
Universidade Federal do Delta do Parnaíba (UFDPa)

Prof. Dr. Flávio Rubens de Carvalho Sousa
Universidade Federal do Ceará (UFC)

Prof. Dr. Ed Porto Bezerra
Universidade Federal da Paraíba (UFPB)

Profa. Dra. Atslands Rego da Rocha
Universidade Federal do Ceará (UFC)

Prof. Dr. Marcos Antonio de Oliveira
Universidade Federal do Ceará (UFC)

Prof. Dr. Regis Pires Magalhães
Universidade Federal do Ceará (UFC)

Prof. MSc. Nécio de Lima Veras
Instituto Federal do Ceará (IFCE)

Prof. Dr. Wladimir Araujo Tavares
Universidade Federal do Ceará (UFC)

PREFÁCIO

É com grande satisfação que apresentamos este livro-texto dos minicursos da VII Escola Regional de Computação Aplicada à Saúde (ERCAS-PI) e do XII Encontro Unificado de Computação do Piauí (ENUCOMPI 2019) realizados na Universidade Federal do Piauí (UFPI), no período de 12 a 14 de novembro de 2019, na cidade de Teresina – Piauí.

A ERCAS tem por objetivo disseminar, discutir as metodologias e promover a integração entre professores, alunos e profissionais que atuam com tecnologias aplicadas a área de saúde. Nesta edição, contou com a realização em paralelo do evento satélite Encontro Unificado de Computação do Piauí. Em sua 12ª edição, o ENUCOMPI se consolida como um dos mais importantes eventos de Computação da região nordeste, especialmente no eixo Maranhão, Piauí e Ceará, visando contribuir para o enriquecimento da ciência regional, por meio do incentivo a pesquisa, a inovação e ao empreendedorismo.

Os eventos se tornam uma oportunidade para que professores e pesquisadores possam detectar afinidades e desempenhar no futuro projetos de pesquisa em conjunto, vislumbrando a submissão conjunta de projetos em resposta a editais do CNPq, FINEP, CAPES e das Fundações de Apoio à Pesquisa.

Tivemos como organizadoras institucionais do evento a Universidade Federal do Piauí (UFPI) e a Universidade Estadual do Piauí (UESPI), sendo parceiro o Instituto Federal de Educação, Ciência e Tecnologia do Piauí (IFPI), além de instituições privadas como a Faculdade Estácio CEUT, dentre outras. Agradecimentos as empresas privadas que foram patrocinadoras do evento, TDA Informática, Livros 360, MobiEduca.me, Maida Health e IntMed, além do apoio a fomento mantido via Edital com a Fundação de Amparo à Pesquisa do Estado do Piauí.

Teresina (PI), 14 de novembro de 2019.

Comissão Organizadora

FICHA CATALOGRÁFICA
Universidade Federal do Piauí
Biblioteca Comunitária Jornalista Carlos Castello Branco
Serviço de Processamento Técnico

M665 Minicursos ERCAS & ENUCMPI 2019 / organizadores, Flávio Henrique Duarte de Araújo ... [et al.]. – Porto Alegre : SBC, 2019.
175 p. : il.

Organizadores: Flávio Henrique Duarte de Araújo, Franklhes Santos Carvalho, Jaclason Machado Veras, Rodrigo Augusto Rocha Souza Baluz, Rodrigo de Melo Souza Veras, Romuere Rodrigues Veloso e Silva.

ISBN 978-85-7669-485-4

1. Computação. 2. Saúde. 3. Encontro Unificado de Computação do Piauí (ENUCOMPI). 4. Escola Regional de Computação Aplicada à Saúde (ERCAS-PI). I. Araújo, Flávio Henrique Duarte de. II. Carvalho, Franklhes Santos. III. Veras, Jaclason Machado. IV. Baluz, Rodrigo Augusto Rocha Souza. V. Veras, Rodrigo de Melo Souza. VI. Silva, Romuere Rodrigues Veloso e.

CDD 004

Capítulo

1

Proteus, um passo além dos projetos com Arduino

Alan Jheyson Ribeiro da Costa, Harilton da Silva Araújo

Abstract

This paper presents an alternative that makes it possible to expand the resources of researchers and developers of the Arduino platform. Getting to know the basic functions of Proteus Software, a powerful PCB production and simulation tool that has a huge library of components. It also presents a manual PCB production alternative using materials that are easy to find in today's market.

Resumo

Este trabalho apresenta uma alternativa que possibilita ampliar os recursos de pesquisadores e desenvolvedores da plataforma Arduino. Permitindo conhecer as funções básicas do Software Proteus, uma poderosa ferramenta de produção e simulação de PCB's, que possui uma enorme biblioteca de componentes. Apresenta também uma alternativa de produção manual de PCB, utilizando matérias fáceis de encontrar no mercado atual.

1.1. Introdução

É possível observar que na última década houve um crescimento no estudo e desenvolvimento de projetos utilizando a plataforma Arduino. No campo acadêmico esta plataforma tornou-se um elemento facilitador, pois permite ao estudante/pesquisador experimentar na prática os conceitos criados. No mercado há uma grande variedade de placas baseadas em Arduino, assim como sensores e atuadores compatíveis com esta plataforma.

Por ser uma placa de desenvolvimento o Arduino possui algumas limitações, já ela é voltada para a prototipação e aprendizagem e não para o desenvolvimento de um produto final. Por sua vez, existem hoje no mercado fermentas, softwares, que permitem você simular de componentes eletrônicos a grandes projetos, dentre eles podemos destacar o Proteus, uma ferramenta que permite criar o diagrama esquemáticos, a placa

de circuito impresso e visualizar a placa e os componentes em 3D. Os objetivos deste trabalho e demonstrar que o Proteus pode ser uma ferramenta para auxiliar no desenvolvimento de um produto desde o conceito com Arduino até o produto final, produzindo em ambiente virtual, Proteus, e confeccionando a placa de circuito impresso com matérias de fácil acesso.

1.2. Arduino

Em 2005, o professor Massimo Banzi procura uma forma econômica de estudante poderem trabalhar com tecnologia. Foi assim que teve início o Arduino, que tinha como objetivo ser barato e que fosse uma plataforma que qualquer pessoa pudesse utilizar [Evans et al, 2013]. Hoje a plataforma Arduino, que é de código Aberto, possui uma serie de versões baseadas no microprocessador ATmega*.

Por suas facilidades, o Arduino ganhou popularidades e encontramos hoje no mercado além das placas, uma grande variedade de sensores, atuadores, *Shields* compatíveis com a plataforma, essa variedade permite que estudantes, professores e pesquisadores possam validar seus conceitos de forma rápida e pratica. Como supracitado o Arduino possui limitações e teoricamente sua utilização deveria aplicar-se ao período de desenvolvimento conceitual, já que para o produto final o ideal seria a utilização de um circuito impresso otimizado para a aplicação desejada.

1.3. Proteus

O Proteus é uma ferramenta desenvolvida pela empresa Labcenter Eletronic, e combina um conjunto de recursos para designer, teste e layouts de circuito impresso, e é destinado tanto para a indústria como a para instituições de ensino[Anacon, 2010]. Um dos grandes diferencias do Proteus e a capacidade de simular circuitos elétricos microcontrolados.

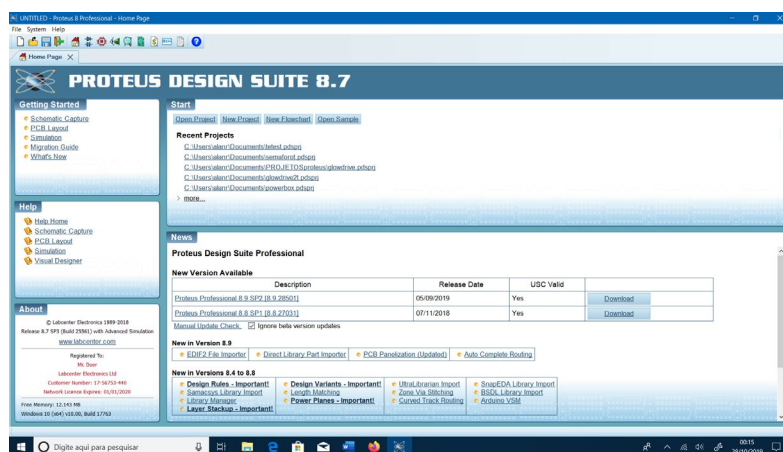


Figura 1.1 - Tela Inicial Proteus 8.7[Elaborada Pelos Autores]

Para nosso trabalho utilizaremos o Proteus 8.7 (Figura 1.1) e três de seus módulos, que trabalham em conjunto, modulo de desenvolvimento esquemático(*Schematic Capture*), modulo de desenvolvimento de *layout's* PCB (*PCB Layout*) e o

modulo de visualização 3D (*3D Visualizer*). Acessado facilmente na barra de ferramenta superior conforme Figura 1.2.

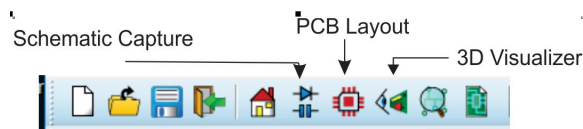


Figura 1.2 – Barra de ferramentas [Elaborada Pelos Autores]

1.3.1. Desenvolvimento Esquemático

O Modulo de desenvolvimento Esquemático apresenta em seu ambiente gráfico uma interface simples e intuitiva. A Figura 1.3 destaca alguns elementos específicos.

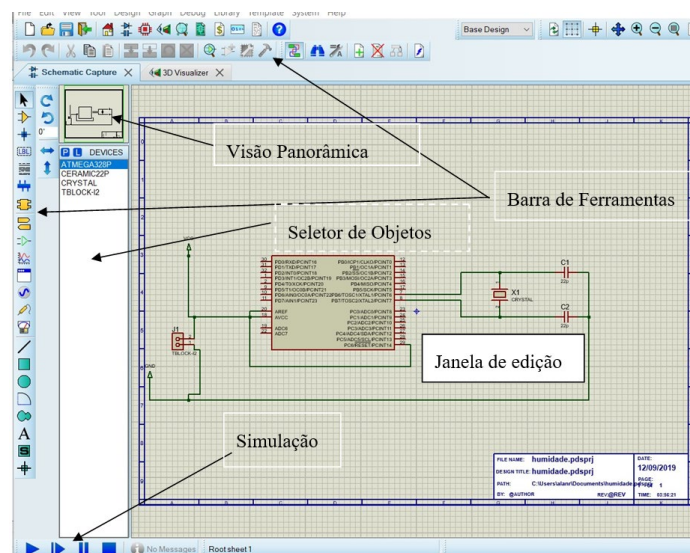


Figura 1.3 – Modulo de Desenvolvimento Esquemático [Elaborada Pelos Autores]

1.3.1.1. Visão Panorâmica

Localizada sempre ao lado esquerdo do *Schematic Capture*, mostra uma visão geral de todo desenho que está na janela de edição ou o componente selecionado no seletor de objetos.

1.3.1.2. Seletor de Objetos

O seletor de objetos armazena os componentes que são utilizados no diagrama esquemático, estes componentes são selecionados no *Pick Devices*, que é acessado pela

letra P logo acima do seletor de Objetos. Na Figura 1.4 podemos ver a opção de seleção de categoria, subcategoria e fabricante do componentes, que iram aparecer na parte central do Pick Devices, a direita na parte superior o desenho do componente que sera utilizado no esquema e na parte inferior uma previa do componente que sera utilizado no modulo de layout PCB. É importante observar quem nem todos os componenes que estão no modulo de esquema estara no de layout.

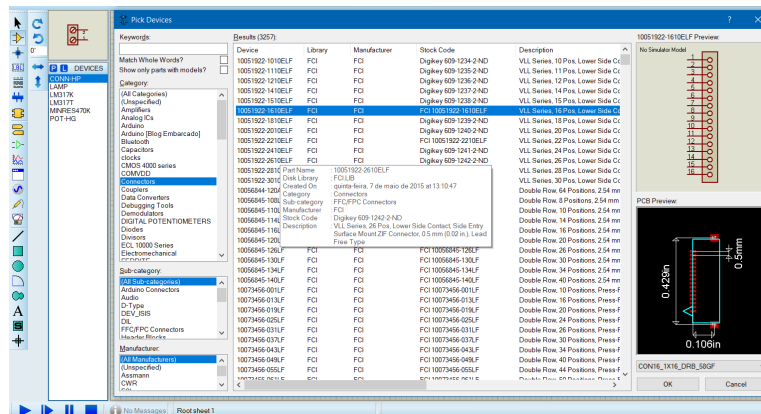


Figura 1.4 – Pick Devices [Elaborada Pelos Autores]

1.3.1.3. Barra de Ferramentas

As barras de ferramentas do *Schematic Capture* estão distribuídas na parte superior e esquerda da janela. Ao selecionar as barras Main Model ou Gadgets componentes e ferramentas apareceram no seletor de objetos (Figura 1.5).

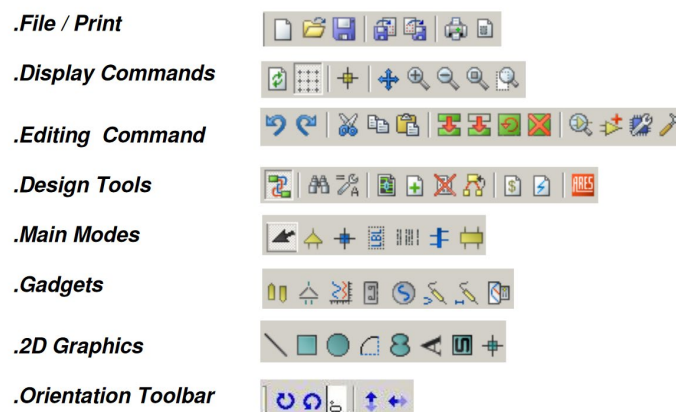


Figura 1.5 – Barras de ferramentas *Schematic Capture*

1.3.2. Layout PCB

O módulo *PCB layout* é a interface gráfica específica para o desenvolvimento do layout, uma boa prática é desenvolver primeiro o diagrama esquemático no *Schematic Capture* e posteriormente no *PCB Layout*, esta ordem facilitará muito no desenvolvimento pois os componentes estarão com seus pinos previamente ligados.

Podemos observar na Figura 1.6 a janela do *PCB Layout*, nela teremos a janela de edição diferentes da janela no *Schematic Capture*, e a substituição de algumas barras de ferramentas. Nesta janela poderemos organizar o componente da melhor forma possível organizar suas trilhas. Uma grande quantidade de recursos está disponível para o desenvolvimento de layout, pois há a possibilidade de criar componentes ou adaptá-los ao seu projeto.

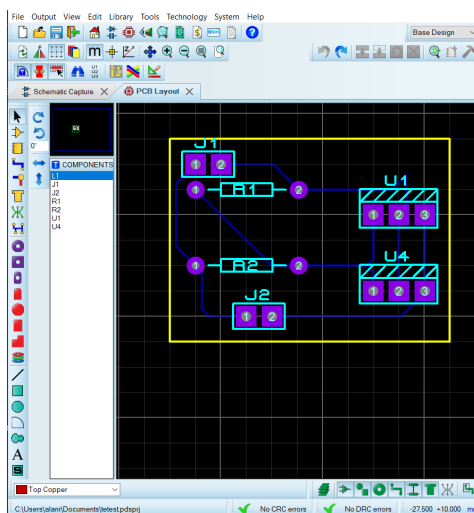


Figura 1.6 - Janela do Layout PCB [Elaborada Pelos Autores]

1.4. 3D Visualizer

Este modulo permite visualizar o produto final em todos os angulos, com ou sem componente, a barra de ferramentas com este comandos fica na parte inferior esquerda(Figura 1.7).

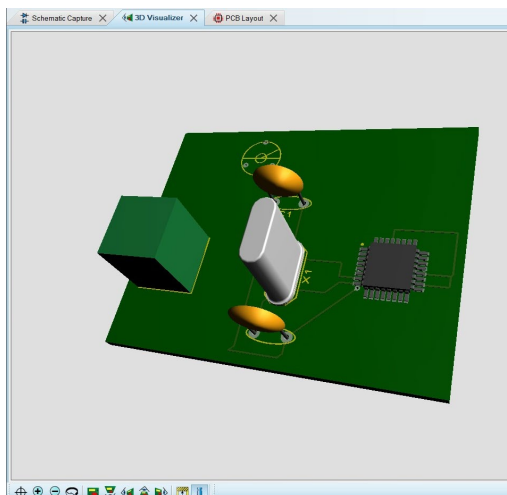


Figura 1.7 Janela 3D Visualizer [Elaborada Pelos Autores]

1.5. Criando Diagrama Esquemático

Pra iniciar a criação do diagrama esquemático devemos criar um novo projeto, há duas formas:

1. Clicando no ícone New Project, que está no início da barra(Figura 1.8).



Figura 1.0.8 - barra de ferramenta file [Elaborada Pelos Autores]

2. Utilizando o comando File > New Project.

Os dois métodos iniciaram o *Wizard*, que auxiliara na criação do projeto. Figura 0.9 apresenta as etapas de criação:

- i. Define nome e local onde projeto será salvo(Figura1.9 (a)).
- ii. Define o formato da folha do esquema(Figura 1.9(b)).
- iii. Marcamos a opção para não criar o PCB layout, apenas posteriormente.

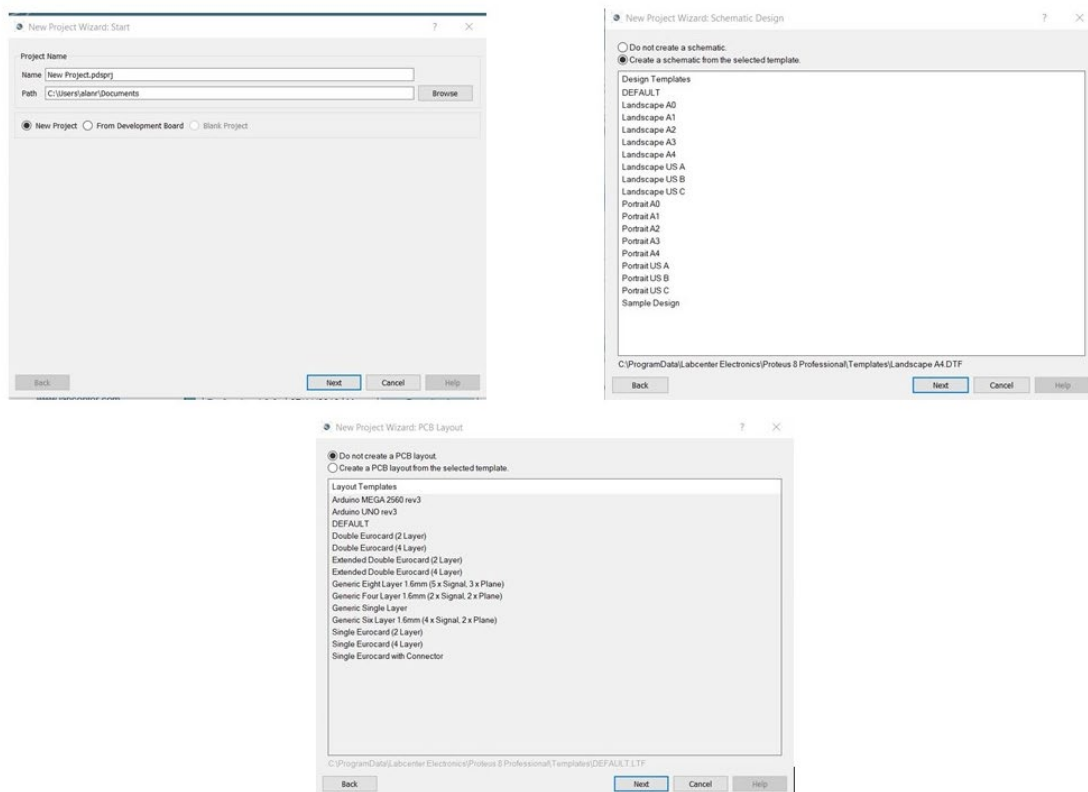


Figura 1.9 – New Project Wizard [Elaborada Pelos Autores]

Ao finalizar o *Wizard* a janela do novo projeto estara pronta para iniciar a edição(Figura 1.10).

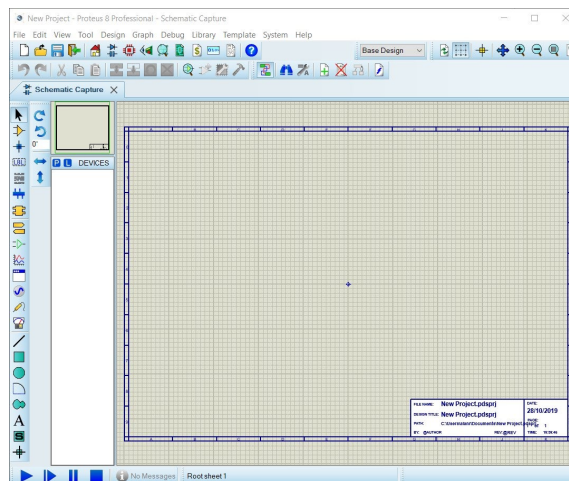



Figura 1.10 - Novo Projeto - Janela Schematic Capture [Elaborada Pelos Autores]

1.5.1. Inserindo Objetos

Após a criação do novo projeto podemos iniciar a edição do diagrama esquemático. Este modulo possui bibliotecas de símbolos e dispositivos, e o usuário pode incrementa-las, criando novos elementos ou importando de outras bibliotecas.

Para inserir um objeto a partir de uma biblioteca, deve-se clicar no ícone *Componentes Mode*  em seguida no botão P no seletor de objetos ou P no teclado,

abrir a janela Pick Devices(Figura 1.4). Nela iremos selecionar cada um dos componentes utilizados no projeto.

Como projeto de demonstração iremos construir um circuito para piscar leds sequenciais, será nosso *Hello Word*. Para este exemplo iremos utilizar os seguintes componentes:

- 4 leds vermelhos;
- 4 resistores de 175ohms
- 1 ATmega16
- 2 capacitores de 22pF
- 1 cristal 16MHz

Os componentes devem ser adicionados no seletor de objetos, depois inseridos na janela de edição(Figura 1.11). No seletor de Objetos cada componente aparece uma única vez, mas pode ser inserido inúmeras vezes na janela de edição. Os componentes podem ter seus valores e propriedades alterados depois de inseridos. Por exemplo, a resistência de 175ohms pode ser alterado para qualquer valor, assim como cristais capacitores e leds.

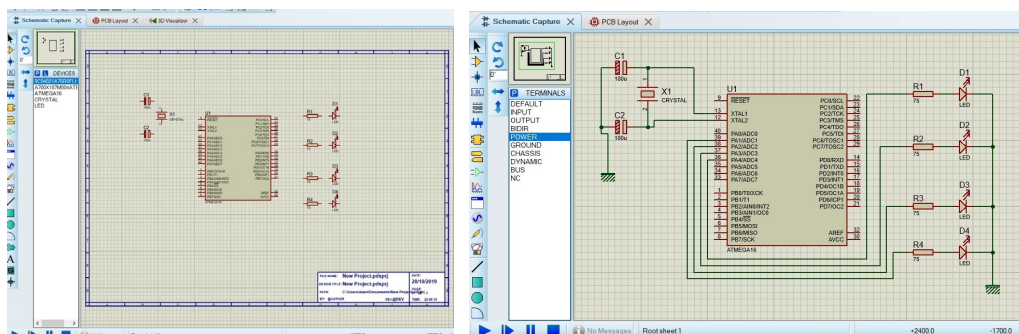


Figura 1.11 Esquema utilizando microcontrolador ATmega16 [Elaborada Pelos Autores]

Ao posicionar o mouse no terminal dos componentes o cursor habilita a função de desenho de trilha, podendo assim interligar os componentes. Neste módulo é possível simular o circuito, e embarcando o código hexadecimal gerado pela IDE Arduino no próprio microcontrolador.

1.5.2. Simulação

Finalizado o diagrama esquemático podemos testar nosso circuito, para isso precisaremos utilizar a IDE Arduino, nela iremos criar o código para nosso projeto. Para utilizar no Proteus precisaremos de um arquivo hexadecimal gerado pela IDE Arduino, pressionando as teclas Ctrl+R iremos iniciar a compilação, finalizada iremos encontrar, na parte inferior da janela o endereço do arquivo .hex gerado pela IDE (Figura 1.12), devemos copiar este endereço.



Figura 1.12 [Elaborada Pelos Autores]

Voltando a janela do Proteus damos dois cliques no microcontrolador, aparecerá a janela *Edit Component* (Figura 1.13), na opção *Program File* colaremos o endereço do arquivo .hex. Para executar basta clicar no botão play da janela.

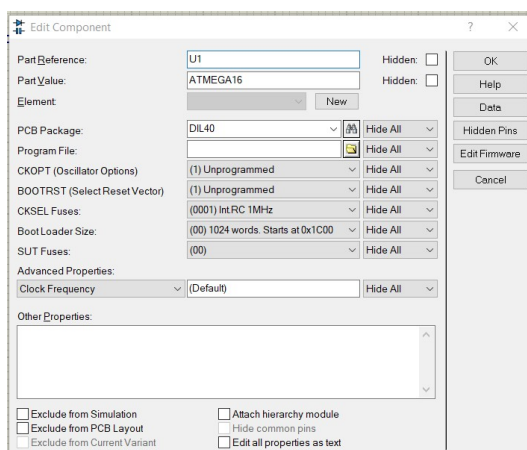





Figura 1.13 [Elaborada Pelos Autores]

1.6. Transformando o Diagrama Esquemático em layout PCB

Concluído o diagrama esquemático, é hora de iniciar o *layout*. Para inserir os componentes basta clicar no ícone *Componentes Mode* , todos os componentes utilizados na diagrama iram aparecer no neste Modulo, no seletor de objetos, agora eles apareceram com a quantidade e o nome utilizados no diagrama. Ao inserir cada elemento, automaticamente ele indicara quais os terminais estaram interligados (Figura 1.14). Permitindo que o usuario crie as trilhas manualmente clicando no icone *Track Mode*  ou posso criar-las de forma automatica clicando no icone *Auto-route* .

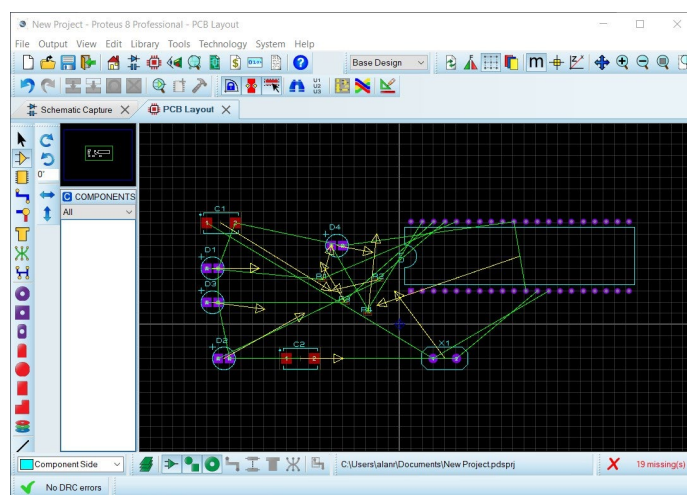


Figura 1.14 – Setas indicando ligação dos componentes [Elaborada Pelos Autores]

Podemos observar na Figura 15 as trilhas organizadas logo apos a execução do *Auto-route*, por padrão o Proteus considera que as placas podem possuir trilhas nas duas faces, mas mesmo executando o *Auto-router* o susuario pode alterar as trilhar posteriormente.

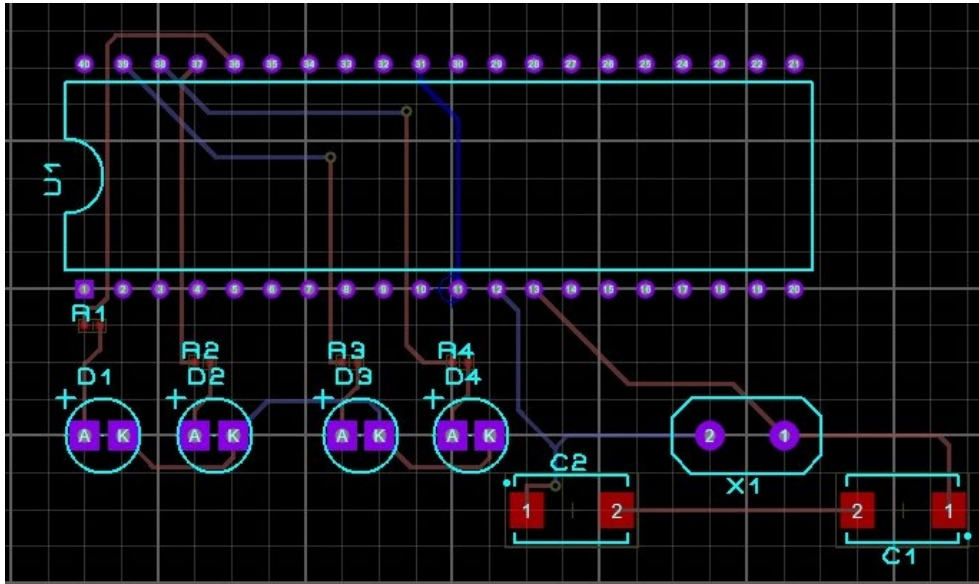



Figura 1.15 - Trilhas Criadas automaticamente [Elaborada Pelos Autores]

Ainda precisamos definir o tamanho da PCB, para isso precisamos delimitar na janela de edição seu tamanho. Para isso iremos clicar no ícone *2D Graphics Box Mode* , selecionar *Board Edge*, na conto inferior esquerdo da janela (Figura 1.16) e desenhar o limite dos compomentes (Figura 1.17).

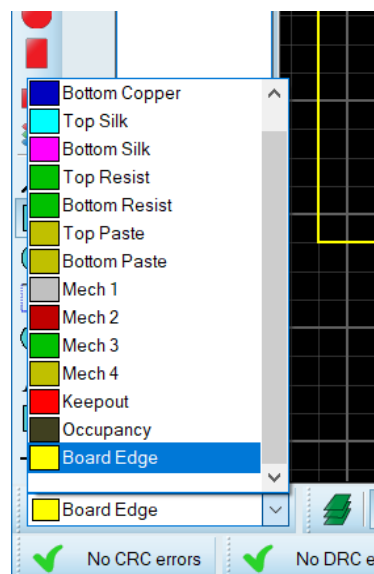


Figura 1.16 [Elaborada Pelos Autores]

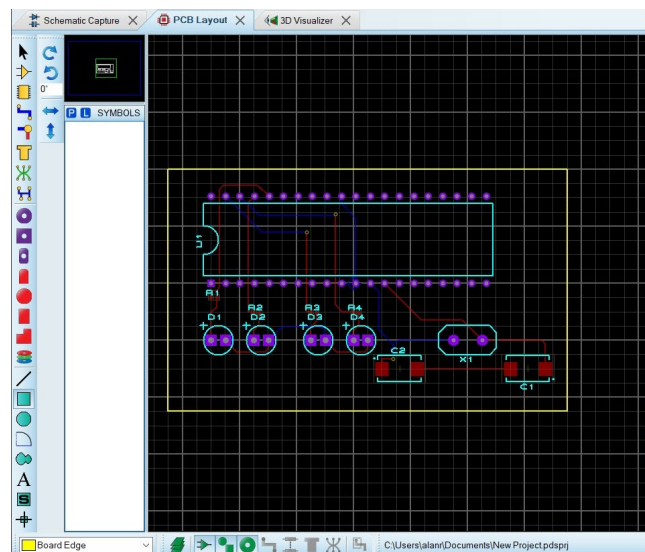


Figura 1.17 [Elaborada Pelos Autores]

Layout concluído, podemos visualizar uma prévia da placa em 3d, para isto basta acessar o módulo *3d Visualizer* (Figura 1.18). Neste módulo podemos visualizar a placa por todos os ângulos.

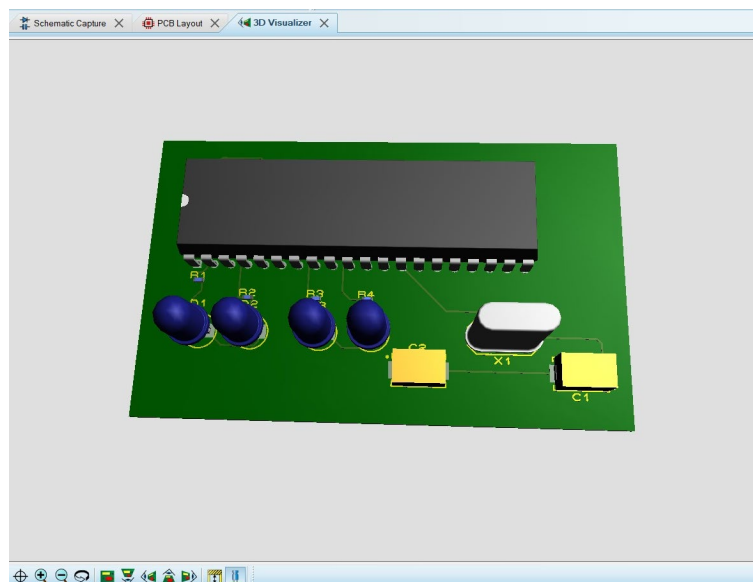


Figura 1.18 - Imagem 3d do projeto [Elaborada Pelos Autores]

1.7. Do layout PCB a placa de Circuito Impresso

Esta é a última etapa que realizaremos no Proteus, ainda no módulo PCB layout iremos acessar o Print Layout, para isso iremos no menu Output > Print Layout, teremos acesso a janela (Figura 1.19). Nesta janela podemos escolher quais elementos serão impresso, assim como escala, posição ou a face que será impressa, em caso de projetos em que a trilhas nas duas faces da placa.

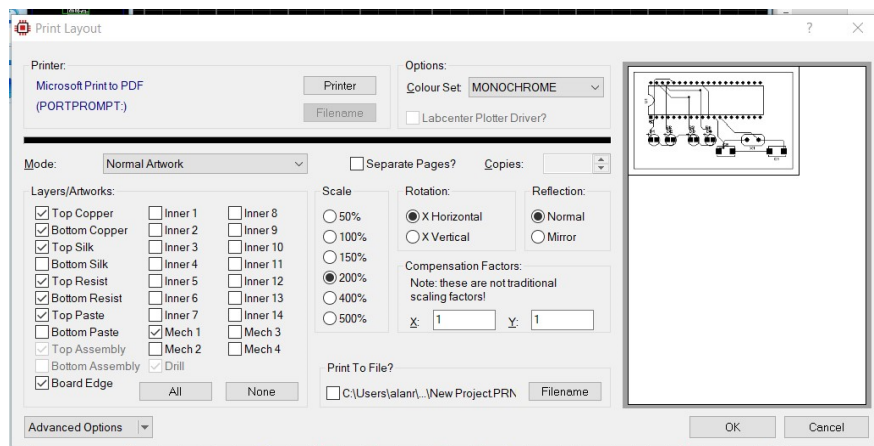


Figura 1.19 [Elaborada Pelos Autores]

O Resultado da impressão será um arquivo (Figura 1.20), onde teremos as trilhas e o local dos pinos de cada componente.

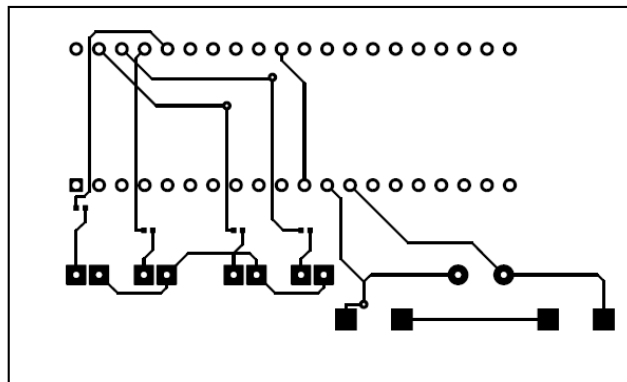


Figura 1.20 - Layout PCB impresso [Elaborada Pelos Autores]

Com o projeto finalizado no Proteus, existem diversas formas de produzir uma placa de circuito impresso, algumas empresas nacionais e estrangeiras produzem placas em pequena quantidade, a partir de arquivos gerados pelo Proteus. Outra maneira é produzir manualmente. Muitas técnicas podem ser usadas para produzir as placas. Mas iremos ver uma bem simples que utiliza apenas:

- Placa de Fenolite;
- Percloroeto de ferro;
- Layout PCB, impresso em transparência;

- Emulsão verde para serigrafia

A placa de fenolite e o percloreto de ferro são encontrados em loja de eletrônica, a emulsão verde é facilmente encontrada em lojas de serigrafia, ele sempre vem acompanhado de um sensibilizante.

O primeiro passo é passar uma fina camada de emulsão verde, já preparada, na placa de fenolite, importante que o local onde a emulsão será manipulada tenha baixa luminosidade, espera secar.

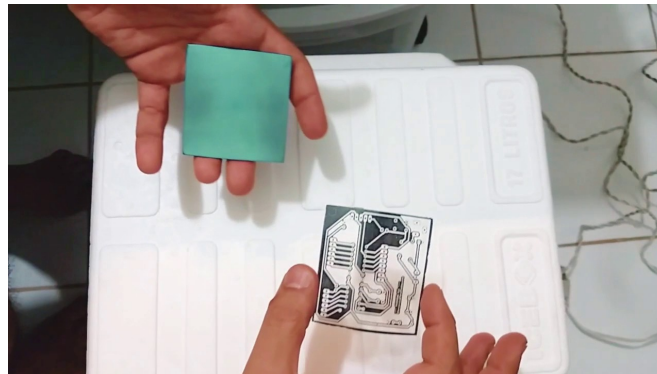


Figura 1.21 Processo de Impressão [You Tube]

Devemos posicionar o layout PCB encima da placa(Figura 1.21), fixando de forma que a transparência não se mova durante o manuseio. Para que a emulsão grave o desenho da PCB devemos posicionar a placa ao sol ou a uma luz ultravioleta por alguns minutos. Retornamos ao local com pouca luminosidade para retirar o excesso de emulsão com água, neste momento deve aparecer as trilhas pintadas no fenolite (Figura 1.22).

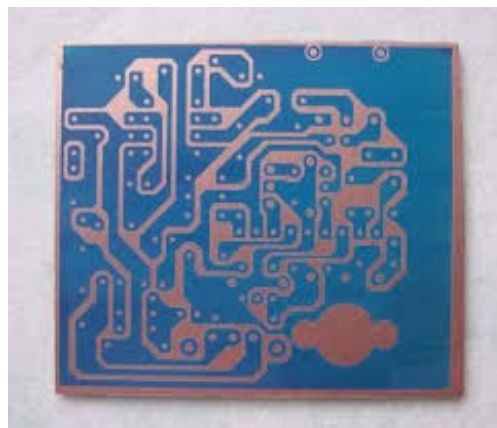


Figura 1.22 PCB com emulsão revelada [Mercado Livre]

Próximo passo é mergulhar a placa no percloreto de ferro, para corroer o cobre, após alguns minutos apenas as trilhas estarão com cobre e a PCB está pronta para receber os componentes.

Referencias

[Evans et all, 2013].Evans, M. Nobel, J. Hochenbaum, J.(2015) Arduino em ação. Editora Novatec.

[Araujo 2011] Araújo, F.M. A.Hardware Livre. Congresso de Tecnologia da Informação. Vol. 6, 2011.

[Jung, 204] Jung, C. F.Metodologia para Pesquisa e Desenvolvimento: aplicada a novas tecnologias, produtos e processos.Axcel Books, 2004

[Arduino,2019] Arduino. Shields.2019.

<https://www.arduino.cc/en/Main/arduinoShields/>, acessado outubro 2019.

[Alves 2013] Alves, R. M. S.; Armando L. C., Pinto, M. C.; Sampaio, F. F. & Elia, M. F. Uso do Hardware Livre Arduino em Ambientes de Ensino-aprendizagem. Jornada de Atualização em Informática na Educação, v. 1, n. 1, p. 162-187, 2013.

[Mercado Livre 2019] Kit caseiro, https://produto.mercadolivre.com.br/MLB-696878951-kit-caseiro-prototipo-pcb-pci-profissional-fotosensivel-_JM?quantity=1. Acessado em 10/10/201.

[Youtube 2017] Confecção de PCB, <https://www.youtube.com/watch?v=j3ihO3BJxPk>, Acessado em 10/10/2019.

[Anacon, 2010] Anacon Eletronica.Terramento Proteus, Versao 7.6.0. 2010.

2

Desenvolvimento de Jogos com Unity3D Engine

Danilo Carvalho, Charles Miranda, Cristina Vidal

Abstract

This meta-paper explains a bit about game development with Unity3D engine. We'll talk about game development, what is a game, instalation of Unity3D through UnityHub, Unity Editor and more. You will understand about different Unity editor panels, creation and adjustment a game cenário, game objects, creating and scripting. Each part of this meta-paper is just a fast explain about some features, just the necessary to do a proposed game. This meta-paper implements a game where a car is moving throught the scenario avoiding the bombs and collecting some itens to win level. Finnaly, we shows how to generate a game for PC, Mac or Linux.

Resumo

Este meta-artigo explica um pouco sobre desenvolvimento de jogos com Unity3D. Nós vamos falar sobre desenvolvimento de jogos, o que é um jogo, instalação da Unity3D através do UnityHub, Unity Editor, entre outros. Você irá entender sobre os diferentes painéis do Unity Editor, criar e ajustar um cenário do jogo, criação de objetos do jogo e codificação através de scripts. Cada parte deste meta-artigo é apenas uma rápida explicação sobre algum recurso, apenas o necessário para se criar o jogo proposto. Este meta-artigo implementa um jogo onde um carro move-se pelo cenário evitando as bombas e coletando alguns itens para ganhar o jogo. Por fim, vamos mostrar como gerar o jogo para as plataformas PC, Mac e Linux.

2.1. Desenvolvimento de Jogos

Um jogo implementa um cenário baseado em regras onde cada jogador tem ao menos um objetivo final e para alcançar esse objetivo, deverá interagir com o cenário do jogo seguindo suas regras. Quando se pensa em um jogo dessa maneira ele não necessariamente é um produto digital. Observando uma mesa de bilhar por exemplo, vemos como é o ambiente geral daquele jogo e mesmo sabendo que o objetivo é acertar as bolas corretas em algum dos buracos não podemos simplesmente arrastar as bolas com as mãos pois a forma de o jogador interagir com o mundo é através do taco. Um jogo é esse conjunto de regras que quando bem planejadas transformam aquele

ambiente em um lugar divertido e desafiador. Os jogos eletrônicos por sua vez, são jogos que aproveitam as vantagens dos computadores para implementar seu ambiente, suas regras e as formas de interação. Dessa forma, pode-se criar jogos que fazem toda a verificação e gera os resultados de forma automática.

Entre as áreas mais comuns do desenvolvimento de jogos, podemos listar o Game Design que é a parte que cuida da experiência do usuário ditando as regras do mundo e formas de interação do jogador buscando criar um ambiente divertido, desafiador e balanceado. Temos ainda as artes visuais onde habilidades como desenho, pintura, modelagem 3D e esculturas são usadas para se criar os objetos gráficos do jogo que vão desde um personagem até os ícones da interface gráfica. Ainda sobre artes temos a música, onde os profissionais criam uma atmosfera mais imersiva e enriquecedora com música de fundo, efeitos sonoros, sons ambientes, música tema entre outros. Uma quarta área comum no processo de desenvolvimento de jogos é a programação, onde os profissionais criam códigos que dão características às entidades do jogo, implementando suas ações, os gatilhos e permitindo que objetos ganhem novas características. Através da programação os objetos criados pelos artistas vão funcionar no mundo de acordo com as definições do game designer. Esses são os papéis mais comuns mas a indústria dos jogos eletrônicos envolve uma série de perfis profissionais que inclui escritores, desenhistas, músicos, estatísticos, jornalistas especializados em games, profissionais de marketing, youtubers, entre outros.

2.2. Unity3D Engine Instalação

As game engines abstraem recursos como renderização, física, áudio, animação, entre outros, o que torna muito mais fácil e rápido o desenvolvimento de um jogo. A Unity3D é uma plataforma que oferece desde uma game engine com editor gráfico para desenvolvimento de jogos, até lojas de recursos e rede social para busca de talentos. Essa infinidade de recursos é gratuita e nessa versão gratuita você pode inclusive vender os jogos criados exceto se você for uma empresa com receita anual bruta superior a 100 mil dólares.

Vamos começar instalando a Unity no computador, para isso você pode baixar a Unity pelo site oficial <https://store.unity.com/download?ref=personal> e caso você não tenha uma conta, crie uma nova conta também pelo site pois você vai precisar dessa conta para usar a Unity. Você irá baixar o UnityHub e ao abrir o programa você terá as opções de downloads do editor da Unity entre outros recursos. Para instalar a Unity, vá em Installs, ADD e escolha a versão estável mais nova da Unity (nesse caso a Unity 2019.2.8f1). As duas versões acima são hoje alfa e beta e por isso não as escolhi. Opcionalmente você pode instalar mais recursos junto a Unity, uma lista será exibida e você pode escolher o que deseja baixar dessa lista. A imagem abaixo mostra a instalação e um exemplo da lista com recursos extras para baixar junto a Unity.

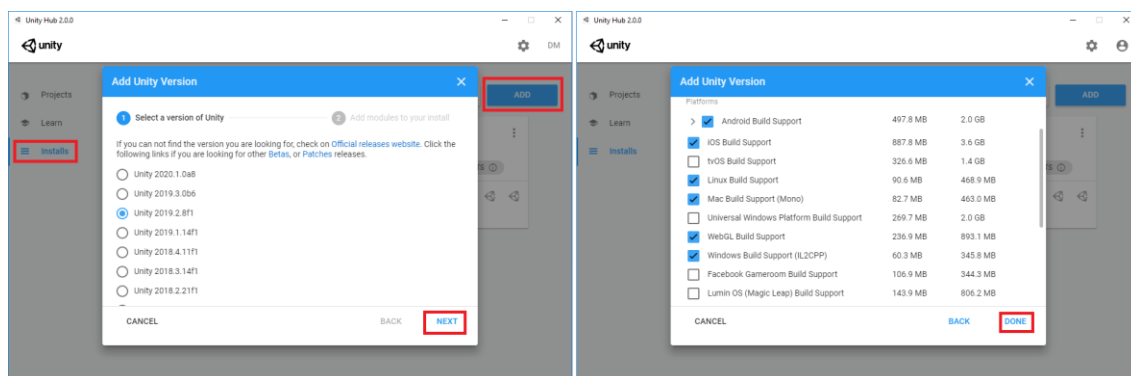


Figura 2.1. Instalação da Unity

Com a Unity instalada crie um novo projeto, para isso no UnityHub acesse o menu Projects e clique em New, selecione a Template 3D, defina um nome para o projeto (no caso coloquei CarProject), defina o local do projeto e clique em Create. Veja a imagem abaixo.

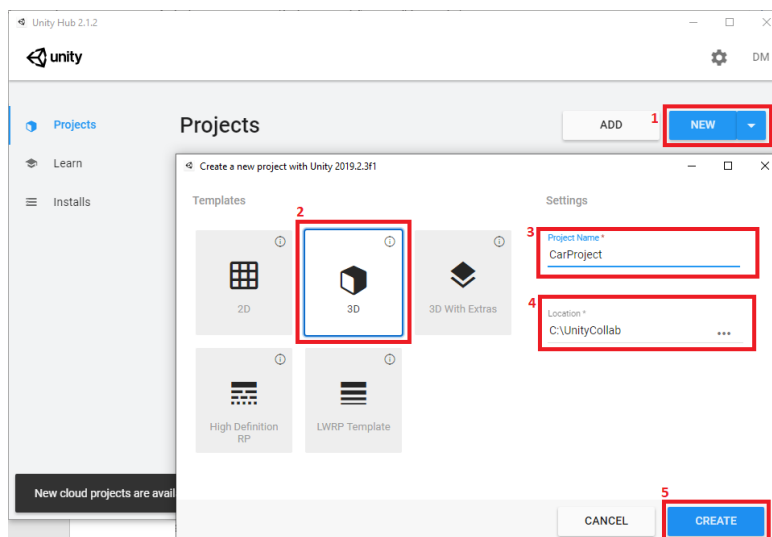


Figura 2.2. Criando o Projeto

2.3. Unity Editor

O editor da Unity é composto de janelas que organizam as funcionalidades oferecidas. Dessa forma, dependendo do que está sendo feito no jogo, uma ou outra janela será útil. A imagem abaixo mostra as principais janelas usadas no desenvolvimento dos jogos. Após a imagem teremos uma explicação sobre cada janela, você pode mexer um pouco nelas para entender melhor os conceitos.

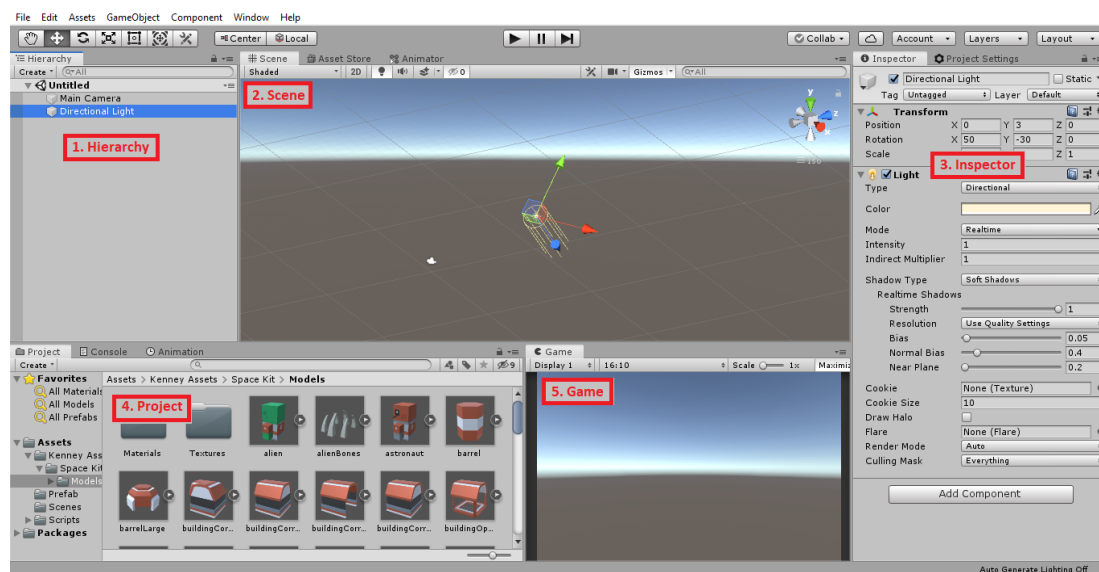


Figura 2.3. Unity Editor

2.3.1. Hierarchy

É a janela onde estão listados os objetos da cena de forma hierárquica. Essa hierarquia, agrupa objetos onde um objeto pai pode ter n objetos filhos, dessa forma, alterações no nó pai irão influenciar nos filhos também. É nessa janela que adicionamos e removemos objetos na cena, e é ainda uma boa opção para acessar rapidamente um objeto que muitas vezes não está tão facilmente visível no cenário. Vamos adicionar um cubo a cena, para isso clicamos com o botão direito do mouse em uma parte vazia da janela Hierarchy e vamos em 3D Object → Cube, veja a figura abaixo.

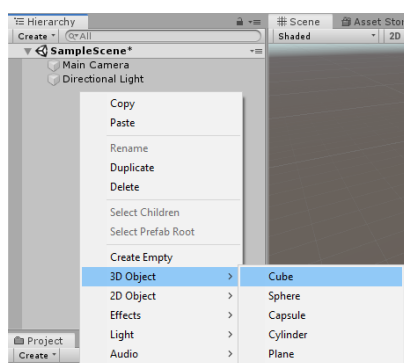


Figura 2.4. Adicionando um Cubo

2.3.2. Scene

A janela Scene serve para edição de cena do jogo. Inicialmente a cena é mostrada como um espaço vazio apenas com uma câmera e uma luz e é nela que você vai adicionando os objetos do jogo. Você pode ver essa edição de cena como se tivesse editando um cenário de cinema que a partir de um espaço vazio constrói-se toda uma cena para os atores. Então é definido por exemplo onde e em que ângulo ficarão a TV e o sofá, como será a iluminação e se lá fora está chovendo ou não.

As ferramentas de controle de objetos da scene view que são representadas pelos botões da imagem abaixo são as seguintes: Q(mover-se pela cena), W(mover objeto da cena), E(rotacionar objetos), R(redimensionar objetos), T(redimensionar objetos baseados em um retângulo). Para navegar na cena, use o mouse e o scroll do mouse que ajudarão muito a ver os objetos em diferentes distâncias e tamanhos. A imagem abaixo mostra os ícones de cada ferramenta de edição e objetos na cena.

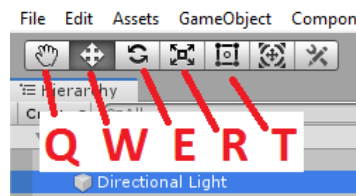


Figura 2.5. Ferramentas de Edição

2.3.3. Inspector

Nesse painel você pode ver, editar e adicionar comportamentos aos objetos da cena e dos arquivos do projeto. Esses comportamentos que são adicionados aos objetos são conhecidos por Componentes e eles podem ser um dos vários já oferecidos pela Unity ou componentes definidos pelo programador através de scripts. Na Unity, um script que estende da classe MonoBehaviour é reconhecido como um novo componente. No seu projeto, uma imagem Player.png por exemplo pode ser adicionada a um objeto da cena através do componente SpriteRenderer que vem com uma série de opções para o uso de imagens no jogo. Durante o curso iremos adicionar components a objetos alguma vezes e esse processo será ilustrado em breve.

2.3.4. Game

A janela Game mostra como o jogo realmente está. Diferente da janela Scene, nessa você não pode editar os objetos e sim ver como estão de fato os objetos no jogo. A posição, tamanho e outros aspectos da imagem do jogo nessa janela irá depender das configurações da câmera. Temos ainda as opções de executar o jogo e testar seu funcionamento, congelar o jogo e passar o jogo frame por frame permitindo testar seu jogo em diferentes resoluções.

2.3.5. Project

É nessa janela que podemos ver e organizar os arquivos do projeto de forma semelhante aos diretórios de um sistema operacional. Esses arquivos podem ser imagens, sons, animações, scripts, texturas, entre outros. Se um arquivo do seu computador for compatível com a Unity, você pode apenas arrastar e soltar ele no painel Project que ele será importado para o projeto.

Você pode ainda adicionar um pacote da Unity ao projeto importando diversos recursos de uma só vez. Vamos importar um pacote de assets do site Kenney.nl, um site muito bom que disponibiliza assets de forma gratuita para uso pessoal ou comercial (licença CC0). Primeiro vamos baixar e extrair o pacote disponível no link <https://www.kenney.nl/assets/nature-pack> então, após extraído vá até o diretório \Unity

package e com a Unity aberta no projeto execute o arquivo NaturePack. Na Unity irá aparecer a lista dos assets que serão importados, basta clicar em Import e aguardar. Assim que finalizada a importação, você poderá encontrar a pasta Nature pack com os assets importados, abra essa pasta, ela deverá estar semelhante a imagem abaixo. Caso a listagem de objetos esteja diferente, de acordo com a imagem abaixo você pode mudar o layout no botão de menu que está com destaque vermelho e selecionar a opção Two Column Layout.

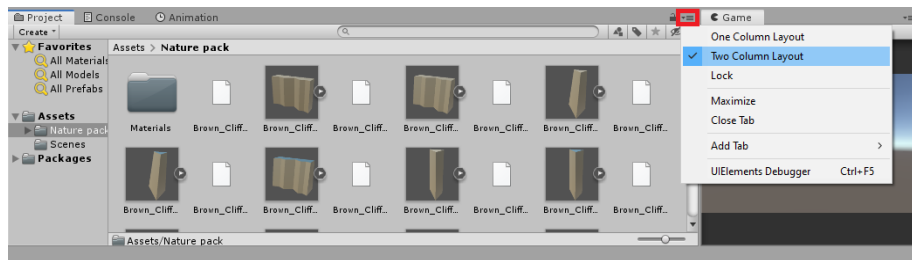


Figura 2.6. UnityPackage Importado

Você também pode importar um só arquivo apenas arrastando esse arquivo para a janela Project ou adquirir seus assets da Unity Asset Store. Para baixar da Asset Store você vai no menu Window → Asset Store e será aberto a janela da loja de assets. Nessa janela, procure pelo asset “Low Poly Police Car Pack” ele é gratuito, então apenas clique em Baixar e após baixar o mesmo botão irá virar o botão Import, ao clicar em Import o processo será semelhante a importação do pacote anterior. A imagem abaixo irá ilustrar um pouco esse processo.

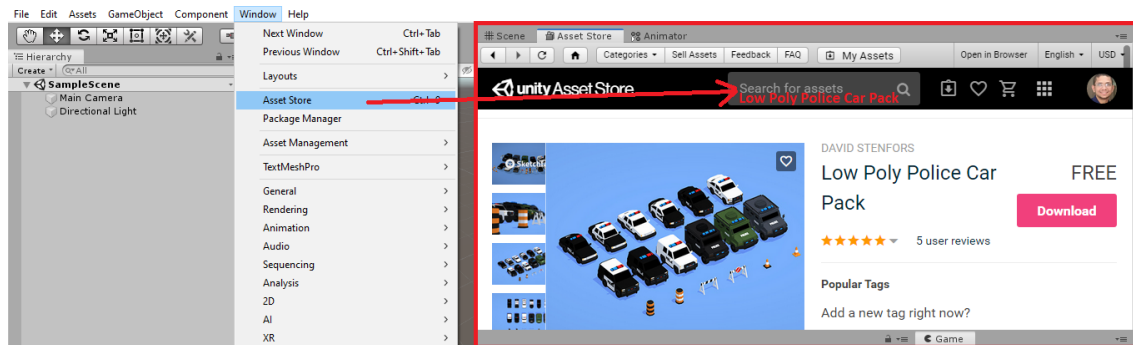


Figura 2.7. Usando a Asset Store

2.4. Scripts na Unity

Nos nossos projetos, muitas vezes precisamos definir ações e características aos game objects do jogo que não vêm prontos na Unity, ou mesmo precisamos definir isso de formas diferentes ao que a Unity oferece. Para definir novos comportamentos e adicioná-los aos objetos da cena, usamos códigos escritos em linguagem C# onde podemos programar por exemplo um personagem para correr e pular, ou para recuperar vida quando pegar um item de cura. Na Unity, para que um script seja adicionado a um

game object como um componente que adiciona comportamentos, esse script deve estender da classe MonoBehaviour. Vamos antes entender um pouco mais sobre isso, primeiro analise a chamada dos métodos Awake, Start, Update e FixedUpdate da imagem abaixo.

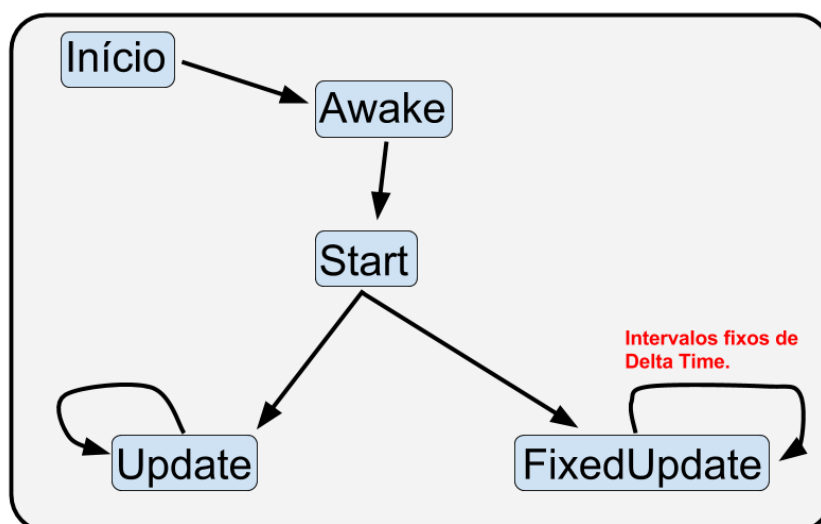


Figura 2.8. Script MonoBehaviour

Em loop a Unity passa várias vezes por segundo por todos os MonoBehaviours ativos do jogo chamando suas funções, então assim que o jogo é iniciado ou um game object é instanciado, o método Awake é chamado, os game objects são criados, então o método Start é chamado. Após isso a Unity chama em loop os métodos Update e FixedUpdate. A diferença entre eles é que no Update devem ser colocadas coisas não relacionadas a física do jogo e no FixedUpdate deve ser colocado tudo o que envolve a física do jogo. Isso porque colocando no FixedUpdate você garante que a física do objeto só receba alterações depois que todo o cálculo necessário da engine de física da Unity tenha concluído o que evita erros. Além dessas funções básicas, temos diversas outras que podem ser usadas, entre elas OnEnable(), OnDestroy(), OnMouseOver(), entre outros. Essas funções não são obrigatórias, são funções de callback muito úteis quando se deseja que algo aconteça naquele momento. A partir de um script podemos também acessar e manipular os outros componentes dos game objects, veremos isso algumas vezes durante esse curso.

2.5. A Física da Unity

A Unity implementa uma física semelhante a física do mundo real e essa implementação pode ser adicionada a objetos através do componente Rigidbody (no caso de jogos 2D usa-se o Rigidbody2D). Você pode modificar de diversas formas os valores que influenciam na física do jogo, por exemplo modificar a massa dos objetos, aceleração da gravidade, coeficientes de atrito e elasticidade entre outros. Vamos iniciar a cena colocando um chão e um carro. Primeiro, com um cubo adicionado a cena, no componente Transform (janela Inspector) defina os seus valores de posição e rotação para (0, 0, 0) e seus valores de escala para (30, 1, 30). Se o cubo estiver muito grande,

afaste a visualização da cena usando o mouse. O cubo deve agora estar no formato de uma plataforma como na imagem abaixo.

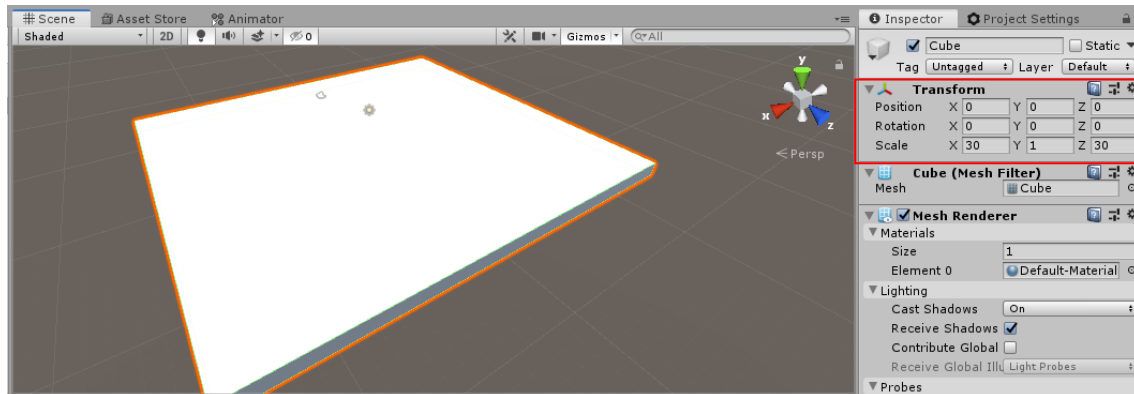


Figura 2.9. Criando um Chão

2.5.1. Rigidbody

Vamos agora importar o carro para a cena e fazer ele cair com a gravidade. Na janela Project vá até o diretório “Low Poly Police Car Pack\ Prefabs\ SWAT Cars” e arraste o Swat Car 2 (ou outro se preferir) para o painel Hierarchy ou diretamente para a cena. O carro estará muito grande, então atualize os valores do componente Transform segundo a imagem a baixo. Além disso, para que ele responda a física da Unity e caia com a gravidade, precisamos adicionar dois components o Box Collider e o Rigidbody em Add Component → Physics. Lembre-se que não é o Physics 2D pois esse é um jogo em três dimensões. Esses componentes irão adicionar características configuráveis de um objeto que responde a física do jogo. Veja a imagem abaixo.

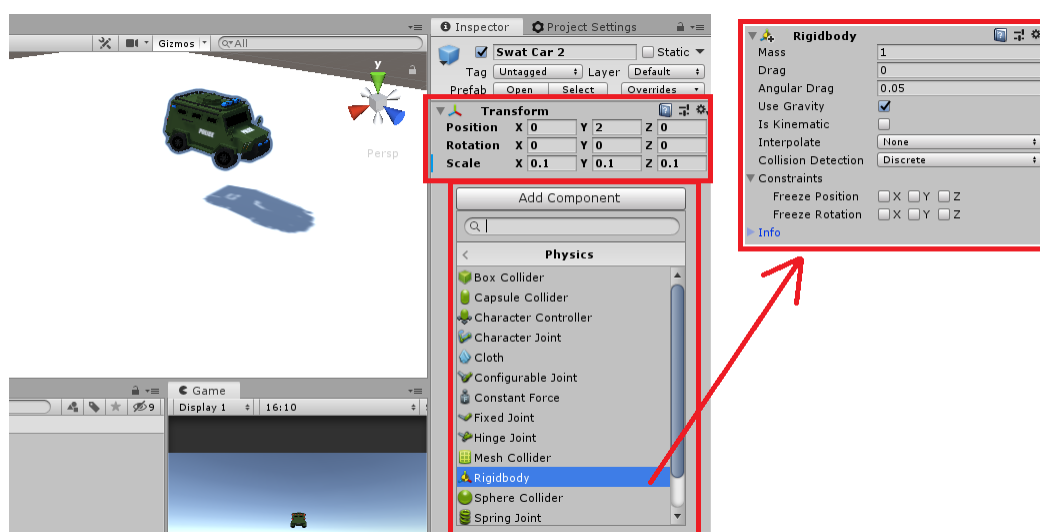


Figura 2.10. Configurando o Carro

Agora execute o jogo e verifique se o carro está caindo corretamente. Lembre-se que o chão é um objeto estático e que não deve receber o Rigidbody. A física das engines implementam a colisão entre objetos usando colisores que podem ter diversas formas onde as mais comuns delas são Retângulos ou Círculos para jogos 2D e Paralelepípedos ou Esferas para jogos 3D. Por isso foi necessário usar um BoxCollider no carro, se fosse colocado um Sphere Collider por exemplo o carro poderia girar como uma bola pois a física segue o formato do colisor e não o formato do objeto. Podem ser usados mais de um colisor para buscar se aproximar mais do objeto ou usar um colisor em malha que são computacionalmente muito caros e devem ser evitados. A imagem abaixo ilustra um pouco esse comportamento. Perceba que a bola azul é mais leve e que tem uma certa elasticidade, o que é diferente da bola cinza. Esses comportamentos são simulados pela engine de física.

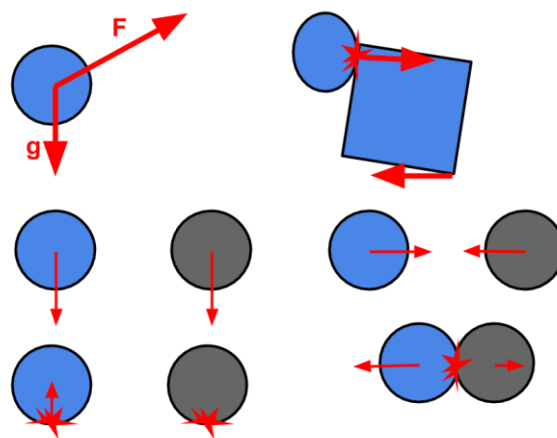


Figura 2.11. Simulações da Física

Vamos agora criar o script do carro e fazer ele se mover pela cena de acordo com os comandos do jogador. Na janela Project, clique com o botão direito do mouse para criar um novo diretório chamado Scripts, dentro desse diretório crie o script CarController.cs e abra o arquivo para editarmos ele. O novo script já é criado herdando de MonoBehaviour e com as funções Start() e Update() prontas para serem implementadas. Vamos alterar o script de forma que ele fique como o código abaixo.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CarController : MonoBehaviour {

    public float Speed;
    public float TurnSpeed;

    private Rigidbody _rgdb;

    // Start is called before the first frame update
    void Start() {
        _rgdb = GetComponent<Rigidbody>();
    }

    // Update is called once per frame
```

```

//void Update()
void FixedUpdate() { //Aqui vamos usar coisas relacionadas a física
    MoveCar(Input.GetAxis("Horizontal"), Input.GetAxis("Vertical"));
}

private void MoveCar(float horizontalAxis, float verticalAxis) {
    _rgdb.angularVelocity = Vector3.up * horizontalAxis * TurnSpeed;
    _rgdb.velocity = new Vector3(0, _rgdb.velocity.y, verticalAxis * Speed);
}
}

```

Observe que foram criadas as variáveis públicas Speed e TurnSpeed para que a partir delas possamos definir a velocidade e o giro para manobrar o carro. A variável _rgdb é inicializada na função Start() onde a função GetComponent() pega do objeto associado o primeiro componente do tipo informado, no caso o primeiro Rigidbody que a função encontrar. No FixedUpdate é chamado a função MoveCar passando por parâmetro os valores dos Inputs das direcionais.

Quando um projeto é criado, os eixos Horizontal e Vertical usados no código acima já vêm configurados para os botões WASD e para as quatro setas do teclado. Então pressionando para a esquerda ou direita você altera o valor do eixo Horizontal de -1 para 1, se não pressionar nenhuma dessas duas direções o valor volta para 0. De forma semelhante, se for pressionado para baixo ou para cima o eixo Vertical altera de -1 a 1, sendo 0 se não for pressionado um botão desse eixo.

Então, os valores dos eixos direcionais são passados para a função MoveCar() que pega o valor do eixo Horizontal (botões esquerda ou direita) e altera o valor da velocidade angular do rigidbody no eixo Y pois Vector3.up é o vetor (0, 1, 0). Já o valor do eixo Vertical (botões para cima ou para baixo) é usado para alterar a velocidade do rigidbody no eixo Z. Para entender melhor essas direções, é importante conhecer os eixos X, Y e Z da cena. É importante saber como vai ser o comportamento de um objeto quando é movido em um certo eixo ou como será a rotação quando rotacionado em torno de outro eixo. Por exemplo, a rotação do carro acima é em torno do eixo Y pois na cena o eixo Y está para cima então. Para entender essa rotação você deve imaginar que esse eixo “fure” o objeto de cima a baixo e gire como se fosse um espeto, a imagem abaixo ilustra essa rotação.

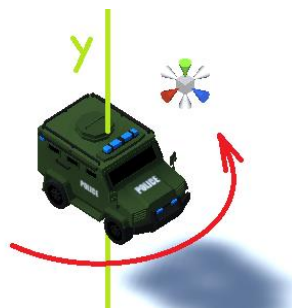


Figura 2.12. Rotação de Objetos

Uma rotação positiva acontece no sentido horário para quem está de frente ao eixo, uma rotação negativa acontece de forma anti-horária. Mas para entender melhor os

conceitos de translação, rotação e escala, vá até o Unity Editor e altere os valores do componente Transform do objeto e observe o que acontece.

Vamos voltar ao Unity Editor e testar o projeto. Primeiro adicione o script CarController usando o botão Add Component e indo na seção Scripts. Certifique-se de que o carro tenha os componentes Rigidbody, BoxCollider e o CarController, você pode ainda ajustar o BoxCollider clicando em Edit Collider (1 na imagem abaixo). Para finalizar defina os valores de Speed e TurnSpeed no componente CarController (2 na imagem abaixo) e por fim execute o jogo (3 na imagem abaixo). O carro deverá cair e se mover quando pressionar os botões de direção WASD ou as setas.

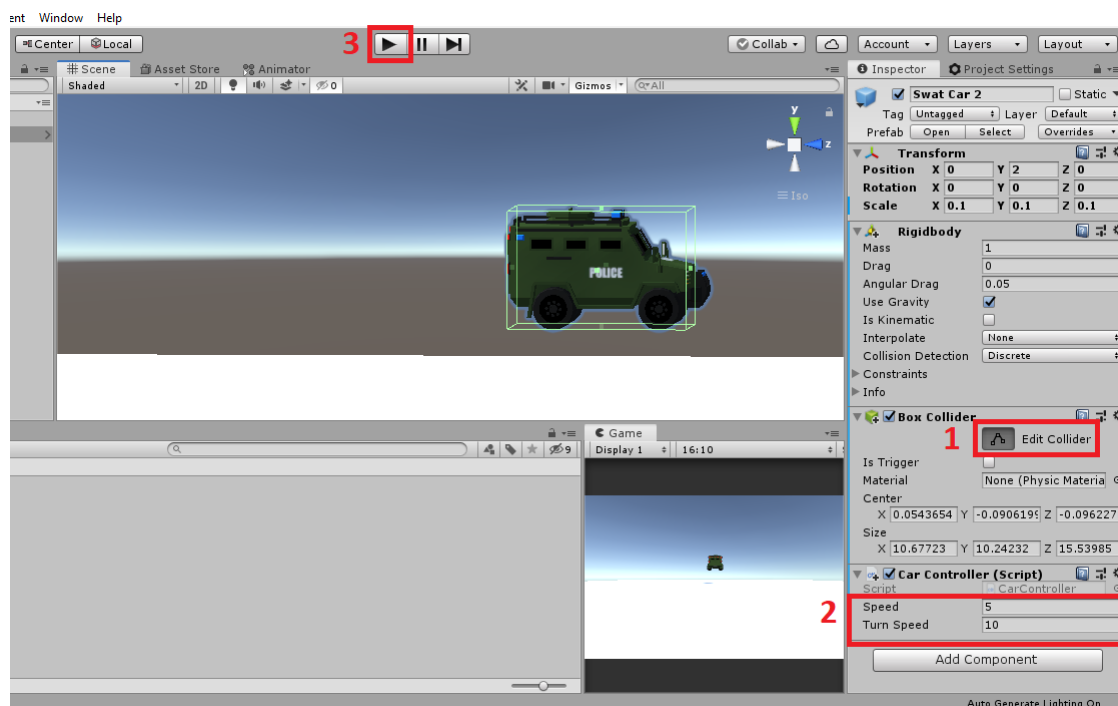


Figura 2.13. Testes da Física

Você já deve ter percebido que ao andar para frente ou para trás ele vai sempre na mesma direção, mesmo com o carro virado para lados diferentes. Isso acontece porque a velocidade definida no Rigidbody do objeto está sendo definida no eixo Z global (eixo Z da cena). Quando o carro gira, seus eixos locais giram juntos mantendo a frente do carro sempre para a frente dele. Porém como o Rigidbody não usa os eixos locais do carro, e sim o global da cena, precisamos calcular para onde o carro está virado pois essa direção muda sempre que o carro vira para algum lado. Inicialmente sua direção é a mesma do eixo Z então vamos usar essa direção como referencial de ângulo de rotação 0°. O giro do carro é feito em torno do eixo Y, então vamos usar a rotação em Y para calcular as direções X e Z, a partir do ângulo em torno de Y. Lembrando que, como a rotação é em torno de Y, a direção nesse eixo não altera. Então vamos mudar a função MoveCar() de acordo com o código abaixo.

```
private void MoveCar(float horizontalAxis, float verticalAxis) {  
    float dirX = Mathf.Sin(Mathf.Deg2Rad * transform.eulerAngles.y);
```

```

float dirZ = Mathf.Cos(Mathf.Deg2Rad * transform.eulerAngles.y);
float intensity = Speed * verticalAxis;

_rgdb.angularVelocity = Vector3.up * horizontalAxis * TurnSpeed;

_rgdb.velocity =
    new Vector3(dirX * intensity, _rgdb.velocity.y, dirZ * intensity);
}

```

O código como está acima vai direcionar corretamente o carro no plano XZ, como o cálculo está sendo feito apenas no eixo Y para rotações fora desse plano a direção não vai funcionar corretamente. Porém, como no nosso caso o carro está girando apenas em torno do eixo Y nosso código vai funcionar corretamente. Vamos agora colocar alguns objetos no cenário, para isso use os assets do diretório Nature pack. Para organizar melhor, na janela hierarchy clique com o botão direito e em Create Empty para criar objetos vazios que irão guardar um grupo de objetos como filho deixando a hierarchy mais organizada. Lembrando que além de colocar alguns elementos na tela, você deve também colocar os colisores neles. Você pode ainda ajustar a câmera do jogo para que dê uma boa visão do cenário. Veja a imagem abaixo.

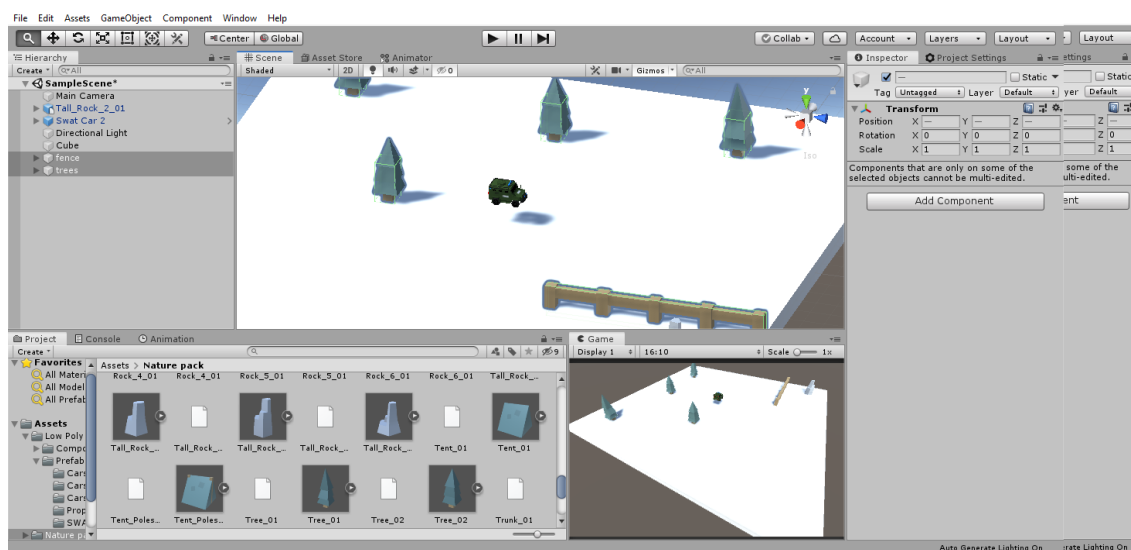


Figura 2.14. Melhorando o cenário

2.5.2. Usando colisores como gatilhos

Como você já viu, os colisores são usados para que a física do jogo entenda quando dois objetos se chocaram, esse choque leva a uma colisão que na Unity é uma classe que carrega informações como objeto A e B, velocidade relativa, entre outros. Mas há uma forma de se usar colisores como gatilhos, nesse caso a colisão não vai acontecer, a Unity apenas vai verificar se os colisores se encontraram e eles atravessam sem que haja o efeito de colisão. Isso é útil por exemplo para moedas ou itens coletáveis onde você quer saber quando o personagem tocou neles para gerar a ação desejada, mas esses objetos coletáveis não vão colidir com o player alterando sua velocidade e direção como acontece por exemplo ao se chocar com uma parede.

Vamos então começar criando um item para o jogador coletar, nesse jogo ele deverá coletar todos para terminar a fase. Você pode usar uma primitiva como uma esfera ou um dos prefabs disponíveis nos pacotes de assets que importamos. Adicione esse objeto a cena, adicione um colisor e marque a opção “Is Trigger” no inspector. Veja a imagem abaixo.

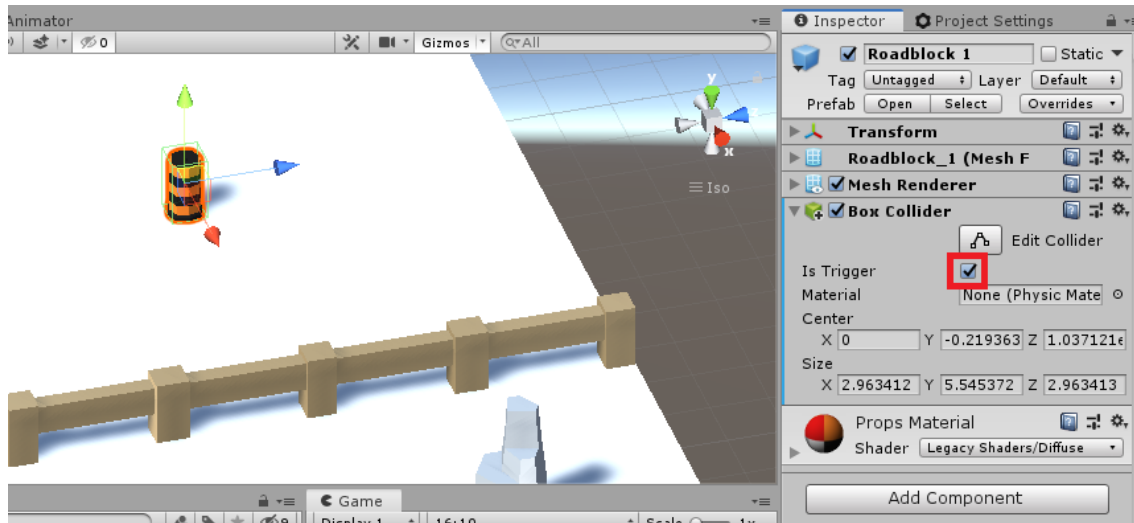


Figura 2.15. Item coletável

Agora crie um novo script chamado ItemController seguindo o código abaixo. Esse script irá verificar quando o jogador tocou o item fazendo ele desaparecer da tela. Não esqueça de adicionar esse script ao GameObject item na janela inspector, execute o jogo, com o carro vá até o item e colete-o. Para usar o colisor como Trigger use o código abaixo.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ItemController : MonoBehaviour {

    private void OnTriggerEnter(Collider other) {
        CarController c = other.gameObject.GetComponent<CarController>();
        if (c != null)
            Destroy(this.gameObject);
    }
}
```

Agora, quando o carro tocar no item ele desaparece. Veja que o script tem apenas a implementação da função OnTriggerEnter. Para esses testes com colidores temos três funções para colisões e 3 para triggers, são eles: OnCollisionEnter, OnCollisionStay e OnCollisionExit para colisões, OnTriggerEnter, OnTriggerStay e OnTriggerExit para triggers e as mesmas 6 para quem está usando a física 2D, no caso de OnCollisionEnter2D até OnTriggerExit 2D.

2.6. Prefabs

Como teremos alguns desses objetos coletáveis na cena com características parecidas, a Unity permite criar prefabs que são objetos pré fabricados onde podemos reusar colocando vários deles na cena e eles dependerão de uma arquivo prefab, então as alterações feitas nesse arquivo prefab irá alterar os objetos da cena que estão ligados a ele. Por exemplo, no seu jogo, as fases têm várias moedas, então um dia você resolve mudar o desenho da moeda. Se todas as moedas na cena estiverem ligadas a um prefab, basta mudar a imagem no prefab que todas as moedas serão atualizadas. Vamos criar o prefab do item coletável que acabamos de criar. Crie a pasta prefabs na janela project e abra, selecione o item coletável na janela hierarchy e arraste para a janela project. Se aparecer uma janela nesse momento, clique em Original prefab. O objeto irá aparecer no diretório da janela project, agora você pode arrastar esse prefab para a cena ou para hierarchy criando cópias dele e espalhando pela fase. A imagem abaixo ilustra essa parte.

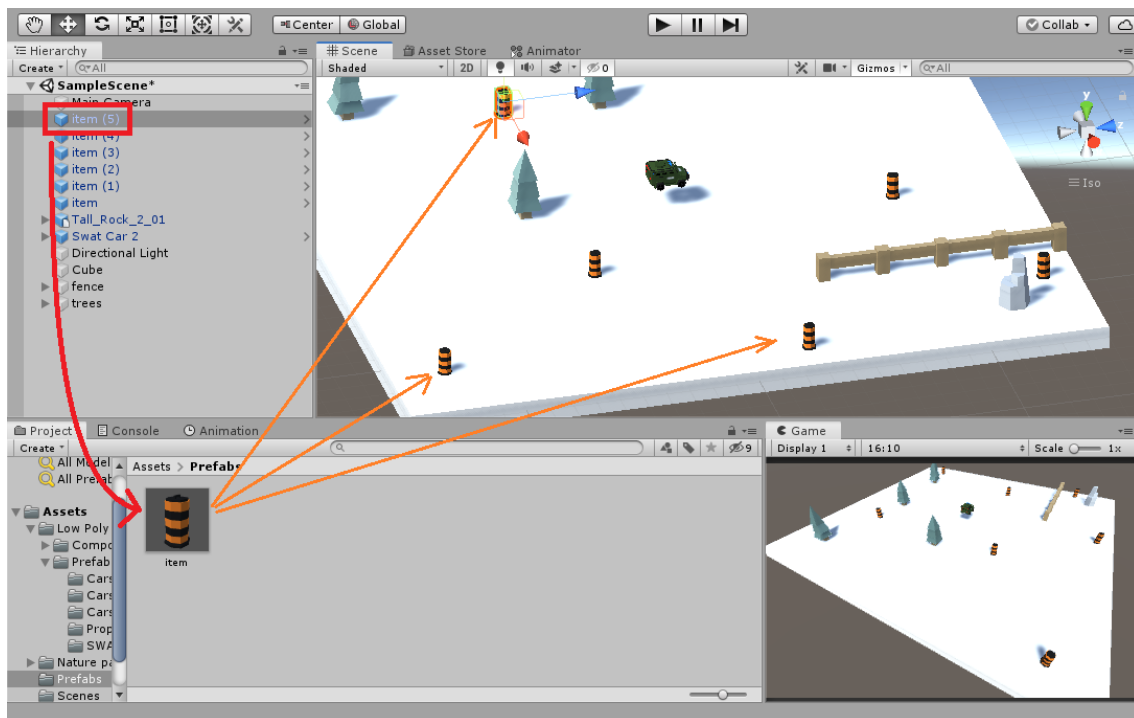


Figura 2.16. Prefab item coletável

Com o prefab criado, se você selecionar ele e ir no inspector mudar as suas características, essas vão ser alteradas em todos os outros que estão sendo usados na cena. Vamos agora colocar no mapa pontos de criação de bombas, essas bombas vão ser criadas de tempo em tempo e vão ao encontro do jogador. Vamos criar o primeiro e então criar o prefab dele. Crie um objeto vazio, posicione em algum lugar do mapa e crie o script BombsCreatorController.cs, adicione o script a ao objeto vazio criado. Em seguida crie uma esfera, crie o script BombController.cs e adicione o script a essa esfera. Certifique-se ainda de que a esfera tenha um colisor e adicione um Rigidbody a ela. Por fim crie o prefab da esfera e delete o gameObject esfera da cena. Após criar a

esfera, ela ficou muito grande, então diminui seu tamanho para (0.5f, 0.5f, 0.5f). Veja a imagem abaixo.

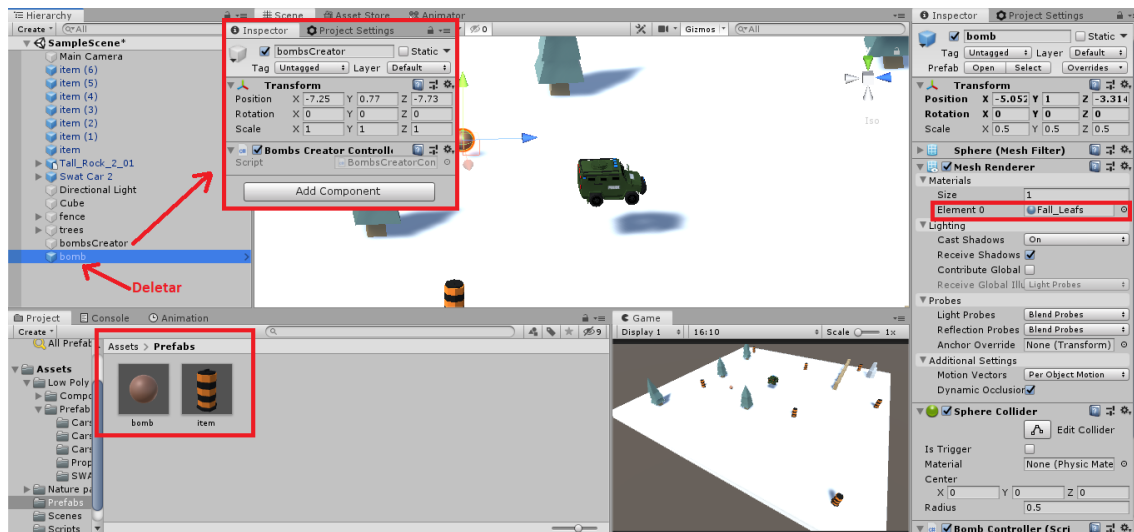


Figura 2.17. Criando as Bombas

Verifique na imagem acima que os dois objetos foram criados, um é o objeto bombsCreator apenas com o Transform e o seu script, o outro é a bomba em formato de esfera. Outra coisa que foi alterada na bomba foi o seu material, como ela veio da mesma cor do chão e ficou ruim de ver, fui até o componente Mesh Renderer e alterei o material para um material com outra cor. Verifique também que foi criado o prefab da bomba e que o objeto pode ser deletado da cena para que inicialmente não tenham bombas na tela. Cada objeto recebeu seu respectivo script, vamos agora editar o script BombsCreatorController.cs que está adicionado ao game object bombsCreator.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BombsCreatorController : MonoBehaviour {

    public BombController Bomb;
    public float TimeToCreateBomb;

    // Start is called before the first frame update
    void Start() {
        Invoke("createBomb", TimeToCreateBomb);
    }

    private void createBomb() {
        Instantiate(Bomb, transform.position, Quaternion.identity, null);
        Invoke("createBomb", TimeToCreateBomb);
    }

}
```

Em seguida vamos ao Unity Editor modificar os dois valores das variáveis públicas. O primeiro valor é o prefab bomb que criamos e esse prefab deve ter o script BombController associado a ele. O outro valor é o float que define a quantidade de segundos entre a criação de uma bomba e outra através do método createBomb. Esse método é passado para o método Invoke que chama ele depois de x segundos. Como o final do método CreateBomb chama novamente o método depois de x segundos novamente através do método Invoke(), o create bomb irá ficar criando uma bomba a cada x segundos. Veja a imagem abaixo.

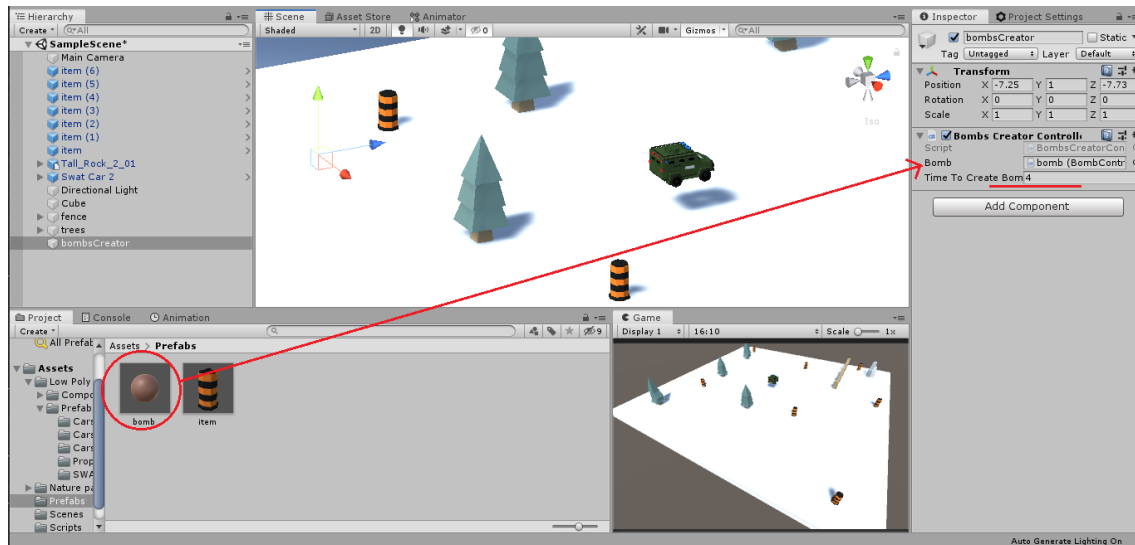


Figura 2.18. Configurando o criador de bombas

Agora, com o gerador de bombas criado, vamos criar um prefab a partir dele assim poderemos a partir de um prefab espalhar o gerador de bombas pela fase. Em seguida vamos implementar o script da bomba para que ela possa se mover em direção ao player quando for iniciada. Ela não vai seguir o player pois a ideia é que tenham bombas vindo em direção ao jogador que terá que evitar a colisão com as bombas enquanto pega os itens para completar a fase. Vamos implementar o script da bomba segundo o código abaixo.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BombController : MonoBehaviour {

    [SerializeField]
    private float _speed;
    [SerializeField]
    private float _timeToRemove;

    private Vector3 _direction;
    private Rigidbody _rgdb;

    void Start() {
        CarController player = FindObjectOfType<CarController>();
        _rgdb = GetComponent<Rigidbody>();
    }
}
```



```

    _direction = player.transform.position - transform.position;
    _direction = _direction.normalized; //Vetor de tamanho 1
}
void FixedUpdate() {
    _rgdb.velocity = _direction * _speed;
    _timeToRemove = _timeToRemove - Time.fixedDeltaTime;
    if (_timeToRemove < 0)
        Destroy(gameObject);
}
}

```

Você pode verificar no código acima que a variável `_speed` e `_timeToRemove` são `SerializeField`, o que faz com que a Unity mostre esses campos no inspector. Para encontrar o jogador na cena usou-se o método `FindObjectOfType()` para o tipo `CarController` que é um método que busca um objeto desse tipo na cena. Como terá apenas um `CarController` na fase, o objeto que o método retornar será o correto. Para representar visualmente o criador de bombas, vamos arrastar um objeto 3D dos assets para a hierarchy, colocar esse objeto novo como filho de `bombsCreator`, definir as posições X e Z do componente transform desse objeto filho para zero e ajustar o valor da posição no eixo Y para tocar o chão, no meu caso ficou (0, -0.5, 0). Agora aplique as mudanças ao prefab do `bombsCreator`. Por fim, espalhe alguns prefabs de `bombsCreator` pelo mapa. Coloquei 3 geradores de bomba na fase tornando a fase mais completa. A imagem abaixo ilustra como eu fiz essa configuração da cena.

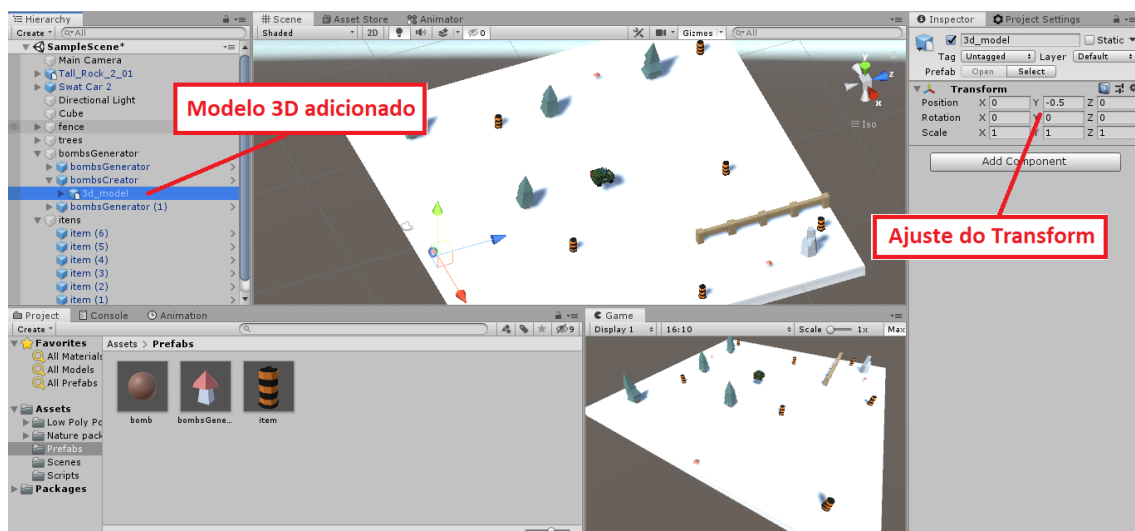


Figura 2.19. Modelo 3D Como Filho do Gerador de Bombas

Verifique que para saber a direção que a bomba vai seguir, é feita a diferença entre a posição do player na cena e a posição da bomba. Como ambas as posições estão no espaço (x, y, z) o vetor resultante da subtração das posições é o vetor direção no espaço 3D. Dessa forma, essa direção funciona em todo o espaço e não apenas em um plano como é o caso do carro que para ir para a frente ele calcula usando seno e cosseno do giro em torno do eixo Y. No caso da bomba, temos o carro para calcular o vetor com a direção e sentido corretos, mas, no caso do carro como poderíamos dessa forma obter o vetor para ele ir para a frente? É bem fácil na verdade. Na janela hierarchy, crie um

objeto vazio (Create Empty) como filho o objeto carro do player e coloque na frente dele que assim você terá sempre uma posição a frente do carro. Então, assim como feito com a bomba, use esse a diferença entre esse objeto e a posição do carro para calcular facilmente no 3D a direção em que o carro está. Essa alteração você não precisa fazer no projeto pois ele já está funcionando corretamente, mas caso você pense em colocar algumas rampas nessa fase, fazendo com que o carro incline, suba morros e outras coisas essa mudança seria uma boa opção. Vamos agora ver melhor esse processo na imagem e no script abaixo.

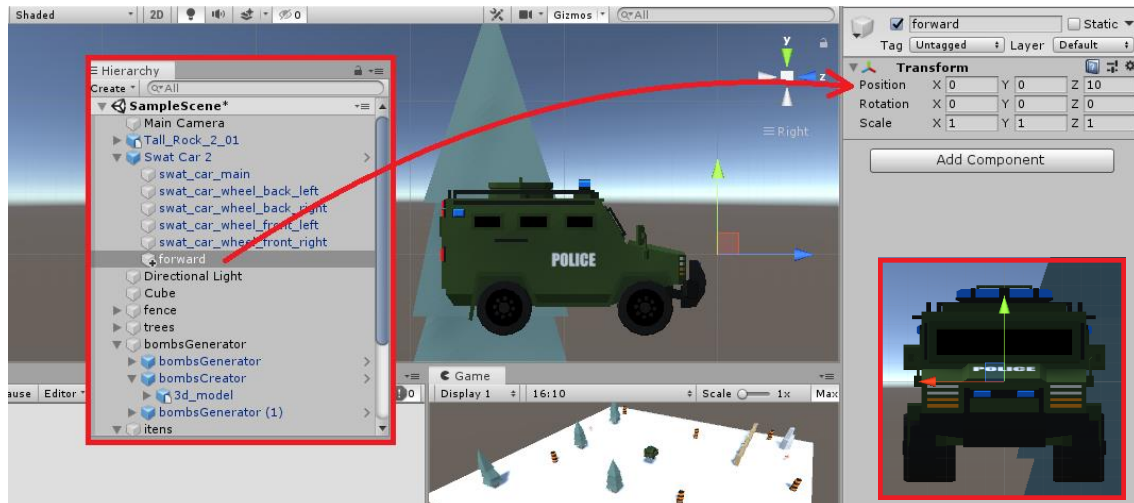


Figura 2.20. Adicionando objeto vazio Forward

Verifique na imagem acima que foi criado o objeto vazio forward como filho do Swat Car 2 e esse objeto está alinhado com o centro do pai mas com um deslocamento para a frente. Logo, se eu fizer a subtração da posição global do forward com a posição global do seu objeto pai, teremos o valor do vetor que aponta sempre para a frente do carro. O código abaixo seria o cálculo do vetor que aponta para a frente e em seguida ele atualiza a velocidade do carro. Veja a alteração abaixo na função MoveCar() do script CarController.cs.

```
private void MoveCar(float horizontalAxis, float verticalAxis) {
    //float dirX = Mathf.Sin(Mathf.Deg2Rad * transform.eulerAngles.y);
    //float dirZ = Mathf.Cos(Mathf.Deg2Rad * transform.eulerAngles.y);

    float intensity = Speed * verticalAxis;

    _rgdb.angularVelocity = Vector3.up * horizontalAxis * TurnSpeed;

    //_rgdb.velocity =
    //new Vector3(dirX * intensity, _rgdb.velocity.y, dirZ * intensity);

    Vector3 forward = (transform.Find("forward").position
        - transform.position).normalized;

    _rgdb.velocity = new Vector3(forward.x * intensity,
        _rgdb.velocity.y, forward.z * intensity);
}
```

Verifique agora que para encontrar a direção da frente do carro é feita a subtração de suas posições usando o `transform.position` de cada objeto. Lembre-se que temos que usar a posição global e não a `localPosition`, pois a `localPosition` é a posição em relação ao pai que no caso do objeto `forward` esse valor não altera com os movimentos e rotações do objeto pai.

2.7. User Interface – Finalizando o Projeto

Vamos agora finalizar o projeto e para isso criamos o menu inicial com a opção de iniciar o jogo e fechar o jogo. Na cena de `gameplay` vamos ter as janelas quando o jogador vence a fase e quando o jogador perde com as opções de tentar novamente e de voltar ao menu inicial. Além disso, iremos verificar quantos itens faltam para o player coletar e terminar a fase. Mas nosso jogo ainda precisa de um menu inicial.

Vamos iniciar criando uma nova cena (`File → New Scene`) salve essa nova cena, para isso vá no menu `File` ou pressione `Ctrl + S` e salve com o nome `MainMenu`. Na janela `hierarchy` crie um novo objeto (`UI → Panel`), dentro desse `Panel` criado, crie dois objetos `UI → Button` como filhos desse `panel`. Dentro de cada `button` tem um `game object “Text”`, mude o texto deles para `Play` e para `Exit` e por fim organize os objetos na tela como preferir. Veja na imagem abaixo um exemplo de como pode ficar.

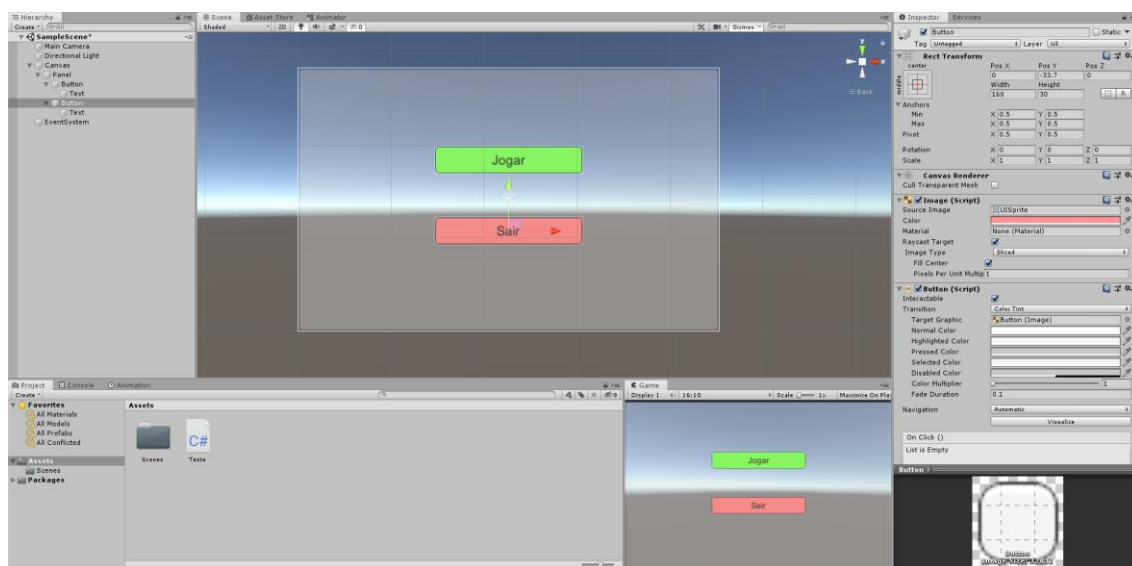


Figura 2.21. Tela de menu principal

A UI se adapta ao tamanho da tela, então quando você executar em uma tela maior ou no dispositivo final, talvez a UI fique ruim de ver ou mal ajustada. Na janela `Game`, ative a opção `Maximize On Play` caso não esteja ativada e tente ajustar corretamente a UI para uma tela maior. Vamos agora criar o script `MainMenu.cs` e adicioná-lo a qualquer `game object` da cena, no caso eu adicionei ao `Painel` criado. Mas antes de implementar as funções `ExitGame()` e `PlayGame()` vamos adicionar as cenas as configurações de `builds` para que a gente possa carregar as cenas no jogo. Para fazer essa configuração vá antes na janela `Project` e vá até o diretório com as cenas de `gameplay` e de `menu`, agora vá em `File → Build Settings...` e a janela `Build Settings` irá aparecer. Essa janela tem o painel `Scenes in Build` onde estão listadas as cenas que serão usadas no jogo. Arraste para esse painel primeiro a cena `MainMenu` e depois a cena de

gameplay que no meu caso é a cena SampleScene. Observação, é importante que a MainMenu seja a primeira cena (a que tem o número 0 a direita), pois assim ao iniciar o jogo, após a splash screen será a primeira cena a ser exibida. Caso você tenha errado a ordem pode arrasta-la para cima para reordenar. A imagem abaixo ilustra essa configuração.

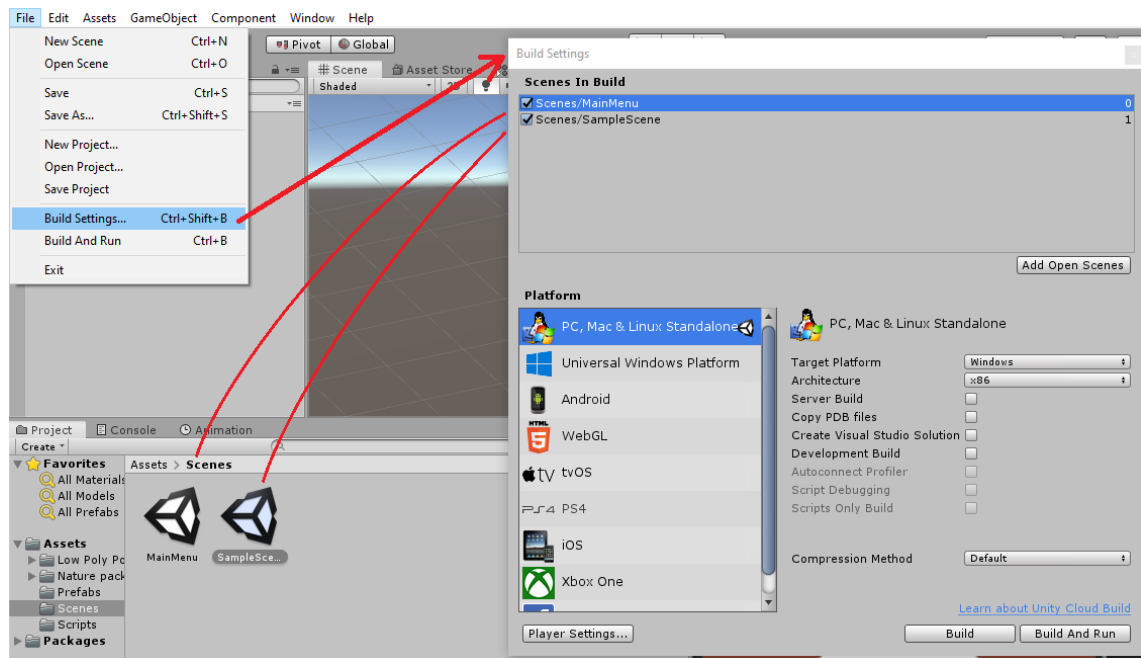


Figura 2.22. Definindo as cenas do Jogo

Agora vamos implementar as funções de iniciar o jogo e sair do jogo no Script MainMenu.cs. Veja o código abaixo.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MainMenu : MonoBehaviour {
    public void PlayGame() {
        UnityEngine.SceneManagement.SceneManager.LoadScene("SampleScene");
    }

    public void ExitGame() {
        Application.Quit();
    }
}
```

Veja que ao chamar o método LoadScene em SceneManager, foi passado o nome da cena de gameplay, no caso “SampleScene”. Vamos agora adicionar essas duas funções aos eventos de click dos botões. Para isso, selecione o botão Play, na janela inspector vá em OnClick() e no espaço vazio coloque o gameObject que recebeu o script MainMenu.cs (no meu caso é o objeto Panel). Selecione uma função para ser

executada ao clicar nesse botão, vá em No Function, selecione MainMenu e selecione PlayGame(). A imagem abaixo ilustra esse processo.

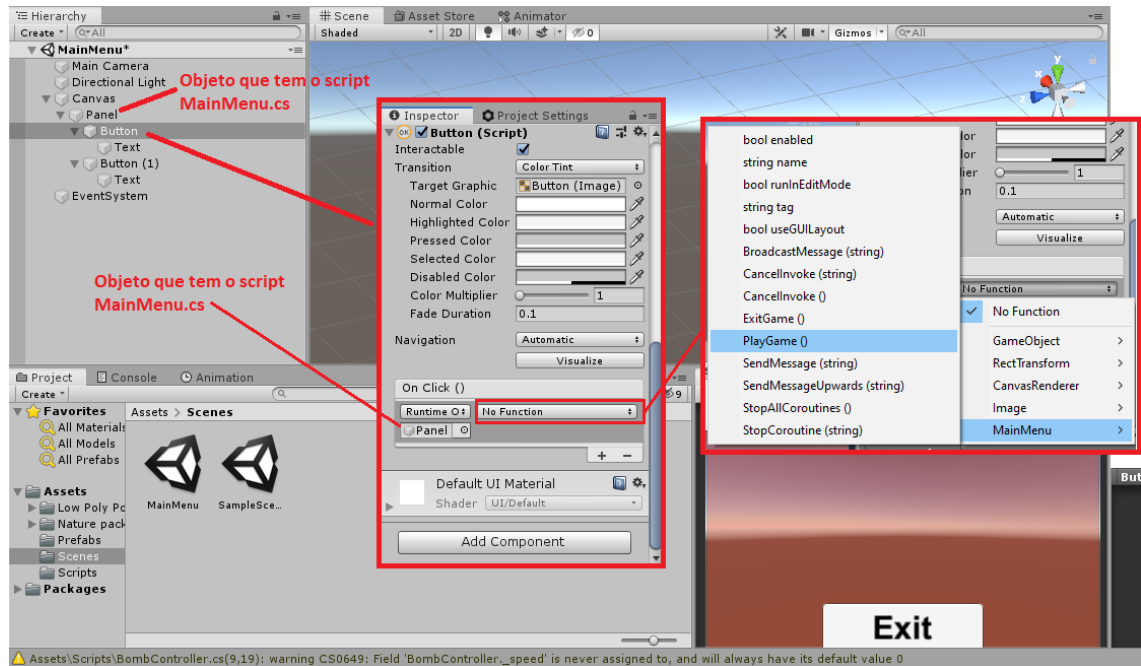


Figura 2.23. Definindo uma ação para no evento OnClick do button

Execute a cena MainMenu e clique no botão Play que criamos e o jogo deverá iniciar. Por enquanto o botão Exit não sai do jogo pois estamos no UnityEditor, mas quando gerarmos a build esse botão irá funcionar corretamente. Agora vamos finalizar a cena de gameplay. Abra a cena de gameplay e crie um painel com os botões Repetir (para caso o jogador queira tentar novamente) e Sair (para voltar ao menu inicial). Em seguida crie o script MenuGameplay.cs. Veja a imagem e o código abaixo.

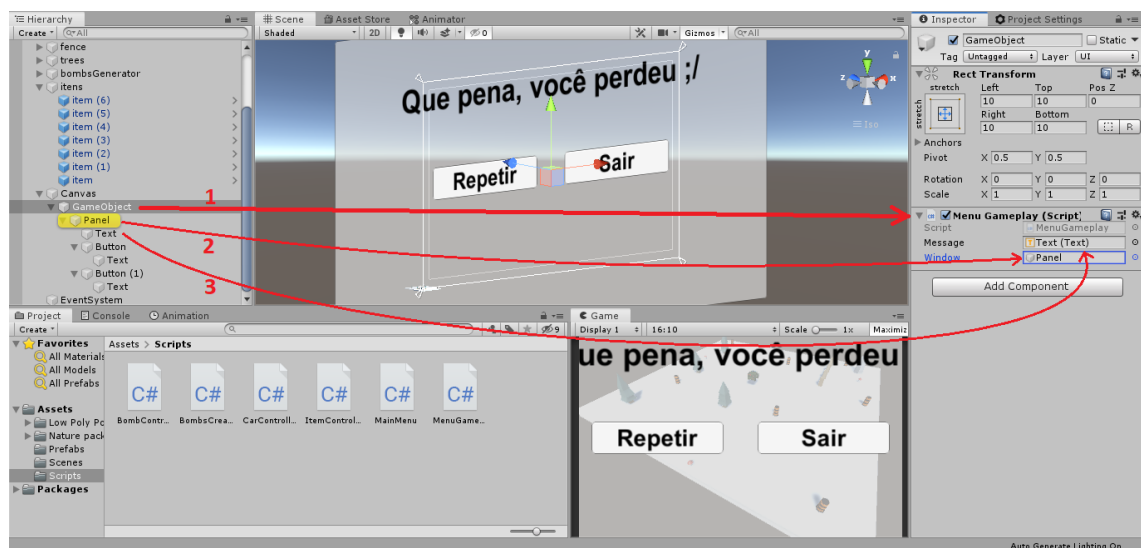


Figura 2.24. Criando a UI da cena de Gameplay

Verifique na imagem acima que temos o (1) GameObject que é um objeto vazio criado e nele foi adicionado apenas o componente MenuGameplay que vem com duas configurações para definir, o Window que é o painel que terá o conteúdo da janela de fim de jogo e o Message que é a mensagem a ser exibida no painel. Essas duas configurações são seus objetos filhos Panel e Text. O script foi colocado no pai da janela e não na janela pois o script irá desabilitar a janela e não queremos que o objeto com o script seja desabilitado. Vamos ver o Script MenuGameplay.cs.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class MenuGameplay : MonoBehaviour {

    public Text Message;
    public GameObject Window;

    void Start() {
        Window.gameObject.SetActive(false);
    }

    public void PlayGame() {
        SceneManager.LoadScene("SampleScene");
    }

    public void Exit() {
        SceneManager.LoadScene("MainMenu");
    }
}
```

Verifique que temos o método Start que desabilita o objeto Window ao iniciar a cena, por isso o Panel como seu filho para que possa ser desabilitado sem prejudicar o funcionamento dos scripts. Agora, vamos verificar se o jogador venceu ou perdeu o jogo, atualizar a mensagem e abrir a tela de fim de jogo. Neste caso, o teste se o jogador venceu ficará no script ItemController.cs modificando a função OnTriggerEnter() e o teste se o jogador perdeu o jogo ficará no script BombController.cs também na função OnTriggerEnter(). Veja os códigos abaixo.

Modificação no script ItemController

```
private void OnTriggerEnter(Collider other) {

    CarController c = other.gameObject.GetComponent<CarController>();

    if (c != null) {

        MenuGameplay window = FindObjectOfType<MenuGameplay>();

        if (transform.parent.childCount == 1 && !window.Window.activeSelf) {
```

```

        FindObjectOfType<MenuGameplay>().Window.SetActive(true);
        FindObjectOfType<MenuGameplay>().Message.text
            = "Parabéns, Você Ganhou!";
    }

    Destroy(this.gameObject);
}
}

```

Modificação no script BombController

```

private void OnTriggerEnter(Collider col) {
    if (col.gameObject.GetComponent<CarController>() != null) {
        MenuGameplay window = FindObjectOfType<MenuGameplay>();

        if (!window.Window.gameObject.activeSelf) {
            FindObjectOfType<MenuGameplay>().Window.SetActive(true);
            FindObjectOfType<MenuGameplay>().Message.text
                = "Que pena, você perdeu ;/";
        }

        Destroy(col.gameObject);
    }

    Destroy(this.gameObject);
}

```

Veja que no teste do script ItemController.cs, é verificado se o pai tem apenas 1 filho. Isso acontece porque eu agrupei todos os Itens em um só objeto pai e esse objeto tem apenas Itens como objetos filhos, logo, quando o último filho foi coletado, ele inicia a janela de fim de jogo como vencedor. Já o teste no objeto Bomb, verifica apenas se a janela não já está ativa. Esse teste com a janela é apenas para caso o jogador já tenha vencido e uma bomba acertou ele depois, então a janela já está como vencedor e não mostra a mensagem de derrota. Agora, semelhante ao que foi feito na tela MainMenu, vamos configurar os botões Repetir e Sair para chamar corretamente as funções para tentar novamente e para sair para a cena de menu que estão no script MenuGameplay.



Figura 2.25. Painel UI da cena de gameplay

2.8. Gerando a Build do Jogo

A Unity permite gerar facilmente as builds para diversas plataformas. Já acessamos a janela de build antes para adicionar as cenas a build e poder mudar de uma cena para a outra. Para gerar a build para computadores vá em File → Build Settings..., verifique se a cena MainMenu é a primeira da lista (com número o 0 a direita), verifique se a plataforma está correta, se o sistema operacional está correto e então clique em Build. Vai abrir uma janela para escolher o local para gerar a build. Após escolher o local é só aguardar a Unity terminar, ir até o diretório escolhido e executar o jogo. Assim que terminar. A imagem abaixo ilustra o processo de geração de build na Unity.

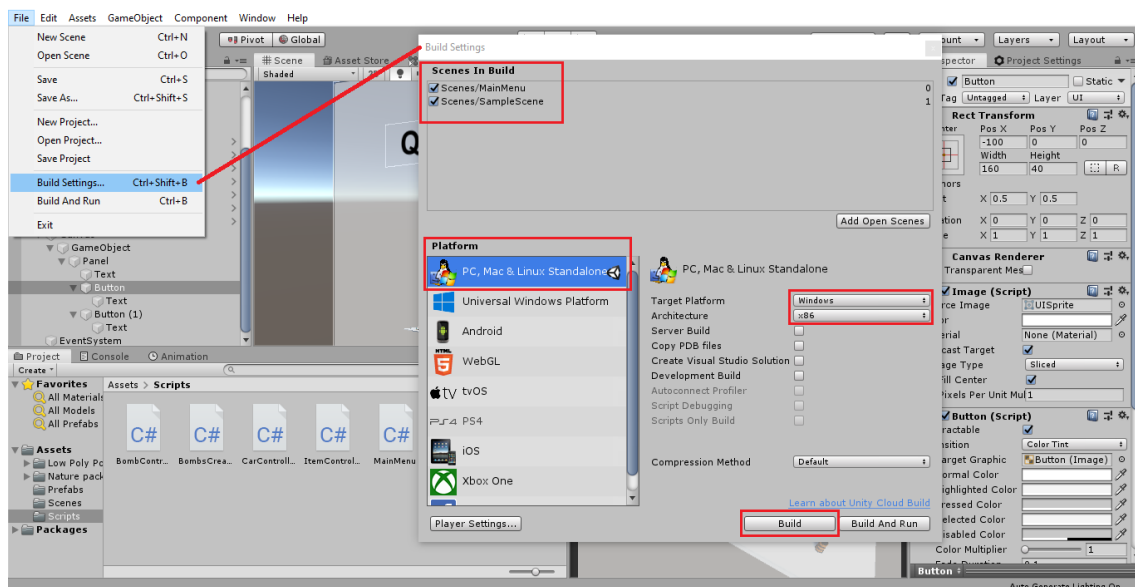


Figura 2.26. Gerando a build

2.9. Considerações Finais

Com a Unity você pode facilmente trabalhar com tecnologias diversas como Realidade Virtual, Realidade Aumentada e Kinect, sem falar que pode gerar projetos para variadas plataformas desde consoles e computadores até smartphones. Muitos recursos podem ser obtidos nas lojas de assets ou em sites que oferecem gratuitamente e auxilia muito caso um pequeno grupo de pessoas ou mesmo uma só pessoa esteja criando seu jogo. Mesmo com essa grande quantidade de ferramentas para auxiliar, desenvolver ainda é um grande desafio pois como é uma ferramenta poderosa e acessível, muitos produtos bons são feitos por pequenas equipes (1 a 3 pessoas por exemplo). Temos ainda as dificuldades que normalmente aparecem na hora de se manter e desenvolver um projeto, manter uma equipe focada e com o mesmo objetivo, entre outros fatores que dificultam o processo de desenvolvimento. Você pode escolher clonar um jogo simples, assim você já sabe o que e como fazer no seu jogo, o que diminui e muito o trabalho com questões de design de games e tomadas de decisões no projeto.

Uma boa opção para quem está começando é participar de um game jam, você pode procurar por uma em sua cidade ou procurar uma online em sites como GameJolt ou Itch.io. Participando de uma game jam você terá no final do evento o protótipo de um jogo e uma equipe formada o que é um bom incentivo para continuar a desenvolver. É bom saber um pouco sobre sua área de desenvolvimento antes de participar de uma game jam, você pode aprender pelo YouTube mesmo ou por cursos online como os da Udemy. Você pode procurar por programação, artes gráficas ou produção de áudio para a área de jogos. Outra coisa muito importante é participar de grupos com desenvolvedores que se ajudam, aqui no Piauí temos o PiauíIndie, um grupo de desenvolvedores de jogos do Piauí.

Gostaria de agradecer a você que chegou até aqui. Espero ter tirado dúvidas e ajudado você. Infelizmente não dá para passar muitas informações nesse tutorial e em cada nova sessão eu sentia falta de explicar muito mais sobre aquela parte, inclusive o código usado aqui não foi feito de forma eficiente. Espero que você tenha se interessado pela área e busque mais, quer tirar mais dúvidas? Pode ficar a vontade para entrar em contato comigo. Muito obrigado.

2.10. Referências

Referências bibliográficas.

References

Jeff W. Murray. (2014) “C# Game Programming Cookbook for Unity 3D”.

Matt Smith, Chico Queiroz (2015), Unity 5.x Cookbook.

Unity, “Learn Unity” <https://unity.com/learn>, December.

Chapter

3

Processamento Digital de Imagens com Python e Scikit-image

Rafael Luz Araujo, Pablo de Abreu Vieira, Romuere Veloso e Silva, Deusimar Damião de Sousa

Abstract

With the increasing popularization of the areas of image processing and computer vision in recent years, research using these techniques to solve various problems has become common, highlighting the area of medicine where there is a growing development of systems that help diagnostics specialists doctors. Thus, this chapter aims to present an introduction to digital medical image processing with the python programming language and some of its main libraries.

Resumo

Com a crescente popularização das áreas de processamento de imagens e visão computacional nos últimos anos, tornaram-se comuns pesquisas utilizando essas técnicas para resolver diversos problemas, destacando-se a área da medicina em que há um crescente desenvolvimento de sistemas que auxiliam especialistas em diagnósticos médicos. Assim, este capítulo tem como proposta apresentar uma introdução ao processamento digital de imagens médicas com a linguagem de programação python e algumas de suas principais bibliotecas.

3.1. Introdução

Com os diversos avanços da ciência e o surgimento de novas tecnologias, a área de aquisição e processamento de imagens ganhou destaque por ser capaz de viabilizar grande número de aplicações. Na área médica ocorreu um crescimento exponencial do desenvolvimento de sistemas automáticos de diagnóstico médico (CAD). Esses sistemas são capazes de auxiliar os médicos em trabalhos repetitivos minimizando erros, com eles é possível medir estrutura anatômicas, monitorar mudanças, diagnosticar e planejar o tratamento dos pacientes [Doi 2005].

Conforme [Marques Filho and Neto 1999] podemos dividir as aplicações de processamento de imagens em duas categorias bem distintas: o aprimoramento de informações pictóricas para interpretação humana (visão computacional) e a análise automática por computador de informações extraídas de uma cena (reconhecimento de padrões).

O Processamento de imagens possui como entrada e saída uma imagem ou quadros de vídeos e tem por objetivo melhorar a imagem para otimizar a extração de informações, facilitando o processo de interpretação dessas imagens [Marengoni and Stringhini 2009]. Como processar imagens é uma tarefa complexa, surgiram diversas bibliotecas que visam auxiliar essa atividade, algumas delas são OpenCV, Matlab e Scikit-image, cada uma delas com suas peculiaridades.

O Matlab necessita a aquisição de licença de custo elevado¹, o OpenCV é complexa e por isso utilizada por desenvolvedores mais avançados². Para esse trabalho foi escolhido o Scikit-image pela sua simplicidade e eficiência³, utilizando a linguagem Python que é amigável e possui uma alta curva de aprendizagem⁴.

O objetivo deste capítulo é apresentar técnicas de processamento de imagens utilizando a linguagem python. Também serão apresentadas algumas das principais bibliotecas utilizadas atualmente como a Scikit-image que é uma coleção de algoritmos para processamento de imagens e está disponível gratuitamente e sem restrições

3.2. Processamento digital de imagens

O tratamento de uma imagem digital, quando realizado por métodos computacionais automatizados, é nomeado de Processamento Digital de Imagens (PDI), que conforme podemos ver em [Marques Filho and Neto 1999] é uma área que já vinha apresentando crescimento expressivo em diversas áreas, já nos anos 2.000, onde o poder computacional só aumentando ela passou a ser utilizada por exemplo na biologia para processar imagens obtidas de microscópios contando o número de um tipo específico de células presente na imagem. Na medicina para auxiliar no diagnóstico automático de doenças. Na Geografia, Sensoriamento Remoto, Geoprocessamento, Meteorologia, dentre muitas outras áreas.

A PDI tem o objetivo de realizar transformações nas imagens afim de melhorá-la para facilitar a extração de informações, sendo uma tarefa complexa que possui cinco etapas fundamentais que formam parte do processamento: aquisição de imagens, pré-processamento, segmentação, extração de características e classificação.

3.2.1. Aquisição

Nesta etapa é realizada a aquisição das imagens, a imagem pode ser transformada em dados digitais por meio de um sensor ou digitalizador. Atualmente existem diversas bases públicas de imagens que podem ser utilizadas.

¹<https://www.mathworks.com/products/matlab.html>

²<https://opencv.org/>

³<https://scikit-image.org/>

⁴<https://www.python.org/>

3.2.2. Pré-processamento

Nesta etapa são realizados diversos tratamentos afim de minimizar defeitos da aquisição da imagem, ou realçar detalhes importantes para o problema em questão. Esse processo visa facilitar a interpretação de informações da imagem.

3.2.3. Segmentação

Nesta etapa, são realizadas delimitações na imagem, afim de isolar a região de interesse. Para isso, podem ser encontrados padrões de agrupamentos na imagem, gerando sub-regiões a serem utilizadas.

3.2.4. Extração de Características

Esta etapa procura extrair características relevantes da imagem. Tem-se como entrada a imagem ou região segmentada da imagem, onde descritores extraem informações de cor, forma e textura e obtém-se como saída um conjunto de dados correspondentes.

3.2.5. Representação e Interpretação

Nesta ultima etapa, representar significa rotular um objeto baseado em suas características. Já a interpretação consistem em atribuir significado a um conjunto de objetos reconhecidos. Como exemplo, tem-se a utilização de aprendizado de máquina para entender os padrões de características e ser capaz de discernir objetos.

3.3. Ferramentas Utilizadas

Como foi visto, a área de processamento de imagens é complexa e envolve várias etapas, para tornar esse trabalho mais simples e minimizar o tempo gasto para realiza-lo, surgiram diversas bibliotecas que trazem diversas funções prontas. Dentre elas o Scikit-image que conforme [Scikit-image 2019] é uma coleção de algoritmos para processamento de imagens. Está disponível gratuitamente e sem restrições, possui código de alta qualidade e com revisão por pares, escrito por uma comunidade ativa de voluntários. Ela inclui algoritmos para segmentação, transformações geométricas, manipulação do espaço de cores, análise, filtragem, operações morfológicas, dentre outras.

Para realizar atividades de análise de dados e processamento de imagens foi utilizado o Anaconda⁵. O Anaconda é uma das plataforma mais populares do mundo, sendo um gerenciador de pacotes gratuito, fácil de instalar, gerenciador de ambiente e Python, com uma coleção de mais de 1.500 pacotes de código aberto com suporte gratuito à comunidade. O Anaconda é independente de plataforma; portanto, é possível usá-lo no Windows, macOS ou Linux. Para instalar, basta acessar o site, realizar o download e executar o instalador na sua máquina [Anaconda 2019].

O Anaconda possui as principais ferramentas utilizadas neste capítulo como o Scikit-image, Jupyter Notebook⁶ e Matplotlib. O Jupyter Notebook é um aplicativo da web de código aberto que permite criar e compartilhar documentos que contêm código ativo, equações, visualizações e texto narrativo. Os usos incluem: limpeza e trans-

⁵www.anaconda.com/distribution/

⁶<https://jupyter.org/>

formação de dados, simulação numérica, modelagem estatística, visualização de dados, aprendizado de máquina e muitos outros [Jupyter 2019].

O Matplotlib é uma biblioteca de plotagem 2D do Python que, com apenas algumas linhas de códigos, é capaz de gerar gráficos, histogramas, espectros de potência, gráficos de barras, gráficos de erros, gráficos de dispersão, etc. O Matplotlib pode ser usado em scripts Python, nos shell Python e IPython, no notebook Jupyter e em servidores de aplicativos da web [matplotlib 2019].

3.4. Manipulação de Imagens

O Scikit-image é um pacote Python de código aberto onde são implementados algoritmos e utilitários com uso em aplicações de pesquisa, educação e na indústria. É uma biblioteca bastante simples e direta, até mesmo para quem não tem muito conhecimento sobre Python.

A manipulação de imagens inclui tarefas simples como carregar uma imagem, exibi-la em tela, realizar operações como cortar e rotacionar. Pode-se também ver informações sobre a imagem, como suas dimensões, além da possibilidade de segmentar alguma região de interesse na imagem.

3.4.1. Carregar imagem

Para que se possa visualizar ou aplicar operações sobre a imagem precisa-se carregá-la. No Código Fonte 3.1 é apresentado um exemplo de como é feito o carregamento de uma imagem e sua exibição na tela.

```
1 from skimage.io import imread # função do pacote skimage para leitura  
  de imagens  
2 import matplotlib.pyplot as plt # função do pacote matplotlib para  
  mostrar as imagens na tela  
3 path_to_image = "imagem.jpg" # endereço da imagem a ser carregada  
4 img = imread(path_to_image) # leitura da imagem  
5  
6 plt.figure(figsize=(10, 20)) # define o tamanho da janela onde a imagem  
  será mostrada  
7 plt.imshow(img) # mostrando imagem carregada na tela
```

Código Fonte 3.1. Carregando uma imagem.



Figure 3.1. Resultado da execução do Código Fonte 3.1.

Às vezes pode-se necessitar que se carregue n imagens de um mesmo diretório de

uma única vez. Fazer isso manualmente como foi feito anteriormente não é uma tarefa fácil, assim, o Código Fonte 3.2 demonstra uma maneira prática e rápida de se fazer isso.

```
1 from skimage.io import imread_collection # função do pacote skimage
   para carregar n imagens
2 extensao = 'dataset/*.jpg' # endereço das imagens seguido de *.
   extensao_da_imagem para carregar todos os arquivos dessa extensão
3 imagens = imread_collection(extensao) # carregando as imagens
4 len(imagens) # verificando a quantidade de imagens carregadas
5 for imagem in imagens: # iterando sobre a lista de imagens carregadas
6     print(imagem.shape)
```

Código Fonte 3.2. Carregando várias imagens.

3.4.2. Salvar imagem

Após carregar uma imagem e ter aplicado algum tipo de operação, talvez seja necessário salvar para reutilizá-la posteriormente. O Código Fonte 3.3 mostra um exemplo de como isso pode ser feito.

```
1 from skimage.io import imsave # função do pacote skimage para salvar de
   imagens
2 path_to_image = 'imagem_salvar.jpg' # endereço para salvar a imagem
3 imsave(path_to_image, imagem) # salvando a imagem
```

Código Fonte 3.3. Salvando uma imagem.

3.4.3. Canais

As imagens coloridas são compostas por combinações de cores primárias representadas por uma série de códigos. Uma imagem comum como as que conhecemos possuem 3 canais de cores. Já as imagens em níveis de cinza possuem apenas um canal.

```
1 fig = plt.figure(figsize=(15,30)) # Definindo a figura e seu tamanho
2
3 canal_vermelho = imagem.copy() # fazendo copia da imagem original
4 canal_vermelho[..., 1] = 0 # zerando o canal verde
5 canal_vermelho[..., 2] = 0 # zerando o canal azul
6 a = fig.add_subplot(1, 3, 1) # fazendo plot a imagem
7 a.axis('off')
8 a.set_title('Vermelho')
9 a.imshow(canal_vermelho)
10
11 canal_verde = imagem.copy()
12 canal_verde[..., 0] = 0
13 canal_verde[..., 2] = 0
14 b = fig.add_subplot(1, 3, 2)
15 b.axis('off')
16 b.set_title('Verde')
17 imgplot = plt.imshow(canal_verde)
18
19 canal_azul = imagem.copy()
20 canal_azul[..., 0] = 0
21 canal_azul[..., 1] = 0
22 c = fig.add_subplot(1, 3, 3)
23 c.axis('off')
```

```

24 c.set_title('Azul')
25 imgplot = plt.imshow(canal_azul)

```

Código Fonte 3.4. Extrair canais da imagem.

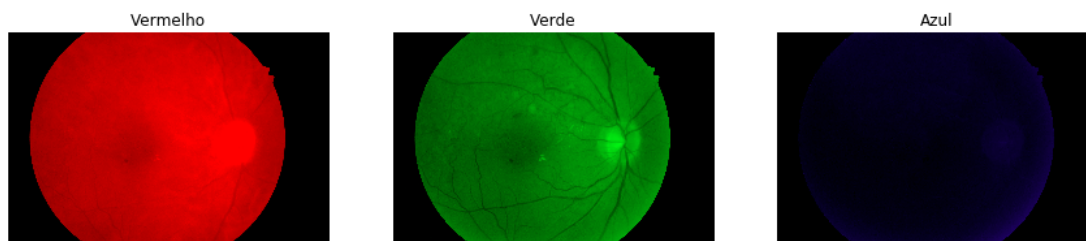


Figure 3.2. Resultado da extração dos canais.

O canal número 0 representa o canal vermelho, o número 1 representa o canal verde e o 2 representa o canal azul. Para que se possa extrair apenas um desses canais, é necessário zerar os outros dois. O Código Fonte 3.4 apresenta como é feita essa extração, e a Figura 3.2 o resultado obtido.

3.4.4. Plotar imagens

Existem algumas maneiras de plotar várias imagens de uma única vez, no Código Fonte 3.4 criou-se uma única linha para plotar três imagens, pois nesse caso foi a melhor maneira de exibir as imagens. Porém, pode ser que necessite-se de plotar várias linhas e várias colunas. O Código Fonte 3.5 apresenta duas alternativas para resolver isso.

```

1 linhas, colunas = 2, 2 # definindo quantidade de linhas e colunas
  desejadas
2 # ----- primeira maneira de plotar as imagens -----
3 fig, ax = plt.subplots(linhas, colunas, figsize=(10, 20)) # criando a
  figura
4 fig.subplots_adjust(bottom=0.1, top=0.4) # definindo estilo
5 # definindo cada posição desejada
6 ax[0,0].set_title('Original') # título da imagem
7 ax[0,0].axis('off')
8 ax[0,0].imshow(imagem)
9 ax[0,1].set_title('Vermelho')
10 ax[0,1].axis('off')
11 ax[0,1].imshow(canal_vermelho)
12 ax[1,0].set_title('Verde')
13 ax[1,0].axis('off')
14 ax[1,0].imshow(canal_verde)
15 ax[1,1].set_title('Azul')
16 ax[1,1].axis('off')
17 ax[1,1].imshow(canal_azul)

```

Código Fonte 3.5. Primeira forma de plotar as imagens em colunas.

Na Figura 3.3, é apresentado o resultado dos Código Fonte 3.5 e 3.6. Pode-se observar que foi colocada uma visualização com 2 linhas e 2 colunas. Isso foi feito de duas maneiras diferentes, e ambas produzem o mesmo resultado. A primeira forma é mostrado no Código Fonte 3.5 e a segunda forma no Código Fonte 3.6.

```

1 # ----- segunda maneira de plotar as imagens -----
2 # cria-se uma lista com as imagens para plot
3 imagens = [imagem, canal_vermelho, canal_verde, canal_azul]
4 # cria-se uma lista de títulos para as imagens do plot
5 titulos = ['Original', 'Vermelho', 'Verde', 'Azul']
6 fig = plt.figure(figsize=(10, 20))
7 for i in range(1, colunas*linhas +1):
8     fig.add_subplot(linhas, colunas, i)
9     plt.imshow(imagens[i-1])
10    plt.title(titulos[i-1])
11    plt.axis('off')
12    plt.subplots_adjust(bottom=0.1, top=0.4)
13 plt.show()

```

Código Fonte 3.6. Segunda forma de plotar as imagens em colunas.

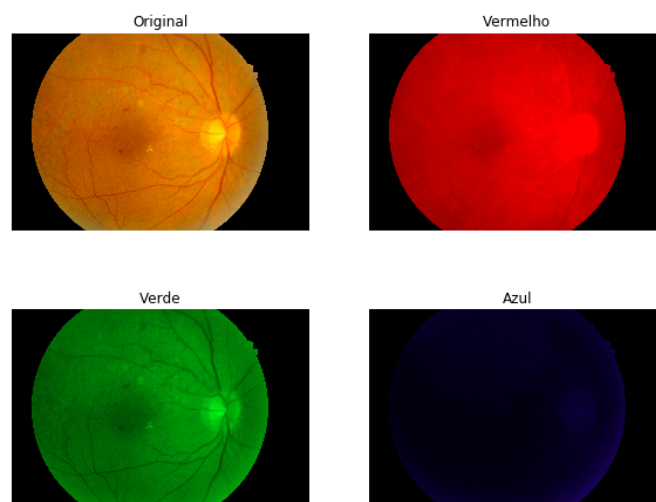


Figure 3.3. Plotando várias imagens de uma vez.

3.4.5. Imagem em níveis de cinza

Há 256 valores possíveis para os pixels de imagens em níveis de cinza. Logo, cada valor equivale a um nível de cinza diferente, que vai desde o preto (0) ao branco (255). Assim, imagens em níveis de cinza possuem apenas uma dimensão.

O Código Fonte 3.7 apresenta como converter uma imagem comum para imagem em níveis de cinza com uma biblioteca disponível no Scikit-image. Na Figura 3.4, é mostrado o resultado da conversão para os níveis de cinza.

```

1 from skimage.color import rgb2gray # função para converter imagens rgb
   para níveis de cinza
2 imagem_cinza = rgb2gray(imagem) # convertendo a imagem
3 fig, ax = plt.subplots(1, 2, figsize=(10, 20)) # plotando o resultado
4 ax[0].imshow(imagem)
5 ax[0].axis('off')
6 ax[0].set_title('Imagem original')
7 ax[1].imshow(imagem_cinza, cmap='gray')
8 ax[1].axis('off')

```



```
9 ax[1].set_title('Imagem em níveis de Cinza')
```

Código Fonte 3.7. Convertendo imagem para níveis de cinza.

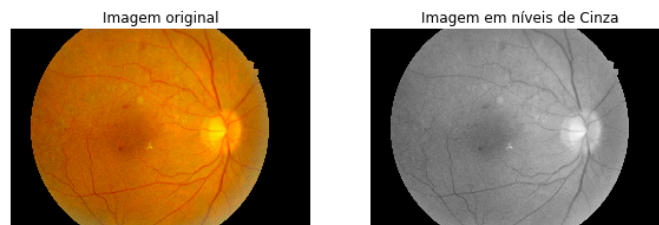


Figure 3.4. Resultado da conversão para níveis de cinza.

3.5. Remoção de Ruídos

A remoção dos ruídos das imagens é uma técnica que pode ser utilizada para várias finalidades, como: melhorar a qualidade da imagem, aplicar pré-processamento, extração de características ou até mesmo em uma etapa pós-processamento para detectar algum objeto na imagem.

3.5.1. Filtro da Mediana

O filtro da mediana executa uma operação de filtragem não linear em que uma janela se move sobre a imagem, em cada ponto, o valor médio dos dados nos quais a janela é tomada como saída. Esse filtro possui algumas propriedades desejáveis que não podem ser alcançadas com algoritmos lineares [Astola et al. 1990].

```
1 from skimage.filters.rank import median
2 from skimage.morphology import disk
3 import numpy as np
4 from skimage import img_as_ubyte # função que converte uma imagem para
  bytes com valores entre 0 e 255
5 from skimage import data # banco de imagens do skimage
6 imagem_ruido = img_as_ubyte(data.camera()) # carrega uma imagem da base
  de dados do Skimage
7 ruído = np.random.random(imagem_ruido.shape) # escolhe pontos aleató
  rios para serem os ruídos
8 # aplicando ruídos na imagem
9 imagem_ruido[ruído > 0.96] = 255 # define 96% dos pixels como branco
10 imagem_ruido[ruído < 0.04] = 0 # define 4% dos pixels como preto
11 # aplica filtro da mediana variando o valor da mediana
12 imagem_mediana_1 = median(imagem_ruido, disk(1))
13 imagem_mediana_2 = median(imagem_ruido, disk(2))
14 imagem_mediana_15 = median(imagem_ruido, disk(15))
15 fig, ax = plt.subplots(2, 2, figsize=(15, 8), sharex=True, sharey=True)
16 ax1, ax2, ax3, ax4 = ax.ravel()
17 ax1.imshow(imagem_ruido, 'gray')
18 ax1.set_title('Imagem com ruído')
19 ax1.axis('off')
20 ax2.imshow(imagem_mediana_1, cmap='gray')
21 ax2.set_title('Mediana = 1')
22 ax2.axis('off')
```

```

23 ax3.imshow(imagem_mediana_2, cmap='gray')
24 ax3.set_title('Mediana = 2')
25 ax3.axis('off')
26 ax4.imshow(imagem_mediana_15, cmap='gray')
27 ax4.set_title('Mediana = 15')
28 ax4.axis('off')

```

Código Fonte 3.8. Aplicando o filtro da mediana para remover ruídos.

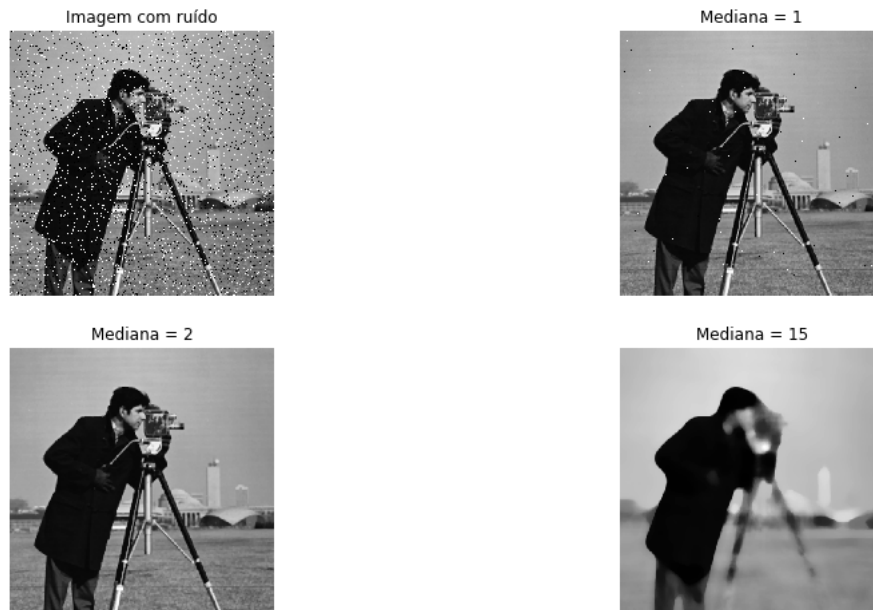


Figure 3.5. Resultados da aplicação do filtro da mediana.

O Código Fonte 3.8 apresenta uma maneira de remover os ruídos das imagens aplicando o filtro da mediana. Primeiro, aplica-se o ruído em uma imagem, ou seja, alguns pixels escolhidos aleatoriamente são substituídos. Depois é aplicado o filtro da mediana, como podemos ver o resultado na Figura 3.5.

3.5.2. Filtro *Square*

```

1 from skimage.morphology import square # aplicando square para retirar
  os ruídos da imagem com diferentes parâmetros
2 imagem_saquare_1 = median(imagem_ruído, square(1))
3 imagem_saquare_3 = median(imagem_ruído, square(3))
4 imagem_saquare_10 = median(imagem_ruído, square(10))
5 fig, ax = plt.subplots(2, 2, figsize=(15, 8), sharex=True, sharey=True)
6 ax1, ax2, ax3, ax4 = ax.ravel()
7 ax1.imshow(imagem_ruído, 'gray')
8 ax1.set_title('Imagem com ruído')
9 ax1.axis('off')
10 ax2.imshow(imagem_saquare_1, cmap='gray')
11 ax2.set_title('Square = 1')
12 ax2.axis('off')
13 ax3.imshow(imagem_saquare_3, cmap='gray')
14 ax3.set_title('Square = 3')
15 ax3.axis('off')

```

```

16 ax4.imshow(imagem_saquare_10, cmap='gray')
17 ax4.set_title('Square = 10')
18 ax4.axis('off')

```

Código Fonte 3.9. Aplicando o filtro *Square* para remover ruídos.

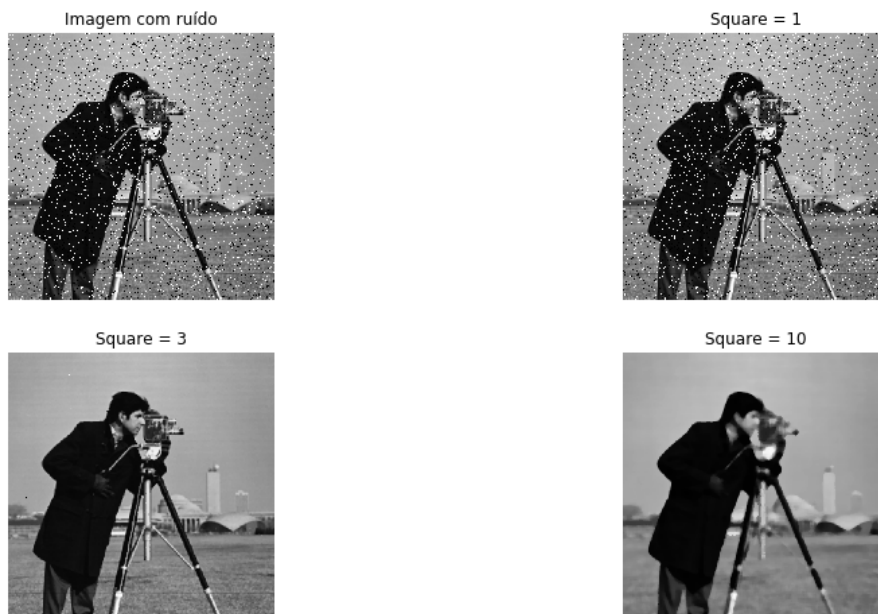


Figure 3.6. Resultados da aplicação do filtro *Square*.

A principal característica do *Square* é que ele gera um elemento estruturado plano e quadrado, ou seja, ele retorna uma caixa quadrada onde cada pixel pertence à vizinhança. A vizinhança é medida pelo diâmetro informado como parâmetro da função. O Código Fonte 3.9 demonstra a aplicação, e o resultado na Figura 3.6.

3.6. Binarização

Para binarizar imagens pode ser utilizado o algoritmo de Otsu, que conforme [Otsu 1979] é um método não paramétrico e não supervisionado de seleção automática de limiar para segmentação de imagem. Um limiar ideal é selecionado pelo critério discriminante, a fim de maximizar a separabilidade das classes resultantes em níveis de cinza. Utilizando otsu é possível gerar uma nova imagem apenas com dois grupos distintos de intensidade.

O Código Fonte 3.10 apresenta um exemplo de binarização por limiar de Otsu. Inicialmente colocamos a imagem em tons de cinza. Em seguida achamos o limiar automaticamente por meio da função *threshold otsu*. Por fim, define-se que na nova imagem gerada os pixels com valor igual ou superior ao limiar ficarão na cor branca e os de valor menor ficarão na cor preta.

```

1 # Primeiro vamos colocar a imagem em tons de cinza
2 from skimage.color import rgb2gray
3 cinza = rgb2gray(imagem)
4
5 # Importando o threshold otsu

```

```

6 from skimage.filters import threshold_otsu
7
8 # Encontrando o valor do limiar
9 otsu = threshold_otsu(cinza)
10 # Binarizando a imagem
11 preto_branco = cinza < otsu
12
13 plt.figure(figsize=(15,30))
14 plt.imshow(preto_branco, cmap = 'gray')

```

Código Fonte 3.10. Binarização com Otsu

A figura 3.7 apresenta a saída do código fonte 3.10. Em que (a) representa a imagem em tons de cinza e (b) é a imagem binarizada por meio do limiar de Otsu.

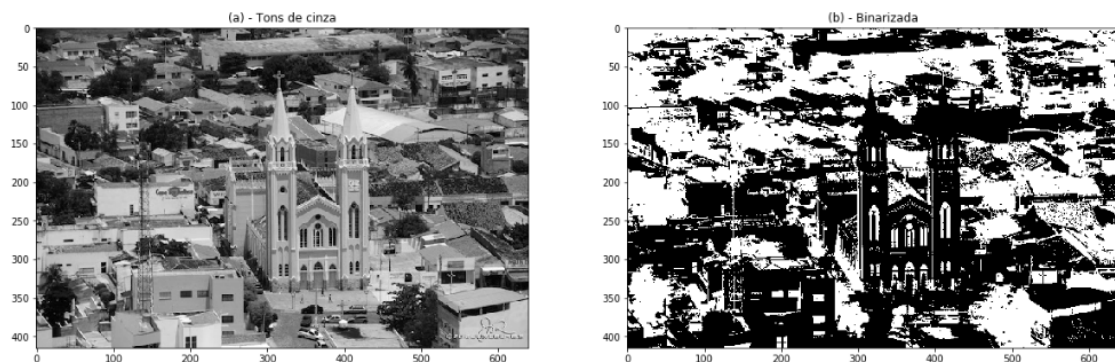


Figure 3.7. Imagem Binarizada com Otsu

3.7. Histograma

O histograma é uma das ferramentas mais simples e úteis para o PDI e mostra a frequência com que um nível de cinza aparece na imagem. Conforme [Gonzalez and Woods 2007] o histograma de uma imagem demonstra a distribuição estatística dos seus níveis de cinza. Em linhas gerais, podemos estimar a probabilidade de ocorrência do nível de cinza na imagem por intermédio de sua frequência relativa.

O Código Fonte 3.11 mostra como calcular o histograma, para isso, é utilizada a função *histogram* do skimage passando como parâmetros a imagem original e o número de posições usadas para calcular o histograma, que nesse caso é a quantidade de variações do valor do pixel.

```

1 # Lendo as imagens
2 claro = imread("picos_claro.jpg")
3 escuro = imread("picos_escuro.jpg")
4 medio = imread("picos_medio.jpg")
5 nitido = imread("picos_nitido.jpg")
6
7 # Importando a função histogram do skimage
8 from skimage.exposure import histogram
9
10 # Calculando histograma
11 hist_claro = histogram(claro, nbins=256)

```

```

12 hist_escuro = histogram(escuro,nbins=256)
13 hist_medio = histogram(medio,nbins=256)
14 hist_nitido = histogram(nitido,nbins=256)

```

Código Fonte 3.11. Calcular Histograma

É possível também plotar o histograma a partir da função "hist" do Matplotlib, como é visto do Código Fonte 3.12 a imagem é transformada em um vetor unidimensional com a função "ravel" e é determinado o intervalo de valores, neste caso 256 pois é o intervalo de valores dos pixels da imagem.

```

1 # Plotando o histograma com o hist do matplotlib (plt.hist)
2 fig,ax = plt.subplots(2,4,figsize=(20,10))
3 ax[0,0].imshow(claro,cmap = "gray")
4 ax[0,1].imshow(escuro,cmap = "gray")
5 ax[0,2].imshow(medio,cmap = "gray")
6 ax[0,3].imshow(nitido,cmap = "gray")
7
8 ax[1,0].hist(claro.ravel(),range(256))
9 ax[1,1].hist(escuro.ravel(),range(256))
10 ax[1,2].hist(medio.ravel(),range(256))
11 ax[1,3].hist(nitido.ravel(),range(256));

```

Código Fonte 3.12. Plotar o Histograma com Matplotlib

A 3.8 apresenta o resultado da plotagem, onde se pode ver em cima as quatro imagens originais: claro, escuro, medio e nitido. Já embaixo, tem-se os quatro histogramas. Como exemplo, percebe-se que para a primeira imagem chamada claro, a intensidade dos pixels está concentrada a direita, informando que a imagem está mais perto da cor branca, ou seja, é uma imagem pouco escura.

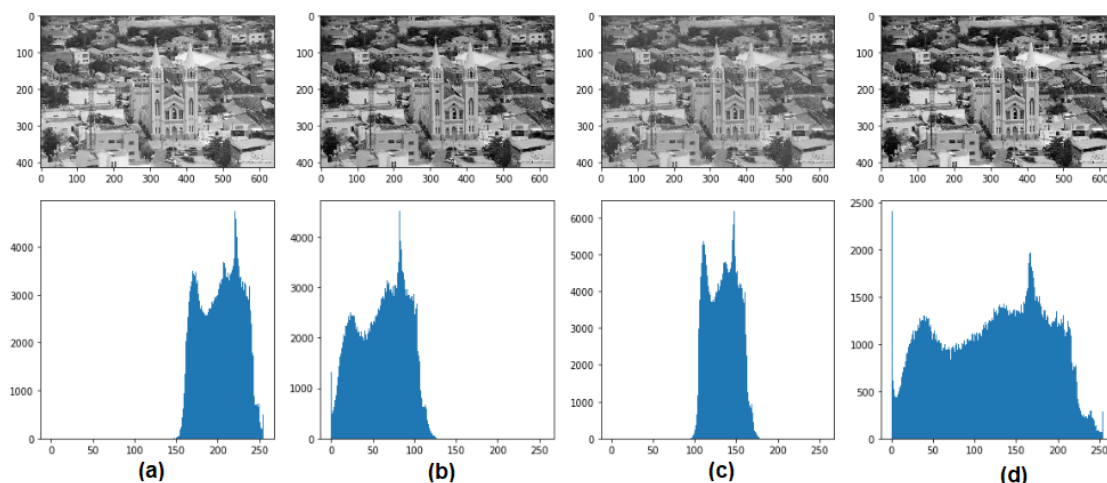


Figure 3.8. Imagens x Histogramas

3.7.1. Equalização do Histograma

Durante o processamento de imagens, diversas vezes faz-se necessário realizar transformações nas imagens a fim de realçar artefatos com baixo contraste ou diminuir a intensidade de algumas regiões. Para isso, é possível realizar a equalização do histograma.

Conforme [Blair et al. 2013] a correta manipulação do histograma permite a equalização dos níveis, para se obter uma imagem melhorada afim de se segmentar regiões ou extrair informações com mais facilidade.

Ao equalizar um histograma obtemos a máxima variância do histograma. O Código Fonte 3.13 apresenta o processo de equalização do histograma, que pode ser realizado com a função "equalize hist" do skimage.

```
1 from skimage.exposure import equalize_hist
2
3 # Equalizando histograma
4 new_claro = equalize_hist(claro)
5
6 # Plotar histograma Equalizado
7 fig, ax = plt.subplots(1, 2, figsize=(20, 5))
8 ax[0].hist(claro.ravel(), range(256));
9 ax[0].set_title('(A) - histograma normal')
10 ax[1].hist(new_claro.ravel()*256, bins=256);
11 ax[1].set_title('(B) - histograma equalizado')
```

Código Fonte 3.13. Equalizar o Histograma

A figura 3.9 apresenta em (a) o histograma da imagem original, que possui maior intensidade a direita identificando que é uma imagem clara. Após a equalização do histograma, pode-se ver em (b) que a intensidade dos pixels ficam bem distribuídas realizando um balanceamento do contraste.

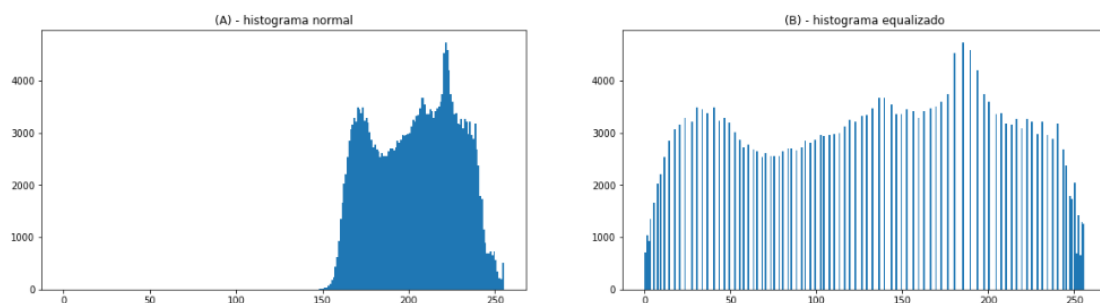


Figure 3.9. Histograma x Histograma Equalizado

Na figura 3.10 em (a) temos a imagem original e em (b) a imagem com histograma equalizado. Percebe-se um aumento no contraste em relação a imagem original, destacando algumas regiões na imagem.

3.8. Filtros de Contorno e Bordas

Bordas nas imagens são áreas com fortes contrastes de intensidade - um salto na intensidade de um pixel para o próximo. A detecção de borda de uma imagem reduz significativamente a quantidade de dados e filtra informações inúteis, preservando as importantes propriedades estruturais de uma imagem [Gupta and Mazumdar 2013].

A detecção de bordas no processamento digital de imagens é bastante utilizada nas etapas de pré-processamento, pois, muitas vezes, ajuda a diminuir a quantidade de

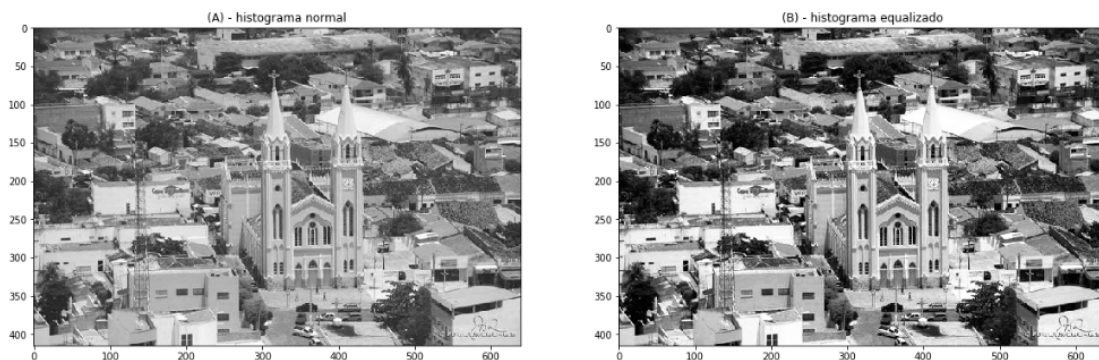


Figure 3.10. Imagem x Imagem com histograma equalizado

informações na imagem e realçar outras importantes. Dependendo do tipo de informação que se queira da imagem, esse processo será útil por eliminar informações redundantes.

No Código Fonte 3.14 é apresentado um exemplo de utilização de alguns dos filtros de bordas disponíveis no Skimage. São eles: Sobel, Roberts, Prewitt e Scharr. Na Figura 3.11 pode-se observar que o melhor resultado obtido na detecção das bordas foi utilizando o filtro Sobel. Portanto, a seguir, será discutido sobre seu funcionamento.

```

1 # Importação dos filtros de bordas
2 from skimage.filters import sobel, roberts, prewitt, scharr
3 import matplotlib.pyplot as plt
4
5 # importação de dados, para utilizar imagens do próprio skimage
6 from skimage import data
7
8 # carregando imagem horse
9 cavalo = data.horse()
10
11 # Pegando as bordas
12 contorno_sobel = sobel(cavalo) # aplicando filtro Sobel
13 contorno_roberts = roberts(cavalo) # aplicando filtro
14 contorno_prewitt = prewitt(cavalo) # aplicando filtro Prewitt
15 contorno_scharr = scharr(cavalo) # aplicando filtro Scharr
16
17 # plotando resultado
18 imagens = [cavalo, contorno_sobel, contorno_roberts, contorno_prewitt,
19            contorno_scharr]
20
21 titulos = ['Original', 'Filtro Sobel', 'Filtro Roberts', 'Filtro
22            Prewitt', 'Filtro Scharr']
23
24 fig = plt.figure(figsize=(10, 20))
25 for i in range(1, 5+1):
26     fig.add_subplot(linhas, colunas, i)
27     plt.imshow(imagens[i-1], cmap='gray')
28     plt.title(titulos[i-1])
29     plt.axis('off')
30     plt.subplots_adjust(bottom=0.1, top=0.4)

```

Código Fonte 3.14. Utilizando Filtros de Bordas.

O filtro Sobel é um dos muitos filtros utilizados no processo de detecção de borda. Ele funciona calculando o gradiente de intensidade da imagem em cada pixel dentro da imagem. O filtro Sobel usa duas matrizes de tamanho 3 x 3. Um para alterações na direção horizontal e outro para alterações na direção vertical. Ambas aplicam operações convolutivas com a imagem original para calcular as aproximações dos pixels vizinhos.

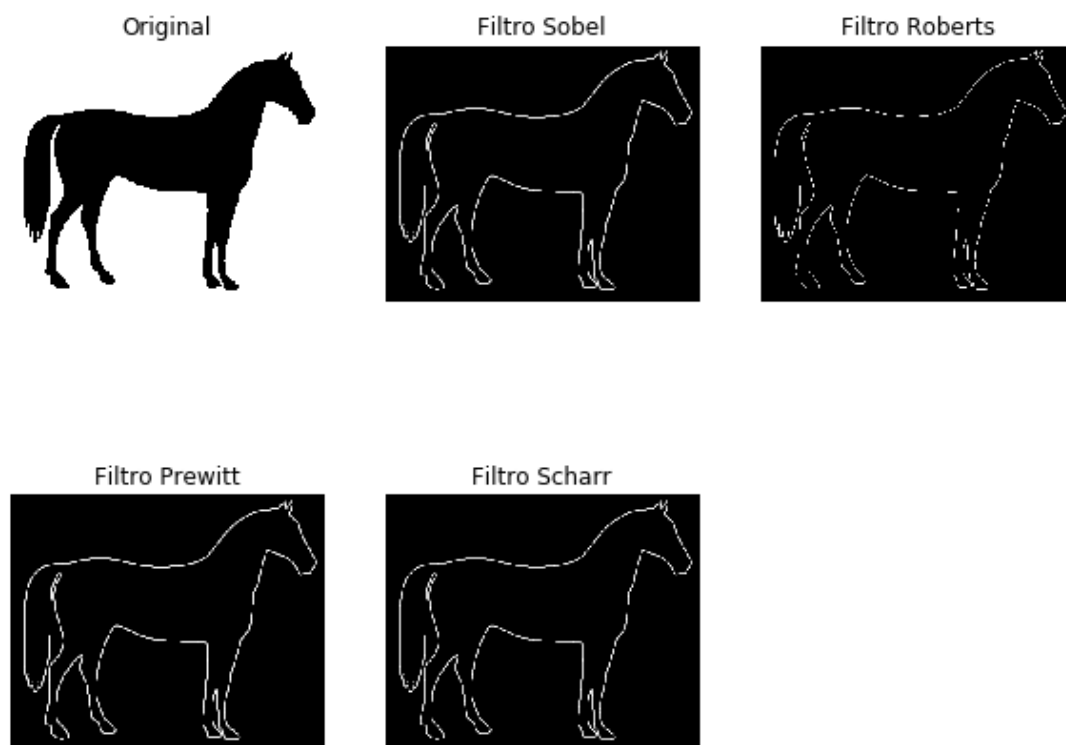


Figure 3.11. Utilizando Filtros de Bordas.

3.9. Segmentação

Durante a análise de imagens, geralmente um dos primeiros passos é a segmentação. Para [Gonzalez and Woods 1992] em PDI, segmentar é uma das tarefas mais difíceis e consiste em subdividir uma imagem em suas partes ou objetos constituintes. O objetivo é isolar a região de interesse e eliminar os artefatos ao redor para que não atrapalhem, uma segmentação efetiva é essencial para o sucesso ou falha da análise.

3.9.1. K-means

É possível utilizar diversas técnicas para segmentação, uma das mais utilizadas é o algoritmo K-means que foi proposto por [MacQueen et al. 1967] e é utilizado para dividir um conjunto de dados de forma automática em k grupos. Seu funcionamento consiste na seleção de k centros de cluster iniciais. Cada instância vai sendo atribuída ao centro de cluster mais próximo e em seguida cada centro de cluster é atualizado através da média

das instâncias que o constituem.

No Código Fonte 3.15 encontra-se a utilização do K-means. Inicialmente, realiza-se a importação dele no sklearn. Para o exemplo foi carregada a imagem de uma folha. Em seguida, o modelo K-means é criado, nele foi definido que a imagem será separada em dois clusters. Ao final é realizada a segmentação. Em forma de comentário no Código Fonte é possível ver como capturar cada um dos clusters de forma separada, bastando apenas igualar o resultado final ao valor do cluster, que pode variar de 0 a n.

```
1 # Importando o KMeans
2 from sklearn.cluster import KMeans
3
4 # Carregando a imagem
5 folha = imread('folha.png')
6
7 # Definindo o modelo com dois clusters (classes):
8 model_kmeans = KMeans(n_clusters=2, random_state=0).fit(np.reshape(
    folha, (-1, 1)))
9
10 # Segmentando a imagem:
11 segmentada1 = model_kmeans.predict(np.reshape(folha, (-1, 1)))
12 segmentada2 = np.reshape(segmentada1, (folha.shape[0], folha.shape[1]))
13
14 # Para visualizar os n clusters basta usar segmentada2 == 0, 1, ..., n
15 # Exemplo:
16 plt.imshow(segmentada2 == 0, cmap='gray')
```

Código Fonte 3.15. Segmentação com KMeans

A figura 3.12 apresenta em (a) a imagem da folha original, em (b) tem-se o cluster 0, que é a segmentação da região da folha. Já em (c) aparece o cluster 1, que representa toda a região ao redor da folha.

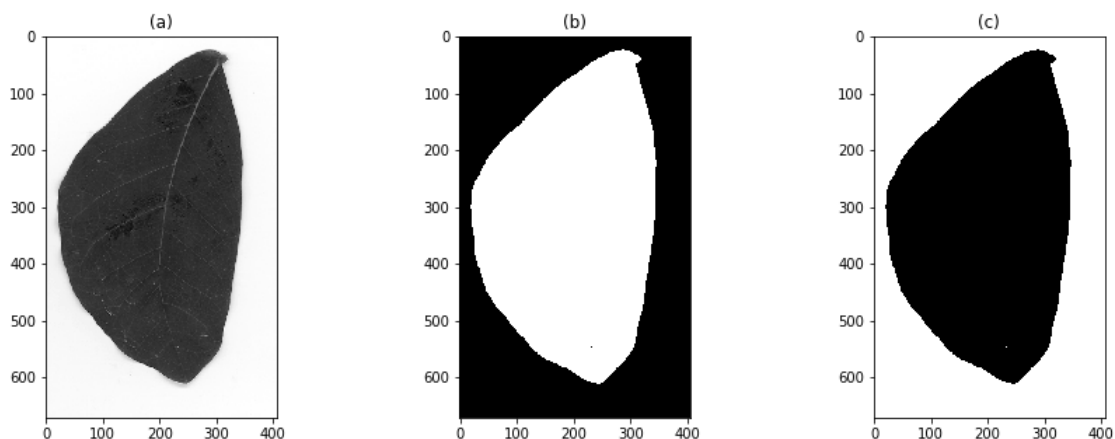


Figure 3.12. Segmentação com KMeans

3.10. Conclusão

Neste capítulo foram mostrados conceitos sobre processamento digital de imagens utilizando a linguagem Python. Para isso, foram apresentadas as etapas da PDI desde a

aquisição de imagens até a representação e interpretação. Também, apresentou-se as principais ferramentas utilizadas atualmente como Scikit-image, Jupyter notebook, Matplotlib e o Anaconda que possui as principais bibliotecas para ciência de dados e processamento de imagens.

Percebe-se que a utilização da biblioteca Scikit-image fornece enormes vantagens ao desenvolver sistemas que incluem maior agilidade e realização de tarefas complexas de forma simples e com pouco código. Diversas operações foram realizadas como manipulações básicas de imagens desde carregar, salvar, utilizar os canais RGB, plotar, transformar em níveis de cinza, até operações mais complexas como a remoção de ruídos, binarização, utilização do histograma, filtros de bordas e por fim, segmentação de regiões utilizando o algoritmo K-means.

Os trabalhos futuros poderão contemplar tarefas mais complexas, realizando as etapas de segmentação, extração de características e classificação em imagens médicas com problemas reais. Para isso, novas tecnologias serão abordadas como a utilização de descritores e das Redes Neurais Convolucionais.

References

- [Anaconda 2019] Anaconda (2019). Anaconda Home Page. <https://www.anaconda.com/>. Online; acessado 26 de Outubro 2019.
- [Astola et al. 1990] Astola, J., Haavisto, P., and Neuvo, Y. (1990). Vector median filters. *Proceedings of the IEEE*, 78(4):678–689.
- [Blair et al. 2013] Blair, C., Robertson, N. M., and Hume, D. (2013). Characterizing a heterogeneous system for person detection in video using histograms of oriented gradients: Power versus speed versus accuracy. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 3(2):236–247.
- [Doi 2005] Doi, K. (2005). Current status and future potential of computer-aided diagnosis in medical imaging. *The British journal of radiology*, 78(suppl_1):s3–s19.
- [Gonzalez and Woods 1992] Gonzalez, R. C. and Woods, R. E. (1992). Digital image processing addison-wesley. *Reading, Ma*, 2.
- [Gonzalez and Woods 2007] Gonzalez, R. C. and Woods, R. E. (2007). Digital image processing. ed iii.
- [Gupta and Mazumdar 2013] Gupta, S. and Mazumdar, S. G. (2013). Sobel edge detection algorithm. *International journal of computer science and management Research*, 2(2):1578–1583.
- [Jupyter 2019] Jupyter (2019). Jupyter Home Page. <https://jupyter.org/>. Online; acessado 26 de Outubro 2019.
- [MacQueen et al. 1967] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

- [Marengoni and Stringhini 2009] Marengoni, M. and Stringhini, S. (2009). Tutorial: Introdução à visão computacional usando opencv. *Revista de Informática Teórica e Aplicada*, 16(1):125–160.
- [Marques Filho and Neto 1999] Marques Filho, O. and Neto, H. V. (1999). *Processamento digital de imagens*. Brasport.
- [matplotlib 2019] matplotlib (2019). Matplotlib Home Page. <https://matplotlib.org/>. Online; acessado 26 de Outubro 2019.
- [Otsu 1979] Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66.
- [Scikit-image 2019] Scikit-image (2019). Image processing in Python. <https://scikit-image.org/>. Online; acessado 26 de Outubro 2019.

Capítulo

4

Processamento e Análise de Sinais Acústicos em Cenários Urbanos com ConvNets: Teoria e Prática

Deborah M. V. Magalhães, Flávio H. D. Araújo, Jederson S. Luz, Myllena C. Oliveira, Fátima N. S. Medeiros

Abstract

The increasing population density in urban centers brings with it several challenges, including the monitoring of public spaces. In this context, sound emerges as an essential source of information about urban life and acoustic monitoring as a low cost and robust alternative to capture relevant information. Despite that, acoustic scene recognition is a challenging task due to the high heterogeneity of sounds present in this environment that overlap with the event to be classified. Convolutional Neural Networks (CNNs) have been successfully applied in the urban scenario classification task as an audio feature extractor. Thus, this chapter aims to introduce the fundamentals of audio processing and the application of CNNs for learning audio characteristics. Finally, we present a tutorial on audio signal processing, followed by the training and use of CNN to support urban sound events detection applied to a New York City dataset.

Resumo

O aumento da densidade populacional em centros urbanos traz consigo diversos desafios, entre eles o monitoramento de espaços públicos. Neste contexto, o som surge como uma importante fonte de informação a respeito da vida urbana e o monitoramento acústico como uma alternativa de baixo-custo e robusta. No entanto, a análise e compreensão de cenas acústicas é uma tarefa desafiadora devido a alta heterogeneidade dos sons presentes no ambiente, bem como, a sobreposição dos mesmos. As Redes Neurais Convolucionais (CNNs) vem sendo aplicadas com sucesso na tarefa de classificação de cenários urbanos como um extrator de características de áudio empregado na etapa de classificação. Com isso, este capítulo tem por objetivo introduzir os fundamentos básicos de processamento de áudio e a aplicação das CNNs para aprendizagem de características de áudio. Por fim, será apresentado um tutorial de processamento dos sinais acústicos,

seguido do treinamento e utilização de CNN na classificação de eventos sonoros urbanos extraídos da cidade de Nova York.

4.1. Introdução

Em 2018, cerca de 55,3% da população mundial vivia em espaços urbanos, esse número chegou a 80% quando tratamos da Europa e América do Norte. A China apresentou um crescimento de 40% em sua parcela de habitantes urbanos nos últimos 50 anos ¹. Até 2030, as áreas urbanas devem abrigar 60% das pessoas em todo o mundo e uma em cada três pessoas viverá em cidades com pelo menos meio milhão de habitantes ². O aumento da densidade populacional estressa a infraestrutura dos centros urbanos e traz consigo diversos desafios relacionados a mobilidade urbana, segurança e saúde pública [Souza et al. 2018].

Neste âmbito, as cidades inteligentes despontam como uma oportunidade para guiar políticas públicas visando oferecer melhores serviços e infraestrutura aos cidadãos [Rathore et al. 2016]. As cidades inteligentes se apoiam na expansão da conectividade com a internet das coisas (IoT) para o monitoramento ubíquo e inteligente através de redes de sensores e *smartphones*. Sendo o som uma importante fonte de informação a respeito da vida urbana [Edwards 2018], a perspectiva de conectividade, captura, análise e classificação automáticas dos sons urbanos podem auxiliar na redução da poluição sonora nas cidades, computação sensível ao contexto e vigilância urbana.

Nos últimos anos, novos métodos foram propostos para analisar eventos sonoros automaticamente [Virtanen et al. 2018]. As Redes Neurais Convolucionais (*ConvNets* ou CNNs) vem sendo aplicadas com sucesso na tarefa de classificação de cenários urbanos [Giannakopoulos et al. 2019, Piczak 2015, Salamon e Bello 2017]. Diferentes arquiteturas têm sido treinadas de forma supervisionada para distinguir entre classes contextuais (parque, biblioteca, etc), funcionando como um extrator de características de áudio empregado na etapa de classificação. Essas redes têm se mostrado capazes de aprender padrões em grandes volumes de dados, estabelecendo relações entre o sinal de áudio e os domínio do tempo e frequência, no entanto, as relações se restringem à representação do sinal de entrada.

Este capítulo provê uma descrição compreensiva do processo de classificação automática de eventos sonoros urbanos que vai desde o pré-processamento do áudio, passando pela extração de características até a detecção do evento em si. Cada etapa conta com *scripts* em python que fazem uso de bibliotecas como, por exemplo, a *LibROSA* ³ para viabilizar o processamento do áudio, e as bibliotecas *Keras* ⁴ com o *TensorFlow* ⁵ para treinamento da CNN a fim de extrair as características utilizadas na etapa de classificação do evento sonoro de interesse.

¹<https://data.worldbank.org/indicator/SP.URB.TOTL.IN.ZS>

²<https://population.un.org/wup/>

³<https://librosa.github.io/>

⁴<https://keras.io/>

⁵<https://www.tensorflow.org>

4.2. Problema Abordado

Dentre as possíveis aplicações de monitoramento acústico, este capítulo foca classificação de eventos sonoros em espaços urbanos com destaque para aplicação de vigilância. Alguns fatores motivadores do desenvolvimento de novas pesquisas focadas na identificação de eventos perigosos a partir do processamento de *streams* de áudio são:

1. O aumento da densidade populacional em centros urbanos projeta, entre outros desafios, a dificuldade de realização de policiamento efetivo e proteção de espaços públicos. Nos últimos anos, enfrentamos no mundo o crescimento da criminalidade e terrorismo [Bello et al. 2018]. No Brasil, em 2012, a cidade de Natal apresentou altos níveis de criminalidade nas categorias de roubo, estupro, fraude e roubos residenciais [Borges et al. 2017];
2. O crescimento das redes de sensores acústicos aliado à ampla adoção dos *smartphones* impulsionou a utilização do microfone como dispositivo vantajoso para monitorar espaços urbanos pois este é menor e mais barato que a câmera. Além disso, o microfone é mais robusto às condições ambientes, como neblina, poluição, chuva e mudanças de luminosidade, e realiza o monitoramento omnidirecional, sendo menos susceptível à oclusão [Bello et al. 2018]. No geral, a captura de som exige menos bateria dos dispositivos envolvidos [Virtanen et al. 2018] quando comparados a aplicações de monitoramento por vídeo.

O processamento e análise de fontes acústicas podem contribuir para localização e rastreamento de atos criminosos, terroristas e tumultos e, a partir dessa identificação, definir características como nível, duração e frequência de tais eventos. Isso pode, por sua vez, oferecer *insights* nas áreas de ciências sociais e políticas públicas sobre a relação entre o som urbano e a criminalidade em uma determinada região, melhorando a efetividade das intervenções tanto da polícia como dos órgãos que prestam socorro às vítimas.

Assim, este capítulo aborda um problema pertinente e complexo, que possui amplo espaço para contribuições. Para a maioria dos humanos e animais, a habilidade de escutar eventos sonoros é uma tarefa trivial, mas desenvolver algoritmos que sejam capazes de automaticamente reconhecer um som é desafiador [Virtanen et al. 2018]. Quando tratamos de ambientes urbanos, onde são encontrados sons naturais não biológicos (vento, chuva), sons naturais biológicos (animais em geral) e sons mecânicos (tráfego, construção, sinais, máquinas, instrumentos musicais), temos um ambiente altamente heterogêneo que pode possuir ilimitados sons.

Ainda, em ambientes realistas, o evento sonoro que se deseja classificar se sobrepõe aos demais presentes no ambiente. Portanto, o áudio capturado é uma superposição de todas as fontes presentes. Além disso, em várias aplicações de monitoramento de eventos sonoros, os microfones usados para capturar áudio costumam ficar significativamente mais distantes das fontes sonoras, o que aumenta a interferência no sinal capturado. Todos esses fatores previnem a correspondência do sinal com os modelos desenvolvidos para reconhecê-lo. Essa situação é bem diferente das aplicações de fala, entre elas, reconhecer a sequência de palavras na fala ou reconhecer a identidade da pessoa que está falando, onde comumente os microfones são utilizados em ambientes fechados e contro-

lados. Portanto, a análise e compreensão de eventos acústicas em espaços urbanos é ainda mais desafiadora.

4.3. Classificação de Eventos Sonoros

A Figura 4.1 ilustra o fluxo de classificação eventos sonoros, onde seu principal objetivo consiste na extração de informação útil do sinal de áudio através de métodos computacionais de processamento de sinais e aprendizagem de máquina para discriminar da melhor forma possível diferentes classes sonoras. Tal processo pode ser aplicado para identificar classes como, por exemplo, o barulho de vidro estilhaçando em função da invasão de uma residência, um tiro ou um acidente de carro. É importante destacar que tal fluxo pode ser extrapolado para identificar o humor da música ou reconhecimento de fala, entre outras aplicações.

A metodologia proposta para a classificação de eventos sonoros conta com cinco etapas: (1) *Aquisição de dados*, consiste na obtenção dos sinais de áudio obtidos através de base de dados pública; (2) *Pré-processamento*, onde os dados serão uniformizados, aumentados e representados em diferentes domínios que serão empregados na etapa (3) Na etapa de *Extração de Características*, temos o aprendizado de características por meio de uma CNN é treinada com espectrogramas dos áudios a fim de extrair características relevantes da representação do sinal e gerar um descritor. De posse do descritor, é possível aplicá-lo diretamente na etapa de (4) Classificação, onde o classificador que será responsável por associar cada áudio a uma classe de evento sonoro. Por fim, temos a etapa de (5) *Validação* que será utilizada para avaliar se o descritor gerado pode se utilizado pelo método de classificação para detectar as classes de eventos sonoros de maneira acurada. Todas essas etapas serão discutidas nas seções a seguir e tem os seus respectivos códigos disponíveis em: github.com/deborahvm/AudioProcessing

4.3.1. Aquisição dos Dados

O desempenho dos métodos de classificação depende diretamente da qualidade e da quantidade dos dados utilizados para treinar e testar os algoritmos. Cenários de classificação desafiadores podem exigir facilmente centenas ou milhares de exemplos para capturar suficientemente a variabilidade dos sons que definem um cenário urbano. Em cenários reais, muitos dos fatores que promovem essas variabilidades como, por exemplo, espaço acústico, posicionamento da fonte em relação ao microfone, tipo do dispositivo de captura, não podem ser totalmente controlados, tornando o processo de aquisição uma tarefa desafiadora.

Uma maneira de iniciar o desenvolvimento do processo de classificação de sinais de áudio é obter os eventos sonoros a partir de base de dados ao invés de capturar manualmente os áudios. Neste capítulo, os áudios bem como a anotação da classe à qual ele pertence foram retirados da base pública de dados chamada UrbanSound8K⁶. Essa base conta com 8732 áudios com duração inferior ou igual a 4 segundos rotulados em 10 classes distintas: ar condicionado, buzina de carro, crianças brincando, latido de cachorro, furadeira, motor de veículo, tiro, britadeira, sirene e música de rua. Com exceção das classes “crianças brincando” e “tiros”, todas as outras classes foram selecionadas devido

⁶<https://urbansounddataset.weebly.com/urbansound8k.html>



Figura 4.1: Metodologia para a classificação dos eventos sonoros em espaços urbanos.

a alta frequência em que aparecem no serviço de reclamações de ruído urbano 311 da cidade de Nova York, nos Estados Unidos da América do Norte [Salamon et al. 2014].

Os áudios estão disponibilizados no formato-padrão WAVEform audio format (WAV), distribuídos em 10 pastas distintas para facilitar a reprodução e comparação com os resultados da classificação automática. Além dos arquivos de áudio, a base contém metadados que incluem informações sobre as classes às quais os áudios pertencem. Essa base de dados é heterogênea, visto que a taxa de amostragem, a codificação e o número de canais podem variar de arquivo para arquivo. Além disso, se trata de uma base desbalanceada, visto que o número de amostras difere de uma classe para outra, chegando a ter 374 amostras da classe “tiro”, 429 da classe “buzina de carro”, 929 amostras da classe “sirene” e 1000 amostras nas demais classes. Essas características devem ser tratadas na

etapa de pré-processamento.

4.3.2. Pré-processamento de Áudio

O pré-processamento do sinal de áudio é realizado antes das etapas de extração e aprendizagem de características. Neste capítulo, o pré-processamento consiste de três etapas, conforme ilustra a Figura 4.2: (1) aumento dos dados decorrente da aplicação de alterações no conjunto de amostras de treinamento, resultando em novos dados; (2) uniformização do sinal de áudio com a aplicação de transformações no áudio para garantir homogeneidade e qualidade do sinal de entrada; (3) geração da representação do sinal de áudio que servirá de entrada para etapas de extração de características e aprendizagem de características. Cada etapa utiliza ferramentas e bibliotecas identificadas na Figura 4.2. As seções seguintes descrevem como essas ferramentas foram empregadas na metodologia adotada.

Pré-Processamento



Figura 4.2: Etapas do pré-processamento dos sinais de áudio.

4.3.2.1. Aumento dos Dados

Uma propriedade importante relacionada ao tamanho da base de dados diz respeito ao balanceamento entre as diferentes classes do problema em estudo. É importante garantir que nenhuma classe seja sub-representada no processo de treinamento pois isso impacta diretamente no desempenho da tarefa de classificação [He e Garcia 2008]. Conforme descrito na Seção 4.3.1, o conjunto de dados utilizado apresenta desbalanceamento entre as classes. Portanto, inspirados em [Salamon e Bello 2017], aplicamos alterações no conjunto de amostras de treinamento a fim de balancear as classes, resultando assim em novos dados. É importante que as alterações aplicadas aos dados rotulados não mudem o significado semântico de seus rótulos. Ao treinar o modelo com os novos dados, espera-se que a rede se torne invariante às alterações aplicadas e generalize melhor para dados ainda não apresentados ao modelo [Salamon e Bello 2017], melhorando o desempenho do mesmo ao estimar ou classificar eventos sonoros.

A classificação automática de eventos sonoros em cenários urbanos tem focado em métodos de aprendizado profundo, como as CNNs. Diferentes arquiteturas têm sido treinadas para distinguir entre classes contextuais (parque, biblioteca, etc), funcionando como um extrator de características de áudio [Giannakopoulos et al. 2019, Piczak 2015,

[Salamon e Bello 2017](#)]. Tais arquiteturas são capazes de aprender padrões em grandes conjuntos de dados e portanto, se beneficiam de técnicas que ampliem o número de amostras.

Na etapa de aumento de dados, dividimos inicialmente o conjunto de dados em aproximadamente 80% de sinais de áudio para treino e 20% desses sinais para teste. Consequentemente, as pastas *fold 1* e *fold 2* do conjunto de dados foram separadas para os testes e não foram submetidas ao processo de aumento dos dados. Já as demais pastas foram empregadas no treinamento e submetidas a alterações nos áudios a fim de aumentar o conjunto de treino.

Antes de aumentar os dados, precisamos gerar arquivos de notação para cada um deles. O formato de notação JAMS ⁷ é baseado em *JavaScript Object Notation* (JSON) em que armazena informações relativas ao áudio como, por exemplo, batidas, acordes, segmentos, *tags*. Esse formato possibilita o armazenamento de informações de forma simples e estruturada sobre as características e transformações aplicadas no áudio de forma independente de idioma e legível por humanos.

O *script* que gera os arquivos de notação é apresentado no Código 4.1. Tal *script* faz uso da biblioteca JAMS ⁸ para definir os parâmetros *nome do autor*, *email do autor*, *fonte dos dados* e *duração do áudio*, configurados nas *linhas 45-48*. Na base de dados utilizada, o tamanho dos áudios variam de 0 a 4 segundos, resultando em representações de áudio de tamanhos distintos. Como a CNN trabalha com entradas de tamanhos fixos, desenvolvemos na *linha 38* uma condição para descartar todos os áudios que fossem inferiores a 2.3 segundos, com isso garantimos um comprimento mínimo para a entrada da CNN. Utilizamos a biblioteca *LibROSA* ⁹ para carregar o áudio e capturar o tempo duração do mesmo conforme descrito nas *linhas 35 e 36*, respectivamente.

```
1 import glob
2 import re #regex
3 import os
4 import jams
5 import librosa
6
7 #Caminho dos áudios de treinamento
8 path_database = '/data/deborah/UrbanSound8K/audio/'
9 audio_format = 'wav'
10 data_source='UrbanSound8K'
11 name='Deborah'
12 email='deborah.vm@ufpi.edu.br'
13
14 def run():
15
16     #Capturando os nomes das pastas
17     names_folders = glob.glob(path_database + '*')
18
19     for names_ in names_folders:
20
21         #Capturando o caminho completo dos áudios
```

⁷<https://jams.readthedocs.io/en/stable/>

⁸<https://pypi.org/project/jams/>

⁹<https://librosa.github.io/>

```

23 path_audios = glob.glob(names_ + '/*.' + audio_format)
24
25 for path_ in path_audios:
26
27     #Capturando o nome do arquivo de áudio (wav)
28     match_obj = re.sub(names_, "", path_)
29     audio_name = re.sub(r'/', "", match_obj)
30
31     #Capturando o caminho completo da pasta
32     fold_path = re.sub(audio_name, "", path_)
33
34     os.chdir(fold_path)
35     #Capturando a duração do sinal
36     y, sr = librosa.load(audio_name, sr=None)
37     duration = librosa.get_duration(y=y, sr=sr)
38
39     #Remove os arquivos de áudio com duração inferior a 2.3s
40     if duration >= 2.3:
41
42         #Gerando o arquivo de anotação jam
43         jam = jams.JAMS()
44
45         #Setando os parâmetros do arquivo de notação
46         jam.file_metadata.duration = duration
47         ann = jams.Annotation(namespace='beat', time=0,
48 duration=jam.file_metadata.duration)
49         ann.annotation_metadata = jams.AnnotationMetadata(
50 data_source=data_source)
51         ann.annotation_metadata = jams.AnnotationMetadata(
52 validation= "")
53         ann.annotation_metadata.curator = jams.Curator(name=
54 name, email=email)
55         jam.annotations.append(ann)
56
57         #Salvando o arquivo de notação para cada áudio
58         jam_name = re.sub(r'\.wav', ".jams", audio_name)
59         jam.save(jam_name)
60
61     else:
62         os.system('rm ' + path_)
63
64 run()

```

Código Fonte 4.1: Geração dos arquivos de notação para o conjunto de treinamento.

Todas as alterações ou transformações aplicadas aos sinais de áudio fizeram uso da biblioteca MUDA ¹⁰, conforme o código 4.2. Tal biblioteca recebe como entrada um arquivo de áudio e seu respectivo arquivo de notação no formato JAMS (*linha 42*), aplica a transformação no áudio (*linhas 45 e 53*) e gera o novo arquivo com seu respectivo arquivo de notação (*linhas 50 e 60*). Na *linha 37*, observamos uma condição que implica em uma maior variação de semitons para as classes 1-buzina de carro e 6-tiro de arma. Isso ocorre porque um dos fins do aumento dos dados é o balanceamento das classes, como as classes

¹⁰<https://muda.readthedocs.io/en/latest/>

1 e 6 possuem menos amostras, elas foram modificadas mais vezes. Portanto, todas as classes sofreram um aumento de 8 vezes o número de amostras, já as classes 1 e 6 foram aumentadas cerca de 10 vezes.

A biblioteca MUDA permite realizar diferentes transformações, nesse capítulo aplicamos 2 tipos:

- Variação de tom: esse tipo de transformação altera o tom do sinal sem modificar seu comprimento. Essa transformação recebe como parâmetro um número indicando o semi tom que se alterar no sinal. Nesse capítulo, cada amostra teve seu tom alterado considerando os seguintes semitons: 1,1.5,2,2.5,3,3.5;
- Ruído de fundo: esse tipo de transformação mistura o áudio com outro áudio contendo como sons de fundo de diferentes tipos de cenas acústicas. De cada áudio que será utilizado como ruído de fundo, são extraídos aleatoriamente n pedaços que são misturados aleatoriamente com o áudio de entrada. Cada áudio original foi misturado com 4 cenas acústicas: trabalhadores na rua, tráfego de rua, pessoas na rua e parque.

```
1 import muda
2 import glob
3 import re #regex
4 import os
5
6 #Caminho dos dados de treinamento (.wav e .jams)
7 path_database = '/data/deborah/UrbanSound8K/audio/'
8 #Caminho dos áudios utilizados como ruído
9 path_background_noise = '/data/deborah/UrbanSound8K/background_noise/'
10 audio_format = 'wav'
11 files = glob.glob(path_background_noise + '/*.' + audio_format)
12
13 def run():
14
15     #Captura os nomes das pastas
16     names_folders = glob.glob(path_database + '*')
17
18     for names_ in names_folders:
19
20         #Captura o caminho completo do áudio
21         path_audios = glob.glob(names_ + '/*.' + audio_format)
22
23         for path_ in path_audios:
24
25             #Captura o nome do áudio sem extensão
26             audio_name = re.sub(names_, "", path_)
27             audio_name = re.sub(r'/', "", audio_name)
28             audio_name_wo_ext = re.sub(r'\.wav', "", audio_name)
29
30             #Captura o caminho completo no arquivo de anotação
31             jam_path = re.sub(r'\.wav', ".jams", path_)
32
33             #Captura o label da classe
34             label = audio_name.split('-')[1]
```

```

35         # A variação de semitons é maior para as classes 1 e 6
36         if label == '1' or label == '6':
37             semitones=[1,1.5,2,2.5,3,3.5]
38         else:
39             semitones=[1.5,2,2.5,3]
40
41         jam_orig = muda.load_jam_audio(jam_path, path_)
42
43         #Gerando variações no tom
44         ps = muda.deformers.PitchShift(n_semitones=semitones)
45         output_name_ps_pattern = names_+'/'+audio_name_wo_ext+ '-ps-
46 _'
47
48         #O áudio modificado é salvo em um novo .wav e as deformaçõ
49 es aplicadas são salvas no seu respectivo .jams
50         for i, jam_out in enumerate(ps.transform(jam_orig)):
51             muda.save(output_name_ps_pattern+str(semitones[i])+'.
52 wav', output_name_ps_pattern+str(semitones[i])+'.jams',jam_out)
53
54         #Gerando deformações de ruído de fundo
55         bg = muda.deformers.BackgroundNoise(n_samples=1, files=
56 files, weight_min=0.1, weight_max=0.5)
57         output_name_bg_pattern = names_+'/'+audio_name_wo_ext+ '-bg-
58 _'
59
60         #O áudio modificado é salvo em um novo .wav e as deformaçõ
61 es aplicadas são salvas no seu respectivo .jams
62         for i, jam_out in enumerate(bg.transform(jam_orig)):
63             bg_noise_name = files[i].split('/')[5]
64             bg_noise_name_w_extension = re.sub(r'\.wav', "",
65 bg_noise_name)
66             muda.save(output_name_bg_pattern+
67 bg_noise_name_w_extension+'.wav', output_name_bg_pattern+
68 bg_noise_name_w_extension+'.jams',jam_out)
69
70 run()

```

Código Fonte 4.2: Geração de transformações nos áudios de treinamento para aumentar o conjunto de treinamento.

4.3.2.2. Uniformização dos Dados

Quando utilizamos arquivos de áudio armazenados em .mp3 ou .wav, por exemplo, esses arquivos já passaram por um processo de digitalização que envolve as etapas de filtragem, amostragem e quantização [Serizel et al. 2018]. A filtragem consiste em limitar a largura de banda do sinal para filtrar alguns ruídos de fundo que são comumente encontrados, especialmente em cenários urbanos. A taxa de amostragem, também conhecida como frequência de amostragem, consiste no número de amostras por unidade de tempo medido em hertz (Hz), onde uma amostra é uma medida da amplitude do sinal em um intervalo de tempo. Quanto maior a taxa de amostragem, maior será a semelhança entre o sinal digital e o sinal analógico original, por conseguinte, melhor será a qualidade do áudio.

A quantização reside no processo de mapear valores de entrada de um grande conjunto (geralmente um conjunto contínuo) para gerar valores em um conjunto menor (contável), geralmente com um número finito de elementos. Neste processo o sinal é discretizado para valores inteiros dentro de um intervalo de possíveis representações, onde os mais usados são os comprimentos de 16-bit, 24-bit e 32-bit. Quanto maior o comprimento, maior a qualidade no som, em contra partida, maior o tamanho do arquivo. Linguagens de alto nível, como *Python*, não possuem um tipo de dado primitivo específico de 24 bits, desse modo, a leitura de arquivos WAV de 24-bit em alguns pacotes dessa linguagem não funcionarão. Situações semelhantes podem ser encontradas com o áudio de 32-bit [Font et al. 2018].

Neste capítulo, como a base de dados utilizada apresenta áudios que foram capturados com configurações de gravação não uniformes, é usual encontrar áudios com diferentes taxas de amostragem e quantização. Portanto, utilizamos a ferramenta SoX¹¹, conforme ilustrado na *linha 37* do código em *Python 4.3*, para garantir que todos os áudios possuam a mesma taxa de amostragem e quantização. Nessa linha, o parâmetro `-r` indica a taxa de amostragem e o parâmetro `-b` indica o comprimento de bits usados para armazenar o áudio. Nós fixamos esses valores em 44.1 kHz e 16-bits que correspondem ao padrão universal *Compact Disc Standard* com qualidade de CD [Font et al. 2018]. Nós desconsideramos todas as amostras com tamanho inferior a 2.3 segundos, conforme definido na condição implementada na *linha 35*. Apesar de já termos definido essa condição na etapa anterior, ela foi implementada apenas para o conjunto de treino. No entanto, ela também vale para o conjunto de teste. É importante destacar que a etapa de uniformização é aplicada tanto ao conjunto de treino quanto ao conjunto de teste.

```
import os
2 import glob
import re #regex
4 import subprocess
import librosa

6
#Caminho dos áudios de treino e teste
8 path_database = '/data/deborah/UrbanSound8K/audio/'
  audio_format = 'wav'

10
def run():
12
    #Capturando o nome das pastas
14     names_folders = glob.glob(path_database + '/*')

16     for names_ in names_folders:

18         #Capturando o caminho completo do áudio
          path_audios = glob.glob(names_ + '/*.' + audio_format)

20
          for path_ in path_audios:

22              #Capturando o nome do arquivo de áudio (wav)
24              match_obj = re.sub(names_, "", path_)
                match_obj = re.sub(r'/', "", match_obj)
```

¹¹<http://sox.sourceforge.net/>

```

26 match_obj4 = re.sub(match_obj, "", path_)
27 match_obj5 = re.sub(r'\.wav', "-resample.wav", path_)
28
29 #Capturando a duração do sinal
30 y, sr = librosa.load(path_)
31 duration = librosa.get_duration(y=y, sr=sr)
32
33 os.chdir(match_obj4)
34 #Desconsidera os áudios com duração inferior a 2.3
35 if duration >= 2.3:
36     #Reamostrando o sinal de áudio e alterando a resolução
37     para 44.1KHz e 16-bits
38     os.system('sox ' + path_ + ' -r 44100 -b 16 ' + match_obj5
39 + ' rate')
40     os.system('rm ' + path_)
41 else:
42     os.system('rm ' + path_)
43
44 run()

```

Código Fonte 4.3: Reamostragem e quantização os áudios para o padrão *Compact Disk*.

4.3.2.3. Representação do Sinal

Interpretar e identificar um evento sonoro a partir da representação no domínio do tempo de um sinal de áudio é uma tarefa difícil. Portanto, é comum os trabalhos na literatura utilizarem representações no domínio do tempo-frequência pois permitem uma interpretação mais alinhada com a percepção humana [Serizel et al. 2018].

Uma propriedade do sinal de áudio é que suas características estatísticas variam ao longo do tempo, por isso, o áudio é quebrado em partes menores de tamanho fixo chamadas de *frames* para que cada um deles seja considerado individualmente, conforme apresentado na primeira Figura 4.3. Aqui, o sinal traz informações referentes ao domínio do tempo, no entanto, outras informações relevantes se apresentam no domínio da frequência.

Portanto, para representar os áudios no domínio da frequência, nós precisamos aplicar uma transformação de Fourier de Tempo Discreto, do inglês *Discrete-time Fourier Transform* (DFT). Todavia, essa transformação implica na inserção de ruídos no sinal. Em vista disso, nós aplicamos uma função de janelamento com o intuito de suavizar os limites dos *frames*, pois mudanças abruptas nesses limites podem gerar distorções no espectro do sinal, conforme é ilustrado na Figura 4.3. Em conjunto com o janelamento, definimos uma taxa de sobreposição, também denominada como *overlap*. O objetivo da sobreposição entre os *frames* é garantir dependência estatística entre eles e suavizar o sinal.

O procedimento de calcular a DFT para os *frames* sobrepostos é conhecido como *Short-Time Fourier Transform* (STFT). Desse modo, aplicamos a STFT nos *frames* de cada áudio a fim alcançar a representação do sinal no domínio da frequência que servirá de entrada para a etapa de extração de características, como é ilustrado na Figura 4.3.

Ainda, a transformação STFT permite definir o espectrograma de frequência li-

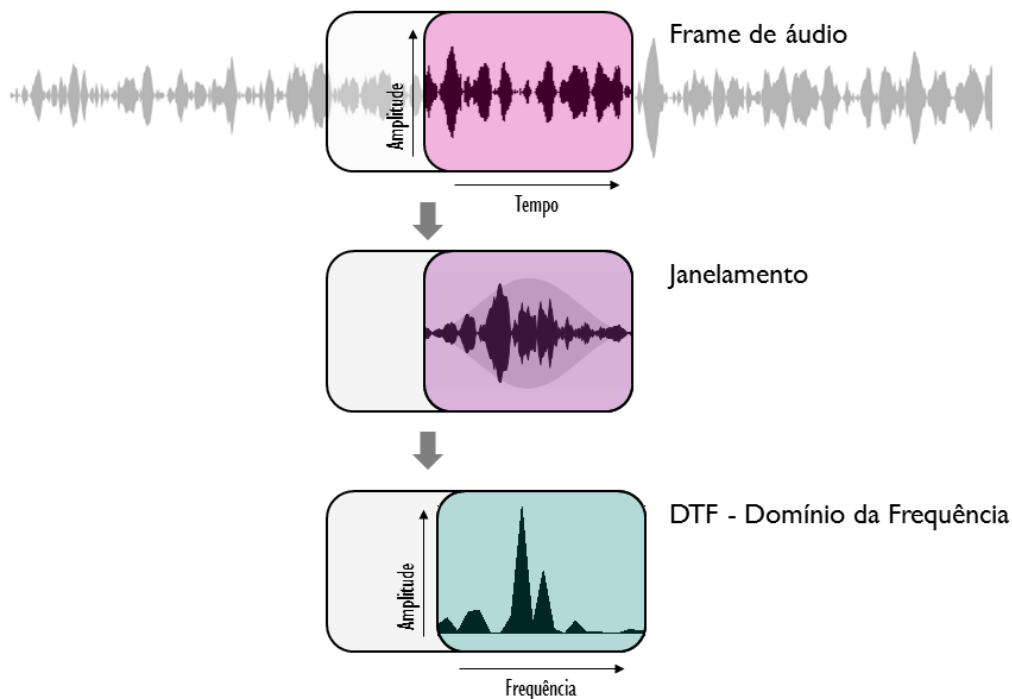


Figura 4.3: Transformações para representação do sinal no domínio da frequência. Adaptado de [Heittola et al. 2018].

near, que é uma representação 2D de um som em que a energia em cada faixa de frequência é dada em função do tempo, representação tempo-frequência. Uma vez que construir um modelo de aprendizado acústico a partir do sinal de áudio original sem um grande custo computacional é uma tarefa desafiadora, grande parte dos trabalhos da literatura que trabalham utilizam o espectrograma como representação de entrada para a CNN. Apesar disso, existem trabalhos que usam diretamente a forma de onda no domínio do tempo como entrada para CNN [Dai et al. 2017]. Neste capítulo, a etapa de aprendizagem de características é realizada através da CNN que, em geral, trabalha com uma entrada 2D. Portanto, nós utilizamos o espectrograma como entrada para a CNN. O código 4.4 é responsável pela geração dos espectrogramas. É importante destacar que esse código é utilizado tanto para gerar os espectrogramas do conjunto de treino quanto do teste.

Neste capítulo, nós utilizamos como representação de entrada da CNN, um log-mel-espectrograma com 100 bandas derivado da STFT com os seguintes parâmetros: taxa de amostragem: 44.100 kHz, função de janelamento: Blackman-Harris, tamanho da janela: 2048 e sobreposição: 50%. Esses parâmetros são definidos nas linhas 18-22 do código 4.4. O tamanho dessa entrada é definido pelo número de amostras *versus* o número de bandas (100) de frequência do sinal, definido pelo parâmetro *n_mels*. No entanto, o número de amostras é determinado em função de um conjunto de parâmetros utilizados no janelamento, entre eles: taxa de amostragem, tamanho da janela, porcentagem de sobreposição e tamanho do áudio. A CNN trabalha melhor com entradas quadradas, ou seja, números de linhas iguais aos números de colunas, portanto, neste capítulo, nós fixamos o tamanho do áudio em 2.3 segundos a fim de gerar espectrogramas de dimensão 100x100, conforme parâmetro definido na linha 56 do código 4.4.

O sistema de audição humana não interpreta o tom de maneira linear, conforme a escala da frequência. Na tentativa de representar como o sistema de audição humano percebe o som, foi criada a escala Mel [Stevens et al. 1937] a partir de um experimento. Tal experimento constatou que o tom é percebido linearmente na faixa de frequência de 0-1000 Hz. No entanto, acima de 1000 Hz, a percepção é descrita na escala logarítmica. Note que o espectrograma é colocado na escala mel, portanto, a escala de frequência f sofre uma transformação não-linear [Fant 1968] para alcançar a escala $mel(f)$, tal transformação é estabelecida pela Equação 1:

$$mel(f) = \frac{1000}{\log 2} \log\left(1 + \frac{f}{1000}\right). \quad (1)$$

A biblioteca *LibROSA* ¹² proveu todas as funções de carregamento do áudio, aplicação da STFT e geração do mel-espectrograma, linhas 56,59 e 63. Em seguida, aplicamos uma normalização dos dados através da função *normalization*, definida nas linhas 29-32, gerando um log-mel-espectrograma. Esses espectrogramas foram armazenados juntamente com suas respectivas classes em objetos no formato *pickle* ¹³ que permite fácil compartilhamento e reconstrução pelos scripts da CNN, conforme as linhas 73-77.

```

import pandas as pd
import numpy as np
import os
import struct
import glob
import re #regex
from scipy.io import wavfile as wav
import subprocess
import librosa
from scipy import signal
import pickle

#Caminho do conjunto de teste e conjunto de treino
path_database = '/data/deborah/UrbanSound8K/teste/'
audio_format = 'wav'

#Parâmetros do espectrograma
sr=44100
hop_length=1024
n_fft=2048
window = signal.blackmanharris(2048)
n_mels=100

#Listas que armazenarão os espectrogramas e classe de cada áudio
specs = []
labels = []

#Função que normaliza os dados do espectrograma
def normalization(data):
    data = np.log10(10000*data+1)
    data = (data-np.mean(data))/np.std(data)

```

¹²<https://librosa.github.io/>

¹³<https://docs.python.org/3/library/pickle.html>

```

32     return data

34 def run():

36     #Capturando o nome das pastas
    names_folders = glob.glob(path_database + '.*')

38

40     for names_ in names_folders:

42         #Capturando o caminho completo do áudio
        path_audios = glob.glob(names_ + '/*.*' + audio_format)

44

46         for path_ in path_audios:

48             #Captura o nome do áudio
            match_obj = re.sub(names_, "", path_)
            match_obj = re.sub(r'/*.*', "", match_obj)
            match_obj2 = re.sub(match_obj, "", path_)

50             #Captura o label da classe
            label = int(match_obj.split('-')[1])

52

54             os.chdir(match_obj2)
            #Captura os primeiros 2.3 segundos do áudio
            y, sr = librosa.load(match_obj, sr=None, duration=2.3)

56

58             #Aplica a transformada STFT
            S = librosa.stft(y, n_fft= n_fft, hop_length= hop_length,
window=window)
60             S = np.abs(S)

62             #Gera o espectrograma com 100 bandas
            X = librosa.feature.melspectrogram(y=y, sr=sr, S=S, n_mels=
n_mels)

64

66             #Aplica a normalização dos dados
            log_spectrogram = normalization(X)

68             specs.append(log_spectrogram)
            labels.append(label)

70

72             os.chdir(path_database)
            #Salvando as listas em disco na forma de objetos
            with open('log_specs_100_teste.pickle', 'bw') as f:
74                 pickle.dump(specs, f)

76             with open('labels_100_teste.pickle', 'bw') as f:
                pickle.dump(labels, f)

78

run()

```

Código Fonte 4.4: Gerando espectrograma de um áudio.

4.4. Aprendizado Profundo para Extração de Características

Técnicas de aprendizado de máquina podem ser utilizadas para a extração automática de características que descrevem os sinais de áudio. Dentre essas técnicas, as Redes Neurais Convolucionais (CNNs) podem ser utilizadas juntamente com os espectrogramas gerados a partir dos áudios, pois elas são fáceis de treinar quando existe grande quantidade de amostras rotuladas que representam as diferentes classes-alvo [Araújo et al. 2017].

CNNs pertencem a uma categoria de algoritmos baseados em redes neurais artificiais que utilizam a convolução em pelo menos uma de suas camadas [Araújo et al. 2017]. Essas redes consistem em várias camadas diferentes empilhadas em uma arquitetura profunda [Piczak 2015].

A Figura 4.4 ilustra a arquitetura de uma *LeNet* [Lecun et al. 1998], que inicialmente foi proposta para reconhecimento de caracteres em imagens, em seguida foi adaptada para outros problemas, tais como classificação de áudios e processamento de linguagem natural. Novas arquiteturas foram propostas nos últimos anos como forma de melhoria da *LeNet*, no entanto essas versões melhoradas compartilhem os mesmos princípios fundamentais, que são as camadas convolucionais, de *pooling* e totalmente conectada.

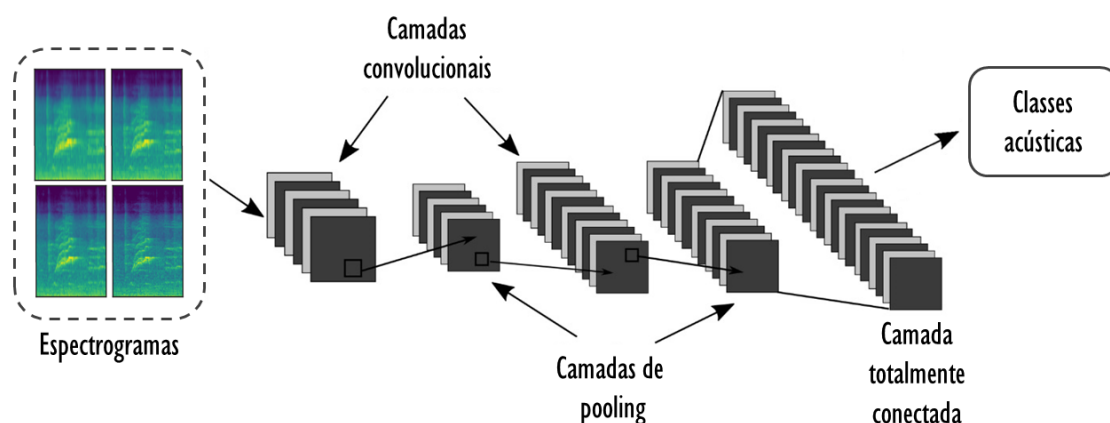


Figura 4.4: Arquitetura da CNN.

As camadas convolucionais são responsáveis por extrair características dos volumes de entradas. As camadas de *pooling* são responsáveis por reduzir a dimensionalidade do volume resultante após as camadas convolucionais e ajudam a tornar a representação invariante a pequenas translações na entrada. As camadas totalmente conectadas são responsáveis pela propagação do sinal por meio da multiplicação ponto a ponto e o uso de uma função de ativação. A saída da CNN é a probabilidade do volume de entrada pertencer a uma das classes para qual a rede foi treinada [Araújo et al. 2017]. Nas próximas sub-seções são detalhadas as principais funções de cada uma dessas camadas.

4.4.1. Camada Convolutiva

As camadas convolucionais consistem de um conjunto de filtros que recebem como entrada um arranjo 3D, também chamado de volume. Cada filtro possui dimensão reduzida, porém ele se estende por toda a profundidade do volume de entrada. Automatica-

mente, durante o processo de treinamento da rede, esses filtros são ajustados para que sejam ativados em presença de características relevantes identificadas no volume de entrada [Karpathy 2017].

Cada um desses filtros dá origem a uma estrutura conectada localmente que percorre toda a extensão do volume de entrada. A somatória do produto ponto a ponto entre os valores de um filtro e cada posição do volume de entrada é uma operação conhecida como convolução. Os valores resultantes após a operação de convolução passam por uma função de ativação, e a mais comum é a função *ReLU* (Rectified Linear Units) [Krizhevsky et al. 2012]. Essa função pode ser calculada pela Equação 2.

$$f(x) = \max(0, x). \quad (2)$$

Os filtros são responsáveis por extrair características diferentes no volume de entrada. Portanto, quanto maior o número de filtros maior o número de características extraídas, porém a complexidade computacional, relativa ao tempo e ao uso de memória, também será maior.

4.4.1.1. Camada de *Pooling*

Após uma camada convolucional, geralmente existe uma camada de *pooling*. O objetivo dessa camada é reduzir progressivamente a dimensão espacial do volume de entrada, consequentemente a redução diminui o custo computacional da rede e evitar *overfitting* [Karpathy 2017]. Na operação de *pooling*, os valores pertencentes a uma determinada região do mapa de atributos, gerados pelas camadas convolucionais, são substituídos por alguma métrica dessa região. A forma mais comum de *pooling* consiste em substituir os valores de uma região pelo valor máximo [Goodfellow et al. 2016]. Essa operação é conhecida como *max pooling* e é útil para eliminar valores desprezíveis, reduzindo a dimensão da representação dos dados e acelerando a computação necessária para as próximas camadas, além de criar uma invariância a pequenas mudanças e distorções locais. Outras funções de *pooling* comumente usadas são a média, a norma L2 e a média ponderada baseada na distância partindo do pixel central [Aggarwal et al. 2001].

4.4.1.2. Camada Totalmente Conectada

A saída das camadas convolucionais e de *pooling* representam as características extraídas do volume de entrada. O objetivo das camadas totalmente conectadas é utilizar essas características para classificar o volume de entrada em uma classe pré-determinada. As camadas totalmente conectadas são exatamente iguais a uma rede neural artificial convencional (*Multi Layer Perceptron* ou MLP) [Haykin et al. 2009] que usa a função de ativação *softmax* [Bishop 2006] na última camada (de saída).

Essas camadas são formadas por unidades de processamento conhecidas como neurônio, e o termo “totalmente conectado” significa que todos os neurônios da camada anterior estão conectados a todos os neurônios da camada seguinte.

A última camada da rede utiliza *softmax* como função de ativação. Essa função recebe um vetor de valores como entrada e produz a distribuição probabilística do volume de entrada pertencer a cada umas das classes na qual a rede foi treinada. Vale destacar que a soma de todas as probabilidades é igual a 1.

Uma técnica conhecida como *dropout* [Goodfellow et al. 2016] também é bastante utilizada entre as camadas totalmente conectadas para reduzir o tempo de treinamento e evitar *overfitting*. Essa técnica consiste em remover, aleatoriamente a cada iteração de treinamento, uma determinada porcentagem dos neurônios de uma camada, readicionando-os na iteração seguinte. Essa técnica também confere à rede a habilidade de aprender atributos mais robustos, uma vez que um neurônio não pode depender da presença específica de outros neurônios.

4.4.2. Código em python

O código 4.5 contém a declaração das principais funções necessárias para o treinamento da CNN. Já o código 4.6 contém a chamada dessas funções e configurações dos parâmetros de treino, esse código considera que os espectrogramas dos áudios foram calculados e salvos na estrutura apresentada no código 4.4. A arquitetura utilizada foi uma LeNet com 3 camadas convolucionais todas de kernel 5x5. Cada uma dessas camadas era seguida por uma camada de max pooling com kernel 2x2. As duas últimas camadas eram totalmente conectadas e possuíam dropout antes delas. A rede foi treinada por 150 épocas e a taxa inicial de aprendizado foi de 0,001.

```
1 #Pacotes utilizados
2 import numpy as np
3 import pandas as pd
4 import pickle
5 from sklearn.metrics import accuracy_score, confusion_matrix
6 from keras import models, optimizers, layers, losses, utils
7 from sklearn.ensemble import RandomForestClassifier
8
9 #Função que recebe o endereço do pickle e retorna o conteúdo no formato
   de numpy array
10 def read_pickle(name):
11     with (open(name, 'rb')) as openfile:
12         while True:
13             try:
14                 one_instance = pickle.load(openfile)
15             except EOFError:
16                 break
17     one_instance = np.asarray(one_instance)
18     return one_instance
19
20 #Função que recebe os espectrogramas e os labels e transforma para o
   formato de entrada da rede aceito pelo Keras
21 def pre_processing_datas_to_cnn_format(X, y):
22     X = X.reshape(X.shape[0], X.shape[1], X.shape[2], 1)
23     y = utils.to_categorical(y)
24     return X, y
25
26 #Função que cria a arquitetura de rede
27 def create_model(shape_in, num_classes = 10, dropout_value = 0.5):
```

```

29     model = models.Sequential()
31     #Primeira camada convolucional
    model.add(layers.Conv2D(32, kernel_size=(5,5), activation = 'relu',
        padding='same', input_shape=(shape_in.shape[0], shape_in.shape[1],
        shape_in.shape[2]), name = 'conv_1'))
33     #Primeira camada de pooling
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2),
        name = 'pool_1', padding='same'))
35
    model.add(layers.BatchNormalization())
37
    #Segunda camada convolucional
39    model.add(layers.Conv2D(64, kernel_size=(5,5), activation = 'relu',
        padding='same', name = 'conv_2'))
41
    #Segunda camada de pooling
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2),
        name = 'pool_2', padding='same'))
43
    model.add(layers.BatchNormalization())
45
    #Terceira camada convolucional
47    model.add(layers.Conv2D(64, kernel_size=(5,5), activation = 'relu',
        padding='same', name = 'conv_3'))
49
    #Terceira camada de pooling
    model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2),
        name = 'pool_3', padding='same'))
51
    model.add(layers.BatchNormalization())
53
    model.add(layers.Flatten(name = 'flatten'))
55
    #Dropout
57    model.add(layers.Dropout(dropout_value))
59
    #Camada totalmente conectada
    model.add(layers.Dense(512, name = 'dense_1', activation='relu'))
61
    #Dropout
63    model.add(layers.Dropout(dropout_value))
65
    #Camada de saída
    model.add(layers.Dense(num_classes, activation='softmax', name = '
        classification'))
67
    #Para visualizar a arquitetura da rede
69    model.summary()
    return model

```

Código Fonte 4.5: Funções necessárias para o treinamento da CNN.

```

1     #Endereços dos pickles contendo os espectogramas e labels

```

```

3 path_train = '/data/deborah/UrbanSound8K/log_specs_100_treino.pickle'
  path_train_labels = '/data/deborah/UrbanSound8K/labels_100_treino.
    pickle'
5 path_test = '/data/deborah/UrbanSound8K/log_specs_100_teste.pickle'
  path_test_labels = '/data/deborah/UrbanSound8K/labels_100_teste.pickle'
7
8 #Nome do modelo que será salvo
9 path_model = 'trained_model_augmented.h5'
  learning_rate = 0.001
11 epochs = 150
  batch_size = 128
13 num_classes = 10
  dropout_value = 0.5
15 decay_ = 1e-3
17
18 X_train = read_pickle(path_train)
  y_train = read_pickle(path_train_labels)
19 X_test = read_pickle(path_test)
  y_test = read_pickle(path_test_labels)
21
22 #Transformando os espectogramas e labels
23 X_train, y_train = pre_processing_datas_to_cnn_format(X_train, y_train)
  X_test, y_test = pre_processing_datas_to_cnn_format(X_test, y_test)
25
26 #Criando o modelo e configurando os parâmetros de treinamento
27 model = create_model(X_train[0], num_classes, dropout_value)
  sgd = optimizers.SGD(lr=learning_rate, decay=decay_)
29 model.compile(loss=losses.categorical_crossentropy, optimizer=sgd,
    metrics=['accuracy'])
31
32 #Treinamento
  model.fit(X_train, y_train, validation_data = (X_test, y_test),
    batch_size=batch_size, epochs=epochs, verbose=1)
33
34 #Salvando o modelo treinado
35 model.save(path_model)

```

Código Fonte 4.6: Configuração de parâmetros e chamada das funções de treino.

O código 4.7 contém a função que faz a extração das características dos áudios. Esse código faz a leitura do modelo de CNN salvo no código 4.5.

```

2 #Função que recebe os espectogramas e o endereço do modelo treinado e
  retorna os características obtidas, que são as saídas da penultima
  camada da rede
3 def extract_features(X_test, path_model):
4     model = models.load_model(path_model)
5
6     intermediate_layer_model = models.Model(inputs=model.input, outputs
      =model.get_layer(index = -2).output)
      features = intermediate_layer_model.predict(X_test)
8
9     features = pd.DataFrame(data=features)
10
11     return features

```

Código Fonte 4.7: Função que faz a extração das características dos espectogramas.

4.5. Classificação de Eventos Sonoros

O objetivo de uma tarefa de classificação é induzir um modelo por meio de um conjunto expressivo de dados previamente rotulados, para classificar novos exemplos ainda não rotulados. Isso é feito por meio do mapeamento entre as relações existentes entre os atributos de predição e o atributo alvo.

A literatura apresenta diversos modelos que realizam a classificação de dados, tais como Redes Neurais, Árvores de Decisão, *Naive Bayes*, dentre outros. Nesse trabalho foi utilizado a *Random Forest* [Breiman 2001], que é um classificador simples, com poucos parâmetros e que produz bons resultados de classificação.

4.5.1. *Random Forest*

O algoritmo *Random Forest* é uma combinação de predições de diversas árvores em que cada árvore depende dos valores de um vetor independente, amostrados aleatoriamente e com a mesma distribuição para todas as árvores da floresta. Aqui, floresta é o que se denomina para uma série de Árvores de Decisão. Após a geração de um grande número de árvores, as classes com maior número de votos são eleitas.

Em uma *Random Forest*, cada nó é dividido usando o melhor dentre um subconjunto de indicadores escolhidos aleatoriamente naquele nó. Esta estratégia um tanto contraditória acaba por funcionar muito bem em comparação com muitos outros classificadores, além de ser robusto a superajuste nos parâmetros. Além disso, é de fácil utilização, no sentido que ele tem apenas dois parâmetros (o número de variáveis no subconjunto aleatório em cada nó e o número de árvores da floresta) e, normalmente, não é muito sensível aos seus valores. A partir de um vetor de atributos, são gerados outros vetores de atributos, que são embaralhados em relação ao vetor original. É gerado um vetor para cada árvore da *Random Forest*. Em seguida, os vetores de atributos são passados como parâmetro para as Árvores de Decisão. Cada árvore irá gerar um resultado para a classificação e, após isso, os resultados são combinados obtendo uma saída unificada. A Figura 4.5 mostra os passos do classificador *Random Forest*.

4.5.2. Métricas de avaliação dos classificadores

A maioria dos critérios de análise dos resultados de uma classificação parte de uma matriz de confusão, que indica a quantidade de classificações corretas e incorretas para cada uma das classes. A partir desses valores podem ser calculadas diferentes métricas, tais como: Acurácia, Precisão, *Recall*, dentre outras. A acurácia foi a métrica utilizada para a avaliação dos resultados, ela representa a porcentagem de elementos corretamente classificados dividido pelo número total de amostras.

4.5.3. Código em *python*

O código 4.8 contém a classificação dos espectogramas do conjunto de teste utilizando a CNN treinada no código 4.5. Já o código 4.9 utiliza a CNN para a extração de caracte-

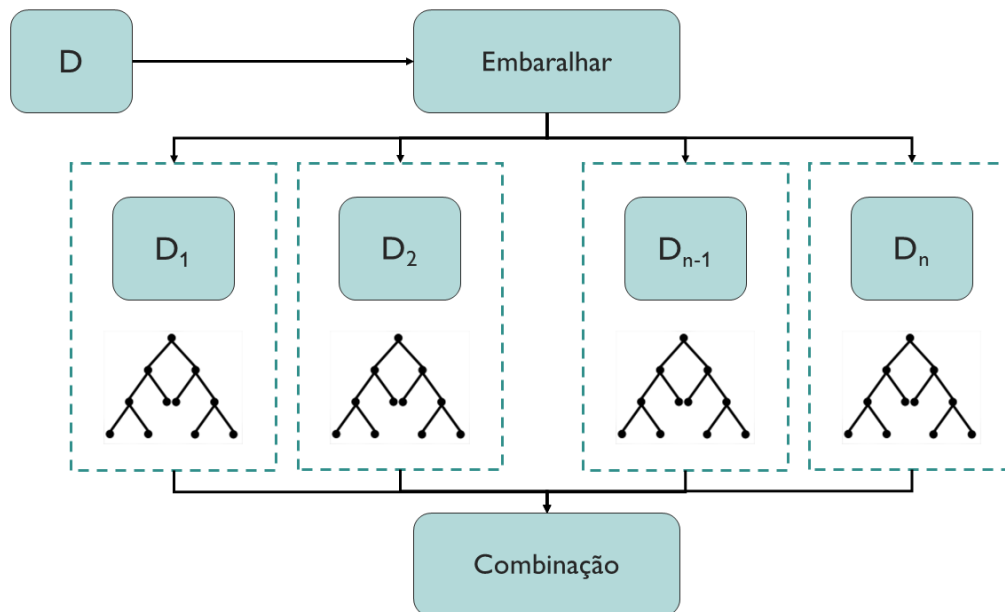


Figura 4.5: Passos do classificador *Random Forest*, onde D é o vetor de atributos original, $D_1, D_2 \dots, D_{n-1}, D_n$ são os vetores de atributos gerados a partir de D .

rísticas dos espectrogramas e em seguida faz o treinamento e classificação utilizando uma Random Forest.

```

2 #Lendo o modelo salvo após o treinamento
  model = models.load_model(path_model)
4
6 #Fazendo a predição dos dados do conjunto de teste com a rede treinada
  pred = model.predict_classes(X_test)
8
8 #Imprimindo a acuracia e a matriz de confusão
  print('Acuracia da rede = ', accuracy_score(pred, np.argmax(y_test, axis
    =1)))
10 print('Matriz de confusão da rede = \n', confusion_matrix(pred, np.
    argmax(y_test, axis=1)))
  
```

Código Fonte 4.8: Função que faz a classificação dos espectrogramas utilizando um modelo de CNN treinado.

```

1
1 #Utilizando a rede treinada para extrair as características dos
  espectrogramas
3 X_train_features = extract_features(X_train, path_model)
  X_test_features = extract_features(X_test, path_model)
5
5 #Treinando uma Random Forest
7 clf = RandomForestClassifier(n_estimators=100)
  clf.fit(X_train_features, np.argmax(y_train, axis=1))
9
9 #Predição com a Random Forest treinada
11 predicted = clf.predict(X_test_features)
  
```

```

13 #Imprimindo a acuracia e a matriz de confusão
print('Acuracia da Random Forest = ', accuracy_score(predicted , np.argmax
(y_test , axis=1)))
15 print('Matriz de confusão da Random Forest = \n', confusion_matrix(
predicted , np.argmax(y_test , axis=1)))

```

Código Fonte 4.9: Função que faz a classificação dos espectrogramas utilizando uma Random Forest.

As acurácias obtidas pela classificação utilizando a CNN e a Random Forest foram de 68,92% e 68,14% respectivamente. As matrizes de confusão são apresentadas nas Tabelas 4.1 e 4.2

Tabela 4.1: Matriz de confusão da classificação utilizando a CNN.

Ar condicionado	Buzina de carro	Crianças brincando	Latido de cachorro	Furadeira	Motor de veículo	Tiro	Britadeira	Sirene	Música de rua
79	0	3	1	7	14	0	0	0	6
0	31	0	0	3	0	0	0	2	2
29	0	178	13	9	9	0	1	15	11
3	0	6	116	4	7	0	0	0	8
3	1	0	0	103	0	1	3	0	5
56	0	0	2	0	134	0	105	3	0
1	0	1	0	1	26	23	0	1	0
19	0	0	0	18	0	0	82	0	4
0	0	3	4	4	4	0	21	147	7
9	0	9	1	8	1	0	2	1	165

Tabela 4.2: Matriz de confusão da classificação utilizando a Random Forest.

Ar condicionado	Buzina de carro	Crianças brincando	Latido de cachorro	Furadeira	Motor de veículo	Tiro	Britadeira	Sirene	Música de rua
100	0	5	1	7	34	0	7	0	8
0	31	0	0	3	0	0	0	4	2
17	0	168	10	7	8	0	1	14	11
6	0	8	118	3	1	1	0	9	1
3	1	1	0	109	14	0	28	0	5
49	0	0	3	0	122	0	93	3	0
1	0	0	1	0	11	23	0	0	0
14	0	0	0	17	1	0	68	0	3
0	0	3	3	4	4	0	15	144	7
9	0	15	1	7	0	0	2	3	163

Referências

- [Aggarwal et al. 2001] Aggarwal, C. C., Hinneburg, A., e Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In *Lecture Notes in Computer Science*, pages 420–434. Springer.
- [Araújo et al. 2017] Araújo, F. H. D., Carneiro, A. C., Silva, R. R. V., Medeiros, F. N. S., e Ushizima, D. (2017). *Redes Neurais Convolucionais com Tensorflow: Teoria e Prática*, volume 1.

- [Bello et al. 2018] Bello, J. P., Mydlarz, C., e Salamon, J. (2018). Sound analysis in smart cities. In *Computational Analysis of Sound Scenes and Events*, pages 373–397. Springer.
- [Bishop 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
- [Borges et al. 2017] Borges, J., Ziehr, D., Beigl, M., Cacho, N., Martins, A., Sudrich, S., Abt, S., Frey, P., Knapp, T., Etter, M., et al. (2017). Feature engineering for crime hotspot detection. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, pages 1–8. IEEE.
- [Breiman 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.
- [Dai et al. 2017] Dai, W., Dai, C., Qu, S., Li, J., e Das, S. (2017). Very deep convolutional neural networks for raw waveforms. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 421–425. IEEE.
- [Edwards 2018] Edwards, J. (2018). Signal processing opens the internet of things to a new world of possibilities: Research leads to new internet of things technologies and applications [special reports]. *IEEE Signal Processing Magazine*, 35(5):9–12.
- [Fant 1968] Fant, G. (1968). Analysis and synthesis of speech processes. *Manual of phonetics*, 2:173–277.
- [Font et al. 2018] Font, F., Roma, G., e Serra, X. (2018). Sound sharing and retrieval. In *Computational Analysis of Sound Scenes and Events*, pages 279–301. Springer.
- [Giannakopoulos et al. 2019] Giannakopoulos, T., Spyrou, E., e Perantonis, S. (2019). Recognition of urban sound events using deep context-aware feature extractors and handcrafted features. In *Artificial Intelligence Applications and Innovations*, pages 184–195. Springer International Publishing.
- [Goodfellow et al. 2016] Goodfellow, I., Bengio, Y., e Courville, A. (2016). *Deep learning (adaptive computation and machine learning series)*. Cambridge: The MIT Press.
- [Haykin et al. 2009] Haykin, S. S., Haykin, S. S., Haykin, S. S., e Haykin, S. S. (2009). *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:.
- [He e Garcia 2008] He, H. e Garcia, E. A. (2008). Learning from imbalanced data. *IEEE Transactions on Knowledge & Data Engineering*, (9):1263–1284.
- [Heittola et al. 2018] Heittola, T., Çakır, E., e Virtanen, T. (2018). The machine learning approach for analysis of sound scenes and events. In *Computational Analysis of Sound Scenes and Events*, pages 13–40. Springer.
- [Karpathy 2017] Karpathy, A. (2017). Convolutional neural networks for visual recognition. <http://cs231n.github.io/convolutional-networks/>.

- [Krizhevsky et al. 2012] Krizhevsky, A., Sutskever, I., e Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., e Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.
- [Lecun et al. 1998] Lecun, Y., Bottou, L., Bengio, Y., e Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.
- [Piczak 2015] Piczak, K. J. (2015). Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE.
- [Rathore et al. 2016] Rathore, M. M., Ahmad, A., Paul, A., e Rho, S. (2016). Urban planning and building smart cities based on the internet of things using big data analytics. *Computer Networks*, 101:63–80.
- [Salamon e Bello 2017] Salamon, J. e Bello, J. P. (2017). Deep convolutional neural networks and data augmentation for environmental sound classification. *IEEE Signal Processing Letters*, 24(3):279–283.
- [Salamon et al. 2014] Salamon, J., Jacoby, C., e Bello, J. P. (2014). A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 1041–1044. ACM.
- [Serizel et al. 2018] Serizel, R., Bisot, V., Essid, S., e Richard, G. (2018). Acoustic features for environmental sound analysis. In *Computational Analysis of Sound Scenes and Events*, pages 71–101. Springer.
- [Souza et al. 2018] Souza, T. I., Magalhaes, D., Aquino, A. L., e Gomes, D. G. (2018). Um método para detecção e diagnóstico de outliers em dados urbanos via análise multidimensional. In *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. SBC.
- [Stevens et al. 1937] Stevens, S. S., Volkman, J., e Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190.
- [Virtanen et al. 2018] Virtanen, T., Plumbley, M. D., e Ellis, D. (2018). Introduction to sound scene and event analysis. In *Computational analysis of sound scenes and events*, pages 3–12. Springer.

Capítulo

5

Desenvolvimento de Aplicativos Multiplataforma com Flutter

Ian Wanderson da Silva Oliveira e Danilo Borges da Silva

Abstract

Mobile development is growing up increasingly motivated by consumers of mobile applications, just in 2018 were made over 194 billion of application downloads. At present, exists several frameworks that promote development for mobile applications and multiplatform. Flutter sticks out as one of those frameworks for grant native apps for any platform (e.g., Android, iOS, Web, Desktop) and agile development. In this chapter, we introduce Flutter as an alternative that uses the same language, Dart, to develop both front-end and back-end solutions. To show Flutter's capability, we developed a multiplatform application that runs on both the web and Android. Lastly, we present the job market perspectives about this framework.

Resumo

O desenvolvimento mobile está evoluindo cada vez mais fomentado por um mercado que consome mais aplicativos móveis, em 2018 foram mais de 194 bilhões de downloads de aplicativos. Existem hoje diversos frameworks que permitem o desenvolvimento para dispositivos móveis e multiplataforma. O Flutter se destaca como um desses frameworks por possibilitar código nativo para qualquer plataforma (e.g., Android, iOS, Web, Desktop) e desenvolvimento ágil. Neste capítulo apresentamos o Flutter como alternativa para quem busca utilizar uma só linguagem, o Dart, para desenvolver tanto no front-end quanto no back-end. Para ilustrar a capacidade do Flutter desenvolvemos uma aplicação multiplataforma para ser executada na Web e em um dispositivo com Android. Por fim, apresentamos as perspectivas do mercado de trabalho acerca desse framework.

5.1. Introdução

Pensar em abrir uma empresa, ou mesmo vender um produto, hoje sem utilizar pelo menos um aplicativo para suporte está se torando inviável. Vivemos na era dos smartphones,

onde temos acesso em tempo real a tudo que nos cerca e que está conectado pela rede (internet) por meio de aplicativos, conhecidos como “apps”. Eles auxiliam nas vendas, no controle de informações e até mesmo na tomada de decisão sendo bastante populares. Esse fato está relacionado a dois fatores: a quantidade de smartphones no mercado e ao aumento de downloads de aplicativos.

Estima-se que até o ano de 2021 existirão mais de 3,8 bilhões de smartphones no mundo, hoje já existem mais de 3 bilhões em operação, segundo a [Statista 2019]. Atualmente, no Brasil existem mais de 260 milhões de smartphones em uso, é o que revela a 30ª Pesquisa Anual de Administração e Uso de Tecnologia da Informação nas Empresas, realizada pela Fundação Getúlio Vargas de São Paulo (FGV-SP) [Época Negócios 2019].

O outro fator determinante é o número de downloads de aplicativos, em 2018 foram contabilizados mais de 194 bilhões de downloads [Statista 2019]. O Brasil aparece entre os principais mercados em número de downloads no mundo, considerando tanto aplicativos para iOS quanto para Android. Segundo um levantamento feito pela App Annie, empresa norte-americana de dados do mercado de aplicativos, mostra que a quantidade de apps baixados em 2017 superou a marca de 175 bilhões de programas, impulsionado principalmente por mercados emergentes como o Brasil e a Índia [Exame 2019]. Outra pesquisa aponta que o Brasil é o segundo mercado de aplicativos que mais cresce no mundo. Segundo levantamento do Adjust, empresa de análise e prevenção de fraudes do setor, nosso país está atrás apenas da Indonésia e a frente da Coreia do Sul [CanalTech 2019].

Esses dados e fatos revelam o quanto o mercado pede por desenvolvedores na área de dispositivos móveis, popularmente conhecidos como desenvolvedores mobile, com ênfase no que se refere ao desenvolvimento multiplataforma. Como existem várias plataformas importantes no mercado, a exemplo, as do iOS da Apple, Android do Google, e Web, uma consequência da abordagem tradicional de desenvolvimento é desenvolver uma solução para cada plataforma, o que exige árduo esforço [Biørn-Hansen et al. 2019]. Usualmente adota-se uma equipe para desenvolver para cada uma dessas plataformas, porém existem hoje SDKs (Kits de Desenvolvimento de Software) que possibilitam ao desenvolvedor mobile exportar sua solução para diversas plataformas utilizando uma única ferramenta, como é o caso do Flutter.

O Flutter é um kit de ferramentas (framework) de interface do usuário (UI) desenvolvido pela Google para criar aplicativos bonitos e compilados nativamente para dispositivos móveis, web e desktop a partir de uma única base de código [Flutter 2019]. Seu forte atrativo estão baseados em três pilares: (a) Desenvolvimento Rápido, (b) UI Expressiva e Flexível e (c) Desempenho Nativo. (a) O Desenvolvimento Rápido, é caracterizado por um rico conjunto de widgets totalmente personalizáveis para criar interfaces nativas em minutos. (b) Seu modo de produzir UI Expressiva e Flexível permite com que os recursos sejam enviados rapidamente, com foco nas experiências nativas do usuário final. A arquitetura em camadas permite a personalização completa, o que resulta em uma renderização incrivelmente rápida. (c) Os widgets do Flutter incorporam todas as diferenças críticas de plataforma, como rolagem, navegação, ícones e fontes, para fornecer Desempenho Nativo completo no iOS e no Android. Dessa forma, o que antes exigia bastante esforço para desenvolver nativamente para cada uma dessas plataformas com o Flutter esse problema é

resolvido. Entende-se pelo termo nativo o desenvolvimento feito baseado na linguagem nativa da plataforma em questão.

Nas próximas seções apresentaremos essa ferramenta bastante poderosa que está em constante crescimento em quantidade de adeptos e exigência do mercado, apesar de ter sido lançada em 2017. Na Seção 5.2, apresentaremos o Dart. Como citamos anteriormente, o Flutter é um framework que por sua vez utiliza o Dart como linguagem de programação. Na Seção 5.3, apresentaremos com mais detalhes o que é o Flutter e as Widgets, especificando a estrutura desta tecnologia e o porque de entender o conceito de Widgets dentro do Flutter. Na Seção 5.4, apresentaremos o desenvolvimento de algumas telas em duas plataformas: Android e Web. Para isso, utilizaremos o estudo de caso de uma aplicação baseada em um chat, intitulado EnuChat. Por fim, na Seção 5.5 falaremos sobre as perspectivas de mercado por desenvolvedores Flutter.

5.2. Dart: a linguagem do Flutter

Esta seção foi baseada no artigo de [Hackernoon 2018], no livro online de [Windmill 2019] e no livro de [Zammetti 2019]. Dart é uma linguagem de programação concisa, fortemente orientada a objetos usada para codificar aplicativos Flutter. Quando o Google começou a trabalhar no Flutter, eles tiveram que tomar uma decisão antecipada: qual linguagem de programação ele usaria? Ao invés de escolher uma linguagem baseada em plataformas já consagradas como por exemplo, Javascript para Web, Android ou Swift para Dispositivos Móveis ou Java para Desktop, a Google optou por trabalhar com uma linguagem que haviam desenvolvido: o Dart. Os motivos para esta escolha apresentaremos nesta seção. Antes disso, falaremos como foi a criação do Dart.

Em 2011 foi criado o Dart, apresentado inicialmente na conferência que aconteceu de 10 a 11 de outubro de 2011 em Aarhus, na Dinamarca. O objetivo da linguagem Dart foi inicialmente a de substituir o JavaScript como a linguagem principal embutida nos navegadores. Em novembro de 2013, foi lançada a primeira versão estável, Dart 1.0, e em agosto de 2018 foi lançado o Dart 2.0, um reboot da linguagem, otimizado para o desenvolvimento *client-side* para Web e dispositivos móveis pensado para o desenvolvimento multiplataforma.

O Dart é uma linguagem elegante que está rapidamente ganhando muita força, principalmente desde a concepção do Flutter, embora seja uma linguagem de uso geral. Ele pode e é usado para criar todos os tipos de aplicações, de aplicativos da Web a códigos de servidor e aplicativos de IoT (Internet das Coisas). O aumento da adoção desta linguagem se reflete em uma pesquisa realizada em 2019 que destaca as mais importantes linguagens para desenvolvedores publicado pela JAXenter¹ mostrando no topo: Dart e Python. Em outra pesquisa realizada por uma grande comunidade de desenvolvedores Stackoverflow² destacou em terceiro lugar, até outubro de 2019, o Flutter como um das ferramentas mais queridas, temidas e desejadas. Isso mostra o quanto o Dart está recebendo bastante atenção pela comunidade de desenvolvedores.

Pode-se destacar que o Dart é uma linguagem influenciada cinco por linguagens

¹<https://jaxenter.com/poll-results-dart-word-2019-154779.html>

²<https://insights.stackoverflow.com/survey/2019>



```
void main() {  
  print('Olá, Mundo!');  
}
```

Figura 5.1: Olá, Mundo escrito em Dart.

de programação: C, C++, Java, C# e Javascript. Embora outras também tenham tido alguma influência essas são as principais. Isso se retrata nos paradigmas que ele implementa: Orientado a Objetos (OO), Imperativo, Reflexivo e Funcional. Suportando na OO: interfaces, classes abstratas, genéricas, tipos opcionais e herança simples.

Para iniciar a implementar com o Dart pode-se utilizar ferramentas online como o DartPad³, para testes simples. Para instalar no computador é preciso instalar seu SDK, disponível em [Dart 2019]. Na Figura 5.1, observa-se a impressão do “Olá, mundo!” feito com Dart. Como em C/C++, a primeira função a ser chamada em um programa escrito em Dart é a função `main()`.

O Dart também pode ser utilizado para escrever scripts simples ou aplicativos completos. Como por exemplo: script de linha de comando ou aplicativo do servidor. Como pode-se observar a Figura 5.2. A tecnologia flexível do compilador do Dart permite executar o código Dart de diferentes maneiras, dependendo da plataforma e dos objetivos de destino:

- **Dart Native:** para programas direcionados a dispositivos (móveis, desktop, servidor, entre outros), o Dart Native utiliza dois compiladores, o compilador Dart VM (*Virtual Machine*) com JIT (*just-in-time*) e o compilador AOT (*Ahead-Of-Time*) para produzir código de máquina; e
- **Dart Web:** para programas direcionados à Web, o Dart Web inclui um compilador em tempo de desenvolvimento (`dartdevc`) e um compilador em tempo de produção (`dart2js`).

Essa robustez entregue pelo Dart SDK faz com que o mesmo seja utilizado para gerar códigos em diversas plataformas, possuindo várias características como:

- **Interface do usuário otimizado:** é possível desenvolver com uma linguagem de programação especializada em torno das necessidades de criação da interface do usuário.
- **Produtividade:** as alterações podem ser feitas de forma interativa usando o *hot reload* para ver o resultado instantaneamente no seu aplicativo em execução. Isso

³<https://dartpad.dartlang.org/>

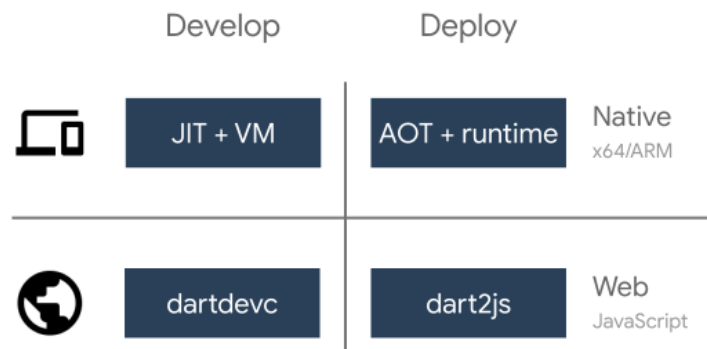


Figura 5.2: Plataformas de desenvolvimento do Dart [Dart 2019]

permite com que não se espere um tempo considerável na compilação para se poder observar as alterações que estão sendo feitas.

- Rápido em todas as plataformas: é possível compilar código de máquina ARM e x64 para dispositivos móveis, desktop e *back-end*. Ou mesmo compilar para JavaScript para a Web utilizando o mesmo desenvolvimento.

Em virtude dessas características o Flutter foi criado sob o Dart. Em muitos casos as pessoas chegam a considerar o Dart e Flutter como um só, porém o Dart é mais genérico e o Flutter vem com suas particularidades para facilitar o desenvolvimento multiplataforma. A seguir listamos cinco recursos do Dart que juntos o tornam indispensável para o Flutter:

1. O Dart é AOT compilado em código nativo rápido, previsível, que permite que quase todo o Flutter seja escrito no Dart. Isso não apenas torna o Flutter rápido, praticamente tudo, incluindo todos os Widgets (Seção 5.3), pode ser personalizado.
2. O Dart também pode ser compilado com JIT para ciclos de desenvolvimento excepcionalmente rápidos e fluxos de trabalho que mudam o constantemente, como em jogos onde a atualização de tela acontece com bastante frequência.
3. O Dart facilita a criação de animações e transições suaves que são executadas a 60fps (frames por segundo). Também permitindo a alocação de objetos e coleta de lixo sem bloqueios. E, como no JavaScript, o Dart evita agendamento preventivo e memória compartilhada.
4. O Dart permite ao Flutter evitar a necessidade de uma linguagem de layout declarativa separada, como JSX ou XML, ou de construtores de interface visual separados, porque o layout programático declarativo do Dart é fácil de ler e visualizar. E com todo o layout está em uma mesma linguagem é fácil para o Flutter fornecer ferramentas avançadas que facilitam o mesmo.
5. O Dart é uma linguagem orientada a objetos produtiva e previsível. Isso facilita a criação de experiências visuais do usuário sem a necessidade de uma linguagem

de marcação. É fácil de aprender e possui recursos familiares aos usuários das linguagens estáticas (e.g., C/C++) e dinâmicas (e.g., Javascript).

A seguir falaremos particularmente do framework Flutter sob o aspecto do que o torna tão simples e rápido de se produzir tanto no *front-end* quanto no *back-end* utilizando uma só linguagem nativa, o Dart, por meio de Widgets.

5.3. Flutter e as Widgets

O Flutter é muito novo, mas uma plataforma promissora, que atraiu a atenção de grandes empresas que já lançaram seus aplicativos utilizando-o, como Alibaba, Grupon, Google e Nubank. A adoção do Flutter se deve a sua simplicidade em comparação ao desenvolvimento de aplicações Web e por causa de sua velocidade em comparação com aplicativos nativos que infere diretamente em sua produtividade e alta performace [Quimbundo 2019]. Como é uma tecnologia nova poucos artigos e livros foram escritos até então.

No trabalho de [Yatsenko et al. 2019] foi realizada uma pesquisa a respeito da popularidade dos frameworks para desenvolvimento nas consultas de pesquisa do Google de 2018 até 2019 presente na Figura 5.3. Observe, no gráfico, que o Flutter desde a sua criação está em constante crescimento. Isso se deve às várias técnicas que o Flutter apresenta que trazem novidades para o programador e melhorando o seu desempenho.

Ao contrário de muitas outras plataformas móveis populares, o Flutter não utiliza JavaScript de nenhuma maneira. O Dart compila diretamente para código binário, e é por isso que ele é executado com o desempenho nativo de Objective-C, Swift, Java ou Kotlin. Além disso, o Flutter não utiliza componentes de UI nativos. Os componentes são implementados no próprio Flutter, não existe um camada de comunicação entre a

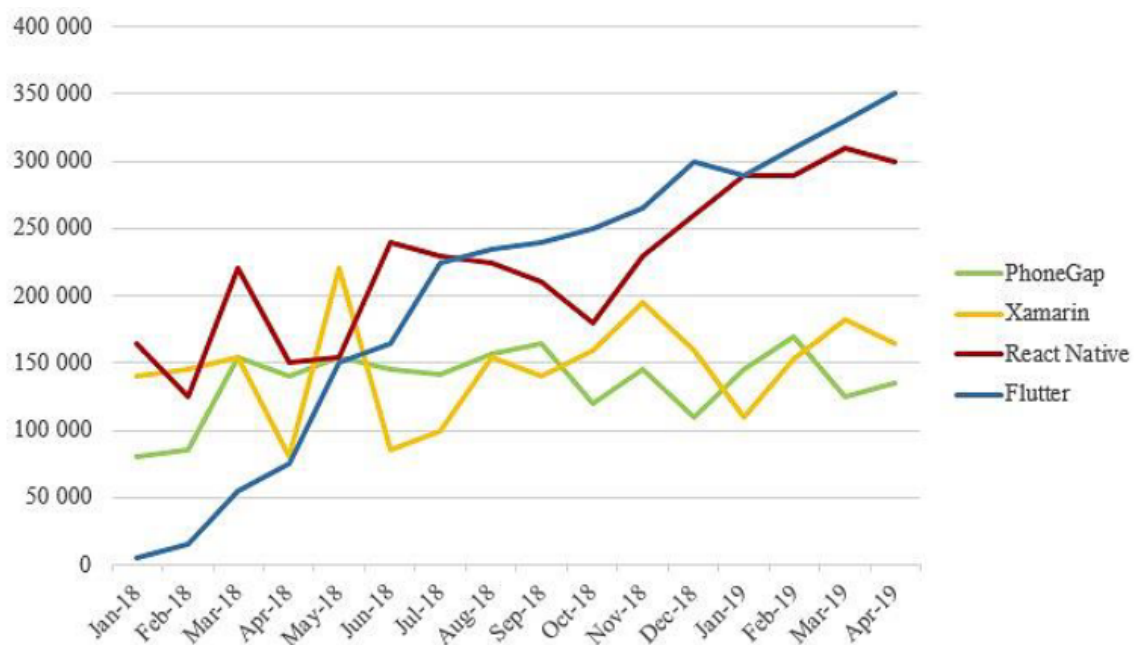


Figura 5.3: Popularidade dos frameworks de acordo com as chaves de pesquisa do Google [Yatsenko et al. 2019].

View e seu código (como acontece no Java). Devido a isso, as aplicações atingem uma melhor velocidade. Ou seja, botões, texto, elementos de mídia, backgrounds são todos desenhados pelo mecanismo de gráficos do Flutter.

O mecanismo de gráficos do Flutter utiliza uma abordagem declarativa, inspirada no framework Web React, para construir sua UI baseada em widgets (chamados “componentes” no mundo web). Para obter mais das widgets, elas são renderizadas apenas quando necessário, geralmente quando seu estado é alterado. Por isso é importante entender o conceito de widgets no Flutter e o contexto desse estado.

No Flutter, tudo são widgets – também vinculada às classes. Assim, os widgets são as partes da interface de usuário. Um *Text* é um widget; *Buttons* são widgets; caixas de seleção são widgets; imagens são widgets, e a lista continua... Na verdade, tudo na interface do usuário é um widget – inclusive o aplicativo em si. Fazendo uma comparação, quem programa em Android ou iOS pode relacionar o conceito de widget no Flutter ao de uma *View* e *UIView*, respectivamente. Porém a ideia é bem mais ampla, pode-se pensar que a widget é um modelo ou template e que o Flutter as usa para criar os elementos de visualização para renderizá-los na tela do dispositivo (Figura 5.4).

Na Figura 5.4, pode-se observar a estrutura de cada elemento do código Flutter que falamos anteriormente tratarem-se de widgets. Esse projeto é o projeto padrão criado por *default*. Vamos fazer uma análise de um trecho do código responsável por criar essa tela, presente na Figura 5.5. Observe que o Flutter trabalha sob uma estrutura de árvore, onde cada nó desta árvore é uma widget, similares a uma classe, que pode possuir ou não atributos. Falaremos mais a respeito dessa árvore na Subseção 5.3.1.

Quando entende-se que os widgets são quase tudo que afeta a aparência e o comportamento da interface, faz sentido que exista mais widgets do que apenas elementos

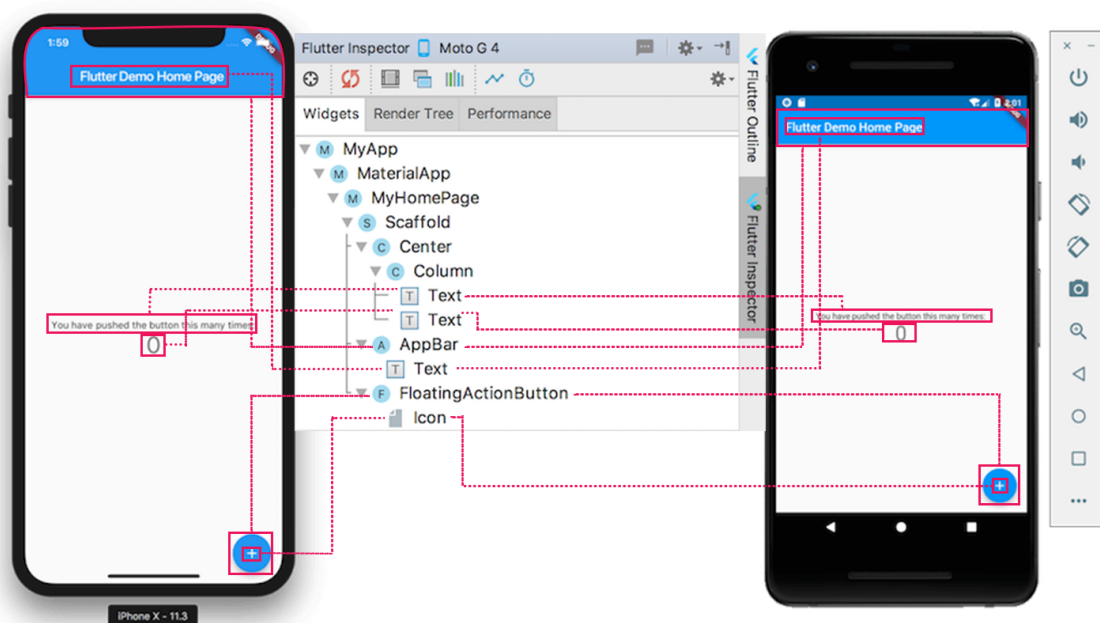


Figura 5.4: Tela inicial de um projeto padrão criado com Flutter.

```
1 @override
2 Widget build(BuildContext context) {
3   return Scaffold(
4     appBar: AppBar(
5       title: Text(widget.title),
6     ),
7     body: Center(
8       child: Column(
9         mainAxisAlignment: MainAxisAlignment.center,
10        children: <Widget>[
11          Text(
12            'You have pushed the button this many times:',
13          ),
14          Text(
15            '$_counter',
16            style: Theme.of(context).textTheme.display1,
17          ),
18        ],
19      ),
20    ),
21    floatingActionButton: FloatingActionButton(
22      onPressed: _incrementCounter,
23      tooltip: 'Increment',
24      child: Icon(Icons.add),
25    ),
26  );
27 }
28 }
```

Figura 5.5: Parte do código responsável pela construção das telas da Figura 5.4.

estruturais, como botões, texto e imagens. Na Figura 5.5 podemos identificar algumas widgets, por exemplo: `Scaffold`, linha 3, responsável por todo o espaço de preenchimento da tela; `AppBar`, linha 4, parte superior da tela onde é armazenado tipicamente o título e abas de navegação; `Center`, linha 7, responsável por fazer o alinhamento de centralização; `Column`, linha 8, cria uma estrutura de coluna; `Text`, linha 5, onde fica localizado o título da tela e linhas 11 e 14, onde são armazenados dois textos centralizados em coluna; `FloatingActionButton`, linha 21, botão no canto inferior direito; e `Icon`, linha 24, imagem de um mais (+). Nesta estrutura todos os widgets possuem finalidades distintas e se encontram separadas em dois tipos `StatelessWidgets` e `StatefulWidgets`. Veremos mais a respeito desses dois tipos de widgets na Subseção 5.3.2 [Macoratti 2019].

5.3.1. Árvores de Widgets

Os widgets são organizados em uma árvore de widgets numa hierarquia pai e filho. Toda a árvore de widgets é o que forma o layout que pode-se visualizar na tela. Por exemplo, na Figura 5.4, a imagem centralizada que mostra a estrutura dos componentes da tela,

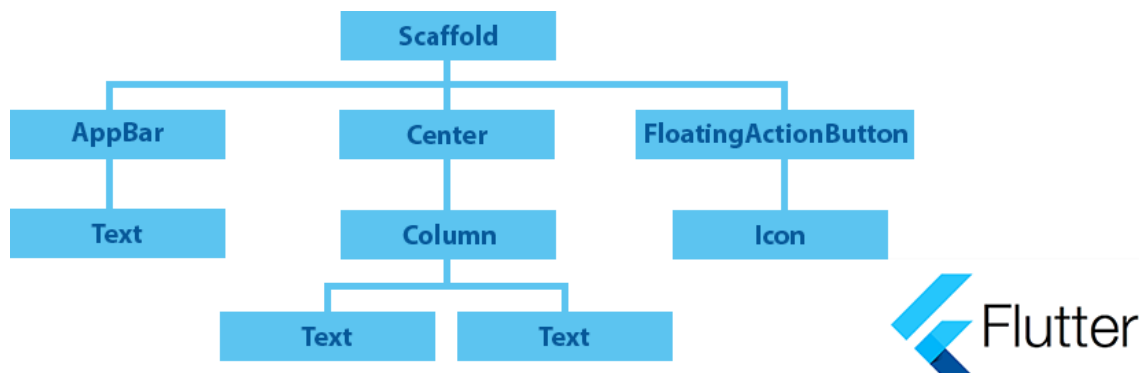


Figura 5.6: Árvore de widgets da aplicação *default* criada no Flutter a partir do `Scaffold` (Figura 5.4).

esta é a árvore de widgets para o projeto *default*. Os widgets visíveis foram marcados em tracejado com linhas vermelhas, para que possa visualizar por outro aspecto de vista produzimos a Figura 5.6. Nessa figura é possível observar melhor a estrutura de árvore criada pelas widgets, além disso é importante observar que as próprias widgets estão dentro de um contexto de herança que as dividem em dois tipos de widgets (Seção 5.3.2).

5.3.2. Tipos de Widgets

Todos os widgets são imutáveis. Ou seja, eles não podem ser alterados. Quaisquer propriedades que eles contêm são finais e só podem ser definidas quando o widget for inicializado. Isso os mantém leves, de modo que é barato recriá-los quando a árvore de widgets for alterada. Existem dois tipos de widgets: (1) `StatelessWidget` (sem estado) e (2) `StatefulWidget` (com estado) [Macoratti 2019].

(1) As widgets sem estado não armazenam nenhum estado. Ou seja, eles não armazenam valores que podem mudar. Por exemplo: um ícone é sem estado – uma vez definida a imagem do ícone quando é criada não é possível alterá-la. (2) O segundo tipo de widget possui estado. Isso significa que ele pode acompanhar as alterações e atualizar a interface do usuário com base nessas alterações. Por exemplo, em uma widget que permite “rolar” uma lista, cada alteração neste comportamento manipula um estado. Na Figura 5.7, pode-se observar a separação entre essas widgets que herdam da classe `Widget`.

As vantagens de optar entre essas duas representações de widgets fazem com que a aplicação possa executar com maior velocidade (como falamos anteriormente). Para mais detalhes a respeito dessa estrutura consulte [Flutter 2019] e as demais referências que usamos neste capítulo. Na Seção 5.4, iremos apresentar o desenvolvimento de um protótipo multiplataforma com Flutter.

5.4. Hands on: EnuChat

Nesta seção vamos construir a interface de usuário (UI) principal do aplicativo, batizado de EnuChat. Neste aplicativo desenvolveremos algumas telas similares aos que os aplicativos de envio de mensagens fazem, como Telegram e Whatsapp. Será possível perceber que em pouco tempo produziremos muito com pouco código. Como editor de código

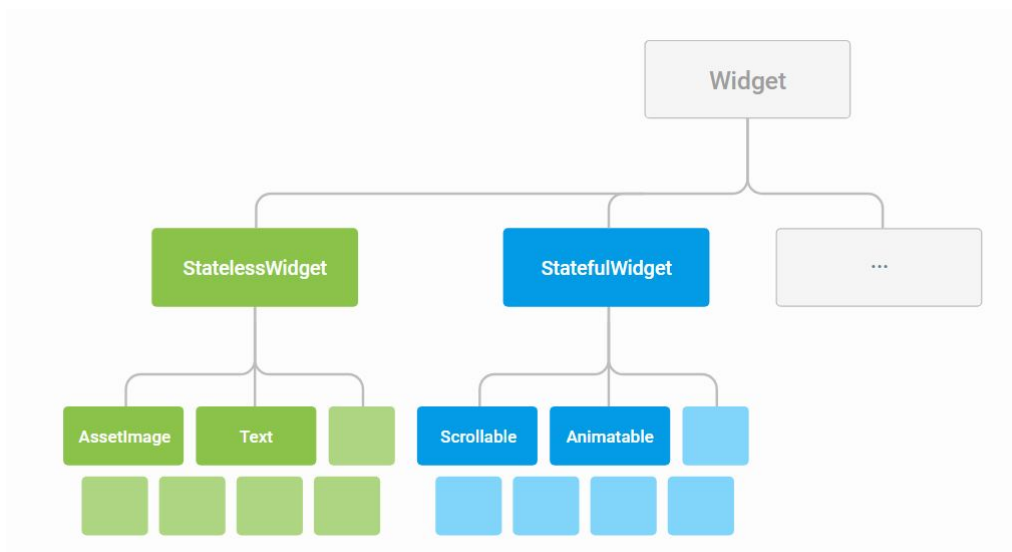


Figura 5.7: Tipos de widgets no Flutter [SJ 2018].

utilizamos o Visual Code⁴. Para deixar mais organizado o desenvolvimento separamos esta seção em algumas subseções.

Na Subseção 5.4.1 mostramos como é realizada a criação de um projeto no Flutter. A elaboração da estrutura do projeto é feita na Subseção 5.4.2. Na Subseção 5.4.3, mostramos como é a estrutura do título da aplicação. A criação das abas de navegação é feita na Subseção 5.4.4. Na Subseção 5.4.5, exibimos a inserção de um botão flutuante na tela e na Subseção 5.4.6 como alterar o ícone do botão para cada aba da aplicação. Na Subseção 5.4.7, realizamos a criação da classe responsável pelo controle das mensagens e a criação da lista contendo todas as mensagens é realizada na Subseção 5.4.8. Finalmente, na Subseção 5.4.9, mostramos como podemos executar o mesmo código criado nas subseções anteriores para web.

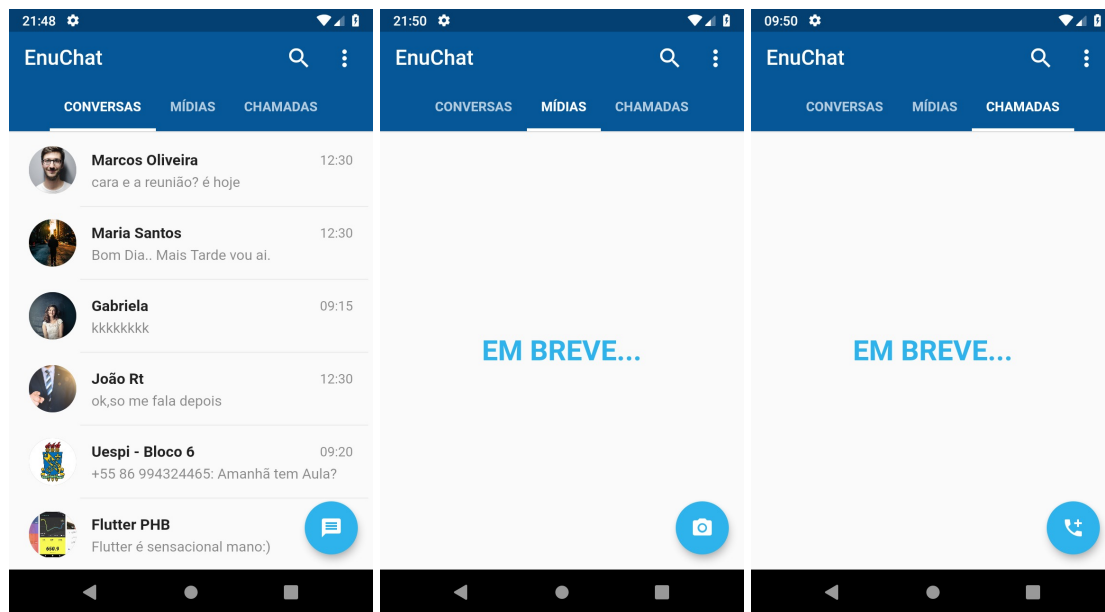
Na Figura 5.8 está o produto final desta seção. Desenvolvemos três modos de visão de acordo com a aba selecionada. Iremos desenvolver particularmente a aba de Conversas as demais abas podem ser feitas de forma similar. Todo o projeto EnuChat encontra-se em um repositório de acesso livre hospedado no Github⁵.

5.4.1. Criação do Projeto

Para criar um novo projeto no Flutter usamos o terminal do Linux executando o seguinte comando: `$ flutter create enuchat`. O parâmetro `enuchat` é o nome do aplicativo, depois disso o Flutter ficará encarregado de criar todos os arquivos para que o projeto possa funcionar. Dentro da pasta **enuchat** estará o projeto. Na Figura 5.9a, pode-se observar os diretórios e arquivos criados pelo comando.

⁴<https://code.visualstudio.com/>

⁵<https://github.com/iang12/enuchat>



(a) Aba de conversas.

(b) Aba das mídias.

(c) Aba das chamadas.

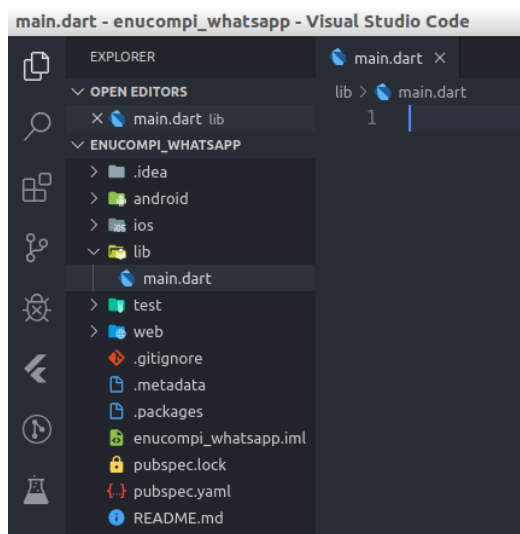
Figura 5.8: Telas da aplicação EnuChat.

5.4.2. Estrutura do Projeto

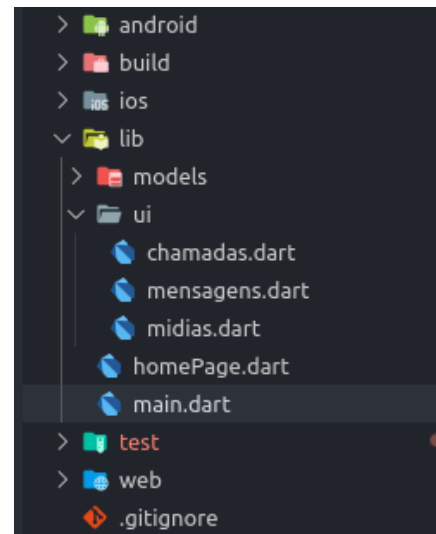
A Figura 5.9a, mostra a estrutura do projeto com pastas, arquivos, uma atenção especial ao arquivo **main.dart** que é essencialmente o ponto de entrada para o aplicativo Flutter (como vimos na Seção 5.2). Em particular trabalharemos todo o código dentro da pasta onde encontra-se o **main.dart**, a pasta **lib**. Cada pasta e arquivo criado pelo Flutter possuem funcionalidades, destacamos algumas:

- **.ideia/**: pasta criada pelos editores para facilitar algumas funcionalidades de edição de código;
- **ios/**, **android/** e **web/**: nessas pastas ficarão os códigos específicos para cada plataforma, incluindo ícones e as configurações do aplicativo.
- **lib/**: contém o código Dart do aplicativo. É possível criar pastas e subpastas internas, porém o arquivo **main.dart** deve estar na raiz.
- **test/**: pasta que contém os arquivos de testes do aplicativo.
- **pubspec.yml**: arquivo onde podemos gerenciar as dependências do projeto; e
- **pubspec.lock**: arquivo usado para gerenciar quais versões das libs (bibliotecas) utilizadas no projeto.

No arquivo **main.dart** colocamos o código descrito na Figura 5.10. Na linha 1, importamos o estilo de material do aplicativo, o pacote `flutter/material.dart`. A partir deste pacote teremos acesso aos componentes para estruturar a aplicação. Na função



(a) Estrutura inicial do projeto.



(b) Arquivos criados em **lib**.

Figura 5.9: Estrutura de diretórios e arquivos do projeto em Flutter.

`main()`, linha 2, é obrigatória neste arquivo e será executada no *bootstrap* da aplicação. Dentro dessa função fizemos uma chamada para a função `runApp()` que é responsável por chamar a tela principal da aplicação por meio do `MaterialApp()` definindo os atributos `title`, `theme` e `home`:

- `title`: nele deve ser atribuído o nome do app, o qual será exibido quando o app for minimizado, por exemplo;
- `theme`: tema utilizado com configurações de cores. O tema possui alguns outros atributos a serem configurados, como: `primaryColor` e `accentColor`, entre outros. No exemplo (Figura 5.10), usamos apenas esses dois para configurar o tema principal do nosso app que será em tonalidades de azul por meio da widget `Color()`; e
- `home`: encontra-se a primeira tela a ser chamada no aplicativo.

No atributo `home` descrevemos outro widget para ser a nossa tela inicial chamado `HomePage()`. Os arquivos que vamos utilizar na aplicação estão organizados na pasta **lib** do projeto, Figura 5.9b. Observe que dentro da raiz dessa pasta colocamos um arquivo chamado **homePage.dart**, neste arquivo criaremos a classe `HomePage()`.

Na Figura 5.9a, definimos o conteúdo do arquivo **homePage.dart**. Nele importamos o pacote `flutter/material.dart`, linha 1, e criamos a classe `HomePage` herdando uma widget com estado, `StatefulWidget`. Na linha 4, sobrecarregamos uma função da classe `StatefulWidget` responsável por criar um estado que por sua vez chamará a função `build`, na linha 9. Será nessa função que definiremos a configuração do *layout* da aplicação. Dessa forma, colocaremos na widget `Scaffold` os componentes que formarão as telas da Figura 5.8, iniciando pela `AppBar`.


```

1 import 'package:flutter/material.dart';
2 void main() {
3   runApp(MaterialApp(
4     title: 'EnuChat',
5     theme: ThemeData(
6       primaryColor: Color(0xff055798),
7       accentColor: Color(0xff2fb3ed),
8     ),
9     home: HomePage(),
10  ));
11 }

```

Figura 5.10: Método `main` da aplicação.

5.4.3. Barra de título

Dentro da widget `Scaffold` definimos a barra de título da nossa aplicação, o código está localizado na Figura 5.12. Na linha 4, essa barra, `AppBar()`, possui as seguintes propriedades: `title`, onde receberá um outro widget do tipo `Text()` que conterá o título do nosso aplicativo, “EnuChat” e `style`, onde definimos o estilo desse texto, como tamanho, fonte e cor. Particularmente, definimos na linha 7 o tamanho e o estilo de fonte. Além desse parâmetro temos:

```

1 import 'package:flutter/material.dart';
2 class HomePage extends StatefulWidget {
3   @override
4   _HomePageState createState() => _HomePageState();
5 }
6
7 class _HomePageState extends State<HomePage> {
8   @override
9   Widget build(BuildContext context) {
10     return Scaffold(
11
12     );
13   }
14 }

```

Figura 5.11: Definição da tela inicial da aplicação, arquivo **homePage.dart**.

- `elevation`: atributo que cria um efeito de elevação na AppBar, dando-lhe uma sombra (linha 9). Para usar essa propriedade, estamos fornecendo um valor do tipo real de 1.0. A elevação acontece no eixo Z da AppBar; e
- `actions`: atributo dedicado a colocar ícones à direita do título da AppBar (linha 10). Ou seja, o canto superior direito do aplicativo, nele estamos usando uma lista que contém duas widgets `Padding` que exibirão na tela dois ícones, um de pesquisa e outro de menu. O `Padding` foi utilizado para dar um espaçamento.

Observe na Figura 5.8 que a AppBar encontra-se fixa em todas as abas de navegação.

5.4.4. Criação das abas de navegação

Na nossa aplicação definimos três abas (*tabs*) de navegação. Inicialmente, vamos definir o atributo `bottom` dentro da AppBar (Figura 5.13). Para criar as abas e colocá-las funcionais na aplicação, precisamos utilizar três componentes básicos: `TabBar`, `TabController` e `TabBarView`. Mostraremos como utilizá-los em três passos.

Primeiro passo, definiremos no atributo `bottom` a `TabBar`, linha 1 da Figura 5.13, com os seguintes atributos:


- `indicatorWeight`: é responsável pela espessura da linha que aparece abaixo da guia selecionada.

```
1 @override
2 Widget build(BuildContext context) {
3   return Scaffold(
4     appBar: AppBar(
5       title: Text(
6         'EnuChat',
7         style: TextStyle(fontSize: 22.0, fontWeight: FontWeight.w600),
8       ),
9       elevation: 1.0,
10      actions: <Widget>[
11        Padding(
12          padding: const EdgeInsets.only(right: 20.0,),
13          child: Icon(Icons.search,size: 28.0,),),
14        Padding(
15          padding: const EdgeInsets.only(right: 16.0,),
16          child: Icon(Icons.more_vert,size: 28.0,),),
17      ],
18    ),
19  ),
```

Figura 5.12: Definição do código da widget AppBar.

- `indicatorColor`: é a cor da linha que aparece abaixo da guia selecionada. Se esse parâmetro for nulo, o valor da propriedade `indicatorColor` recebe a cor do tema utilizado;
- `controller`: é o que mantém o funcionamento das abas, sincronizando a seção e o conteúdo selecionados em cada aba. Este é o trabalho do `TabController`, falaremos um pouco mais sobre ele no segundo passo;
- `isScrollable`: é responsável por definir a forma de interação das abas, caso esse atributo seja verdadeiro, cada guia terá a largura necessária para seu rótulo e toda a `TabBar` poderá ser rolada. Caso contrário, cada guia recebe uma parcela igual do espaço disponível na tela;
- `labelPadding`: é o espaçamento adicionado entre os rótulos da aba e as barras inferiores da aba; e
- `tabs`: é definida a lista de widgets para compor as abas.

Definiremos, no segundo passo, a criação do `TabController`. O `TabController` controlará o movimento entre as guias definidas nas linhas 8, 13 e 18 da Figura 5.13. Para isso, alteramos no arquivo **homePage.dart** a classe `_HomePageState` para utilizar a



```
1 bottom: TabBar(  
2   indicatorWeight: 3.0,  
3   indicatorColor: Colors.white,  
4   controller: _tabController,  
5   isScrollable: true,  
6   labelPadding: EdgeInsets.only(left: 15.0, right: 15.0),  
7   tabs: <Widget>[  
8     Tab(  
9       child: Text(  
10        'CONVERSAS',  
11        style: TextStyle(fontWeight: FontWeight.bold),  
12      )),  
13     Tab(  
14       child: Text(  
15        'MÍDIAS',  
16        style: TextStyle(fontWeight: FontWeight.bold),  
17      )),  
18     Tab(  
19       child: Text(  
20        'CHAMADAS',  
21        style: TextStyle(fontWeight: FontWeight.bold),  
22      )),  
23   ],  
24 ),
```

Figura 5.13: Construção do widget `TabBar ()`

```

1 class _HomePageState extends State<HomePage> with
2   TickerProviderStateMixin{
3   TabController _tabController;
4   @override
5   void initState() {
6     super.initState();
7     _tabController = TabController(
8       vsync: this,initialIndex: 0,length: 3
9     );
10  }

```

Figura 5.14: Construção do TabController.

```

1  body: TabBarView(
2    controller: _tabController,
3    children: <Widget>[
4      Mensagens(),
5      Midias(),
6      Chamadas(),
7    ],
8  ),

```

Figura 5.15: Construção da estrutura da TabBarView.

classe `TickerProviderStateMixin` e colocamos o atributo `_tabController`, Figura 5.14. Dentro dessa última classe sobrecarregamos o método `initState()`, linha 5, o primeiro método chamado pela classe, nele declaramos `_tabController`, linha 7, e definimos no atributo `vsync`, na linha 8, a quantidade das abas de navegação (`length`) e a aba que será exibida inicialmente na tela (`initialIndex`).

No terceiro e último passo criamos as `TabView`, nesta estrutura vamos definir quais classes serão chamadas para cada aba. Particularmente, o que a `TabView` fará é selecionar qual conteúdo será visível e exibido na tela do aplicativo. Colocamos o código dentro do atributo `body`, logo após o fechamento da `AppBar`, presente na Figura 5.15. Dentro desse atributo definimos o `TabBarView` com dois atributos: o `controller`, definido no arquivo **homePage.dart** (`_tabController`), e uma lista de widgets em `children` que ficará associado a cada aba de navegação. Com isso teremos abas funcionais faltando agora somente definir o conteúdo em cada aba definida pelas classes:

Mensagens, na linha 4, Mídias, na linha 5, e Chamadas, na linha 6 (Figura 5.15).

5.4.5. Botões flutuantes

No nosso aplicativo colocamos alguns `FloatingActionButton` que no material design é um botão em uma tela vinculada a uma ação óbvia que um usuário normalmente faria nessa tela específica. Neste contexto, definimos mais um atributo ao `Scaffold`, logo após o fechamento do `body`, na linha 1 da Figura 5.16. Fornecemos dentro da classe do botão um ícone para indicar a ação da tela, para isso utilizamos a propriedade `child`, colocando a widget `Icon()` que conterá uma variável chamada `iconButton` com o valor ícone. Em cada aba o ícone mudará para ficar de acordo com a aba, como pode ser observado nas figuras da Figura 5.8 nos círculos redondos em azul.

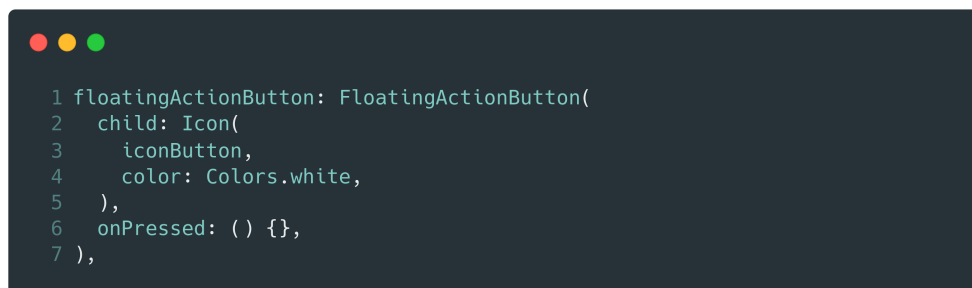
5.4.6. Modificando os ícones dos botões para interagir com as abas

Vamos modificar novamente a função `initState()`, do arquivo `homePage.dart`, para mudar o ícone exibido no `FloatingActionButton`. Na Figura 5.17, colocamos o código responsável para isso. Colocamos um `listener` responsável por capturar uma ação e dependendo de qual aba estiver selecionada, linha 11, escolhemos qual o ícone que será exibido na tela. Observe que nessa estrutura de seleção é que criamos a variável `iconButton` no qual a classe `FloatingActionButton`, do `Scaffold`, irá utilizar para definir o ícone a ser exibido (Figura 5.16, linha 3).

Na próxima subseção vamos definir a estrutura presente na classe `Mensagens` referenciada na `TabBarView` (Figura 5.15, linha 4).

5.4.7. Construção da classe de Mensagens

Para compor a aba de Conversas criamos uma estrutura para armazenar as mensagens recebidas, o armazenamento pode ser feito em um banco de dados ou consumir de alguma API (*Application Programming Interface*), porém adotaremos uma estrutura mais simples com o objetivo de simular o comportamento da tela. Para isso, criamos dentro da pasta `models` o arquivo `model_message.dart` contendo uma lista de mensagens



```
1 floatingActionButton: FloatingActionButton(  
2   child: Icon(  
3     iconButton,  
4     color: Colors.white,  
5   ),  
6   onPressed: () {},  
7 ),
```

Figura 5.16: Construção do widget `FloatingActionButton`

ficções para serem listadas na tela principal da aplicação. A classe `Mensagem` será responsável de adquirir as informações de: `nome`, `mensagem`, `horario` e `imageUrl` e posteriormente declaramos uma lista chamada `listaMensagens` que receberá cada mensagem a ser exibida na tela, Figura 5.18. Colocamos somente duas entradas na lista para demonstrar, no repositório da aplicação pode-se observar as mensagens definidas no arquivo `model_message.dart`.

Depois de criada a estrutura da mensagem definimos a widget `Mensagens`, presente dentro no arquivo `ui/mensagens.dart`, herdado de `StatelessWidget`. Nesta widget retornamos um `ListView.builder` que conterá as mensagens recebidas em nosso aplicativo. O construtor `builder()` do `ListView` constrói uma lista de itens com base em dois parâmetros principais: o `itemCount` para o número de itens na lista e o `itemBuilder` para construir cada item da lista; o mesmo será chamado apenas com índices maiores ou iguais a zero e menores que `itemBuilder`. O formato de exibição de cada item da lista definiremos no `ListTile`, a iteração dos itens será de acordo com o `position` definido na linha 10 na Figura 5.19.

```
1 class _HomePageState extends State<HomePage> with
2   TickerProviderStateMixin {
3   TabController _tabController;
4   var iconButton = Icons.message;
5   @override
6   void initState() {
7     super.initState();
8     _tabController = TabController(vsync: this,
9     ..addListener(() {
10      setState(() {
11        switch (_tabController.index) {
12          case 0:
13            iconButton = Icons.message;
14            break;
15          case 1:
16            iconButton = Icons.camera_alt;
17            break;
18          case 2:
19            iconButton = Icons.add_call;
20            break;
21        }
22      });
23    });
24  }
```

Figura 5.17: Alteração no código do `TabController` para alteração dos ícones do `FloatingActionButton`

```
1 class Mensagem{
2   final String nome;
3   final String mensagem;
4   final String horario;
5   final String imageUrl;
6   Mensagem({this.nome,this.mensagem,this.horario,this.imageUrl});
7 }
8
9 List<Mensagem> listaMensagens = [
10   new Mensagem(
11     nome: "Marcos Oliveira",
12     mensagem: "cara e a reunião? é hoje",
13     horario: "12:30",
14     imageUrl: "https://images.pexels.com/photos/220453/pexels-photo-220453.jpeg?
15       auto=compress&cs=tinysrgb&dpr=2&h=650&w=940"
16   ),
17   new Mensagem(
18     nome: "Gabriela",
19     mensagem: "kkkkkkkk",
20     horario: "09:15",
21     imageUrl: "https://images.pexels.com/photos/756453/pexels-photo-756453.jpeg?
22       auto=compress&cs=tinysrgb&dpr=2&h=650&w=940"
23   ),
24 ];
```

Figura 5.18: Definição da classe da Mensagem e da lista de mensagens.

```
1 import 'package:enucompi_whatsapp/models/model_message.dart';
2 import 'package:flutter/material.dart';
3
4 class Mensagens extends StatelessWidget {
5   @override
6   Widget build(BuildContext context) {
7     return ListView.builder(
8       padding: EdgeInsets.all(5.0),
9       itemCount: listaMensagens.length,
10      itemBuilder: (BuildContext context, int position) {
11        return Column(
12          children: <Widget>[
13
14          ],
15        );
16      },
17    );
18  }
19 }
20
```

Figura 5.19: Construção da classe Mensagens.

5.4.8. Definição da exibição da lista de mensagens

Definimos uma estrutura para exibir a lista de itens dentro da lista `Widget`, Figura 5.19, linha 13. O widget `ListTile()` geralmente é usado para preencher um `ListView` no Flutter e o mesmo será o conteúdo filho, `children`, da nossa lista de mensagens contendo as seguintes propriedades, Figura 5.20:

- `leading`: recebe geralmente uma imagem ou um ícone no início do `ListTile`, usaremos para mostrar a foto do usuário que enviou a mensagem (linha 2);
- `title`: é o título de cada item do `ListView`, usaremos o `title` para mostrar o nome da pessoa que enviou a mensagem (linha 9); e
- `subtitle`: é um texto menor abaixo do título, nesse caso vamos mostrar um trecho da mensagem recebida (linha 23).

```
1 ListTile(
2   leading: CircleAvatar(
3     foregroundColor: Theme.of(context).primaryColor,
4     backgroundColor: Colors.grey,
5     radius: 25.0,
6     backgroundImage:
7       NetworkImage(listaMensagens[position].imageUrl),
8   ),
9   title: Row(
10    mainAxisAlignment: MainAxisAlignment.spaceBetween,
11    children: <Widget>[
12      Text(
13        listaMensagens[position].nome,
14        style: new TextStyle(fontWeight: FontWeight.bold),
15      ),
16      Text(listaMensagens[position].horario,
17        style: TextStyle(
18          fontSize: 14.0,
19          color: Colors.black45,
20        )),
21    ],
22  ),
23  subtitle: Container(
24    padding: EdgeInsets.only(top: 5.0),
25    child: Text(listaMensagens[position].mensagem,
26      style: TextStyle(
27        fontSize: 15.0,
28        color: Colors.black45
29      )),
30  ),
31 ),
32 Divider(height: 5.0, indent: 70.0),
```

Figura 5.20: Construção do widget `ListTile`.

Na linha 32, na mesma figura, colocamos um widget bem simples chamado `Divider()` para colocar uma linha que separa cada item do nosso `ListView`.

O resultado final dos códigos exibidos anteriormente estão na Figura 5.8a. A criação das demais abas da aplicação não mostramos no capítulo, ficando assim como exercício, no repositório da aplicação colocamos o conteúdo responsável por exibir os textos apresentados nas Figuras 5.8b e 5.8c.

5.4.9. Executando o aplicativo na web

Como falamos na Subseção 5.4.2 o Flutter cria uma pasta para cada plataforma em que se deseja executar a aplicação. Nesse caso, o suporte da Web está localizado na pasta **web**, com uma implementação compatível com código do Flutter que é renderizada usando tecnologias da Web baseadas em padrões: HTML, CSS e JavaScript. Para isso basta usar o terminal e configurar alguns parâmetros dentro da pasta **enuchat**, para mais detalhes consultar [FlutterWeb 2019]. A partir do 1.9, o Flutter oferece suporte antecipado à execução de aplicativos da web. Ainda faltam recursos e problemas conhecidos de desempenho. Portanto, não é recomendado para uso em produção no momento. Na Figura 5.21 vemos a mesma aplicação executada no Chrome.

5.5. Considerações Finais

O Flutter veio como uma excelente alternativa para o desenvolvimento mobile, pois oferece desempenho, integração de plataformas móveis, desenvolvimento rápido e criação de código nativo multiplataforma. Isso faz com que seja uma boa alternativa para quem pretende desenvolver para qualquer plataforma ou que já conheça outras tecnologias, como React Native, para aumentar o leque de possibilidades no mercado. Diariamente aparecem vagas para desenvolvedor Flutter com diferentes faixas salariais chegando até cinco

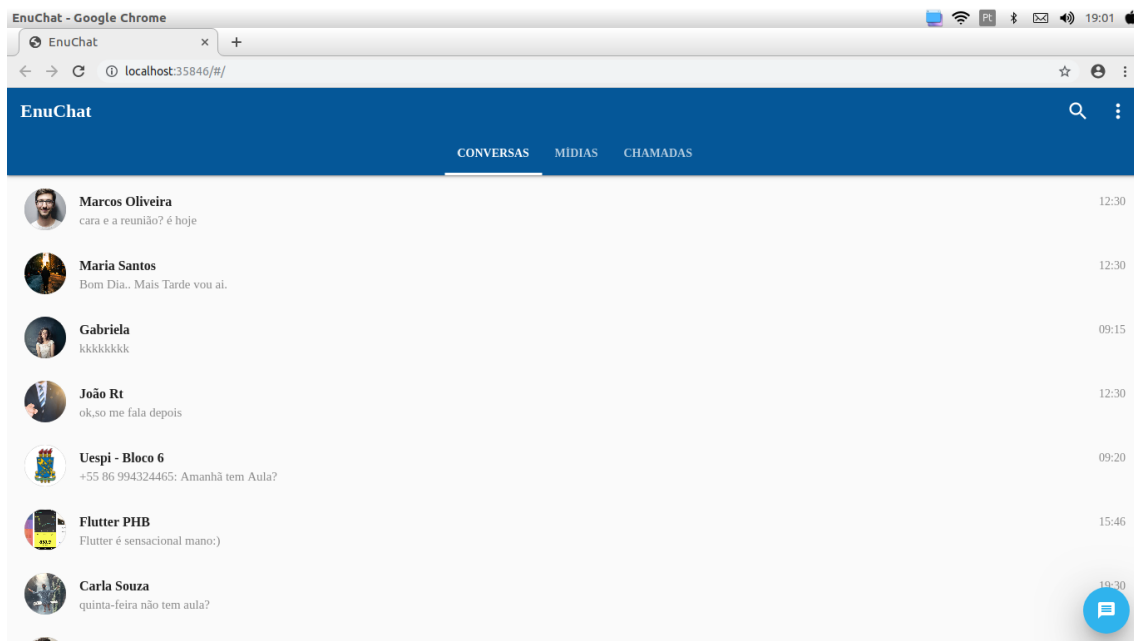


Figura 5.21: Aplicativo Enuchat executando no Google Chrome.

dígitos [ProgramaThor].

Este capítulo teve como objetivo principal apresentar o Flutter e mostrar o desenvolvimento de um app multiplataforma utilizando-o. A aplicação apesar de simples revelou que se pode fazer muito com pouco código e de forma rápida, com grandes possibilidades de criação, onde a criatividade pode ser explorada de forma mais construtiva, como se fossem legos. Para isso é importante entender a lógica por trás das widgets. Embora não tenhamos utilizado, pode-se fazer tudo no Flutter inclusive o desenvolvimento *back-end*, como uso de banco de dados, comunicação com serviços REST (*Representational State Transfer*), geolocalização, entre outros.

Observando isso, mesmo sendo uma tecnologia recente, aprender Flutter é uma aposta que pode dar muito certo. Algumas grandes empresas (Alibaba, Groupon, Google, Nubank, etc...) notaram isso e estão migrando para Flutter [Santana 2019]. Além disso, aprender uma nova tecnologia pode ser bastante divertida, a comunidade Flutter está crescendo cada vez mais. Hoje temos duas comunidades brasileiras que estão ganhando bastante evidencia: Flutterando e Flutter PHB⁶. Essa última foi criada na cidade de Parnaíba/PI e promove encontros (*meetups*) mensais. Desse modo, desenvolver para Flutter jamais será uma jornada solitária, facilmente encontra-se auxílio dentro do Brasil ou pelas comunidades ao redor do mundo.

Referências

- [Biørn-Hansen et al. 2019] Biørn-Hansen, A., Grønli, T.-M., Ghinea, G., and Alouneh, S. (2019). An empirical study of cross-platform mobile development in industry. *Wireless Communications and Mobile Computing*, 2019.
- [CanalTech 2019] CanalTech (2019). Brasil é o segundo mercado de apps que mais cresce no mundo. <https://canaltech.com.br/apps/brasil-e-o-segundo-mercado-de-apps-que-mais-cresce-no-mundo-139241/>. Accessed: 2019-10-16.
- [Dart 2019] Dart (2019). Site dart. <https://dart.dev/>. Accessed: 2019-10-24.
- [Exame 2019] Exame (2019). O brasil aparece entre os principais mercados em número de downloads no mundo, considerando tanto aplicativos de ios quanto de android. <https://exame.abril.com.br/tecnologia/brasileiro-gasta-200-minutos-por-dia-em-aplicativos-diz-estudo/>. Accessed: 2019-10-16.
- [Flutter 2019] Flutter (2019). <https://flutter.dev/web>. Accessed: 2019-10-15.
- [FlutterWeb 2019] FlutterWeb (2019). <https://flutter.dev/docs/get-started/web>. Accessed: 2019-10-28.
- [Hackernoon 2018] Hackernoon (2018). Why flutter uses dart. <https://hackernoon.com/why-flutter-uses-dart-dd635a054ebf>. Accessed: 2019-10-15.
- [Macoratti 2019] Macoratti, J. C. (2019). Flutter – parte 01: apresentando widgets. <https://imasters.com.br/mobile/flutter-parte-01-apresentando-widgets>. Accessed: 2019-10-15.

⁶<https://www.instagram.com/flutter-phb/>

- [ProgramaThor] ProgramaThor. Vagas de desenvolvedor flutter. <https://programathor.com.br/jobs-flutter>. Accessed: 2019-10-27.
- [Quimbundo 2019] Quimbundo, M. (2019). Introdução ao flutter: O básico. <https://medium.com/flutter-angola/introdu%C3%A7%C3%A3o-ao-flutter-o-b%C3%AAsico-f8c8302be95c>. Accessed: 2019-10-15.
- [Santana 2019] Santana, F. (2019). Flutter: porque você deveria apostar nesta tecnologia. <https://medium.com/android-dev-moz/flutter-conhecendo-o-flutter-70d31772afa5>. Accessed: 2019-10-15.
- [SJ 2018] SJ, F. R. (2018). Flutter: Conhecendo e testando o framework da google para criar apps nativos para android/ios. <https://fabiorogerosj.com.br/2018/03/07/Flutter-Conhecendo-e-testando-o-Framework-da-Google-para-criar-apps-nativos-para-Android-iOS/>. Accessed: 2019-10-15.
- [Statista 2019] Statista (2019). Statista. <https://www.statista.com/>. Accessed: 2019-10-16.
- [Windmill 2019] Windmill, E. (2019). Flutter in action. <https://livebook.manning.com/book/flutter-in-action/>. Accessed: 2019-10-15.
- [Yatsenko et al. 2019] Yatsenko, R., Obodiak, V., and Yatsenko, V. (2019). Comparative analysis of cross-platform frameworks for mobile applications development. *Λ'OOΣ*, (4):132–136.
- [Zammetti 2019] Zammetti, F. (2019). Flutter: A gentle introduction. In *Practical Flutter*, pages 1–36. Springer.
- [Época Negócios 2019] Época Negócios (2019). Brasil tem 230 milhões de smartphones em uso. <https://epocanegocios.globo.com/Tecnologia/noticia/2019/04/brasil-tem-230-milhoes-de-smartphones-em-uso.html>. Accessed: 2019-10-16.

Capítulo

6

Deep Learning em Imagens: aplicando CNNs com Keras e TensorFlow

Luis Vogado, Maíla Claro, Justino Santos and Rodrigo Veras

Abstract

The considerable amount of data being made available plus the increased computational power made it possible to create more accurate Machine Learning (ML) methods, promoting significant advances in the areas of signal processing, natural language, computer vision. Deep Learning techniques are based on artificial neural networks and are applied to complex fields, such as searching for patterns in images. This short course focuses on presenting the fundamentals and technologies for developing such models of Deep Learning. In particular, prepares the participant to understand and develop convolutional neural networks; apply the templates to problem-solving in the image classification, detection, and pattern recognition domain. The Python programming language is used in conjunction with the Keras API to implement convolutional neural network models during the class.

Resumo

O acentuado volume de dados sendo disponibilizado somado ao aumento do poder computacional tornaram possível a criação de métodos de Machine Learning (ML) mais precisos, promovendo significativos avanços nas áreas de processamento de sinais, linguagem natural e visão computacional. Técnicas de ML denominadas Deep Learning são baseadas nas redes neurais artificiais e podem ser aplicadas em campos complexos, como na busca por padrões em imagens. Este minicurso tem como foco apresentar os fundamentos e tecnologias para desenvolver tais modelos de Deep Learning. Em especial, o minicurso prepara o participante para entender e desenvolver redes neurais convolucionais; aplicar os modelos à resolução de problemas do domínio de classificação de imagens, detecção e reconhecimento de padrões. A linguagem de programação Python é utilizada em conjunto com a API Keras para implementação dos modelos de redes neurais convolucionais no decorrer do minicurso.

6.1. Introdução

Os desafios mais recentes que a Inteligência Artificial (IA) tenta resolver, são problemas com soluções intuitivas. Em outras palavras, são tarefas executadas de modo fáceis por pessoas, porém difíceis de serem descritas formalmente como o reconhecimento da fala, padrões e imagens. A intuição humana está relacionada com a habilidade de entender e interpretar conhecimentos passados para predizer ou resolver problemas atuais. Com isso surgiu o Aprendizado de Máquina (em inglês *Machine Learning* - ML), uma subárea da IA, que utiliza métodos computacionais para emular esse processo de intuição humana. Em ML, a aprendizagem é feita por meio de treinamentos em banco de dados, que representam eventos e experiências passadas, possibilitando a construção de sistemas capazes de aprender de forma automática [Dundas and Chik 2011].

Desde sua concepção, ML têm contribuído para o avanço de diversas áreas do conhecimento. Atualmente, um subgrupo específico de ML que é a técnica de *Deep Learning* foi a que mais se beneficiou com o surgimento do método, sendo um dos campos que vêm ganhando grande importância, tanto pelo desenvolvimento quanto pela utilização de construções de aplicação de IA. Esse subgrupo geralmente utiliza redes neurais profundas e dependem de muitos dados para o treinamento.

6.1.1. Contextualização

Reconhecimentos de padrões continua sendo um dos desafios fundamentais de sistemas de visão computacional. A área de Processamento Digital de Imagens (PDI) vem sendo objeto de crescente interesse por permitir viabilizar grande número de aplicações em duas categorias bem distintas: (1) o aprimoramento de informações para interpretação humana; e (2) a análise automática por computador de informações extraídas de uma cena. Atualmente muitos esforços de PDI estão concentrados nas Redes Neurais Convolucionais (em inglês *Convolutional Neural Networks* - CNNs).

De 1964 aos dias atuais, a área de processamento de imagens vem apresentando crescimento expressivo e suas aplicações permeiam quase todos os ramos da atividade humana. Em Medicina, o uso de imagens no diagnóstico médico tornou-se rotineiro e os avanços em processamento de imagens vêm permitindo tanto o desenvolvimento de novos equipamentos quanto a maior facilidade de interpretação de imagens produzidas por equipamentos mais antigos, como por exemplo o de raio X [Marques Filho and Neto 1999].

6.1.2. Representação Digital de Imagens

Uma imagem pode ser definida como sendo a representação visual de um objeto. Do ponto de vista matemático, uma imagem é considerada uma função bidimensional $f(x, y)$ onde x e y são coordenadas planas, e a amplitude de f em qualquer par de coordenadas (x, y) é chamada de intensidade ou nível de cinza da imagem no referido ponto. Quando (x, y) e a amplitude de f fazem parte de um conjunto de valores finitos, ou discretos, a imagem é chamada de imagem digital [Gonzalez et al. 2002].

Em outras palavras, uma imagem digital pode ser representada através de uma matriz $n \times m$ onde cada elemento da matriz corresponde à intensidade ou nível de cinza $f(x, y)$ em um determinado ponto da imagem. Estes elementos de imagens digitais são os *pixels*, que na prática possuem uma intensidade que representa uma cor em um determi-

nado ponto da imagem. Para imagens binárias (em preto e branco) os valores dos *pixels* podem assumir os valores 0 e 1. Para imagens em tons de cinza estes valores podem variar de 0 a 255 e, finalmente, para imagens coloridas, tem-se que o valor do pixel é representado por três valores variando de 0 a 255 cada um. Ao olharmos para a Figura 6.1 podemos visualizar um exemplo de imagem digital (a) e uma pequena porção da sua imagem (b) representada na escala de valores de cinza (c).

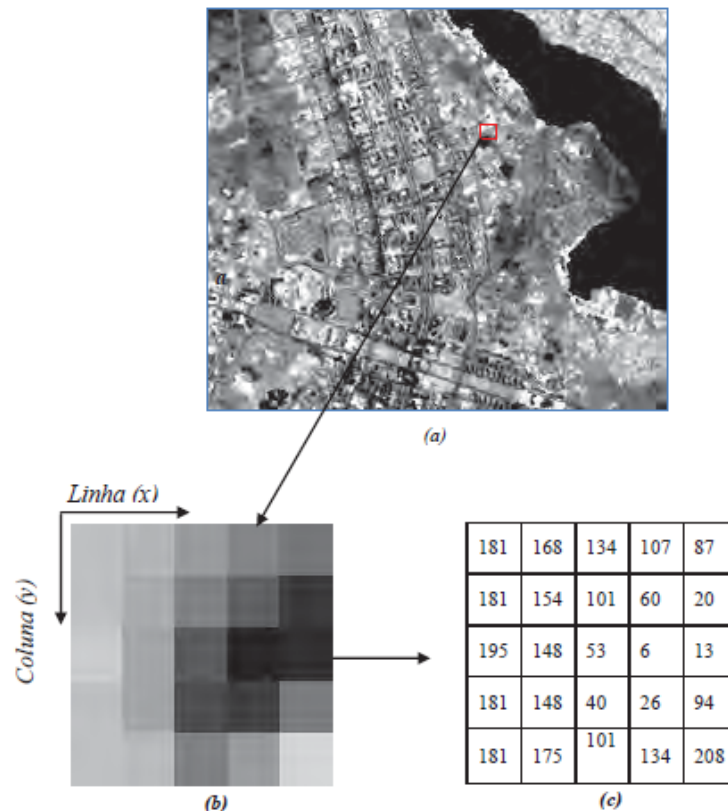


Figura 6.1. Imagem digital (a) com área em zoom de grupo de *pixels* em valores de cinza (b) e correspondentes valores digitais (c). Fonte: [Meneses et al. 2012]

Assim, as linhas são contadas de cima para baixo e as colunas da esquerda para a direita. Essa convenção é utilizada, pois a maioria das imagens é obtida no sentido de órbita descendente do satélite, de norte para sul. Nesse caso, o topo da imagem indica o sentido para norte. Quando uma imagem é obtida no sentido ascendente da órbita, para visualizá-la no monitor e orientá-la para norte, faz-se uma rotação na imagem. O tamanho de uma imagem é uma expressão do tipo: $\text{linha} \times \text{coluna}(\text{byte}) \times \text{número de bandas}$.

6.1.3. Aprendizagem e Redes Neurais Artificiais

Deep Learning ou aprendizagem profunda é uma técnica de aprendizado de máquina desenvolvida a partir das Redes Neurais Artificiais (RNAs). As RNAs são modelos matemáticos que tentam simular algumas das estruturas neurais biológicas, possuindo capacidade computacional adquirida através do aprendizado e generalização [Haykin 2007]. Pode-se dizer então que as RNAs são capazes de reconhecer e classificar padrões e posteriormente generalizar o conhecimento adquirido. Na Figura 6.2 observamos um exemplo de RNA

genérica com duas camadas densas.

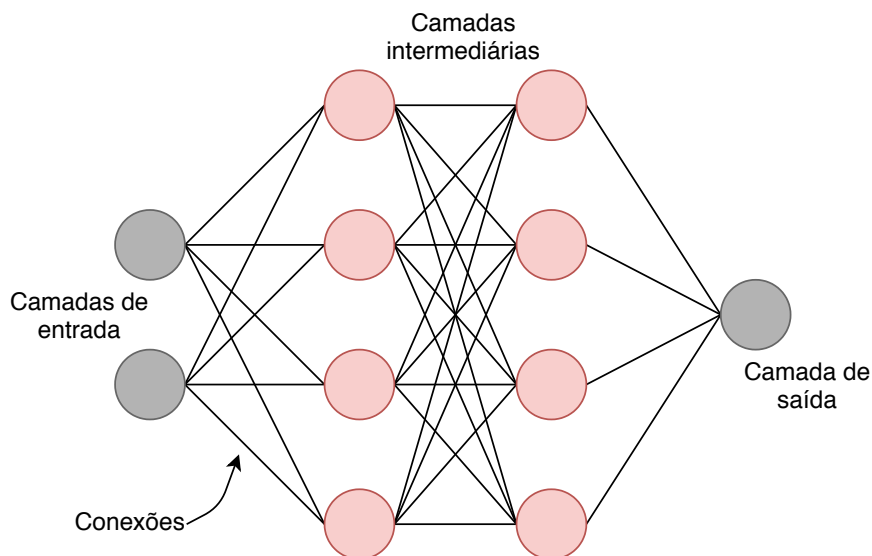


Figura 6.2. Exemplo de uma rede neural artificial.

A propriedade mais importante das redes neurais é a habilidade de aprender de seu ambiente e, com isso, melhorar seu desempenho. O aprendizado em RNAs está normalmente associado à capacidade de as mesmas adaptarem os seus parâmetros como consequência da sua interação com o meio externo. O processo de aprendizado é iterativo e por meio dele a RNA deve melhorar o seu desempenho gradativamente à medida que interage com o meio externo [Rezende 2003]. Pode-se denominar o algoritmo de aprendizado como um conjunto de regras bem definidas para a solução de um problema de aprendizado [Haykin 2007]. Existem muitos tipos de algoritmos de aprendizado específicos para determinados modelos de redes neurais. Estes algoritmos diferem entre si, principalmente, pelo modo como os pesos são modificados.

O *Deep Learning* passa por grandes evoluções e vem se destacando no seu paradigma de “habilitar o computador a aprender a partir da observação dos dados”. O Aprendizado Profundo passou a ser aplicado em diversas áreas, em particular, mas não exclusivamente, as áreas de Visão Computacional, Processamento de Imagens, Computação Gráfica, entre outras mais [Ronneberger et al. 2015, Bejnordi et al. 2018]. As Redes neurais convolucionais começaram a aparecer como base para métodos do estado da arte em diversas aplicações. A competição *ImageNet* [Deng et al. 2009] teve grande impacto nesse processo, começando uma corrida para encontrar o modelo que seria capaz de superar o atual campeão nesse desafio de classificação de imagens, além de segmentação de imagens, reconhecimento de objetos, entre outras tarefas.

Uma das vantagens dos algoritmos de *Deep Learning* é sua capacidade de aprendizagem em grandes quantidades de dados de uma forma não-supervisionada, sendo assim uma ferramenta valiosa para *Big Data Analytics* onde a maioria dos dados são desta natureza, também designados por dados não-estruturados.

6.1.4. Aplicação de *Deep Learning*

O aprendizado de máquina e o *Deep Learning* são amplamente utilizados em muitos domínios, tais como: na medicina, em documentos, nos bancos, no processamento de linguagem natural, na recuperação de imagens, entre outros domínios.

Na medicina pode ser utilizada na detecção de células cancerígenas, restauração de imagem de ressonância magnética cerebral, impressão de genes, entre outras. Nos documentos, essas áreas poderão auxiliar na resolução de imagens de documentos históricos e segmentação de texto em imagens de documentos. Para o bancos poderá ser realizada uma previsão de ações e decisões financeiras [Pacheco and Pereira 2018].

O Processamento de Linguagem Natural são sistemas de recomendação como por exemplo: Netflix que utiliza um sistema de recomendação para sugerir filmes aos usuários com base em seu interesse, análise de sentimentos e marcação de fotos. Para a recuperação de Informações poderão ser aplicadas os mecanismos de busca, pesquisa de texto e pesquisa de imagens como as utilizadas pelo Google, Amazon, Facebook e LinkedIn.

Deep Learning pode ser usada para:

- ⇒ Pré-processamento: que é uma maneira de realizar um ajuste ou melhoramento nos dados;
- ⇒ Extração de recursos: que é um processo para reconhecer algum padrão entre os dados e tem como intuito tornar o processo de decisão mais fácil durante a classificação;
- ⇒ Classificação: que é uma tarefa de predição e até uma classificação de várias classes.

Existem outras funcionalidades do *Deep Learning*, tais como regressão, reconhecimento de objetos, previsões, entre outras.

6.2. Redes Neurais Convolucionais

As CNNs fazem parte do conjunto de técnicas de *Deep Learning*. Essas redes convolucionais são uma classe de RNAs que modelam abstrações em alto nível através de imagens e camadas convolucionais dispostas de forma sequencial ou não. O conceito de CNN foi apresentado por Yann LeCun [Lecun et al. 1998] e Fukushima [Fukushima 1988] na década de 90. No entanto, apenas no século XXI essa tecnologia foi desenvolvida com eficácia.

Atualmente as CNNs são empregadas nos mais diversos problemas que envolvem a classificação, segmentação e detecção em imagens. Uma das principais vantagens na utilização de CNNs é a alta capacidade de aprender os mais diversos padrões que não são perceptíveis para outras técnicas tradicionais que utilizam descritores. No entanto, essa vantagem demanda um alto custo computacional e a necessidade de grandes bases de dados para o treinamento.

Camadas Convolucionais: Como as demais redes neurais artificiais, as CNNs apresentam estruturas no formato de camadas que auxiliam na extração de características, redução e classificação. A principal é a que denomina esse tipo de rede. As camadas convolucionais são compostas por uma quantidade c de filtros com tamanho $d \times d$ que irão extrair mapas de características ao serem convoluídos com as imagens de entrada ou com saídas de outras camadas. A saída da convolução é denominada mapa de características ou *features map*. Esses mapas geralmente representam características gerais extraídas da imagem, como cor, borda, textura e forma. Um exemplo da operação de convolução é apresentada na Figura 6.3.

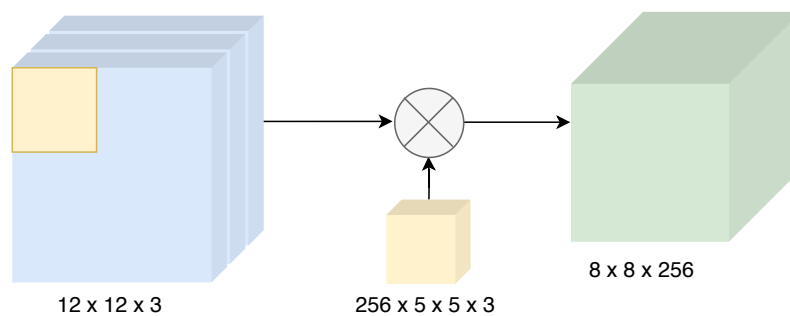


Figura 6.3. Operação realizada nas camadas convolucionais.

Camadas de Pooling: Após as camadas convolucionais, geralmente são empregadas as de *pooling*. Esse tipo de camada reduz a dimensão espacial dos mapas gerados por camadas anteriores, consequentemente reduzindo o custo computacional. Dentre os diversos tipos, de *pooling* existe o *maxpooling*. Essa operação utiliza uma janela deslizante de tamanho $n \times n$ no mapa de características e para cada passo realizado, o valor máximo daquela janela é retirado. Na Figura 6.4 essa operação é ilustrada com um mapa aleatório.

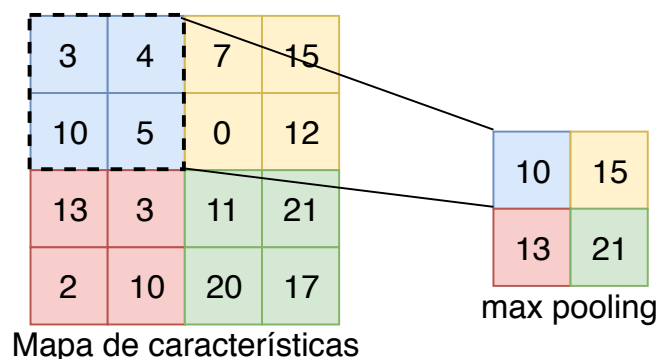


Figura 6.4. Operação de *maxpooling* realizada em um mapa de características aleatório

Camadas Totalmente Conectadas: As camadas densas ou totalmente conectadas (*Fully-Connected layers* - FCs) foram apresentadas inicialmente nas RNAs. São constituídas de neurônios que representam pesos e guardam o aprendizado da rede neural. Nas CNN's

elas apresentam a mesma função e geralmente aparecem ao final da arquitetura, após inúmeras camadas convolucionais. A operação que transforma os mapas de características em um vetor de neurônios é realizada pela camada *flatten*. A camada de classificação presente nas CNNs é do tipo densa, no entanto, apresenta uma função de ativação do tipo *softmax*, enquanto as FCs contam com ativação por meio da ReLu.

6.2.1. Hiper-parâmetros

Além dos parâmetros, existem os hiper-parâmetros da CNN. Diferente dos parâmetros que serão ajustados com o treinamento, os hiper-parâmetros são algumas variáveis utilizadas na etapa de configuração da CNN. Os mais comuns são: função de ativação, funções de custo, otimizadores, taxa de aprendizado e tamanho do *mini batch*.

Funções de ativação: Geralmente, após as camadas convolucional ou densas, são utilizadas as funções de ativação. Essas funções realizam transformações não-lineares nos dados, de modo a gerar saídas linearmente separáveis. Uma das funções mais comumente utilizadas com este propósito é a *Rectified Linear units* (ReLu) [Glorot et al. 2011], apresentada na Equação 1. Esta função de ativação substitui os *pixels* dos mapas de características com valores negativos por zero.

$$f(x) = \max(0, x) \quad (1)$$

Essas camadas também são responsáveis pela etapa de predição dos vetores obtidos nas camadas densas ao final da arquitetura. As funções mais conhecidas são: *softmax* e *sigmoid*.

Treinamento, função de custo e otimizadores: Diante da estrutura apresentada, assim como as RNAs, as CNNs apresentam parâmetros que serão ajustados durante o treinamento. A quantidade total de parâmetros é calculada a partir da soma de todos os parâmetros de cada camada.

Para medir a qualidade atual do treinamento de uma CNN, existe o **custo** ou *loss* associado a cada época de treino. O valor do custo está associado a qualidade da predição realizada e representa o quão distante os parâmetros atuais estão da predição ideal. Para calcular o custo, existem as **funções de custo** ou *loss function*. Existem diversas funções propostas no estado da arte, no entanto, a entropia cruzada ou *cross-entropy* é a mais utilizada. Essa função calcula a soma das entropias cruzadas entre a predição e a classe real considerando um único exemplo.

A partir do custo calculado durante a época atual de treino, a rede neural irá ajustar os parâmetros através do *backpropagation* e de algoritmos de otimização. Dentre eles, o mais utilizado é o Gradiente Descendente (GD). Esse algoritmo calcula de forma iterativa para cada parâmetro da rede neural os valores que minimizam a função de objetivo. Após encontrar os valores, o *backpropagation* irá reajustar os parâmetros. O GD também é responsável por definir a mudança no passo em direção ao gradiente através da taxa de aprendizado ou *learning rate*.

O GD é considerado um algoritmo de otimização simples para CNNs com muitos parâmetros e que necessitam de grande bases de dados para treinamento. Sendo assim, para aplicações mais robustas envolvendo *Deep Learning*, outros algoritmos de otimização como o *Stochastic Gradient Descent* (SGD), Adam, AdaGrad e RMSProp são utilizados. Esses algoritmos possuem diferenças no desempenho e tendem a modificar a taxa de aprendizado com o passar das épocas.

6.2.2. Arquiteturas e ImageNet

A popularidade das CNNs cresceu durante o *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) [Russakovsky et al. 2015]. Essa competição ocorreu a partir de 2010 por 7 anos ininterruptos e teve como principais desafios a classificação de imagens e detecção de objetos em larga escala. O *dataset* popularmente conhecido como ImageNet possui mais de 1 milhão de imagens e cerca de mil classes.

Durante essa competição foram propostas arquiteturas que atualmente compõem o estado da arte, dentre elas temos a AlexNet [Krizhevsky et al. 2012], Inception ou GoogLeNet [Szegedy et al. 2015], VGGNet [Simonyan and Zisserman 2014] e ResNet [He et al. 2016]. Essas arquiteturas foram propostas por diversas companhias do ramo de tecnologia e cada uma possui características específicas que acabaram influenciando o desenvolvimento das posteriores. Abaixo elencaremos as mais relevantes nos respectivos anos de desafios e sua contribuição na literatura.

AlexNet: Em 2012 [Krizhevsky et al. 2012] propuseram a AlexNet, uma CNN com cinco camadas convolucionais seguidas de *maxpooling*, duas camadas totalmente conectadas e uma de classificação. Essa CNN obteve excelentes resultados no ILSVRC 2012 quando comparados com os anos anteriores. Em 2010 a taxa de erro era de 28,2%, em 2011 a evolução não foi como esperado e o erro baixou para 25,8%. Em 2012 o erro baixou para 16,4% com a utilização da AlexNet.

Nessa competição observamos a efetividade das CNNs quando comparadas com outras abordagens. Entretanto, os autores constataram que o desempenho foi obtido devido a profundidade da arquitetura e que isso representa um alto custo computacional. Ao todo, são 62,3 milhões de parâmetros, sendo assim necessário a utilização de GPUs para processar todos os dados em tempo hábil. Na Figura 6.5 observamos o *design* da AlexNet.

GoogLeNet: Com o aumento popularidade do evento proporcionada pela AlexNet, as grandes companhias da tecnologia como a Google e a Microsoft montaram times para participar da competição. Em 2014, a Google e o seu time obtiveram os melhores resultados na competição com a GoogLeNet [Szegedy et al. 2015]. Essa CNN faz uso de módulos denominados *inception*. Eles são capazes de armazenar múltiplas camadas convolucionais em paralelo, mudando o paradigma e a rede deixa de ser sequencial. Utilizando essa estratégia, a GoogLeNet alcançou uma taxa de erro de 6,7% para a classificação de imagens. No entanto, para a localização de objetos, a erro foi de 26,4%, maior que o erro obtido pela VGG, sua principal concorrente na competição.

A estrutura da GoogLeNet conta com topologia de 22 camadas com cerca de 4 milhões de parâmetros. O *design* dessa CNN foi inspirado na LeNet [Lecun et al. 1998]

A ResNet venceu a competição ILSVRC-2015 com taxas de erro de apenas 3,56%, considerada uma taxa menor que a dos seres humanos que alcançam entre 5% a 10%. Os autores propuseram a ResNet utilizando um estado de ablação que resultou em cinco arquiteturas com diferentes profundidades, sendo elas com 18, 34, 50, 101 e 152 camadas. Na Figura 6.7 apresentamos uma ilustração da CNN ResNet.

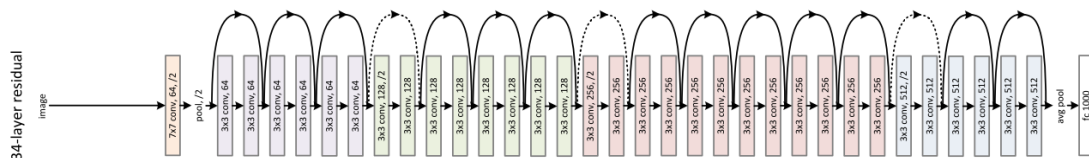


Figura 6.7. Residual Network com 34 camadas treináveis. Fonte:[He et al. 2016]

Em 2016, o vencedor da competição de classificação do ILSVRC foi a metodologia Trimps-Soushen. A taxa de erro obtida ao final da competição foi 2,99%. Nessa edição do evento os vencedores não apresentaram nenhuma inovação relacionada a estrutura das CNNs como nos anos anteriores. Diante disso, essa abordagem não se tornou conhecida como as demais CNNs.

Já em 2017, a *Squeeze-and-Excitation Network* (SENet) [Hu et al. 2018] foi a vencedora com 2,25% de taxa de erro. A denominação utilizada nesse tipo de rede neural vem dos blocos convolucionais *Squeeze-and-Excitation*. Esses blocos recalibram adaptativamente as características de saída de cada canal, modelando explicitamente as interdependências entre os canais. Diante disso, as redes neurais convolucionais consegue generalizar efetivamente diferentes bases de dados com a mesma estrutura.

Outras arquiteturas famosas não foram inicialmente propostas para o concurso, dentre elas temos a CaffeNet [Jia et al. 2014], InceptionV3 [Szegedy et al. 2016], Xception [Chollet 2017] e DenseNet [Huang et al. 2017]. Além das CNNs utilizadas para classificar imagens, foram propostas algumas com o objetivo de segmentar regiões.

U-Net: Sendo desenvolvida para segmentar imagens biomédicas, a U-Net é uma CNN que realiza a segmentação de uma determinada região tendo como base a sua marcação real[Ronneberger et al. 2015]. A ideia desenvolvida para essa CNN é de utilizar os mapas de características para contrair e expandir o vetor da imagem segmentada. Para realizar essa operação, a arquitetura é dividida em três partes, a contração, o gargalo e a expansão. A etapa de contração é realizada por blocos com camadas convolucionais com filtros 3×3 seguidos por *maxpooling* com janelas 2×2 .

A parte de expansão é considerada a mais importante, uma vez que possui blocos com camadas convolucionais e de *upsampling* que possuem o mesmo tamanho das camadas da fase de contração. Além dos blocos, são concatenados mapas de características da etapa de contração que permitem um melhor aprendizado durante a reconstrução da segmentação. O efeito proporcionado pela expansão mantém a simetria original da imagem de entrada, fazendo com que a saída seja de mesmo tamanho. Na Figura 6.8, apresentamos uma ilustração da arquitetura onde observamos as etapas descritas previamente.

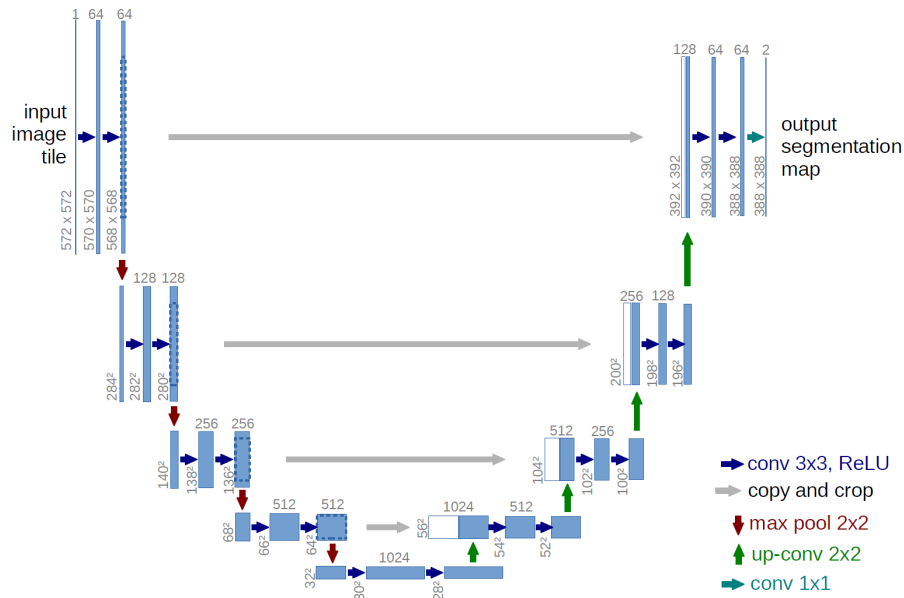


Figura 6.8. Ilustração da CNN U-Net e as etapas de contração e expansão.
 Fonte: [Ronneberger et al. 2015]

6.2.3. Arquitetura Genérica

Para exemplificar a construção de uma CNN e a definição das camadas convolucionais, totalmente conectadas e seus tamanhos, apresentamos uma arquitetura genérica desenvolvida para solucionar um problema simples de classificação. Neste caso, utilizamos o *dataset* MNIST. Essa base contém imagens com dimensão 28×28 e está dividida em 10 classes que representam os números de 0 a 9. Cada imagem possui um número escrito a mão livre e o objetivo é classificar os números de acordo com a sua respectiva classe.

Diante do problema apresentado, propomos uma CNN com duas camadas convolucionais, duas de *maxpooling* e uma camada totalmente conectada. Na Tabela 6.1, apresentamos a estrutura da CNN com o tamanho dos filtros, entrada e saída.

Tabela 6.1. Estrutura de uma CNN genérica.

Tipo de camada	Entrada	Qtd. de filtros	Tam. dos filtros	Passo	Saída
conv 1	$28 \times 28 \times 3$	64	$5 \times 5 \times 3$	1	$22 \times 22 \times 64$
maxpool 1	$22 \times 22 \times 64$	-	2×2	2	$11 \times 11 \times 64$
conv 2	$11 \times 11 \times 64$	32	$3 \times 3 \times 32$	1	$9 \times 9 \times 32$
maxpool 2	$9 \times 9 \times 32$	-	2×2	2	$4 \times 4 \times 32$
flatten	$4 \times 4 \times 32$	-	-	-	1×512
fc 1	-	1×512	-	-	512
fc 2	512	-	-	-	10

6.3. Construção da CNN com Keras

Existem diversos *frameworks* e bibliotecas que possibilitam o desenvolvimento de CNNs. Dentre elas, o *Pytorch*, *Tensorflow* [Abadi et al. 2015], *Caffe* e *Theano* são os mais popu-

lares. O *Tensorflow* é uma das bibliotecas mais completas, sendo *open source* e desenvolvida pela Google. Inicialmente disponibilizada em 2015, ela consegue atender todos os principais sistemas operacionais e oferece suporte para a utilização de CPUs, GPUs e TPUs durante o treinamento dos modelos. Diante de todos os benefícios providos pelo *Tensorflow*, o desenvolvedor é capaz de reutilizar arquiteturas já apresentadas no estado da arte e desenvolver novas. Para auxiliar nesse desenvolvimento, nesse trabalho utilizamos o Keras [Chollet et al. 2015].

O Keras é uma API de alto nível para redes neurais e executa algumas bibliotecas apresentadas anteriormente, como o *Tensorflow*. Sendo assim, o Keras dispõe de todos os modelos de camadas presentes em uma CNN e possibilita uma rápida prototipagem.

A partir do problema apresentado e da CNN construída na Tabela 6.1, implementamos o modelo utilizando o Keras na linguagem Python. O código desenvolvido pode ser observado na Figura 6.9.

```
1 from keras import Model
2 from keras.layers.core import Dense, Activation, Flatten
3 from keras.layers.convolutional import Conv2D, MaxPooling2D
4 from keras.layers import Input
5
6 entrada = Input(shape=(24,24,3))
7
8 # Definição da primeira camada convolucional com 64 filtros de tamanho 3x3.
9 conv_1 = Conv2D(64, (3, 3))(entrada)
10 maxpool_1 = MaxPooling2D(pool_size=(2, 2))(conv_1)
11
12 # Definição da segunda camada convolucional com 32 filtros de tamanho 3x3.
13 conv_2 = Conv2D(32, (3, 3), strides=1)(maxpool_1)
14 maxpool_2 = MaxPooling2D(pool_size=(2, 2))(conv_2)
15
16 flat = Flatten()(maxpool_2)
17 fc_1 = Dense(512, activation='relu')(flat)
18 prediction = Dense(10, activation='softmax')(fc_1)
19
20 CNN_Generica = Model(inputs=entrada, outputs=[prediction])
21
22 # Essa função apresenta um sumário de cada camada com sua respectiva saída e quantidade de parâmetros.
23 CNN_Generica.summary()
```

Figura 6.9. Código em python para a CNN genérica desenvolvida na subseção anterior.

Na Figura 6.9, observamos a definição da entrada da CNN com a dimensão da imagem e os canais de cores. A partir da entrada, definimos a primeira camada convolucional com 64 filtros. As camadas são conectadas uma as outras através da chamada da camada anterior após a sua definição.

O Keras permite a alteração de diversos parâmetros durante a definição de uma camada em específico. No entanto, optamos por deixar por padrão hiper-parâmetros existentes nessas camadas para facilitar a absorção do aprendizado. Nas camadas de *maxpooling* definimos ambas com uma janela 2×2 . Definimos ainda uma camada *flatten* para dispor os mapas em um vetor único de características e a conectamos na camada totalmente conectada. A camada de predição é a final, possuindo 10 neurônios e ativação do tipo *softmax* por se tratar de um problema multi-classes. Finalmente, para finalizar o modelo, utilizamos a função *Model*, onde repassamos a camada de entrada e de saída, como a CNN foi completamente conectada, ela está apta para a etapa de treinamento.

Na função *summary()*, o Keras retorna a visualização do modelo construído. Na Figura 6.10, observamos o sumário da CNN com a quantidade total de parâmetros treináveis e saídas de cada camada.

Model: "model_6"

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	(None, 24, 24, 3)	0
conv2d_11 (Conv2D)	(None, 22, 22, 64)	1792
max_pooling2d_11 (MaxPooling)	(None, 11, 11, 64)	0
conv2d_12 (Conv2D)	(None, 9, 9, 32)	18464
max_pooling2d_12 (MaxPooling)	(None, 4, 4, 32)	0
flatten_6 (Flatten)	(None, 512)	0
dense_11 (Dense)	(None, 512)	262656
dense_12 (Dense)	(None, 10)	5130
Total params: 288,042		
Trainable params: 288,042		
Non-trainable params: 0		

Figura 6.10. Sumário da CNN construída com Keras.

6.4. Transferência de aprendizado

Algoritmos supervisionados de aprendizagem de máquina utilizam modelos estatísticos para fazer previsões. Esses modelos são construídos (na etapa de treino) com base em exemplos cuja variável a ser predita é conhecida previamente. É importante ressaltar que tradicionalmente o algoritmo é planejado para realizar sua tarefa para domínios similares nos conjuntos de treino e teste. Uma mudança de domínio, a princípio, necessitaria de uma reconstrução do modelo (novo treinamento). Em contraste, utilizar transferência de aprendizado permite que os domínios, tarefas e distribuições usados no treinamento e teste sejam diferentes [Pan et al. 2010].

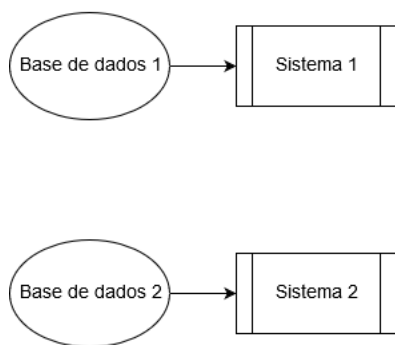
[Goodfellow et al. 2016] trata a transferência de aprendizado como a situação onde o que foi aprendido em um cenário é explorado para melhorar a generalização em outro. É fácil observarmos tais situações no cotidiano: saber andar de bicicleta ajuda no aprendizado de pilotagem de moto, por exemplo.

A Figura 6.11 mostra uma comparação entre a técnica tradicional de aprendizado de máquina e a de transferência de aprendizado.

A criação de uma CNN robusta advém de um minucioso estudo de ablação para definir sua arquitetura. Além disso, facilmente essas redes chegam a casa de dezenas de milhões de parâmetros treináveis. Tal característica torna a etapa de treino de uma CNN uma tarefa de alto custo computacional além de requerer uma grande quantidade de dados

Aprendizagem de máquina tradicional:

Ocorre isoladamente. O conhecimento/aprendizado adquirido em uma tarefa não é reaproveitado em outra.



Transferência de aprendizado:

A aprendizagem em novas tarefas utiliza o conhecimento adquirido em sistema anteriormente definidos.

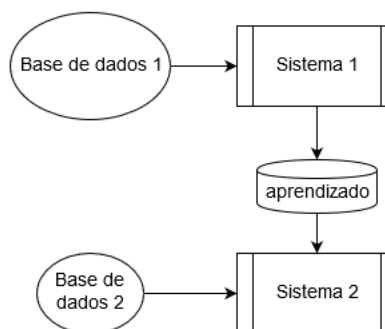


Figura 6.11. Aprendizagem tradicional vs transferência de aprendizado. Fonte: adaptado de [Sarkar 2018]

(imagens) para atingir resultados e poder de generalização satisfatórios.

É muito comum que na prática não se disponha desses elementos (tempo/poder de processamento, grandes bases de imagens) em aplicações específicas. Estes são casos típicos onde a transferência de aprendizado é avaliada como opção para a criação dos sistemas.

No caso das redes neurais, o aprendizado/conhecimento reside nos pesos sinápticos que foram ajustados durante as várias épocas na fase de treino. Partindo deste ponto vamos detalhar duas formas de realizar a transferência de aprendizado: a **extração de características** e o **ajuste fino**.

6.4.1. Extração de Características

Sistemas de *deep learning* baseiam-se em arquiteturas multicamadas que retêm diferentes características em diferentes camadas, como bordas, texturas, brilho etc. [Sarkar et al. 2018]. A transferência de aprendizado em CNN sob a forma de extração de características consiste na utilização de uma rede pré-treinada com o fim de gerar um conjunto de atributos da imagem de entrada. Tais atributos são utilizados como entrada em outros algoritmos de aprendizagem.

Conceitualmente, o processo de transferência está em utilizar a habilidade adquirida (decorrente do treinamento) pela CNN em ponderar as características relevantes das imagens por entre as suas camadas. Na Figura 6.12 ilustramos o procedimento.

As arquiteturas propostas em razão da ILSVRC, são comumente utilizadas para tal finalidade, uma vez que a *dataset* onde foram treinadas possui um grande número de imagens, além de ser bem variada e ampla no que se refere ao número de classes. Isso requer das CNNs uma boa capacidade de generalização e de extração de características relevantes.

O código da Figura 6.13 mostra uma função em python com uma forma de fazer o “corte” na CNN para que ela atue como extrator de características. O retorno desta função é um vetor de atributos que será utilizado para treinar algum métodos de aprendizagem,

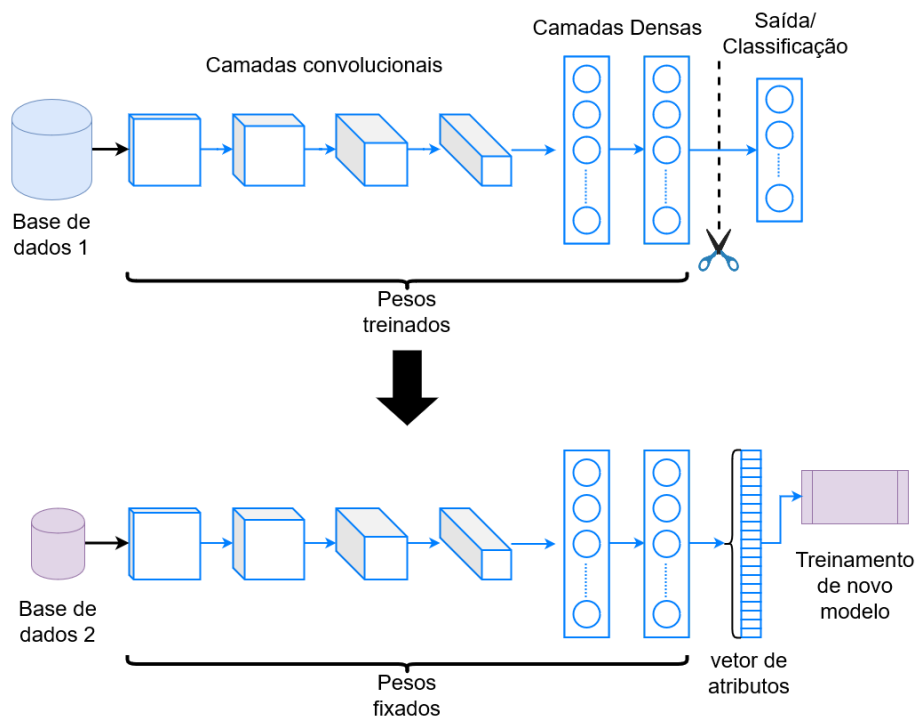


Figura 6.12. Transferência de Aprendizado por meio de extração de características. De uma rede robusta previamente treinada, se extrai um vetor de atributos de uma de suas camadas. Esse vetor serve para treinar um novo modelo específico.

por exemplo os classificadores Máquina de Vetor de Suporte (em inglês *Support Vector Machine* - SVM), *k-nearest neighbor*(KNN), entre outros.

A técnica de extração de características é bastante utilizada, especialmente quando a quantidade de imagens é insuficiente para o treino de uma CNN. Além disso, é possível combinar diversos descritores oriundos de CNNs diferentes em um único vetor, combinando o poder de extração de cnns distintas como foi avaliado em [Santos et al. 2019].

6.4.2. Ajuste fino

Outra forma de realizar a transferência de aprendizado em CNNs é por meio do ajuste fino. Trata-se de utilizar os pesos de uma rede já treinada como ponto de partida. A partir dessa inicialização, algumas camadas da rede são retreinadas na nova base (com baixa taxa de aprendizagem) ajustando seus pesos à nova aplicação.

A utilização dessa técnica, ajuda o novo modelo a obter melhor desempenho com menos tempo de treinamento [Sarkar et al. 2018]. A figura 6.14 ilustra o funcionamento do ajuste fino. A título de exemplo, mantivemos os pesos da parte convolucional e retreinamos a parte densa. Observe que uma vez que as tarefas sejam diferentes, a camada final também deverá ser adaptada à saída desejada.

A Figura 6.15 traz o trecho de código onde a rede VGG-16 treinada na base da imagenet é tomada para ajuste fino. Neste exemplo os pesos transferidos para as camadas convolucionais são mantidos e os das camadas densas são treináveis. A última camada também foi adaptada para um problema de classificação binário.

```

1 def vgg16_f_extractor(input_data, layer_name='fc2'):
2     """Retorna um array de características correspondente ao input_data
3     input_data: array com as imagens. shape=(num_imagens,224,224,3)
4     layer_name: nome da camada da rede VGG-16 para extração.
5
6     retorno: vetor com os atributos. shape=(num_imagens,num_features)
7
8     Ex. caso layer_name seja 'fc2', num_features=4.096.
9     """
10
11     from keras.applications.vgg16 import VGG16, preprocess_input
12
13     # CNN VGG-16 pre-treinada com os pesos da imagenet
14     vgg16_model = VGG16(weights='imagenet', include_top=True)
15
16     # modelo VGG-16 'cortado' antes da camada de classificação
17     ext_model = Model(inputs=vgg16_model.input,
18                       outputs=vgg16_model.get_layer(layer_name).output)
19
20     input_data = preprocess_input(input_data)
21
22     # o predict de ext_model não dá a classificação como saída,
23     # mas o vetor de atributos de saída camada layer_name
24     features = ext_model.predict(input_data)
25
26     return features

```

Figura 6.13. Função em python para extração de características da rede VGG-16.

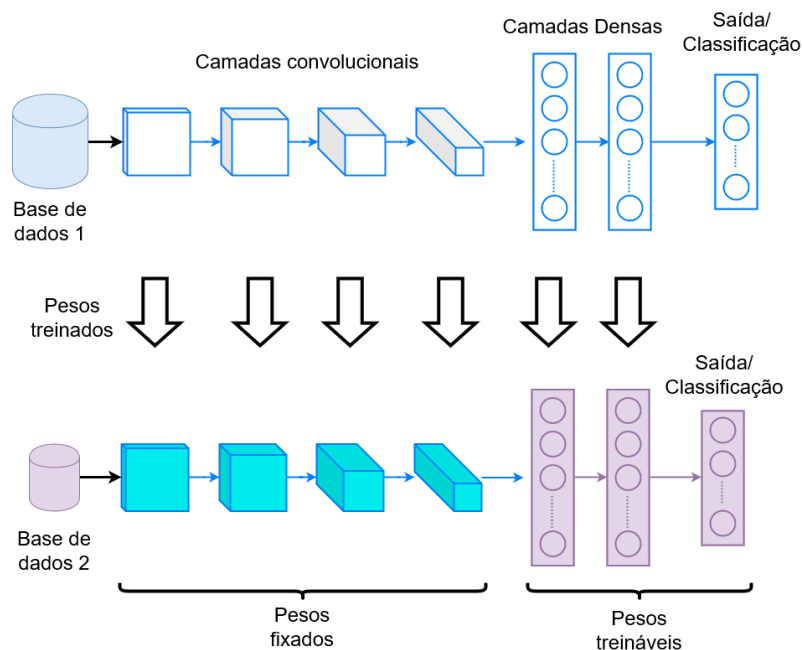


Figura 6.14. Esquema de transferência de aprendizado por ajuste fino. Neste exemplo as camadas convolucionais mantiveram seus pesos e as densas tem os pesos treináveis.

6.5. Estudo de Caso

Atualmente as CNNs são uma das técnicas mais eficazes no diagnóstico de imagens médicas. No entanto, essas redes exigem um alto custo computacional e em sistemas com

```

1  from keras.layers import Dense
2  from keras.applications.vgg16 import VGG16
3  from keras.models import Model
4  from keras.optimizers import SGD
5
6  # modelo pré-treinado da VGG-16
7  vgg16_model = VGG16(weights='imagenet', include_top=True)
8
9  # guardar a penúltima camada (excluída a de classificação)
10 fc2 = vgg16_model.get_layer('fc2')
11
12 # nova camada de classificação binária
13 newlayer = Dense(2, activation='softmax')(fc2.output)
14
15 # modelo com a nova camada de classificação
16 model = Model(inputs=vgg16_model.input, outputs=newlayer)
17
18 # fixar os pesos das camadas iniciais.
19 # Treináveis só as 3 últimas.
20 for l in model.layers[:-3]:
21     l.trainable = False
22
23 # compilação do modelo com pequena taxa de aprendizagem
24 model.compile(optimizer=SGD(lr=1e-3,
25                             decay=1e-6,
26                             momentum=0.9,
27                             nesterov=True),
28               loss='categorical_crossentropy',
29               metrics=['accuracy'])
30
31 model.summary()
32
33
34 '''
35 O modelo está pronto, e a transferência de aprendizado já está feita.
36 Daqui podemos seguir normalmente com treinamento
37 do modelo na nova base usando o método model.fit().
38 '''

```

Figura 6.15. Código em python para o ajuste fino da CNN VGG-16. os pesos das camadas convolucionais foram mantidos da imagenet, já as camadas densas são treináveis i.e. serão ajustados após execução do método fit().

baixo poder de processamento e armazenamento, a utilização dessa técnica se torna dispendiosa.

6.5.1. Problema

A leucemia é um dos tipos de câncer que mais afeta a população. De acordo com a *American Cancer Society* (ASC)¹, cerca de 61.780 novos casos da doença foram estimados para 2019, com aproximadamente 22.840 mortes. Essa doença não possui etiologia definida e afeta a produção de glóbulos brancos na medula óssea. Com a presença da enfermidade, células jovens ou blastos são produzidos de forma anormal, substituindo células sanguíneas normais (glóbulos brancos, vermelhos e plaquetas). Consequentemente o indivíduo sofre com problemas no transporte de oxigênio e no combate a in-

¹<https://cancerstatisticscenter.cancer.org/!/cancer-site/Leukemia>

fecções [Bodey et al. 1966]. Dentre as formas de diagnóstico da leucemia, o hemograma e o mielograma são as mais utilizadas. No entanto, existem meios menos usuais de diagnosticar a doença como a citometria de fluxo e a punção lombar [Agaian et al. 2016].

O tratamento da leucemia e o diagnóstico precoce auxilia na sobrevivência do paciente. Portanto, para prover um diagnóstico auxiliar e eficaz para os especialistas, existem os sistemas de auxílio ao diagnóstico. Através de imagens de exames, esses sistemas utilizam técnicas de processamento de imagens e aprendizado de máquina para diagnosticar a patologia objetivo [Vogado et al. 2018]. A Figura 6.16, observamos exemplos lâminas com a presença de leucemia.

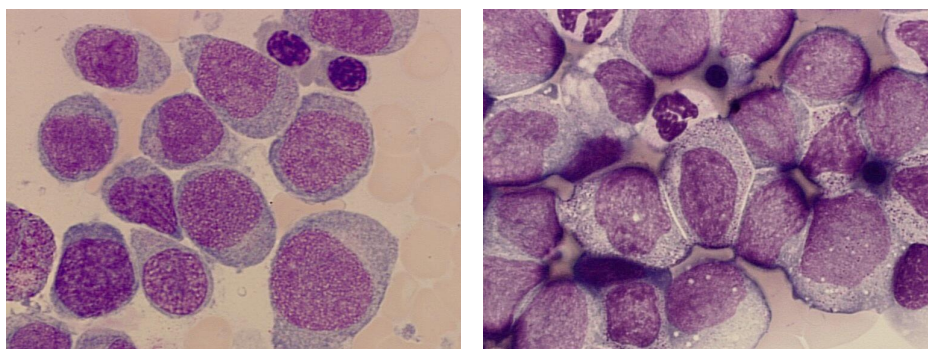


Figura 6.16. Exemplos de lâminas de sangue com leucemia. Fonte: <http://leukemia-images.com>

6.5.2. Base de Dados

As redes neurais convolucionais possuem uma grande quantidade de parâmetros a serem otimizados, como já mencionado anteriormente. A CNN é composta por diversas camadas (*layers*), cada uma com diversos neurônios, as arquiteturas CNN passam por um processo de treinamento (com um *dataset* de treino) em uma etapa inicial.

Esse processo de otimização permite, ao longo de várias iterações, encontrar os pesos ideais para cada um dos neurônios da CNN – quanto mais “ideal” os pesos forem, melhor será a acurácia do seu modelo na teoria (Observar Figura 6.17).

Assim, para problemas do mundo real, normalmente não temos um *dataset* adequado para treinar a CNN. Pode ser que a quantidade de fotos que temos sobre um determinado objeto seja muito pequena para treinar os milhares (ou milhões) de parâmetros, ou que todas as fotos que temos foram batidas do mesmo ângulo.

Uma das consequências de trabalharmos com um *dataset* de treino desse tipo, é que a CNN não vai ter a generalidade necessária para trabalhar com o conjunto de dados de teste. O modelo vai sofrer do indesejado efeito de *overfitting*. As consequências práticas serão que o modelo apresentará um desempenho excelente quando rodado com as imagens de treino, com acurácia altíssima, porém um mau desempenho quando rodamos o *dataset* de teste.

Para reduzir o *overfitting* existem muitos métodos, como técnicas de normalizações dos pesos, método do *dropout* (remover aleatoriamente algumas conexões entre neurônios de *layers* subjacentes), ou *batch normalization*, cada uma com suas vantagens

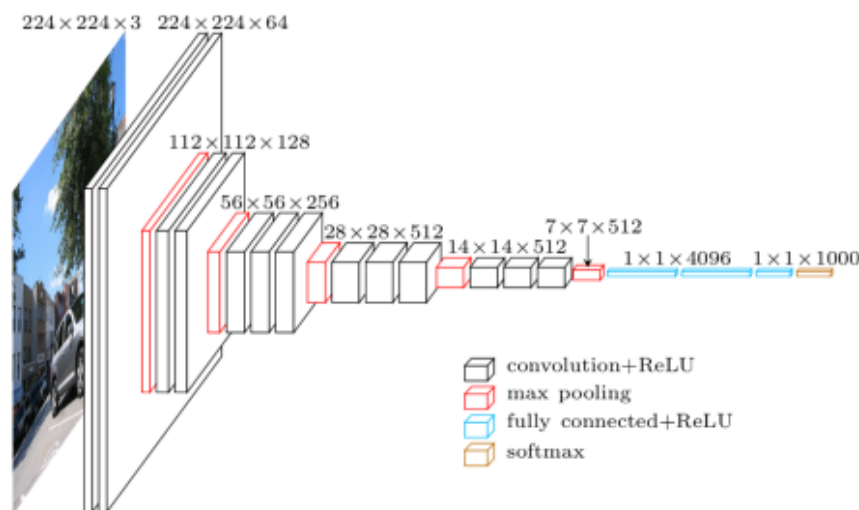


Figura 6.17. As CNN modernas possuem milhares ou até milhões de parâmetros para serem otimizados.

e características. O método conhecido como *data augmentation*, que pode ser implementado facilmente com a biblioteca keras – e que vai ajudar você a minimizar o problema nas suas CNN.

6.5.2.1. Data Augmentation

Data Augmentation é uma técnica para gerar novos exemplares de dados de treinamento a fim de aumentar a generalidade do modelo. Basicamente, toda modificação feita em um algoritmo com a intenção de reduzir o erro de generalização (mas não o erro de treinamento) é uma técnica de regularização. Bom, a técnica de *data augmentation* se encaixa exatamente nesse perfil.

Aplicando essa técnica, veremos a nossa precisão no treinamento piorar, porém a precisão sobre o *dataset* de teste vai melhorar: o modelo CNN se tornará mais genérico. Existem muitas maneiras possíveis para gerar outras imagens, mas os métodos mais comuns são aplicando combinações de operações sobre a imagem original, como: translação, rotação, modificação a perspectiva, achatamento e alongamento, distorção de lentes.

Uma rede neural convolucional que pode classificar objetos de maneira robusta, mesmo que se diga que ele é colocado em orientações diferentes, possui a propriedade chamada invariância. Mais especificamente, uma CNN pode ser invariável à translação, ponto de vista, tamanho ou iluminação (ou uma combinação dos itens demonstrados na Figura 6.18).

Essa é essencialmente a premissa do aumento de dados. No cenário do mundo real, podemos ter um conjunto de dados de imagens tiradas em um conjunto limitado de condições. Porém, nosso aplicativo de destino pode existir em uma variedade de condições, como orientação, localização, escala, brilho diferentes entre outras condições. Essas

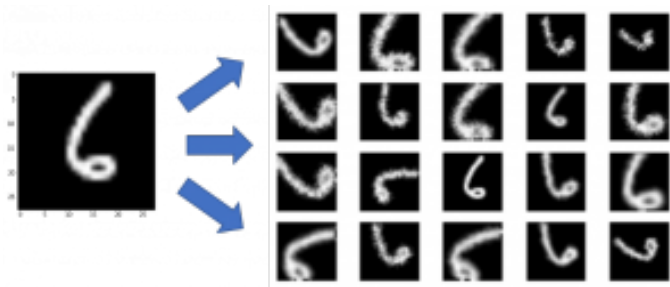


Figura 6.18. Exemplo de *Data Augmentation*. Fonte: [Raj 2018]

situações são treinadas na rede neural com dados adicionais modificados sinteticamente.

6.5.3. Métodos

O Código Fonte apresentado na Figura 6.19 contém a declaração das principais funções necessárias para o treinamento da CNN.

```

1  from keras.models import Sequential
2  from keras.applications.vgg16 import preprocess_input
3  from keras.models import Model
4  from keras.layers import Dense
5  from keras import backend as K
6  from keras.optimizers import Adam
7  from keras.layers import Input, Dense
8  from keras_preprocessing.image import ImageDataGenerator
9  import matplotlib.pyplot as plt
10 import numpy as np
11 import keras as keras
12
13 # Carregando o modelo pré-treinado da vgg16 com os pesos da imagenet.
14 vgg16_model = keras.applications.vgg16.VGG16(include_top=True, weights='imagenet')
15
16 # Definimos o novo modelo como sequencial.
17 novo_model = Sequential()
18
19 # Para todas as N camadas da vgg16, adicionamos N-1 no novo modelo, ou seja, todas menos a de classificação.
20 for layer in vgg16_model.layers[:-1]:
21     novo_model.add(layer)
22
23 # Adicionando uma nova camada densa com um neurônio e função de ativação sigmoid.
24 novo_model.add(Dense(1, activation='sigmoid'))
25
26 # Você pode definir quais camadas são treináveis e não-treináveis, facilitando o processo de ajuste.
27 for layer in novo_model.layers[:-1]:
28     layer.trainable = False
29
30 novo_model.summary()

```

Figura 6.19. Modificando os pesos da CNN VGG-16.

O Código Fonte apresentado na Figura 6.20 contém a leitura do conjunto de dados de treino e de teste e treinamento da CNN.

O Código Fonte apresentado na Figura 6.21 demonstra como utilizar a CNN para classificar o conjunto de teste e calcular a acurácia.


```

31
32 # Definição do conjunto de treinamento utilizado.
33 diretorio_treino = '/Users/Luís_Henrique/database/treino'
34
35 # Operações realizadas para o conjunto de treinamento.
36 train_datagen = ImageDataGenerator(
37     rotation_range=40,
38     width_shift_range=0.2,
39     height_shift_range=0.2,
40     rescale=1./255,
41     shear_range=0.2,
42     zoom_range=0.2,
43     horizontal_flip=True,
44     vertical_flip = True,
45     fill_mode = 'reflect'
46 )
47
48 # Aplicando as operações no diretório de treino e criando os batches.
49 batches_treino = train_datagen.flow_from_directory(diretorio_treino,
50     target_size=(224,224),
51     color_mode='rgb',
52     class_mode='binary',
53     batch_size= 32
54 )
55 # Definição do conjunto de validação utilizado no treino.
56 diretorio_val = '/Users/Luís_Henrique/database/teste'
57
58 # Operações realizadas para o conjunto de teste.
59 val_datagen = ImageDataGenerator(rescale=1./255)

```

Figura 6.20. Leitura dos conjuntos de dados de treinamento e teste da CNN.

```

60 # Aplicando as operações no diretório de teste e criando os batches.
61 batches_val = val_datagen.flow_from_directory(diretorio_val,
62     target_size=(224,224),
63     color_mode='rgb',
64     class_mode='binary',
65     batch_size = 32
66 )
67 # Algoritmo de otimização selecionado com taxa de aprendizado de 0.001.
68 sgd = SGD(lr=0.001, momentum=0.8, decay = 0.0001)
69
70 # Compilando o modelo com otimizador, função de custo e as métricas de avaliação.
71 novo_model.compile(optimizer = sgd, loss='binary_crossentropy', metrics=['accuracy'])
72
73 # Treinando o modelo e definindo a quantidade de epochs de treino, assim como os steps para cada epoch.
74 history = novo_model.fit_generator(batches_treino,
75     steps_per_epoch= 32,
76     epochs=50,
77     validation_data = batches_teste,
78     validation_steps = 32,
79     verbose=1
80 )
81
82 # Utilizando o histórico: o gráfico da função de custo ao longo das épocas.
83 plt.plot(history.history['loss'])
84 plt.plot(history.history['val_loss'])
85 plt.ylabel('Loss')
86 plt.legend(['train', 'test'], loc='upper left')
87 plt.show()
88
89 # Utilizando o histórico: o gráfico da acurácia ao longo das épocas.
90 plt.plot(history.history['binary_accuracy'])
91 plt.plot(history.history['val_binary_accuracy'])
92 plt.ylabel('Accuracy')
93 plt.legend(['train', 'test'], loc='upper left')
94 plt.show()

```

Figura 6.21. Utilização da CNN para classificar o conjunto de teste e calcular o valor da acurácia.

Referências

- [Abadi et al. 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
- [Agaian et al. 2016] Agaian, S., Madhukar, M., and Chronopoulos, A. T. (2016). A new acute leukaemia-automated classification system. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 0(0):1–12.
- [Bejnordi et al. 2018] Bejnordi, B. E., Mullooly, M., Pfeiffer, R. M., Fan, S., Vacek, P. M., Weaver, D. L., Herschorn, S., Brinton, L. A., van Ginneken, B., Karssemeijer, N., Beck, A. H., Gierach, G. L., van der Laak, J. A. W. M., and Sherman, M. E. (2018). Using deep convolutional neural networks to identify and classify tumor-associated stroma in diagnostic breast biopsies. *Modern Pathology*, 31:1502–1512.
- [Bodey et al. 1966] Bodey, G. P., Buckey, M., Sathe, Y. S., and Freireich, E. J. (1966). Quantitative Relationships Between Circulating Leukocytes and Infection in Patients with Acute Leukemia. *Annals of Internal Medicine*, 64(2):328–340.
- [Chollet 2017] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1800–1807.
- [Chollet et al. 2015] Chollet, F. et al. (2015). Keras. <https://keras.io>.
- [Deng et al. 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.
- [Dundas and Chik 2011] Dundas, J. and Chik, D. (2011). Implementing human-like intuition mechanism in artificial intelligence. *arXiv preprint arXiv:1106.5917*.
- [Fukushima 1988] Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119 – 130.
- [Glorot et al. 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G. J. and Dunson, D. B., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323.
- [Gonzalez et al. 2002] Gonzalez, R. C., Woods, R. E., et al. (2002). Digital image processing.
- [Goodfellow et al. 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

- [Haykin 2007] Haykin, S. (2007). *Redes neurais: princípios e prática*. Bookman Editora.
- [He et al. 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE Computer Society.
- [Hu et al. 2018] Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141.
- [Huang et al. 2017] Huang, G., Liu, Z., v. d. Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269.
- [Jia et al. 2014] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia, MM '14*, pages 675–678.
- [Krizhevsky et al. 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.
- [Lecun et al. 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- [Lecun et al. 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.
- [Marques Filho and Neto 1999] Marques Filho, O. and Neto, H. V. (1999). *Processamento digital de imagens*. Brasport.
- [Meneses et al. 2012] Meneses, P. R., Almeida, T. d., et al. (2012). Introdução ao processamento de imagens de sensoriamento remoto. *Universidade de Brasília, Brasília*.
- [Pacheco and Pereira 2018] Pacheco, C. A. R. and Pereira, N. S. (2018). Deep learning conceitos e utilização nas diversas áreas do conhecimento. *Revista Ada Lovelace*, 2:34–49.
- [Pan et al. 2010] Pan, S. J., Yang, Q., et al. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.
- [Raj 2018] Raj, B. (2018). Data augmentation | how to use deep learning when you have limited data—part 2.
- [Rezende 2003] Rezende, S. O. (2003). *Sistemas inteligentes: fundamentos e aplicações*. Editora Manole Ltda.

- [Ronneberger et al. 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597.
- [Russakovsky et al. 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.
- [Santos et al. 2019] Santos, J. D., Veras, R. d. M. S., Veloso, R. R., Aldeman, N. L. S., Aires, K. R. T., Bianchi, A. G. C., et al. (2019). Classificação de imagens de biópsias renais com glomeruloesclerose segmentar e focal ou com lesões mínimas utilizando transfer learning em cnn. In *Anais do XIX Simpósio Brasileiro de Computação Aplicada à Saúde*, pages 82–93. SBC.
- [Sarkar 2018] Sarkar, D. (2018). A comprehensive hands-on guide to transfer learning with real-world applications in deep learning. <https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a>.
- [Sarkar et al. 2018] Sarkar, D., Bali, R., and Ghosh, T. (2018). *Hands-On Transfer Learning with Python: Implement advanced deep learning and neural network models using TensorFlow and Keras*. Packt Publishing Ltd.
- [Simonyan and Zisserman 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.
- [Szegedy et al. 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826.
- [Szegedy et al. 2015] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9.
- [Vogado et al. 2018] Vogado, L. H. S., Veras, R. M. S., Araújo, F. H. D., e Silva, R. R. V., and Aires, K. R. T. (2018). Leukemia diagnosis in blood slides using transfer learning in cnns and SVM for classification. *Engineering Applications of Artificial Intelligence*, 72:415–422.

Capítulo

7

Sustentando Microsserviços em Ambiente de Cloud Computing utilizando Kubernetes e Service Mesh

Luciano de Aguiar Monteiro, Washington Henrique Carvalho Almeida, Anderson Cavalcanti de Lima, Sahra Karolina Gomes e Silva e Fernando Escobar

Abstract

Microservices are the current architectural standard in the information technology market and have replaced monolithic systems for cloud computing applications. Support for microservices should be based on software industry standards and best practices. This chapter introduces the tools for supporting microservices in highly complex environments. The tools presented are Kubernetes and Service Mesh. The proposal is to present an architecture for operation in cloud computing environments with high scalability and resilience as well as an overview of their deployment.

Resumo

Os microsserviços são o padrão arquitetural em voga no mercado de tecnologia da informação tendo substituído sistemas monolíticos para aplicações em ambiente de Cloud Computing. A sustentação de microsserviços deve ser baseada em padrões adotados pela indústria de software e nas melhores práticas. Este capítulo apresenta as ferramentas para a sustentação de microsserviços em ambientes de alta complexidade. As ferramentas apresentadas são o Kubernetes e o Service Mesh, a proposta é apresentar uma arquitetura para operação em ambientes de computação em nuvem com alta escalabilidade e resiliência bem como uma visão geral de sua implantação.

7.1. Introdução

Transformações digitais impulsionam uma evolução na arquitetura dos softwares. Em um primeiro momento, os sistemas eram desenvolvidos para serem executados em modo local (modelo conhecido como *standalone*), no qual tanto a Aplicação quanto o Banco de Dados eram executados no mesmo sistema operacional do usuário. Essa arquitetura evoluiu para um modelo cliente/servidor, em que o usuário passou a acessar

a aplicação via programa local (cliente), interagindo com o servidor por meio de requisições encaminhadas pela rede de computadores.

O padrão arquitetural de software teve sua grande transformação a partir da disponibilização dos serviços de cloud computing, iniciado em 2006 com a Amazon AWS (VERAS, 2013), na qual a arquitetura de software deixou de ser monolítica, ou seja, um único código-fonte para toda a aplicação, migrando para microsserviços, constituídos por um conjunto de serviços pequenos e independentes executados em seus próprios processos, desenvolvidos e implantados de forma autônoma (INDRASIRI; SIRIWARDENA, 2018).

Estes microsserviços são executados dentro de containers, que são abstrações de sistemas operacionais (SO) disponibilizados a partir de isolamento de processos e recursos, gerenciados pelo Docker (MCKENDRICK; GALLAGHER, 2018). Nesse sentido, aplicações com pequenas quantidades de microsserviços são fáceis de gerenciar e de rápida solução. No entanto, ao aumentar o número de microsserviços, tanto a gestão do container como a interconexão entre eles tornam-se uma tarefa inexecutável.

Uma proposta para a solução desta problemática será o foco deste capítulo, no qual serão apresentados os conceitos e a arquitetura do Kubernetes, projeto de código aberto criado pela Google para a orquestração de containers, com o objetivo de garantir sua estabilidade e resiliência (SAYFAN, 2017). Outro aspecto estudado será a comunicabilidade e interconexão entre os microsserviços providos pelo Service Mesh, que introduz uma camada de infraestrutura dedicada sobre os microsserviços sem impor modificações em suas implementações (LI *et al.*, 2019a). Neste trabalho, o Istio será apresentado como exemplo de arquitetura de implementação do Service Mesh.

A organização deste ambiente será focada na arquitetura de cloud computing, considerando principalmente os fatores de alta disponibilidade, escalabilidade e resiliência. Os principais provedores de nuvem, como Amazon AWS, Azure e Google Cloud Platform, possuem implementações para execução do Kubernetes em suas plataformas.

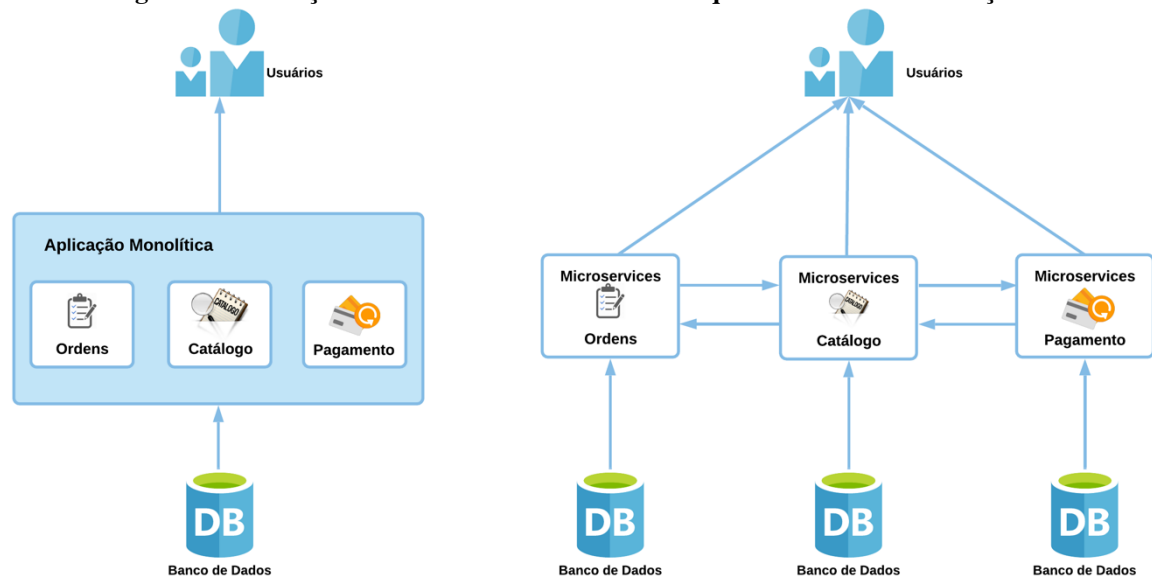
7.2. Arquitetura de Microsserviços

Evoluções tecnológicas proporcionadas com o desenvolvimento de novas soluções tecnológicas têm sido viabilizadas por meio do uso frequente de arquitetura computacional baseada em Cloud Computing.

Nesse contexto, os microsserviços são adotados como estilo arquitetural que busca aproveitar o potencial de processamento distribuído proporcionado pela cloud computing, dividindo o software em serviços menores, independentes entre si, que possam ser executados em processos distintos.

Microsserviços são serviços autônomos, modelados em torno de um domínio específico de negócio, que são executados em processos independentes e que interagem entre si por meio de mensagens, contrapondo-se, assim, ao modelo aplicado em softwares monolíticos. A Figura 1 apresenta uma demonstração da diferença entre uma aplicação monolítica e um sistema desenvolvido com arquitetura de microsserviços.

Figura 1. Diferença entre o modelo monolítico e a arquitetura de microsserviços.



Fonte: dos autores (2019).

Segundo Dragoni *et al.* (2016), um software monolítico é aquele cujo módulos não podem ser executados de maneira independente, dificultando o uso desse tipo de software em sistemas distribuídos sem a adoção de frameworks específicos para integração. Uma arquitetura monolítica considera que todas as funcionalidades de negócio são agrupadas em uma única aplicação e são distribuídas como um único produto de software (INDRASIRI; SIRIWARDENA, 2018).

De acordo com Indrasiri e Siriwardena (2018), o software monolítico possui uma série de características que se apresentam como limitações para sua implantação, execução e manutenção. Para Dragoni *et al.* (2016), algumas dessas características são: (a) dificuldade na manutenção e evolução do software monolítico por conta de seu tamanho e complexidade; (b) qualquer mudança nos módulos de um software monolítico requerem a implantação do software todo; (c) limitação da escalabilidade do software; (d) dificuldade na adoção de novas tecnologias, considerando que a solução toda deve compartilhar o mesmo padrão de desenvolvimento; (e) subutilização dos recursos de implantação, visto que o ambiente deve ser dimensionado para executar a solução completa.

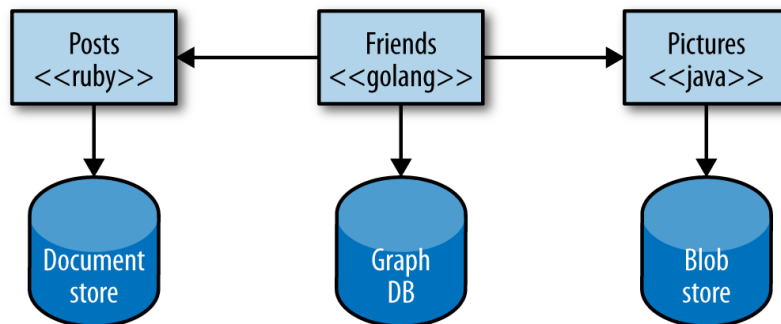
A arquitetura de microsserviços surge como um estilo arquitetural para sobrepor as limitações presentes em softwares monolíticos, ao tempo que permite potencializar os benefícios apresentados pela computação distribuída oferecido pelo modelo de cloud computing.

Segundo Amundsen et al. (2018), os softwares baseados em microsserviços compartilham importantes características: (a) pequenos no tamanho; (b) habilitados a troca de mensagens; (c) restritos a um contexto de negócio específico; (d) desenvolvidos de maneira autônoma; (e) distribuídos de maneira independente; (f) descentralizados; (g) construídos e distribuídos com processos automatizados.

Para Newman (2015), os principais benefícios da arquitetura de microsserviços consistem nos elementos a seguir.

- Heterogeneidade de tecnologia – a arquitetura de microsserviços permite que sejam usadas tecnologias distintas no desenvolvimento de cada microsserviço. Isso permite que sejam usadas as ferramentas mais apropriadas para cada trabalho, em contraposição a se adotar uma ferramenta única que atenda a todos os aspectos técnicos da solução completa. A Figura 2 apresenta um exemplo do uso de diferentes linguagens e Bancos de Dados para implementação de microsserviços que conversam entre si.

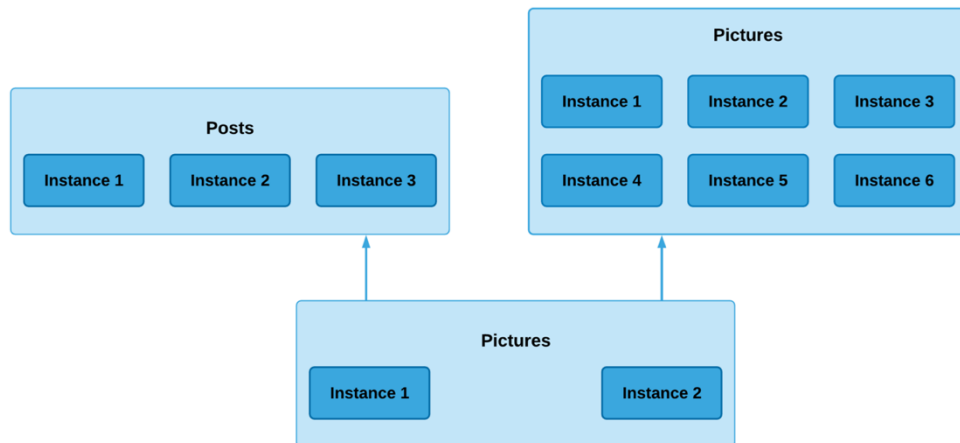
Figura 2. Exemplo de adoção de tecnologias diferentes para microsserviços.



Fonte: Newman (2015).

- Resiliência – capacidade de isolar um eventual problema ocorrido no microsserviço, de modo que a falha não seja propagada para o resto do sistema, permitindo que ele possa continuar a funcionar. Além disso, a implementação de mecanismos de redundância se torna especialmente facilitada quando o software é distribuído utilizando a arquitetura de microsserviços.
- Escalonamento – facilidade de criação de diversas instâncias de um microsserviço sem precisar alocar recursos para a solução completa. Isso é possível em razão do tamanho reduzido do microsserviço e da limitação de recursos necessários apenas para atender à funcionalidade de negócio por ele implementada. Esse benefício proporciona, ainda, a economia de recursos por escalar apenas os microsserviços que necessariamente sejam demandados pela aplicação, mantendo o número de microsserviços correspondentes às demais funcionalidades do sistema a níveis reduzidos. A Figura 3 demonstra o escalonamento de diferentes microsserviços.
- Facilidade de instalação – capacidade de implantar modificações em determinadas funcionalidades do sistema representada por um microsserviço específico sem precisar implantar a aplicação completa. Esse benefício está também relacionado à antecipação da implantação de novas funcionalidades no sistema implementadas como microsserviços, tornando a entrega de software mais rápida para os clientes.
- Alinhamento organizacional – capacidade de organizar as equipes de desenvolvimento designadas para a codificação do software na forma de microsserviços de modo que melhor se adapte a estrutura organizacional da empresa.

Figura 3. Demonstração do escalonamento de diferentes microserviços.



Fonte: Newman (2015).

- Versatilidade – possibilidade de reutilização de microserviços, representando funcionalidades do sistema, de diferentes maneiras e para diferentes propósitos. Isso permite que os microserviços possam ser acessados a partir de diferentes plataformas, tais como aplicações web, desktop ou para dispositivos móveis, permitindo estabelecer uma arquitetura adaptável aos diferentes meios de envolvimento dos clientes de uma organização.
- Otimização para substituíbilidade – facilidade de substituição de membros da equipe de desenvolvimento em caso de desligamento da organização. Em organizações de médio e grande porte, a rotatividade de profissionais da equipe de desenvolvimento pode ser alta, dificultando a manutenção ou evolução de funcionalidades em sistemas corporativos em decorrência da dependência de conhecimento técnico mantido pelo desenvolvedor. A arquitetura de microserviços restringe o domínio de negócio mantido por um componente de software, o que facilita o compartilhamento do conhecimento, além de tornar mais fácil, em último caso, a reescrita do microserviço, em caso de necessidade.

Nos tópicos a seguir, serão abordados como a arquitetura de microserviços pode ser implementada na prática por meio de soluções de abstração do sistema operacional, no caso containers disponibilizados com Docker, com recursos de orquestração e garantia de resiliência oferecidos pelo Kubernetes e a capacidade de comunicabilidade de interconexão disponibilizada pelo Service Mesh, neste trabalho, implementado com o Istio.

7.3. Container Docker

Um ambiente factível para execução de microserviços são os containers, que se caracterizam por ser uma abstração da arquitetura de um sistema operacional, sendo capaz de isolar recursos de hardware e processos, compartilhando o *kernel* do sistema hospedeiro.

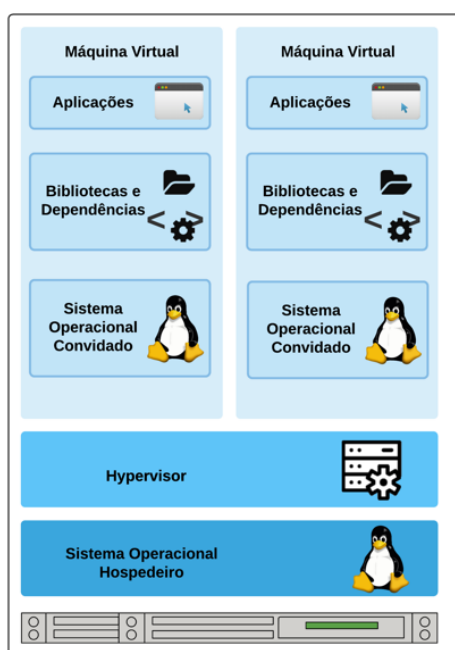
Os containers são baseados em uma técnica de virtualização em nível de sistema operacional que fornece ambientes virtuais utilizando Linux Containers (LCX),

facultando o isolamento do processo e da rede por meio de *chroot*, *cgroups* e *namespaces* (CITO *et al.*, 2017).

A abordagem de container pode acarretar dúvidas quando comparados com máquinas virtuais (MV). Considerando a semelhança entre os modelos, será apresentado um comparativo em relação ao uso de container e máquinas virtuais.

A Virtualização é o processo de dividir a máquina física em vários componentes virtuais, nos quais cada um pode hospedar sistemas operacionais diferentes, cabendo a gestão destes SO ao *hypervisor* (SALAH *et al.*, 2017). A Figura 4 apresenta um ambiente com esta arquitetura. Um aspecto a se ressaltar é o consumo de hardware: cada sistema convidado possui um kernel em execução.

Figura 4. Máquinas Virtuais.



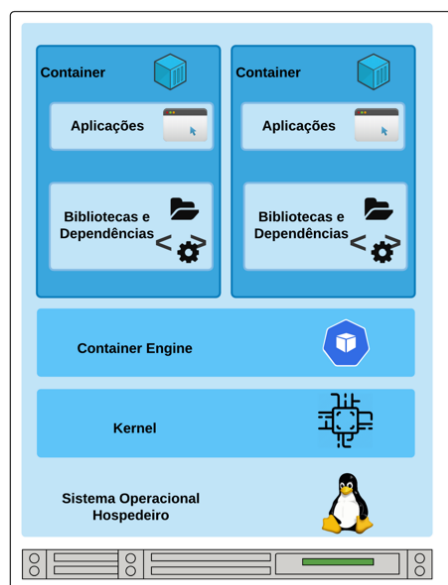
Fonte: Bisong e Learning (2019).

Na arquitetura de container, bibliotecas, *system calls* e recursos do SO hospedeiro são compartilhados com os containers. Neste ambiente, cada container compartilha ainda o mesmo *kernel* do SO hospedeiro, tornando a sua execução mais rápida. A Figura 5 apresenta um ambiente com esta abordagem executando dois containers.

Funcionalmente, cada componente destacado a seguir proporciona uma estrutura para a execução do SO no container:

- *chroot* – disponibiliza um diretório-raiz;
- *cgroups* – fornecem mecanismos para contabilizar e limitar os recursos que os processos podem utilizar em cada container (BUI, 2015);
- *namespaces* – criam grupos de processos de maneira que um grupo de processo não seja visualizado por outro, garantindo o isolamento dos containers.

Figura 5. Container.



Fonte: Bisong e Learning (2019).

Existem disponíveis várias ferramentas de gestão de container, como o Docker, o CoreOS, o Mesos e o LXC. O presente estudo concentra-se somente no Docker, considerada a ferramenta mais usada para este propósito, conforme evidenciado por Carter (2018), em seu Relatório de Uso de Container, apresentado na Figura 6.

Figura 6. Relatório de Uso de Container.



Fonte: Carter (2018).

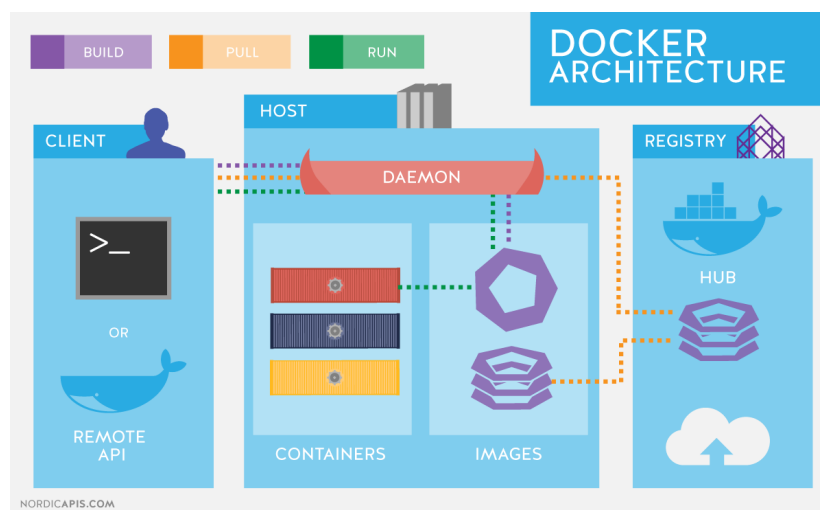
No Docker, cada container é baseado em uma imagem e em um conjunto de dados de configuração. As imagens são cópias estáticas da configuração dos containers (TOSATTO; RUIU; ATTANASIO, 2015). Todas as imagens de um container ficam hospedadas em um servidor remoto, chamado Docker Hub, no qual o usuário pode fazer seu cadastro e disponibilizar sua imagem personalizada.

Para compreender como ocorre a gestão dos containers com Docker, se faz necessário conhecer a arquitetura deste ambiente, apresentada na Figura 7. A interação com o ambiente é realizada por meio do **Docker Client**, a partir do qual é possível realizar o download de imagens, criação de container, entre outros.

O componente responsável para atender as requisições do Docker Client é o **Docker Daemon**, atendendo as solicitações da API do Docker (do inglês *Application Programming Interface* - Interface de Programação de Aplicações) e gerenciando objetos, como imagens, containers, redes e volumes. O download das imagens dos

containers e as suas publicações é feito a partir de um repositório de registros de imagens, o Docker Registry. A loja oficial do Docker é o Docker Hub. Todavia, existem outros ambientes para armazenamento de imagens, como o Amazon Elastic Container da AWS e o Google Cloud Platform e a Azure, que usam a nomenclatura padrão para o serviço, chamando de Docker Registry.

Figura 7. Arquitetura Docker.



Fonte: Docker (2019).

A administração de container com Docker não será abordada neste capítulo, considerando que a orquestração do ambiente proposto é realizada pelo Kubernetes, havendo somente o pré-requisito do Docker instalado. As orientações quanto a sua instalação encontram-se no site oficial, transcritos nos links abaixo:

- **Ubuntu** - <https://docs.docker.com/install/linux/docker-ce/ubuntu/>
- **Debian** - <https://docs.docker.com/install/linux/docker-ce/debian/>
- **CentOS** - <https://docs.docker.com/install/linux/docker-ce/centos/>

A arquitetura de microsserviços torna-se mais vantajosa ao tempo em que eles são implementados por meio de containers. Convém ressaltar que os microsserviços podem ser executados em qualquer arquitetura de sistema operacional. No entanto, a arquitetura de container possibilita separar componentes, agrupando o aplicativo com todas as suas dependências em um software independente, que, por sua vez, pode ser executado em qualquer plataforma (CASALICCHIO, 2019), agregando agilidade ao processo com a adoção de ferramentas de automação, versionamento de código e infraestrutura como código.

7.4. Kubernetes: Arquitetura, Implantação e Administração

Uma dúvida comum e recorrente ao abordar a adoção de orquestradores de container, como o Kubernetes, decorre da eventual necessidade de sua utilização, considerando

que o Docker já realiza a administração de containers. É justamente este aspecto que difere o Docker do Kubernetes. A fundamentação do Docker é a gestão do container, desde a criação, administração e finalização do mesmo. O Docker não cuida da automação e da resiliência do ambiente. O Kubernetes provê uma plataforma para automatizar a implantação, o dimensionamento e as operações de containers de aplicativos por meio de clusters de hosts (MOORE, 2019).

De acordo com Hightower, Burns e Beda (2017), o cluster disponibilizado pelo Kubernetes agrega ainda as características relacionadas a seguir:

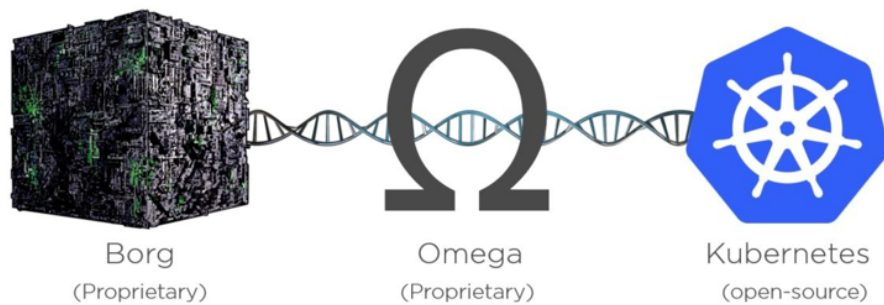
- velocidade – proporciona um ambiente altamente disponível, mesmo durante a atualização de funcionalidades do sistema e novas entregas;
- escalabilidade – alcançada favorecendo arquiteturas desacopladas para o time de desenvolvimento. Em uma arquitetura desacoplada, cada componente é separado dos demais, por APIs definidas e balanceadores de carga de serviço. Outro exemplo de desacoplamento é a arquitetura de microsserviços, provida neste tipo de ambiente;
- abstração da infraestrutura - os desenvolvedores passam a criar seus aplicativos em termos de imagens de container e os implementam em termos de APIs Kubernetes portáteis, transferir seu aplicativo entre ambientes ou mesmo executar em ambientes híbridos é apenas uma questão de enviar a configuração declarativa para um novo cluster; e,
- eficiência – o kubernetes provê ferramentas que automatizam a distribuição de aplicativos em um cluster de máquinas, garantindo níveis mais altos de utilização do que é possível com as ferramentas tradicionais;

O Google usa container para serviços em produção, tais como gMail, Google Search, Google Maps, e outros, utilizando um gestor de container proprietário interno chamado Borg. O Borg era essencialmente um sistema de gerenciamento centralizado que alocava e escalonava containers, fortemente acoplado às tecnologias proprietárias e internas do Google (THAMES *et al.*, 2019).

O Ômega foi um segundo projeto do Google de gestão de container. Ele foi construído desde o início para apresentar uma arquitetura mais consistente e baseada em princípios, mas ainda implementava arquitetura proprietária.

Em 2014, o Google criou um projeto de código aberto visando a orquestração de containers, chamado Kubernetes, o qual teve sua primeira versão publicada por volta de julho de 2015 sob o modelo de licença Apache 2.0. Kubernetes é uma palavra de origem grega que significa “timoneiro, piloto”. Também é chamado de k8s, por conta da sua letra inicial “K”, as oito letras no meio e depois “S”. A Figura 8 apresenta a evolução dos sistemas de gestão de containers da Google.

Figura 8. Origem Kubernetes.



Fonte: (POULTON, 2018)

7.4.1. Arquitetura do Kubernetes

Para arquitetura de um ambiente Kubernetes, se faz necessária a implantação de um *cluster* de pelo menos três nós, sendo um *master node* e dois *worker nodes*. Outro pré-requisito é o *docker engine* instalado em todos os nós.

O *Master Node* é o ponto de entrada de todas as funções de administração que são responsáveis por controlar e fazer a gestão do cluster, sendo responsável pelos *worker nodes*. É recomendável mais de um Master Node no ambiente de produção, de modo a assegurar uma alta disponibilidade da solução.

O *Worker Node* é um único *host*. Pode ser uma máquina física ou virtual. Seu trabalho é executar Pods - POD é a menor unidade para o Kubernetes; porém, um POD não precisa ser exclusivamente um container. Cada nó do Kubernetes executa vários componentes do Kubernetes, como um *kube-proxy* e um *kubelet*. Os nós são gerenciados por um *Master Node* do Kubernetes. A Figura 9 representa a arquitetura de um Cluster Kubernetes com seus componentes.

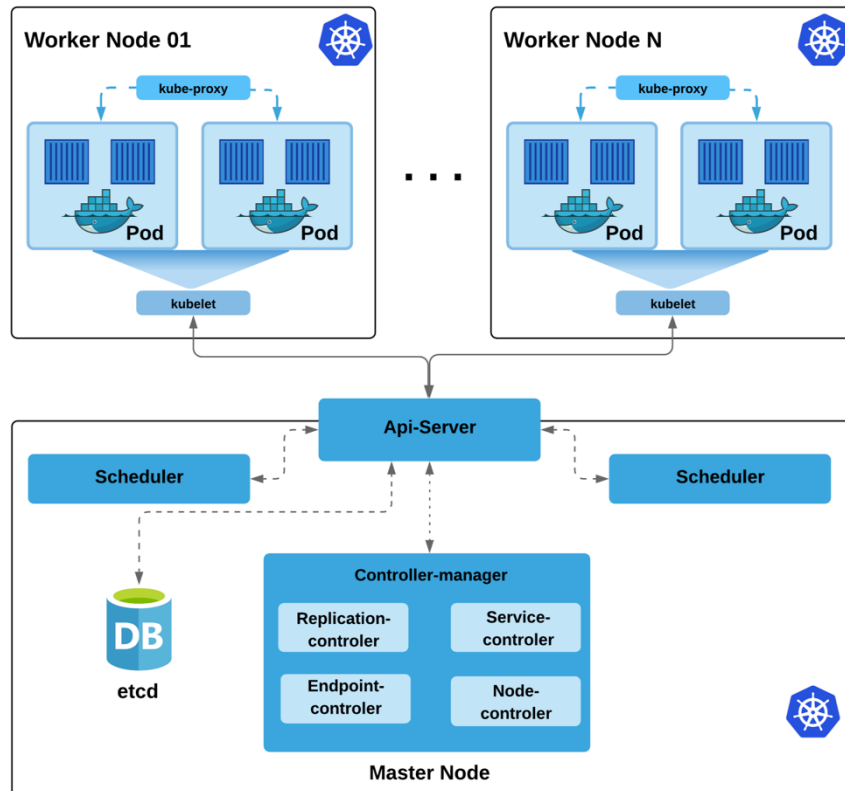
Um *Master Node* é composto por 4 (quatro) componentes essenciais (POULTON, 2018), listados a seguir.

- API-Server – o servidor da API é a interface de gerenciamento central no cluster Kubernetes para interações do usuário e informações de status. É possível enviar e receber dados do *api-server*, pois é um servidor de API RESTful. Toda definição de carga de trabalho é enviada para esse servidor de API e lida com o armazenamento dos dados no Etcd. Como o Kubernetes é uma plataforma orientada a API, o *api-server* é o componente mais crítico no plano de controle.
- Etcd – é um repositório de dados do Kubernetes no qual todas as configurações, as informações de tempo de execução e os status são armazenados. Os estados reais e os estados desejados de recursos são armazenados no Etcd, que é o único componente com estado nos componentes principais. O Etcd é um armazenamento de chave/valor de código aberto desenvolvido pelo CoreOS e é um dos componentes cruciais que tornam o Kubernetes confiável. O Etcd pode ser instalado em vários servidores principais, além de estar acessível dentro do cluster Kubernetes.
- Scheduler - é responsável por atribuir container de carga de trabalho aos nós, levando em consideração a capacidade, os requisitos e o ambiente de

infraestrutura. Pode ser considerado como um *loop* contínuo para verificar cargas de trabalho não atribuídas e encontrar nós apropriados.

- Controller-manager - É um processo *daemon* do Kubernetes para gerenciar o ciclo de vida dos recursos. Os controladores devem ler as novas informações quando uma alteração for vista. Em seguida, eles implementam as alterações necessárias para atingir o estado desejado.

Figura 9. Arquitetura Kubernetes.



Fonte: dos autores.

Os *Workers Nodes* compreendem os seguintes componentes (MOORE, 2019):

- Kubelet – é um agente de comunicação entre o *Master Node* e *Worker Node*, é o responsável por enviar periodicamente ao *Master Node* informações sobre a saúde do nó e dos *Pods*.
- Kube-proxy – é o serviço responsável pela rede nos servidores do nó. Como os containers e o sistema *host* são isolados em termos de rede, este é o serviço que encaminha solicitações para os containers e os torna acessíveis a partir do mundo externo.
- Pods - Um Pod é a unidade de trabalho no Kubernetes. Cada Pod contém um ou mais containers. Os Pods são sempre agendados juntos (sempre executados na mesma máquina). Todos os containers em um Pod têm o mesmo endereço IP e a

mesma porta. Eles podem se comunicar usando o *localhost* ou a comunicação entre processos padrão.

7.4.2. Modelos e Implantação Kubernetes

Ao planejar a implantação de um ambiente Kubernetes, é essencial entender em que estrutura o ambiente deve ser executado, considerando que existem diversos modelos de implantação, sendo alguns deles apresentados neste trabalho. Importante destacar que a abordagem deste capítulo é focada em ambiente de cloud computing.

7.4.2.1. Minikube

É recomendado para os usuários iniciantes, em fase de aprendizagem. Consiste em um ambiente permissível, para testes com Kubernetes. O **Minikube** é uma ferramenta que simula um *cluster* de Kubernetes. A ferramenta cria um ambiente virtualizado, fazendo um papel de *cluster* com um único nó. É importante ressaltar que o Minikube é somente para testes, não devendo ser usado para ambiente de produção.

Antes da instalação, deve-se testar o suporte e a virtualização do SO.

Linux: `grep -E --color 'vmx|svm' /proc/cpuinfo`

Mac OS: `sysctl -a | grep -E --color 'machdep.cpu.features|VMX'`

A instalação do Minikube deve seguir os seguintes passos:

Linux:

```
curl -LoMinikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
\&&chmod +x minikube
```

```
sudo mkdir -p /usr/local/bin/sudo install
```

```
minikube /usr/local/bin/
```

```
minikube start
```

Mac OS:

```
curl -Lominikube https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-amd64
\&&chmod +x minikube
```

```
sudo mv minikube /usr/local/bin
```

```
minikube start
```

Caso haja necessidade de finalizar o Minikube digite:

```
minikube delete
```

7.4.2.2. Modelo de Implantação OnPremises

Em algumas situações, empresas têm a necessidade de executar o cluster Kubernetes em seu próprio hardware local. Desta forma, não existe nenhum impeditivo da sua implantação em uma infraestrutura interna.

Havendo a necessidade da instalação de no mínimo três servidores com sistema operacional Linux, todos com o Docker, instalados conforme orientação do item 1.3. Em todos os nós devem ser instalados o *kubeadm*, *kubelet* e *kubect*, conforme orientação a seguir:

```
apt-get update && apt-get install -y apt-transport-https curl
```

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF
apt-get update
apt-get install -y kubeletkubeadmkubectl
apt-mark hold kubeletkubeadmkubectl
```

Em todos os nós, desative o uso da memória *swap* como o comando: `swapoff -a`.

No Master Node inicialize o *cluster* Kubernetes: `kubeadm init`.

Ao finalizar a configuração do cluster, será apresentado o resumo, o *token* que deve ser usado pelos *Worker Nodes* para ingresso no Master Node, bem como os comandos de configuração do arquivo de manifesto.

```
Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

You can now join any number of machines by running the following on each node
as root:

kubeadm join 10.0.0.20:6443 --token h717sr.y8tyxeq0164787fc --discovery-token-ca-cert-hash
sha256:c4a3d31a4d30190dcabda8c098752bfc670feff18ca7167e09f392e5dfacc94e3
```

Para que os *Worker Nodes* ingressem no cluster, deve-se executar o comando apresentado `kubeadm join`, seguido do ip do *Master Node* e do *token*. Após a inserção de todos os *Worker Nodes* no cluster Kubernetes, pode-se visualizar o *status* dos nós com o comando `kubectl get nodes`, conforme disposto a seguir:

```
root@kubernetes-master:~# kubectl get nodes
NAME          STATUS    ROLES    AGE   VERSION
kubernetes-master  NotReady  master   17m   v1.10.4
kubernetes-no1    NotReady  <none>   6m    v1.10.4
kubernetes-no2    NotReady  <none>   1m    v1.10.4
```

É possível, ainda, visualizar com o comando `kubectl get Pods` todos os Pods criados.

```
root@kubernetes-master:~# kubectl get pods --all-namespaces
NAMESPACE   NAME                                     READY   STATUS    RESTARTS   AGE
kubernetes-system  etcd-kubernetes-master                1/1     Running   0           8m
kubernetes-system  kube-apiserver-kubernetes-master      1/1     Running   0           8m
kubernetes-system  kube-controller-manager-kubernetes-master  1/1     Running   0           8m
kubernetes-system  kube-dns-86f4d74b45-ghvsh            0/3     Pending   0           19m
kubernetes-system  kube-proxy-54ddf                      1/1     Running   0           19m
kubernetes-system  kube-proxy-8pgkn                      1/1     Running   0           8m
kubernetes-system  kube-proxy-tlvfj                      1/1     Running   0           3m
kubernetes-system  kube-scheduler-kubernetes-master      1/1     Running   0           8m
```

7.4.2.3. Modelo de Implantação Cloud Computing

Nativos de nuvem tornaram-se um pragmático padrão arquitetural de implantação adotado por grandes empresas, como Netflix, Nubank e Uber, tendo sido fomentado pela Cloud Native Computing Foundation (CNFC), criada em 2015, que tem o Kubernetes como um dos seus projetos mais conhecidos.

É por meio desta prerrogativa, além das características elencadas anteriormente referentes às vantagens da adoção da arquitetura em nuvem, que se norteiam os padrões apresentados neste capítulo para o uso de Kubernetes e Service Mesh. Os principais

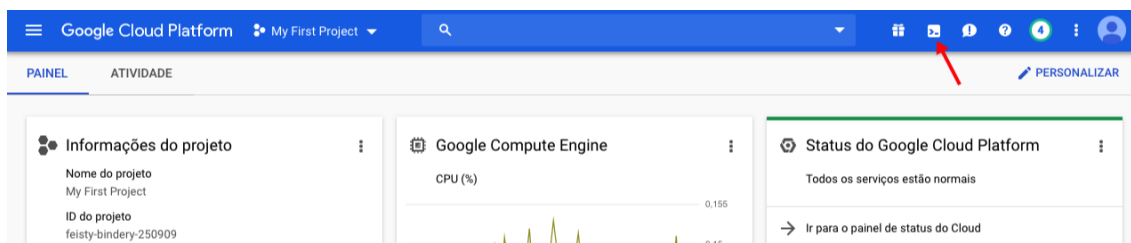
provedores de nuvem possuem ferramentas para implantação de Kubernetes. Na Amazon AWS, temos o Amazon Elastic Kubernetes Service (EKS). Na Azure, o Serviço de Kubernetes da Azure (AKS). Já a IBM possui o Cloud Kubernetes Service e a Google, o Google Kubernetes Engines (GKE).

Considerando o grande volume de informações acerca da abordagem de configurações dos diversos provedores de nuvem apresentados, o capítulo será focado na nuvem da Google Cloud Platform (GCP), especificamente por ser o criador da tecnologia do Kubernetes. Todavia, os conceitos aqui apresentados podem ser aplicados em qualquer provedor de nuvem. A única diferença é a maneira de implantação do ambiente.

7.4.2.3.1. Criação do cluster Kubernetes na CGP

Para implantação de um *cluster* Kubernetes no Google Kubernetes Engines (GKE), é necessária uma conta no GGP, com cobrança ativa e a ferramenta *gcloud* instalada. É possível, ainda, acessar o Cloud Shell via *browser*, conforme apresentado na Figura 10.

Figura 10. Cloud Shell.



Fonte: dos autores.

Após ter acessado o Cloud Shell, é necessário informar o id do projeto que você irá trabalhar. Caso não saiba, no canto superior direito ao lado do nome “Google Cloud Platform” tem o nome do Projeto, clicando nele, será apresentado o id do projeto.

```
server-master-kubernetes@cloudshell:~$ gcloud config set project feisty-bindery-250909
Updatedproperty [core/project].
```

Informe também a zona *default* que será criada o *cluster* Kubernetes. Em nosso exemplo, iremos criar na zona *us-east1-b*.

```
server-master-kubernetes @cloudshell:~ (feisty-bindery-250909)$ gcloud config set compute/zone us-east1-b
Updatedproperty [compute/zone].
```

Para criarmos um novo cluster, devemos usar o comando `gcloud container cluster create`. Em nosso exemplo iremos criar um cluster com o nome *istio-tutorial*, utilizar uma máquina do tipo *n1-standard-2* e o cluster terá três nós.

```
gcloud container clusters create istio-tutorial \
  --machine-type=n1-standard-2 \
  --num-nodes=3
```

Caso esteja usando um *cluster* existente, verifique se possui credenciais de *cluster* para o *kubectl* na máquina em que deseja executar os comandos de instalação.

Por exemplo, o comando a seguir obtém as credenciais para um *cluster* chamado *istio-tutorial*

```
gcloud container clusters get-credentials istio-tutorial
```

Verifique se a versão do *kubectl* é a mesma ou mais recente que o seu cluster. Conceda permissões de administrador de *cluster* ao usuário atual. Serão necessárias estas permissões para criar as regras de controle de acesso baseado em função (RBAC) para o *Istio*:

```
kubectl create clusterrolebinding cluster-admin-binding \
--clusterrole=cluster-admin \
--user="$(gcloud config get-value core/account)"
```

Para verificarmos o estado dos nós podemos utilizar o comando: **kubectl get nodes**.

```
MacBook-Pro-de-Luciano:~ lucianoaguaiar$ kubectl get nodes
NAME                                STATUS    ROLES    AGE      VERSION
gke-istio-tutorial-default-pool-5fe93765-8zcb    Ready    <none>    7d       v1.13.10-gke.0
gke-istio-tutorial-default-pool-5fe93765-16sm    Ready    <none>    7d       v1.13.10-gke.0
gke-istio-tutorial-default-pool-5fe93765-ps9b    Ready    <none>    7d       v1.13.10-gke.0
```

Podemos visualizar os *Pods* que foram criados com o comando: **kubectl get pods -n kube-system**.

```
MacBook-Pro-de-Luciano:~ lucianoaguaiar$ kubectl get pod -n kube-system
NAME                                READY    STATUS    RESTARTS   AGE
event-exporter-v0.2.4-5f88c66fb7-vps8p    2/2      Running    0           7d
fluentd-gcp-scaler-59b7b75cd7-56wp2      1/1      Running    0           7d
fluentd-gcp-v3.2.0-dn8f1                 2/2      Running    0           7d
fluentd-gcp-v3.2.0-kjgtm                 2/2      Running    0           7d
fluentd-gcp-v3.2.0-pwfd8                 2/2      Running    0           7d
heapster-v1.6.1-76cd74b8d5-d4twc         3/3      Running    0           7d
kube-dns-79868f54c5-12cqr                4/4      Running    0           7d
kube-dns-79868f54c5-znhk6                4/4      Running    0           7d
kube-dns-autoscaler-bb58c6784-bxbw4       1/1      Running    0           7d
kube-proxy-gke-istio-tutorial-default-pool-5fe93765-8zcb    1/1      Running    0           7d
kube-proxy-gke-istio-tutorial-default-pool-5fe93765-16sm    1/1      Running    0           7d
kube-proxy-gke-istio-tutorial-default-pool-5fe93765-ps9b    1/1      Running    0           7d
17-default-backend-fd59995cd-8rfnx        1/1      Running    0           7d
metrics-server-v0.3.1-57c75779f-2958b    2/2      Running    0           7d
prometheus-to-sd-5bjgm                    1/1      Running    0           7d
prometheus-to-sd-qfpn4                    1/1      Running    0           7d
prometheus-to-sd-vpx4v                    1/1      Running    0           7d
```

7.5. Service Mesh

A implantação de um ambiente de microserviços em uma arquitetura de cloud computing similar ao apresentado na Figura 1, não apresenta tanta dificuldade. A gestão deste ambiente a nível de rede, por conta da pequena quantidade de microserviços, pode ser feita pelo desenvolvedor.

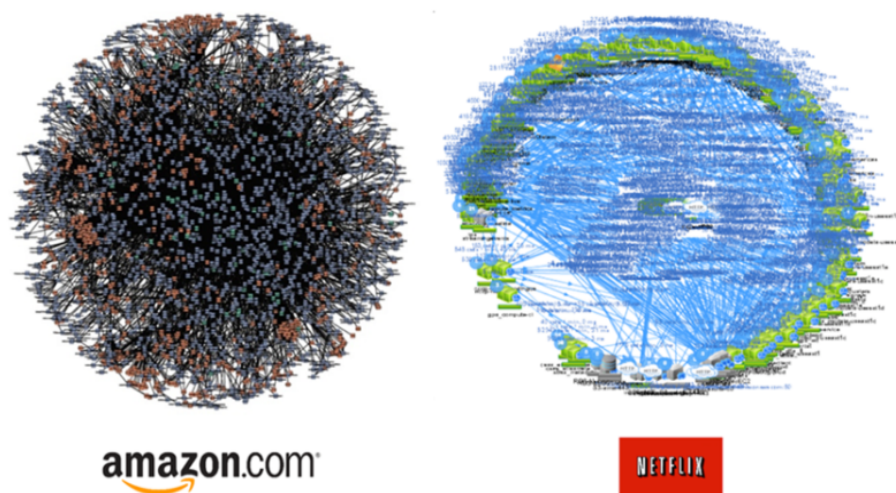
Por outro lado, em arquiteturas complexas, como aquelas apresentadas pela Amazon e pela Netflix com uma malha gigantesca de microserviços (Figura 11), se faz necessária uma infraestrutura com capacidade de prover a comunicação adequada entre os microserviços, tornando transparente para o desenvolvedor a gestão da integração entre eles.

A solução para este problema de comunicação entre os microserviços é o Service Mesh: uma camada de infraestrutura dedicada para lidar com a comunicação serviço a serviço. O Service Mesh é responsável pela entrega confiável de solicitações por meio da topologia complexa de serviços que inclui um aplicativo nativo da nuvem moderno (LI *et al.*, 2019b).

O Service Mesh proporciona serviços de rede com base em políticas para cargas de trabalho conectadas à rede, impondo o comportamento desejado da rede em face de condições e topologia em constante mudança. Estas mudanças podem ser referentes a

configurações, recursos e cargas de trabalho sendo implantadas. São construídas usando-se proxy de serviços, que interceptam de forma transparente a comunicação usando regras *iptables* no *namespace* do *Pod*.

Figura 11. Arquitetura Estrela da Morte.



Fonte: Appcentrica (2016).

A arquitetura de Service Mesh é um conjunto de *proxys* inteligentes e infraestrutura de controle adicional de componentes. Os *proxys* são implantados em todos os nós do seu *cluster*. Os *proxys* interceptam toda a comunicação entre os serviços e podem fazer muitos trabalhos em seu nome que anteriormente tinham que ser feitos pelo serviço (SAYFAN, 2019).

O Service Mesh torna-se um serviço complementar ao Kubernetes. Enquanto o Kubernetes é responsável principalmente pelo agendamento de *Pods* e pelo fornecimento da rede plana descoberta de modelo e serviço, para que diferentes *Pods* e serviços possam se comunicar um com o outro, o Service Mesh assume e gerencia a comunicação serviço a serviço de uma maneira muito mais refinada responsabilizando-se pelo balanceamento de carga e políticas da rede.

O diagrama apresentado na Figura 12 ilustra como o Service Mesh é incorporado em um cluster Kubernetes.

7.5.1. Istio

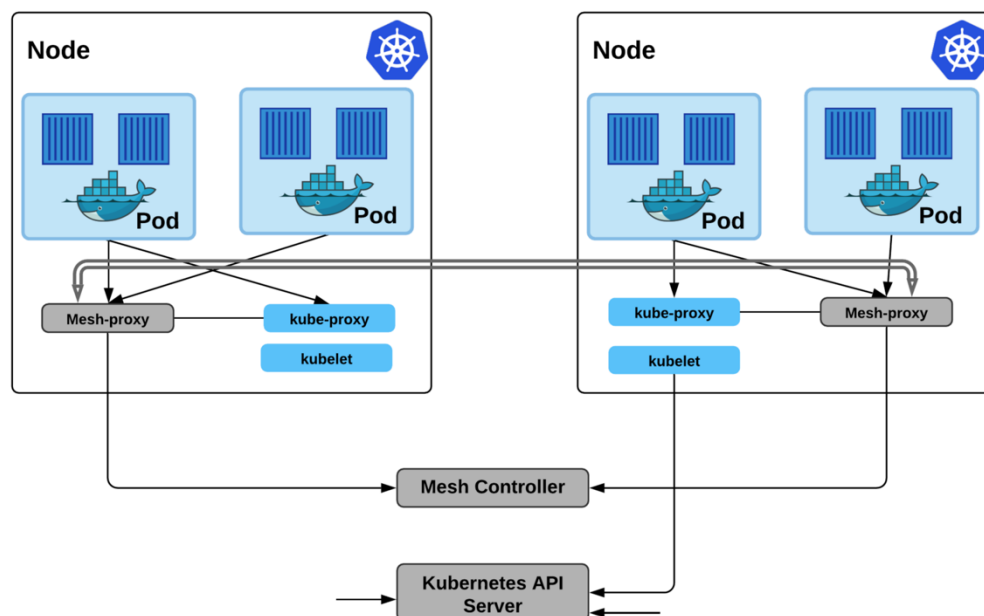
Istio é uma implementação de Service Mesh criada pelo Google, IBM e Lyft, por meio de um esforço colaborativo entre essas organizações (CALCOTE; BUTCHER, 2019); trata-se de uma solução de código fonte aberto e faz parte de projetos da Cloud Native Computing Foundation (CNFC).

Uma das principais vantagens do Istio é o suporte ao gerenciamento de fluxos de tráfego entre microsserviços, aplicando políticas de acesso e agregação de dados de telemetria, tudo sem exigir alterações no código de microsserviço (INDRASIRI *et al.*, 2018a).

O Istio fornece alguns recursos importantes de maneira uniforme para o Service Mesh, dentre eles podemos destacar (INDRASIRI *et al.*, 2018b):

- gestão do tráfego - Permitindo controlar facilmente o fluxo de tráfego e chamadas de API entre serviços, simplificando a configuração de propriedades de nível de serviço, como disjuntores, tempos limite e novas tentativas, facilitando a configuração de tarefas importantes, como testes A / B, lançamentos de canários e lançamentos em etapas com divisões de tráfego baseadas em porcentagem.
- segurança - o Istio fornece o canal de comunicação seguro subjacente e gerencia a autenticação, a autorização e a criptografia da comunicação de serviço em escala, as comunicações de serviço são protegidas por padrão, permitindo que se aplique políticas de maneira consistente em diversos protocolos e tempos de execução.
- observabilidade – disponibiliza recursos de rastreamento, monitoramento e registro, fornecendo informações detalhadas sobre a implantação do Service Mesh. Proporcionando um entendimento real de como o desempenho do serviço afeta a ambiente upstream e downstream com os recursos de monitoramento do Istio, enquanto seus painéis personalizados fornecem visibilidade do desempenho de todos os seus serviços e permitem que veja como esse desempenho está afetando outros processos.

Figura 12. Cluster com Service Mesh.



Fonte: Sayfan (2019).

O detalhamento de todos os componentes envolvidos na arquitetura do Istio será descrito a seguir:

- Pilot - responsável pela descoberta de serviços independente de plataforma, com balanceamento dinâmico e roteamento. Ele traduz regras de roteamento de alto nível e resiliência de sua própria API de regras em uma configuração do *Envoy*.

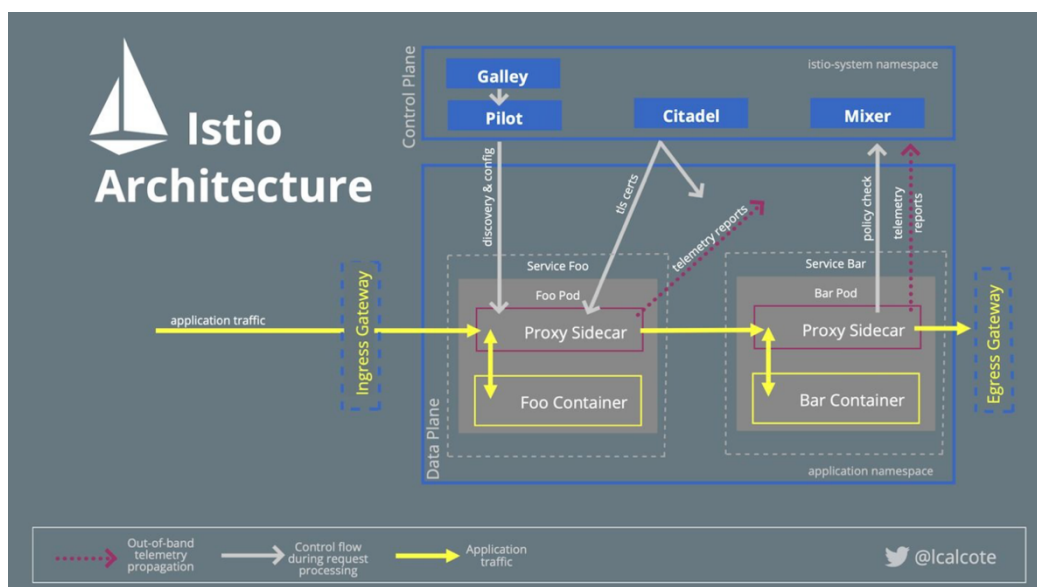
Essa camada de abstração permite que o Istio execute em múltiplas plataformas de orquestração.

- Mixer - é responsável por abstrair a coleção e as políticas de métricas. Esses aspectos geralmente são implementados nos serviços acessando APIs diretamente para *back-end* específicos. Isso tem o benefício de descarregar esses ônus dos serviços desenvolvedores e colocando o controle nas mãos dos operadores que configuram o Istio.
- Citadel - é responsável pelo gerenciamento de certificados e chaves no Istio. Integra com várias plataformas e se alinha com seus mecanismos de identidade. Por exemplo, no Kubernetes, ele usa contas de serviço como AWS e GCP.
- Galley - é um componente relativamente simples. Seu trabalho é abstrair a configuração do usuário em diferentes plataformas. Ele fornece a configuração ingerida para Pilot e Mixer.

A arquitetura do Istio e todos os seus componentes são apresentados na Figura

13.

Figura 13. Arquitetura Istio.



Fonte: Calcote (2019).

7.5.1.1. Implementação do Istio na GCP

Todos os conceitos e as arquiteturas percorridos de maneira construtiva e escalar ao longo do capítulo, formaram uma base teórica e prática para a solução do problema elencado, ao qual o fechamento deste arco se faz com a implementação de uma malha de serviço em cloud computing utilizando o Istio.

As configurações do Istio serão feitas no cluster Kubernetes já criado no GKE, conforme disposto no item 7.4.2.3.1.

O comando descrito a seguir deve ser executado para baixar e extrair o arquivo de instalação do Istio correspondente ao sistema operacional no qual se deseja executar o seu cliente:

```
curl -L https://git.io/getLatestIstio | sh -
```

Acesse o diretório de instalação do Istio para executar os comandos de configuração, a versão atual durante a construção deste capítulo é a 1.3.3. Portanto, o comando seria **cd istio-1.3.3**.

```
cd istio-VERSÃO
```

Adicione o `istioctl` como uma das várias do sistema operacional

```
export PATH=$PWD/bin:$PATH
```

Inicialmente é necessário criar um *namespace* para o ambiente do Istio. *namespace* é uma funcionalidade do Kubernetes que provê gerenciamento de seus recursos através de particionamento do cluster. Desta forma, é possível estabelecer um *namespace* para o ambiente de teste e outro de produção.

Execute o seguinte comando de criação do namespace:

```
kubectl create namespace istio-system
```

Para consultar os *namespaces* criados execute o comando **kubectl get namespaces**, observe que o *namespace istio-system* já é apresentado como criado.

```
MacBook-Pro-de-Luciano:~ lucianoaguiar$ kubectl get namespaces
NAME          STATUS    AGE
default       Active   7d
istio-system   Active   24s
kube-public    Active   7d
kube-system    Active   7d
```

O ambiente está pronto para realizar a instalação dos componentes do Istio dentro do *namespace* criado e habilitado a gerenciar os microsserviços a serem disponibilizados.

Defina os recursos personalizados que serão utilizados pelo ambiente a partir de um template utilizando o *helm* conforme comando a seguir:

```
helm template install/kubernetes/helm/istio-init \
--name istio-init --namespace istio-system | kubectl apply -f -
```

O Istio deste ambiente será instalado com um perfil padrão.

```
helm template install/kubernetes/helm/istio \
--name istio --namespace istio-system | kubectl apply -f -
```

Para verificar se os serviços do Istio foram instalados, execute o comando **kubectl get service -n istio-system**.

```
kubectl get service -n istio-system
```

```
MacBook-Pro-de-Luciano:~ lucianoaguiar$ kubectl get service -n istio-system
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
istio-citadel	ClusterIP	10.27.244.219	<none>	8060/TCP,15014/TCP	1m
istio-galley	ClusterIP	10.27.242.201	<none>	443/TCP,15014/TCP,9901/TCP	1m
istio-ingressgateway	LoadBalancer	10.27.246.78	34.74.164.214	15020:32283/TCP,80:31380/TCP,443:31390/TCP,31400:31400/TCP,15029:30922/TCP,15030:30096/TCP,15031:32189/TCP,15032:32114/TCP,15443:30938/TCP	1m
istio-pilot	ClusterIP	10.27.252.225	<none>	15010/TCP,15011/TCP,8080/TCP,15014/TCP	1m
istio-policy	ClusterIP	10.27.241.72	<none>	9091/TCP,15004/TCP,15014/TCP	1m
istio-sidecar-injector	ClusterIP	10.27.247.251	<none>	443/TCP,15014/TCP	1m
istio-telemetry	ClusterIP	10.27.241.205	<none>	9091/TCP,15004/TCP,15014/TCP,42422/TCP	1m
prometheus	ClusterIP	10.27.254.225	<none>	9090/TCP	1m

Podemos ainda consultar os Pods que foram criados do Istio com o comando **kubectl get pod -n istio-system**.

```
MacBook-Pro-de-Luciano:~ lucianoaguiar$ kubectl get pods -n istio-system
```

NAME	READY	STATUS	RESTARTS	AGE
istio-citadel-6cb95997f8-768kv	1/1	Running	0	7m
istio-galley-b877d99f4-52mtr	1/1	Running	0	7m
istio-ingressgateway-67fbf57b85-x26ln	1/1	Running	0	7m
istio-pilot-8687d5696-wtgw6	2/2	Running	0	7m
istio-policy-9f5d7d7b4-42k2w	2/2	Running	1	7m
istio-sidecar-injector-6c65cfff5-xrhj2	1/1	Running	0	7m
istio-telemetry-75cbc855b5-4kqrh	2/2	Running	2	7m
prometheus-5679cb4dcd-v9nbb	1/1	Running	0	7m

Desta forma, o *cluster* Kubernetes, implementado na Google Cloud Platform, está apto a receber qualquer aplicação com arquitetura de microsserviços. Conforme abordado anteriormente, os desenvolvedores irão se preocupar somente com a codificação do microsserviço, sendo que a comunicação entre eles ficaria sob a responsabilidade do Istio e a resiliência à cargo do Kubernetes.

7.5.1.2. Implantando uma Aplicação

Como fechamento deste capítulo e a título de ilustração, será apresentado um exemplo de implantação de uma aplicação no cluster Kubernetes criado com Istio. A aplicação é denominada Bookinfo, um aplicativo exemplo de livreria disponibilizado junto com o Istio, composto por quatro microsserviços separados, desenvolvidos utilizando-se Python, Java, Node JS e Ruby, conforme ilustrado na Figura 14. Maiores detalhes sobre a arquitetura da aplicação podem ser encontrados no endereço <https://istio.io/docs/examples/bookinfo/>.

Para atualizar o *cluster* criado com a aplicação Bookinfo, devemos usar o comando **kubectl apply**. O arquivo manifesto da aplicação encontra-se dentro do diretório **sample**, foi feito o seu download junto com o Istio. Para instalar a aplicação execute o comando a seguir:

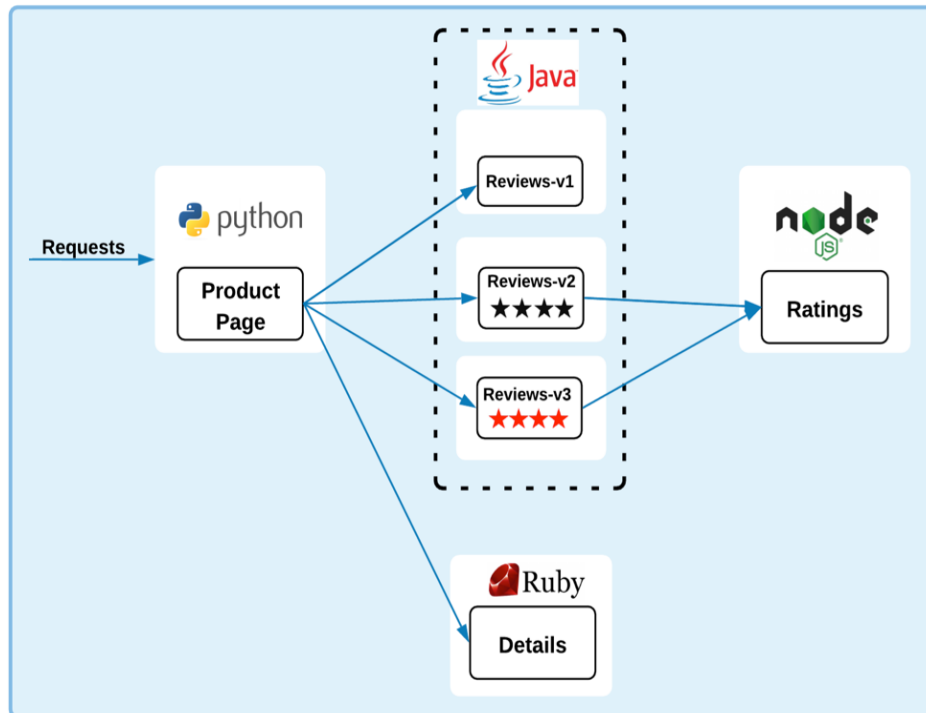
```
kubectl apply -f <(istioctl kube-inject -f samples/bookinfo/platform/kube/bookinfo.yaml)
```

Confirme se a aplicação foi implantada no *cluster* corretamente e se seus serviços estão em execução, com o comando **kubectl get services**:

```
MacBook-Pro-de-Luciano:~ lucianoaguiar$ kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
details	ClusterIP	10.27.250.105	<none>	9080/TCP	37s
kubernetes	ClusterIP	10.27.240.1	<none>	443/TCP	7d
productpage	ClusterIP	10.27.242.38	<none>	9080/TCP	36s
ratings	ClusterIP	10.27.250.166	<none>	9080/TCP	36s
reviews	ClusterIP	10.27.255.199	<none>	9080/TCP	36s

Figura 14. Aplicação Bookinfo.



Fonte: Istio (2019).

Verifique ainda se todos os Pods da aplicação foram criados corretamente. Observar com atenção especial a coluna STATUS, na qual todos os Pods devem estar em modo Running, execute o comando **kubectl get pods**:

```
MacBook-Pro-de-Luciano:~ lucianoaguair$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
details-v1-576b9c9d5f-d45tx         2/2     Running   0           1m
productpage-v1-75f797845f-cnkt       2/2     Running   0           1m
ratings-v1-5cb96d77cf-75z9f         2/2     Running   0           1m
reviews-v1-88c77b6d7-265b8          2/2     Running   0           1m
reviews-v2-6dc48dcc74-v45k4         2/2     Running   0           1m
reviews-v3-56f9b47cfc-nq687        2/2     Running   0           1m
```

Para que a aplicação receba requisições externas da Internet, deve-se definir o roteamento do gateway de entrada, conforme especificado a seguir:

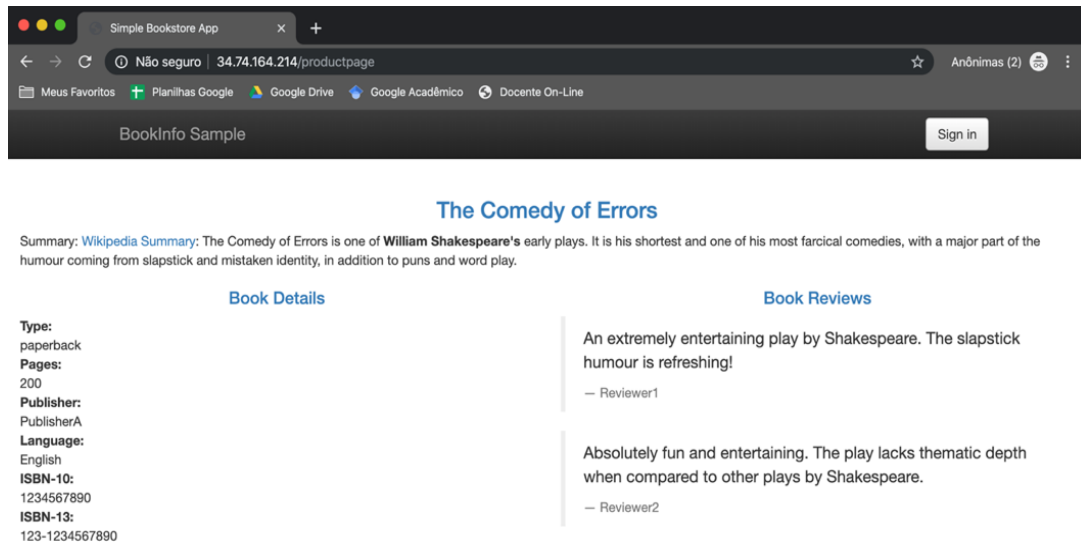
```
kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml
```

O teste do aplicativo pode ser feito na Internet. Para isso, basta enviar a requisição para o IP Externo do Gateway, com o comando **kubectl get svc istio-ingressgateway -n istio-system** para obter o endereço:

```
MacBook-Pro-de-Luciano:~ lucianoaguair$ kubectl get svc istio-ingressgateway -n istio-system
NAME                                TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)                                     AGE
istio-ingressgateway               LoadBalancer   10.27.246.78  34.74.164.214  15020:32283/TCP,80:31380/TCP,443:31390/TCP,31400:31400/TCP,15029:30922/TCP,15030:30096/TCP,15031:32189/TCP,15032:32114/TCP,15443:30938/TCP  21m
```

Em um navegador, digite o endereço do `http://IP_Externo/productpage`. Caso esteja tudo correto, na configuração do *cluster* e na implantação, a aplicação Bookinfo é exibida (Figura 15).

Figura 15. Teste da Aplicação Bookinfo



Fonte: dos autores.

7.6. Conclusões

A escalabilidade do alcance da Internet, bem como a sua velocidade e aplicações nativas de nuvem, proporcionaram, tanto às empresas de tecnologia quanto aos usuários finais, novos parâmetros de qualidade e exigências.

Ao longo deste capítulo, foram abordadas tecnologias nativas de computação em nuvem que visam atender a essas novas exigências na qualidade do software. Foi apresentado o padrão arquitetural de desenvolvimento baseado em **microserviços** e a sua diferença em relação ao modelo monolítico. Adicionalmente, foram apresentados os **containers** como abstrações de sistemas operacionais que possuem como grande vantagem o empacotamento de aplicações e a sua execução em qualquer ambiente, independente do sistema operacional hospedeiro. Em seguida, foi apresentado o **Kubernetes**, na qual a abordagem principal do capítulo foi descrever a sua arquitetura e seu funcionamento, além de elencar alguns modelos de implantação. Essa descrição permitiu detalhar as diferenças entre o Kubernetes e o Docker, constatando-se, assim, que o Kubernetes proporciona adicionalmente, em relação ao Docker, resiliência e saúde do ambiente.

Para atingir a completude arquitetural do modelo de implantação de software nativo de nuvem, foi discutida a tecnologia **Service Mesh**, apresentando sua arquitetura, modelo de funcionamento e implantação com o software Istio em um cluster Kubernetes da Google. Pode-se compreender que a principal vantagem neste ambiente é a segurança, gestão de tráfego e observação dos microserviços.

O modelo arquitetural de implantação de software nativo de nuvem, utilizando container Docker, Kubernetes, Service Mesh e microserviços apresentado no capítulo constitui uma tendência de adoção, considerando a série de vantagens agregadas aos ambientes apresentados ao longo deste capítulo, bem como o fato de serem tecnologias chanceladas e já testadas por grandes empresas, como Google, Netflix, Nubank, Amazon e outras.

Por fim, é importante ressaltar que as unidades abordadas ao longo do capítulo não atingiram a completude do assunto, dada sua vasta quantidade de informações, mas serviram para contextualizar conceitos, arquiteturas e compreender o modelo. Para aprofundamento no assunto, os livros usados como referência para este capítulo são fortemente recomendados, especialmente os títulos “DevOps nativo de nuvem com Kubernetes” e “Istio: Up and Running”.

7.7.Referências

AMUNDSEN, M. *et al.* **Microservice Architecture**. 1. ed. Estados Unidos da América: O'Reilly Media, 2016.

APPCENTRICA. **The Rise of Microservices**. Disponível em: <<https://www.appcentrica.com/the-rise-of-microservices/>>. Acesso em: 13 out. 2018.

BISONG, E.; LEARNING, B. M. **Containers and Google Kubernetes Engine**. p. 655–670, 2019.

BUI, T. **Analysis of Docker Security**. 2015.

CALCOTE, L.; BUTCHER, Z. **Istio: Up and Running**. 1. ed. California: O'Reilly Media, 2019.

CARTER, E. **Docker Usage Report** 2018. Disponível em: <<https://sysdig.com/blog/2018-docker-usage-report/>>. Acesso em: 6 out. 2019.

CASALICCHIO, E. **Container Orchestration: A Survey**. n. Vm, p. 221–235, 2019.

CITO, J. *et al.* **An Empirical Analysis of the Docker Container Ecosystem on GitHub**. IEEE International Working Conference on Mining Software Repositories, p. 323–333, 2017.

CITO, J. *et al.* **An Empirical Analysis of the Docker Container Ecosystem on GitHub**. IEEE International Working Conference on Mining Software Repositories, p. 323–333, 2017.

DOCKER. **Docker Architecture**. Disponível em: <<https://docker.com>> Acesso em 10 de outubro, 2019.

DRAGONI, N. *et al.* **Microservices: Yesterday, Today, and Tomorrow**. In: Present and Ulterior Software Engineering. Springer, Cham, 2017, p. 195-216.

HIGHTOWER, K.; BURNS, B.; BEDA, J. **Kubernetes: Up and Running: Dive into the Future of Infrastructure**. [s.l.: s.n.]. Disponível em: <<http://oreilly.com/safari>>.

INDRASIRI, K. *et al.* **Developing Services**. In: Microservices for the Enterprise. [s.l.] Apress, 2018a. p. 89–123.

INDRASIRI, K. *et al.* **Service Mesh**. In: **Microservices for the Enterprise**. [s.l.] Apress, 2018b. p. 263–292.

INDRASIRI, K.; SIRIWARDENA, P. **Microservices for the Enterprise**. [s.l: s.n.].

INDRASIRI, K.; SIRIWARDENA, P. **Microservice for the enterprise – Designing, Developing and Deploying**. 1. ed. Estados Unidos da América: Apress, 2018.

ISTIO. Bookinfo Application. Disponível em: <
<https://istio.io/docs/examples/bookinfo/>> Acesso em: 15 de outubro, 2019.

LI, W. et al. **Service Mesh: Challenges, state of the art, and future research opportunities**. Proceedings - 13th IEEE International Conference on Service-Oriented System Engineering, SOSE 2019, 10th International Workshop on Joint Cloud Computing, JCC 2019 and 2019 IEEE International Workshop on Cloud Computing in Robotic Systems, CCRS 2019, p. 122–127, 2019a.

LI, W. et al. **Service Mesh: Challenges, state of the art, and future research opportunities**. Proceedings - 13th IEEE International Conference on Service-Oriented System Engineering, SOSE 2019, 10th International Workshop on Joint Cloud Computing, JCC 2019 and 2019 IEEE International Workshop on Cloud Computing in Robotic Systems, CCRS 2019. Anais...Institute of Electrical and Electronics Engineers Inc., 3 maio 2019b

MCKENDRICK, R.; GALLAGHER, S. **Mastering Docker**. 3. ed. Birmingham: [s.n.].

MOORE, J. D. **Kubernetes: The Complete Guide**. 2. ed. [s.l: s.n.].

NEWMAN, S. **Building Microservices**. 1. ed. Estados Unidos da América: O'Reilly Media, 2015.

POULTON, N. **The Kubernetes Book**, Jan 2018. 2.2 ed. [s.l: s.n.].

SALAH, T. *et al.* **Performance Comparison between Container-based and VM-based Services**. v. 2, p. 185–190, 2017.

SAYFAN, G. **Hands-On Microservices with Kubernetes: Build, deploy, and manage scalable microservices on Kubernetes**. [s.l: s.n.].

SAYFAN, G. **Mastering Kubernetes Automating Container deployment and management**. [s.l.] Packt Publishing, 2017.

THAMES, W. *et al.* **DevOps nativo de nuvem com Kubernetes**. 1. ed. São Paulo: Novatec, 2019.

TOSATTO, A.; RUIU, P.; ATTANASIO, A. **Container-based orchestration in cloud : state of the art and challenges**. 2015.

VERAS, M. **Arquitetura de Nuvem: Amazon Web Services (AWS)**. 1. ed. Rio de Janeiro: Brasport, 2013.

Capítulo

8

Introdução à Análise Exploratória de Dados com Python

Gesiel Rios Lopes, Alessandro Wilk Silva Almeida,
Alexandre C. B. Delbem, Cláudio Fabiano Motta Toledo

Resumo

O avanço da tecnologia, observados nos últimos anos, aliado com a popularização da Internet e com o aumento na quantidade e complexidade dos serviços oferecidos por ela, contribuíram de forma significativa à geração massiva de dados. A manipulação e a análise de forma inteligente desse volume de dados têm se tornado um dos grandes desafios computacionais da atualidade. Para este fim, a Análise Exploratória de Dados (AED) é bem adequada, uma vez que é bem conhecida em estatística e ciências. A abordagem operacional da análise de dados visava melhorar a compreensão e a acessibilidade dos resultados. Sem esquecer a solidez dos modelos estatísticos e a formulação de hipóteses, que está intrinsecamente ligada ao conceito de “análise” em seu significado científico, o foco é transferido para a “exploração”. A AED se relaciona com o processo de revelar informações ocultas e desconhecidas dos dados de tal forma que o analista obtém uma representação imediata, direta e fácil de entender. Gráficos visuais são um elemento obrigatório desta abordagem, devido à capacidade intrínseca do cérebro humano de obter uma interpretação mais direta e confiável de similaridades, diferenças, tendências, clusters e correlações através de uma imagem, ao invés de uma série de números. Neste minicurso, será apresentado conceitos sobre como utilizar a linguagem de programação Python para explorar um conjunto de dados, o que é essencial para obter uma boa compreensão e possíveis problemas de um conjunto de dados, além de auxiliar na geração de hipóteses que podem ser extraídas a partir da análise de um conjunto de dados.

8.1. Introdução à análise exploratória de dados

A Análise Exploratória de Dados (AED) teve seu início com J.W. Tukey [Tukey 1977] e visa aumentar o conhecimento do pesquisador sobre uma população a partir de uma amostra. Dessa forma, podemos descrever a AED como um conjunto de métodos adequados para a coleta, a exploração e descrição e interpretação de conjuntos de dados numéricos.

Estes métodos permitem a exploração dos dados com intuito de identificar padrões de interesse e a representação dos dados caracterizados por estes padrões.

Embora as técnicas da AED sejam simples, geralmente são métodos robustos (válidos para uma grande gama de situações e modelos) e resistentes (insensíveis aos erros grosseiros ou dados estranhos).

Não é exagero reafirmar a citação de TUKEY [Tukey 1977]:

“A AED é trabalho de detetive, procurando pistas e evidências; análise confirmatória de dados é trabalho judicial ou quase-judicial, que analisa, avalia e julga as provas e as evidências.”

8.2. Por que usar python para análise exploratória de dados ?

Python é uma linguagem de programação muito utilizada em atividades de análise de dados e ao contrário de linguagens como R, Python é uma linguagem de programação de propósito geral, ou seja, não é exclusiva para atividades de análise de dados. Esta característica tem contribuído de forma significativa para aumentar o seu uso dentro das empresas, uma vez que muitas equipes de desenvolvimento de *software* já possuem um certo conhecimento com Python, o que facilita a implantação em produção de modelos desenvolvidos na mesma plataforma.

No desenvolvimento de aplicações científicas é recomendável o utilizar a distribuição Anaconda¹, uma vez que ela é de código aberto e é a maneira mais fácil de executar aplicações científicas desenvolvidas em Python no Linux, Windows e Mac OS X. Dentro da distribuição Anaconda você também encontrará o Jupyter Notebook, uma interface muito interessante para criar seus modelos e compartilhar com quem quiser.

Uma alternativa similar ao Jupyter Notebook e não requer configuração para ser usada é uma ferramenta desenvolvida pelo Google chamada Colaboratory². Para usar este ambiente, é necessário apenas ter uma conta gmail, pois todo notebook ficará armazenado no Drive.

O código do seu notebook é executado em uma máquina virtual dedicada à sua conta. As máquinas virtuais são recicladas após um determinado tempo ocioso, ou caso a janela seja fechada. Para restaurar seu notebook, talvez seja necessário refazer o *upload* do arquivo .csv e executar as opções “Runtime” e “Restart and run all”.

8.3. Fundamentos de python para análise exploratória de dados

As características mais relevantes do Python são a sua capacidade de integração com outras linguagens e seu sistema de bibliotecas bem maduro. As bibliotecas apresentadas a seguir são fortemente analíticas e oferecerão uma completa caixa de ferramentas de ciência de dados composta de funções altamente otimizadas para trabalhar, configuração ideal de memória, pronta para realizar operações de *script* com desempenho ideal [Boschetti and Massaron 2015].

Parcialmente inspirados por ferramentas semelhantes presentes nos ambientes R

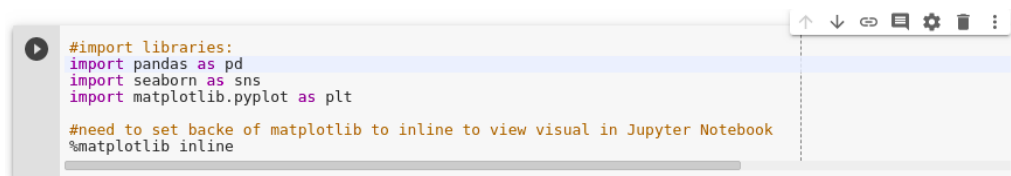
¹Disponível em: <https://www.anaconda.com/distribution/>

²Disponível em: <https://colab.research.google.com/>

e *MATLAB*, serão exploradas juntamente com alguns comandos Python, de forma a permitir que se lide com dados com eficiência e depois explore, transforme, experimente e aprenda com os mesmos sem precisar escrever muito código ou reinventar a roda.

- **Pandas:** Pandas³ é uma biblioteca licenciada com código aberto que oferece estruturas de dados de alto desempenho e de fácil utilização voltado a análise de dados para a linguagem de programação Python [Coelho 2017].
- **Seaborn:** Seaborn⁴ é uma biblioteca de visualização de dados Python baseada no *matplotlib*. Ela fornece uma interface de alto nível para desenhar gráficos estatísticos atraentes e informativos.
- **Matplotlib:** O Matplotlib⁵ é uma biblioteca de plotagem 2D do Python que produz números de qualidade de publicação em vários formatos de cópia impressa e ambientes interativos entre plataformas. O Matplotlib pode ser usado em *scripts* Python, nos *shell* Python e IPython, *notebooks* Jupyter e em servidores *web*.

A Figura 8.1 é apresentado a declaração das bibliotecas que serão necessárias para AED.



```
#import libraries:
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#need to set backe of matplotlib to inline to view visual in Jupyter Notebook
%matplotlib inline
```

Figura 8.1. Bibliotecas para AED.

Para se trabalhar com Pandas e o Seaborn, é necessário estar familiarizado com duas estruturas de dados que são: *Séries* e *Dataframes*, descritos resumidamente a seguir.

8.3.1. Séries

A estrutura *Série*, é um array unidimensional, semelhante a uma lista em Python, no entanto criado sobre o *numpy*. Além da velocidade de processamento, a principal característica que o difere de uma lista comum é que seus índices podem ser mutáveis.

A Figura 8.2 apresenta a criação e manipulação de uma série. A primeira linha define a série *my_serie* com 5 elementos, a segunda linha apresenta a série *my_serie*, a terceira linha apresenta o valor armazenado no índice 2 e a quarta linha altera os índices da série pelas letras A,B,C,D,E e logo em seguida a série é impressa novamente com os novos índices.

³Disponível em <https://pandas.pydata.org/>

⁴Disponível em <https://seaborn.pydata.org/>

⁵Disponível em <https://matplotlib.org/>



```
my_series = pd.Series([10,20,30,40,50])
print(my_series)
print(my_series[2])
my_series.index = ['A', 'B', 'C', 'D', 'E']
print(my_series)
```

0 10
1 20
2 30
3 40
4 50
dtype: int64

A 10
B 20
C 30
D 40
E 50
dtype: int64

Figura 8.2. Criação e manipulação de uma série.

8.3.2. Dataframes

Dataframe é uma estrutura de dados tabular bidimensional e mutável em tamanho, potencialmente heterogênea, com eixos rotulados (linhas e colunas). Para se criar um Dataframe a partir das estruturas de dados nativas em Python, pode-se passar um dicionário de listas para seu construtor. Usando-se o parâmetro *columns*, define no construtor como as colunas serão ordenadas. Por padrão, o construtor do Dataframe irá ordenar as colunas em ordem alfabética.



```
# initialise data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'],
        'Age':[20, 21, 19, 18]}

# Create DataFrame
df = pd.DataFrame(data)

# Print the output.
df
```

	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

Figura 8.3. Criação um dataframe.

A Figura 8.3 ilustra a criação e apresentação de um Dataframe contendo informações referentes à idade de quatro pessoas fictícias. A primeira coluna da Figura 8.3 apresenta os índices padrões.

Em geral, é muito mais comum a fazer a leitura de uma base de dados a partir de uma base externa, o Pandas oferece alguns métodos com essa finalidade, dentre os quais se destaca a função *read_csv* para leituras de arquivos CSV e o *pydataset* que fornece acesso instantâneo a muitos conjuntos de dados diretamente do Python, a versão atual conta com 757 datasets prontos para serem importados e utilizados juntamente com o Pandas. A Figura 8.4 ilustra a criação de um dataframe a partir de um arquivo CSV. Na primeira linha é possível observar a utilização a função *read_csv* e na segunda linha temos

o uso da função *head* com fornece os cinco primeiros registros de um Dataframe.



```
data = pd.read_csv("nba.csv")
data.head()
```

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0

Figura 8.4. Criação um dataframe a partir de um arquivo CSV.

Séries e Dataframes formam o núcleo dos modelos de dados do Pandas em Python. Os conjuntos de dados são lidos primeiro nos Dataframes e, em seguida, várias operações (por exemplo, *groupby*, agregações, etc.) podem ser aplicadas muito facilmente às suas colunas. A Tabela 8.1 apresenta um pequeno resumo de alguns comandos úteis para manipulação de informações em um Dataframe.

Tabela 8.1. Comandos mais comuns em um Dataframe.

Função	Retorno
<i>df.shape()</i>	Quantidade de linhas e colunas do Dataframe
<i>df.head()</i>	Primeiros 5 registros do Dataframe
<i>df.tail()</i>	Últimos 5 registros do Dataframe
<i>df.columns()</i>	Colunas presentes no Dataframe
<i>df.count()</i>	Contagem de dados não-nulos
<i>df.sum()</i>	Soma dos valores de um Dataframe
<i>df.min()</i>	Menor valor de um Dataframe
<i>df.max()</i>	Maior valor de um Dataframe
<i>df.describe()</i>	Resumo estatístico do Dataframe

Para demonstrar o processo exploratório de análise de dados, além de fornecer um exemplo para programadores Python que desejam praticar o trabalho com dados, será utilizado dados da Pesquisa Nacional por Amostra de Domicílio (PNAD) ⁶, valores de ações disponibilizados pela MetaTrader ⁷ e os *datasets* disponibilizados pela função *load_dataset()* do *seaborn*, cuja descrição dos *datasets* pode ser encontrada em: <https://github.com/mwaskom/seaborn-data>.

Explorar dados por meio de visualizações bem construídas e estatísticas descritivas é uma ótima maneira de se familiarizar com os dados com os quais você está trabalhando e formular hipóteses com base em suas observações.

8.4. Medidas de tendência central

As medidas de tendência central ou “de posição” procuram caracterizar a tendência do conjunto (valor “típico”) a um valor numérico que “represente” o conjunto. Esse valor

⁶Disponível em: <https://www.ibge.gov.br/>

⁷Disponível em: <https://www.metatrader5.com/>

pode ser calculado levando em conta todos os valores do conjunto ou apenas alguns valores ordenados [Tukey 1977, Stevenson and De Farias 1981].

8.4.1. Média \bar{x}

A média aritmética simples é a soma dos valores observados dividida pelo número desses valores. A média de uma amostra é representada pelo símbolo \bar{x} (leia-se “*x barra*”), definida pela Equação 1

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n} = \frac{\sum_{i=1}^n x_i}{n} \quad (1)$$

A Figura 8.5 apresenta uma das formas de se calcular a média de um conjunto de dados. A função do pandas para cálculo da média de um conjunto de valores é apresentado na Figura 8.6



```
media_idade = pnad_df['idade'].sum() / len(pnad_df['idade'])
print(media_idade)
```

34.2846

Figura 8.5. Uma forma de calcular a média de um conjunto de dados.




```
pnad_df['idade'].mean()
```

34.2846

Figura 8.6. Cálculo da média de um conjunto de dados através da função `mean()` do pandas.

8.4.2. Mediana M_e

A mediana é o valor médio ou a média aritmética de dois valores centrais de um conjunto de números ordenados (crescente ou decrescente). Para calculá-la, primeiramente temos de reorganizar os dados em ordem crescente (ou decrescente) e, em seguida, escolher o valor central. Se o número de dados for ímpar, então este valor central é único; se for par, fazemos a média dos dois valores centrais. O pandas disponibiliza a função `median()` para determinar a mediana de um conjunto de dados, como pode ser observado da Figura 8.7.



```
pnad_df['rendimento'].median()
```

1200.0

Figura 8.7. Cálculo da mediana de um conjunto de dados através da função `median()` do pandas.

8.4.3. Moda M_o

A moda é o valor que ocorre com maior frequência em um dado conjunto de dados. Se todos os valores aparecem um número igual de vezes (em geral, uma vez cada), dizemos que o conjunto de dados não tem moda, ou seja, a moda pode, muitas vezes, não existir. Uma outra particularidade importante da moda é que ela é a única. O pandas disponibiliza a função `mode()` para determinar a moda de um conjunto de dados, como pode ser observado da Figura 8.8.



Figura 8.8. Cálculo da moda de um conjunto de dados através da função `mode()` do pandas.

8.4.4. Outras médias

Algumas vezes é interessante calcular médias de forma diferente para dados com características especiais.

8.4.4.1. Média geométrica G

As médias geométricas são bastante empregadas para observações positivas referentes a crescimentos exponenciais (como taxas de avanço de doenças, números de habitantes de regiões em colonização, crescimento de produtividade, e etc...) [Stevenson and De Farias 1981]. A média geométrica (G) é a raiz n -ésima do produto dos n valores de um conjunto de dados (x_1, x_2, \dots, x_n) , conforme definido na Equação 2.

$$G = \sqrt[n]{x_1 \times x_2 \times \dots \times x_n} = \sqrt[n]{\prod_{i=1}^n x_i} \quad (2)$$

8.4.4.2. Média harmônica H

Para fenômenos que dependem fortemente do menor dos dados, em geral, utiliza-se médias harmônicas calculadas como o inverso da média dos inversos de um conjunto de dados (x_1, x_2, \dots, x_n) , definida conforme a Equação 3.

$$H = \frac{1}{\frac{1}{x_1} + \frac{1}{x_2} + \dots + \frac{1}{x_n}} = \frac{n}{\sum_{i=1}^n \frac{1}{x_i}} \quad (3)$$

Para o cálculo das médias geométricas e harmônicas utilizaremos as funções `gmean()` e `hmean()`, respectivamente, do módulo `scipy.stats` da biblioteca `SciPy` [Jones et al. 2001]. O módulo `scipy.stats` contém um grande número de distribuições de probabilidade, bem como uma crescente biblioteca de funções estatísticas.

A Figura 8.9 apresenta o cálculo das médias geométrica e harmônicas.



```
from scipy import stats

var = {'t1': [2, 6, 15, 59], 't2': [2000, 20, 250, 1500]}
other_means_df = pd.DataFrame(var)
|
other_means_df.head()

geometric_mean = stats.gmean(other_means_df['t1'], axis=0)
print(geometric_mean)

harmonic_mean = stats.hmean(other_means_df['t2'], axis=0)
print(harmonic_mean)
```

10.151521276336176
72.50755287009062

Figura 8.9. Cálculo das médias geométricas e harmônicas através do módulo *scipy.stats* da biblioteca *SciPy*.

8.5. Medidas de dispersão ou variabilidade

As medidas de dispersão ou variabilidade indicam se os valores de conjunto de dados estão relativamente próximos uns dos outros, ou separados. Todas elas têm na média o ponto de referência [Tukey 1977, Stevenson and De Farias 1981].

As medidas de dispersão oferecem as condições necessárias para analisar até que ponto os valores de um conjunto de dados oscilam para mais ou para menos em relação a uma medida de posição fixada, em geral a média [Stevenson and De Farias 1981].

8.5.1. Amplitude

A amplitude ou intervalo total (I_t) de um conjunto de dados (x_1, x_2, \dots, x_n) é a diferença entre o maior valor e o menor valor, conforme definido pela Equação 4

$$I_t = x_{\max} - x_{\min}, \quad (4)$$

onde:

x_{\max} = valor máximo do conjunto de dados, isto é, $\max\{x_1, x_2, \dots, x_n\}$;

x_{\min} = valor mínimo do conjunto de dados, isto é, $\min\{x_1, x_2, \dots, x_n\}$.

A Figura 8.10 apresenta um exemplo do cálculo da amplitude.



```
amplitude = pnad_df['rendimento'].max() - pnad_df['rendimento'].min()
print(amplitude)
```

99985.0

Figura 8.10. Cálculo da amplitude.

8.5.2. Variância S^2

A variância de uma amostra de valores (dados não agrupados), x_1, x_2, \dots, x_n , é definida como sendo a média dos quadrados dos desvios das medidas em relação à sua média \bar{x} . A variância da amostrada é dada pela Equação 5.

$$S^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n} \quad (5)$$

A Figura 8.11 apresenta o cálculo da variância.



```
pnad_df['rendimento'].var()
```

```
7722933.596037565
```

Figura 8.11. Cálculo da variância.

8.5.3. Desvio padrão

O desvio padrão é definido como a raiz quadrada da média aritmética dos quadrados dos desvios em relação à média. É a mais importante medida de variabilidade. O desvio padrão é dado pela Equação 6.

$$S = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n}} \quad (6)$$

A Figura 8.12 apresenta o cálculo do desvio padrão.

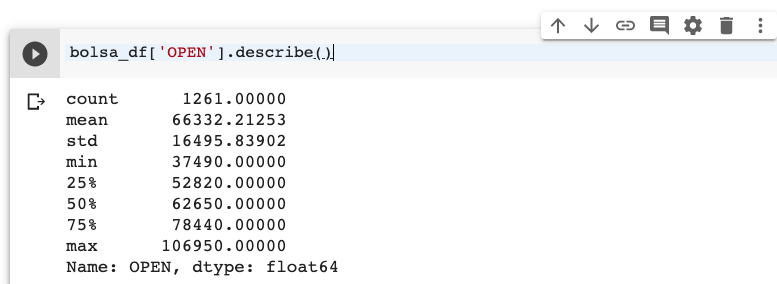


```
pnad_df['rendimento'].std()
```

```
2779.0166599064432
```

Figura 8.12. Cálculo do desvio padrão.

O *pandas* disponibiliza a função *describe()* que gera as estatísticas descritivas que resumem a tendência central, a dispersão e a forma da distribuição de um conjunto de dados, excluindo os valores de NaN. Analisa séries numéricas e de objetos, bem como conjuntos de colunas DataFrame de tipos de dados mistos. A Figura 8.13 é apresentado o resultado da função *describe()*.



```
bolsa_df['OPEN'].describe()
```

```
count      1261.000000
mean       66332.21253
std        16495.83902
min         37490.00000
25%         52820.00000
50%         62650.00000
75%         78440.00000
max        106950.00000
Name: OPEN, dtype: float64
```

Figura 8.13. Resultado da função *describe()* disponibilizada pelo *pandas*.

A Tabela 8.2 destaca as principais funções para estatísticas básicas disponíveis no Pandas.

Tabela 8.2. Funções estatísticas básicas do Pandas.

Função	Retorno
<i>mean()</i>	Média
<i>median()</i>	Mediana
<i>mode()</i>	Moda
<i>var()</i>	Variância
<i>std()</i>	Desvio padrão
<i>describe()</i>	Resumo Estatístico

8.6. Visualização gráfica de dados

A apresentação dos dados estatísticos através de tabelas ou medidas de centralidade e variabilidade nem sempre proporciona um entendimento adequado dos dados. Assim, com a finalidade de melhorar esse processo, muitos recorrem ao uso dos gráficos. Para isso, é necessário saber o que se pretende mostrar, como elaborar o gráfico e qual o tipo de gráfico mais apropriado para cada tema abordado. Dependendo da mídia em que o gráfico será levado a público, é importante considerar aspectos técnicos da sua elaboração como: o uso de cores e texturas, a espessura das linhas e as fontes dos textos na composição das legendas [Tukey 1977, Stevenson and De Farias 1981]. Nesta seção serão apresentados os gráficos mais utilizados dentro da análise de dados.

8.6.1. Histogramas

O histograma representa a distribuição dos dados, formando compartimentos ao longo do intervalo dos dados, desenhados por barras para mostrar a frequência de observações dos dados em cada compartimento [Vigni et al. 2013, Stevenson and De Farias 1981].

O *seaborn* disponibiliza a função *distplot()* para “plotar” histogramas. Essa função combina a função *hist* do *matplotlib* com as funções *kdeplot()* e *rugplot()* do próprio *seaborn* [Haslwanter 2016]. A Figura 8.14 apresenta um exemplo de um histograma a partir da função *distplot()* do *seaborn*.

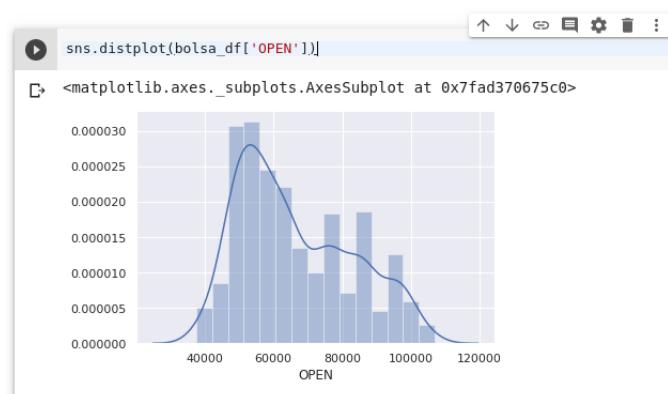


Figura 8.14. Histograma.

8.6.2. Gráficos de barra e linhas

No gráfico de barras, cada categoria é representada por uma barra de comprimento proporcional à sua frequência, conforme identificação no eixo horizontal.

O *seaborn* disponibiliza a função `catplot()` para “plotar” gráficos de barra. Esta função fornece acesso a várias funções no nível de eixos que mostram o relacionamento entre uma série numérica de uma ou mais variáveis categóricas, usando uma das várias representações visuais. O parâmetro `kind` seleciona a função de nível de eixo subjacente a ser usada [Haslwanter 2016]. As Figuras 8.15(a) e 8.15(b) apresentam um exemplo de um gráfico de barra a partir da função `catplot()` do *seaborn*.

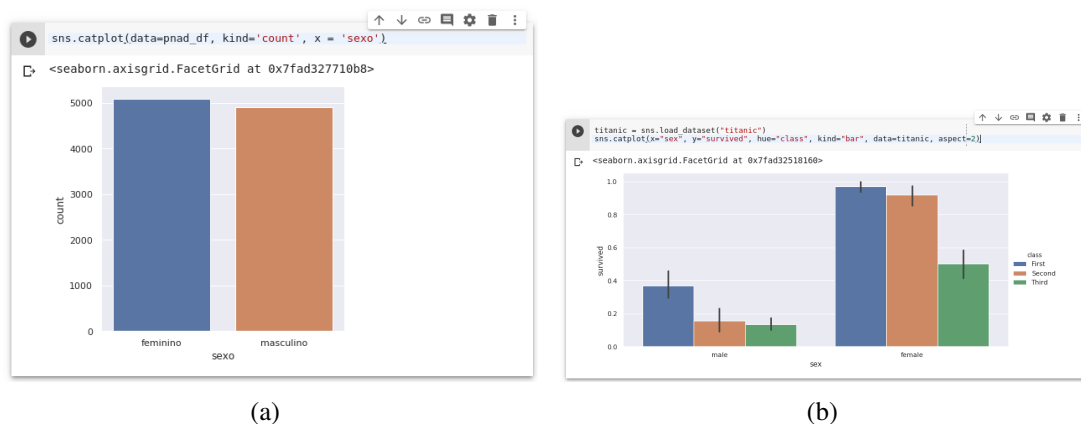


Figura 8.15. Gráficos de barra.

Para “plotar” gráficos de linha, o *seaborn* disponibiliza a função `relplot()`. Essa função fornece acesso a várias funções diferentes no nível de eixos que mostram o relacionamento entre duas variáveis com mapeamentos semânticos de subconjuntos. O parâmetro `kind` seleciona a função de nível de eixo subjacente a ser usada [Haslwanter 2016].

A Figura 8.16 apresenta um exemplo de um gráfico de linha a partir da função `relplot()` do *seaborn*.



Figura 8.16. Gráfico de linha.

8.6.3. Gráficos de setores

Este gráfico é constituído com base em um círculo, e é empregado sempre que desejamos ressaltar a participação do dado em relação ao total. O total é representado pelo círculo, que fica dividido em tantos setores quantas são as partes. Os setores são tais que suas áreas são proporcionais aos dados. Cada setor é obtido por meio de uma regra de três simples e direta, lembrando que o total corresponde a 360° [Jelihovschi 2014, Stevenson and De Farias 1981].

Existem algumas recomendações devem ser seguidas ao se elaborar gráficos de setores. Os valores devem ser apresentados em ordem decrescente a partir da parte superior do gráfico e no sentido horário. Ao lado de cada setor, podem-se colocar os percentuais e os nomes de cada parcela. Não é recomendado a utilização deste gráfico usado quando há muitas parcelas e nem quando existem muitas parcelas com valores muito semelhantes, sob pena de se perder uma das suas principais funções: a da comparação [Haslwanter 2016, Stevenson and De Farias 1981].

Para “plotar” gráficos de setores é necessário a utilização da função `matplotlib.pyplot.pie` do `matplotlib` [Haslwanter 2016]. As Figuras 8.17 e 8.18 apresentam exemplos de gráficos de setores (pizza) a partir da função `pie()` do `matplotlib`.

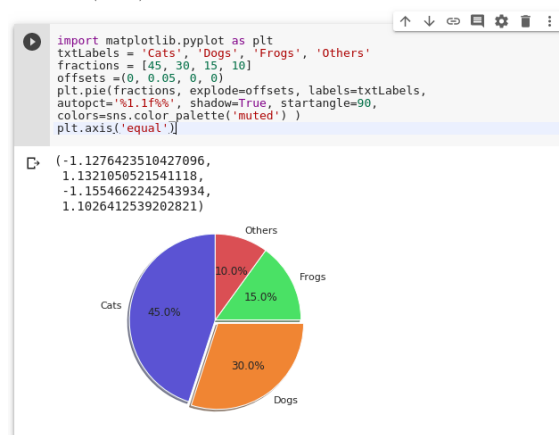


Figura 8.17. Gráfico de setores (pizza).

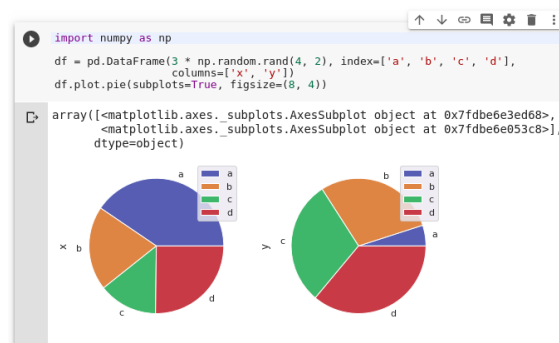


Figura 8.18. Gráfico de setores (pizza).

8.6.4. Boxplots

Um *boxplot* (diagrama de caixa) é um gráfico apresentado em formato de caixa, em que a aresta inferior da caixa representa o primeiro quartil (Q_1), a aresta superior representa o terceiro quartil (Q_3) e um traço interno à caixa representa a mediana (Q_2) de uma amostra.

Vale ressaltar que o *boxplot* não é paramétrico, ou seja, apresenta a variação das amostras de uma população estatística sem fazer qualquer suposição da distribuição estatística subjacente [Haslwanter 2016, Stevenson and De Farias 1981, Spiegel and Stephens 2000].

Para “plotar” *boxplot*, o *seaborn* disponibiliza a função *boxplot()*. Os dados de entrada para a função *boxplot()* podem ser transmitidos em vários formatos, incluindo:

- Vetores de dados representados como listas, matrizes *numpy* ou objetos da série *pandas* passados diretamente para os parâmetros *x*, *y* e/ou *matizes*;
- Um *DataFrame* de “formato longo”; nesse caso, as variáveis *x*, *y* e *matizes* determinarão como os dados são plotados;
- Um *DataFrame* de formato amplo, de modo que cada coluna numérica seja plotada; ou
- Uma matriz ou lista de vetores.

Na maioria dos casos, é possível usar objetos *numpy* ou *Python*, no entanto, objetos do *pandas* são preferíveis pois os nomes associados serão usados para definir os eixos. Além disso, é possível usar tipos categóricos para as variáveis de agrupamento para controlar a ordem dos elementos da plotagem.

As Figuras 8.19(a) e 8.19(b) apresentam exemplos de *boxplots* a partir da função *boxplot()* do *seaborn*.

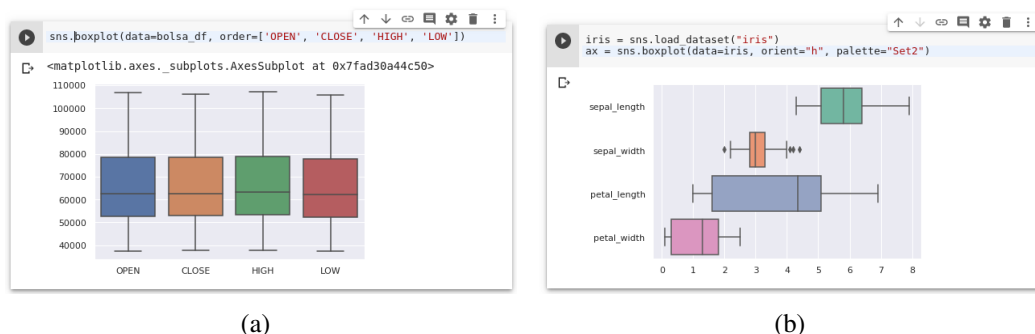


Figura 8.19. *Box plot*. (a) orientação vertical e (b) orientação horizontal.

8.6.5. Gráfico de dispersão

Gráficos de Dispersão são utilizados para pontuar dados em um eixo vertical e horizontal com a intenção de exibir quanto uma variável é afetada por outra.

Para “plotar” gráficos de dispersão, o *seaborn* disponibiliza a função *catplot()*, representação padrão dos dados na função *catplot()*. O *seaborn* disponibiliza dois gráficos de dispersão diferentes e eles adotam abordagens diferentes para resolver o principal desafio na representação de dados categóricos através de gráficos de dispersão, ou seja, todos os pontos pertencentes a uma categoria caem na mesma posição ao longo do eixo correspondente à variável categórica [Haslwanter 2016]

A Figura 8.20 apresenta um exemplo de um gráfico de dispersão a partir da função *catplot()* do *seaborn*.

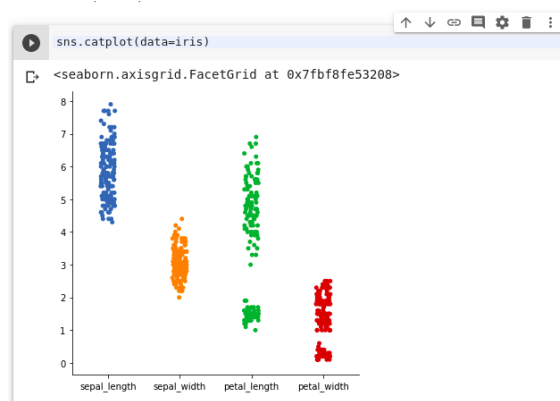


Figura 8.20. Gráfico de dispersão.

A Figura 8.21 apresenta uma abordagem que ajusta os pontos ao longo do eixo categórico usando um algoritmo que os impede de se sobreporem. Essa forma pode dar uma melhor representação da distribuição das observações, embora funcione bem apenas para conjuntos de dados relativamente pequenos. Às vezes, esse tipo de gráfico é chamado de “*beeswarm*” e é desenhado pelo *seaborn* através da função *swarmplot()*, que é ativado pela configuração de *kind = "swarm"* do *catplot()*.

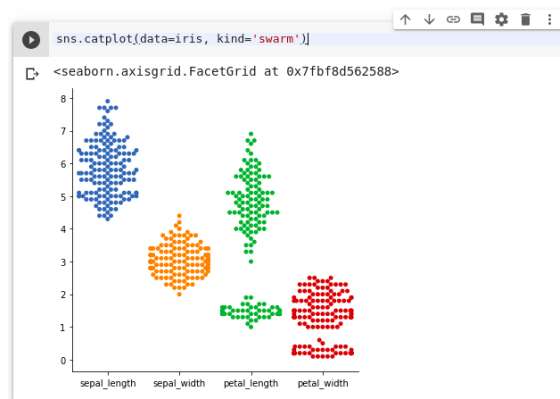


Figura 8.21. Gráfico de dispersão *beeswarm*.

8.7. Correlação

O termo correlação representa, sob o ponto de vista da estatística, uma medida de associação entre duas ou mais variáveis. Por definição, se forem considerados numa população,

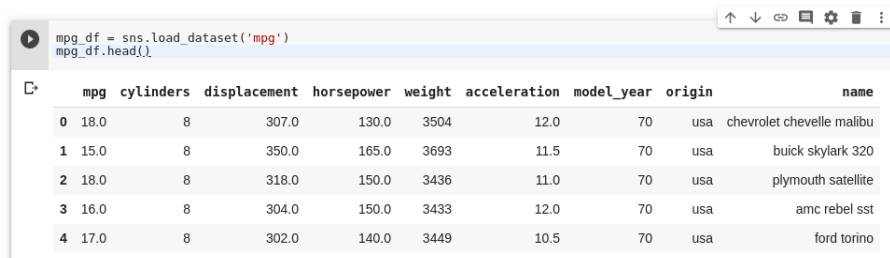
os pares de valores de duas variáveis (x_i, y_i) , a correlação pode ser definida pela Equação 7 [Spiegel and Stephens 2000, Stevenson and De Farias 1981].

$$\rho = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\left[\sum_{i=1}^n (x_i - \bar{x})^2 \right] \left[\sum_{i=1}^n (y_i - \bar{y})^2 \right]}} \quad (7)$$

O valor da correção, conhecido como coeficiente de correlação, assume valores no intervalo de -1 a 1 , de acordo com o grau de associação entre as variáveis em questão. Este grau de associação pode ser interpretado da seguinte forma:

- 0,9 a 1 positivo ou negativo indica uma correlação muito forte.
- 0,7 a 0,9 positivo ou negativo indica uma correlação forte.
- 0,5 a 0,7 positivo ou negativo indica uma correlação moderada.
- 0,3 a 0,5 positivo ou negativo indica uma correlação fraca.
- 0 a 0,3 positivo ou negativo indica uma correlação desprezível.

O *pandas* disponibiliza a função *corr()* para calcular a correlação entre duas colunas. Para exemplificar o uso da função *corr()* para calcular a correlação entre duas colunas, será utilizado o conjunto de dados “Auto-Mpg Data” (mpg), conforme pode ser observado na Figura 8.22.



```
mpg_df = sns.load_dataset('mpg')
mpg_df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

Figura 8.22. Obtenção e cinco primeiros valores do dataset Auto-Mpg Data.

A ideia é verificar qual característica do veículo está mais associada ao seu consumo de combustível (mpg: *miles per gallon* - milhas(1,6 km) por galão(3,78 litros)). A Figura 8.23 é possível verificar a correlação entre o peso do veículo e seu desempenho mpg do dataset mpg.

É possível também verificar todas as correlações de forma simultânea, como observado na Figura 8.24.

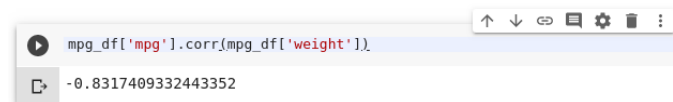


Figura 8.23. Correlação entre o peso do veículo e seu desempenho mpg do *dataset* mpg.

```
mpg_df.corr()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	0.579267
cylinders	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	-0.348746
displacement	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	-0.370164
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361
weight	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457	-0.306564
acceleration	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000	0.288137
model_year	0.579267	-0.348746	-0.370164	-0.416361	-0.306564	0.288137	1.000000

Figura 8.24. Correlação entre todas as variáveis do *dataset* mpg.

8.8. Considerações finais

Neste capítulo foram apresentados conceitos gerais sobre a Análise Exploratória de Dados (AED) através da linguagem de programação Python e a biblioteca Pandas, Matplotlib e Seaborn, demonstrando o quanto é simples e rápida a sua utilização na AED.

Foi apresentado uma breve descrição como se pode calcular medidas de tendência central e de dispersão, além de mostrar a apresentação dos dados estatísticos graficamente. Muitos outros dados e inferências poderiam ter sido abordados e outros gráficos também poderiam ser abordados, porém, como é apenas uma introdução esses outros pontos ficarão para trabalhos futuros.

Referências

- [Boschetti and Massaron 2015] Boschetti, A. and Massaron, L. (2015). *Python data science essentials*. Packt Publishing Ltd.
- [Coelho 2017] Coelho, A. S. (2017). Introdução a análise de dados com python e pandas. *Anais Eletrônicos ENUCOMP*, pages 862–876.
- [Haslwanter 2016] Haslwanter, T. (2016). *An Introduction to Statistics with Python*. Springer.
- [Jeliahovski 2014] Jeliahovski, E. (2014). *Análise Exploratória de Dados usando o R*. Editus.
- [Jones et al. 2001] Jones, E., Oliphant, T., Peterson, P., et al. (2001). *Scipy: Open source scientific tools for python*.
- [Spiegel and Stephens 2000] Spiegel, M. R. and Stephens, L. J. (2000). *Estatística: Coleção Schaum*. Bookman.

- [Stevenson and De Farias 1981] Stevenson, W. J. and De Farias, A. A. (1981). *Estatística aplicada à administração*.
- [Tukey 1977] Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley, 1st edition.
- [Vigni et al. 2013] Vigni, M. L., Durante, C., and Cocchi, M. (2013). Chapter 3 - exploratory data analysis. In Marini, F., editor, *Chemometrics in Food Chemistry*, volume 28 of *Data Handling in Science and Technology*, pages 55 – 126. Elsevier.