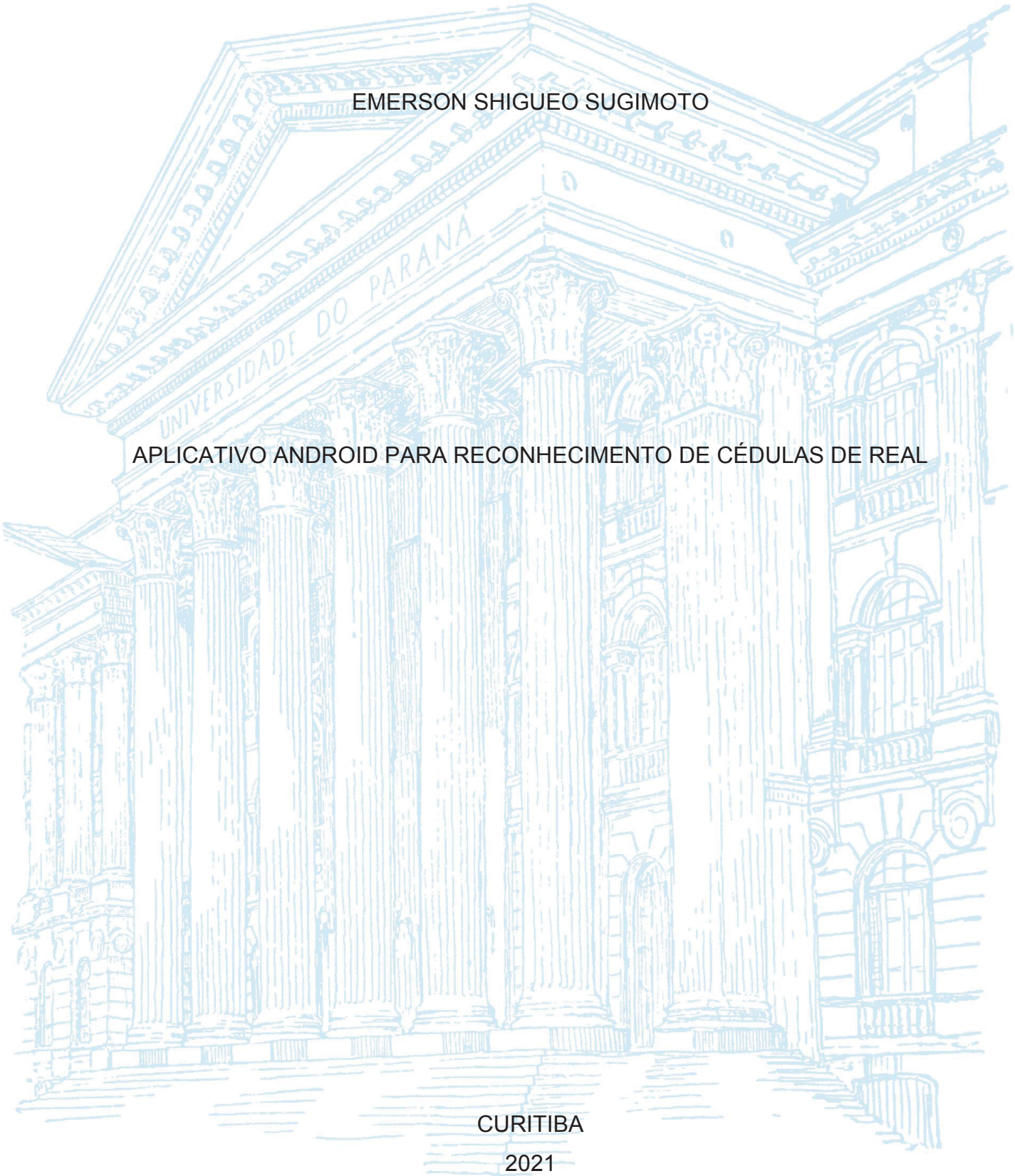
UNIVERSIDADE FEDERAL DO PARANÁ

EMERSON SHIGUEO SUGIMOTO



APLICATIVO ANDROID PARA RECONHECIMENTO DE CÉDULAS DE REAL

CURITIBA

2021

EMERSON SHIGUEO SUGIMOTO

APLICATIVO ANDROID PARA RECONHECIMENTO DE CÉDULAS DE REAL

Trabalho de conclusão de curso apresentado ao curso de Especialização em Inteligência Artificial Aplicada, Setor de Educação Profissional e Tecnológica, Universidade Federal do Paraná, como requisito parcial à obtenção do título de Especialista em Inteligência Artificial Aplicada.

Orientador: Prof. Dr. Razer Anthom Nizer Rojas Montaño

CURITIBA

2021

MINISTÉRIO DA EDUCAÇÃO
SETOR DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
CURSO DE PÓS-GRADUAÇÃO INTELIGÊNCIA ARTIFICIAL APLICADA - 40001016348E1

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INTELIGÊNCIA ARTIFICIAL APLICADA da Universidade Federal do Paraná foram convocados para realizar a arguição da Monografia de Especialização de **EMERSON SHIGUEO SUGIMOTO** intitulada: **APLICATIVO ANDROID PARA RECONHECIMENTO DE CÉDULAS DE REAL**, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua __APROVAÇÃO _____ no rito de defesa.
A outorga do título de especialista está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 01 de Julho de 2021.

RAZER ANTHOM NIZER ROJAS MONTAÑO
Presidente da Banca Examinadora (UNIVERSIDADE FEDERAL DO PARANÁ)

RAFAEL ALANTOVANI FONTANA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

RESUMO

O objetivo central do trabalho é o desenvolvimento de uma aplicação para dispositivos móveis Android de reconhecimento de cédulas de real através da câmera do celular. A aplicação deve ser simples, visando o uso de portadores de deficiência visuais. Como forma de identificação das cédulas e comunicação ao usuário, a aplicação deve vocalizar a classe reconhecida. Para detecção das classes das imagens capturadas pela câmera do celular, foi treinada uma rede neural especializada através da plataforma gratuita de desenvolvimento colaborativo em nuvem do Google Colab. Como método neste trabalho realizou-se a coleta e treinamento de 1412 imagens de cédulas de real nos valores de R$ 2,00, R$ 5,00, R$ 10,00, R$ 20,00, R$ 50,00 e R$ 100,00, frente e verso rotacionadas em ângulos de 90º, 180º e 270º. Como resultado o aplicativo desenvolvido para Android é capaz de reconhecer e classificar as cédulas de real treinadas na rede neural, bem como vocalizar a classificação de saída. Como conclusão espera-se que este trabalho possa auxiliar pessoas portadoras de alguma deficiência visual através de um aplicativo de reconhecimento de cédulas de real.

Palavras-Chave: Colab. TensorFlow. TensorFlow Lite. Rede neural. Classificação de imagens. Aplicativos Móveis. Smartphone.

ABSTRACT

The main objective of the work is the development of an application for Android mobile devices to recognize real banknotes through the cell phone camera. The application should be simple, aimed at the use of visually impaired people. As a way of identifying the ballots and communicating to the user, the application must vocalize the recognized class. To detect the classes of images captured by the cell phone camera, a specialized neural network was trained through the free collaborative development platform in the Google Colab cloud. As a method in this work, 1412 images of banknotes in the amounts of R$ 2,00, R$ 5,00, R$ 10,00, R$ 20,00, R$ 50,00 and R$ 100,00, front and back rotated at 90º, 180º and 270º angles. As a result the application developed for Android is able to recognize and classify the real banknotes trained in the neural network, as well as vocalize the output classification. As a conclusion, it is hoped that this work can help people with some visual impairment through an application for recognition of real banknotes.

Keywords: Colab. TensorFlow. TensorFlow Lite. Neural network. Image classification. Mobile Applications. Smartphone.

LISTA DE FIGURAS

FIGURA 1 - HIERARQUIA DE APRENDIZADO....24
FIGURA 2 - NEURÔNIO ARTIFICIAL....27
FIGURA 3 - FUNÇÕES DE ATIVAÇÃO....28
FIGURA 4 - FUNÇÃO DE ATIVAÇÃO RELU....28
FIGURA 5 – REDES NEURAIS MULTICAMADAS....30
FIGURA 6 - PADRÕES DE CONEXÃO REDE MULTICAMADAS....30
FIGURA 7 – REDES NEURAIS RECORRENTES....31
FIGURA 8 - OBJETOS LINEARMENTE SEPARÁVEIS....33
FIGURA 9 - IMAGEM DE ENTRADA....38
FIGURA 10 - *KERNEL* (FILTRO)....38
FIGURA 11 - *KERNEL* X IMAGEM ENTRADA 1....38
FIGURA 12 - MATRIZ RESULTANTE 1....38
FIGURA 13 - *KERNEL* X IMAGEM ENTRADA 2....38
FIGURA 14 - MATRIZ RESULTANTE 2....38
FIGURA 15 - *KERNEL* X IMAGEM ENTRADA 3....39
FIGURA 16 - MATRIZ RESULTANTE 3....39
FIGURA 17 - *KERNEL* X IMAGEM ENTRADA 4....39
FIGURA 18 MATRIZ RESULTANTE 4....39
FIGURA 19 *KERNEL* X IMAGEM ENTRADA 5....39
FIGURA 20 - MATRIZ RESULTANTE 5....39
FIGURA 21 - *KERNEL* X IMAGEM ENTRADA 6....40
FIGURA 22 - MATRIZ RESULTANTE 6....40
FIGURA 23 - *KERNEL* X IMAGEM ENTRADA 7....40
FIGURA 24 - MATRIZ RESULTANTE 7....40
FIGURA 25 - *KERNEL* X IMAGEM ENTRADA 8....40
FIGURA 26 - MATRIZ RESULTANTE 8....40
FIGURA 27 - *KERNEL* X IMAGEM ENTRADA 9....41
FIGURA 28 - MATRIZ RESULTANTE 9....41
FIGURA 29 - IMAGEM ENTRADA....41
FIGURA 30 - *PADDING*....41
FIGURA 31 - FILTRO / *KERNEL*....42
FIGURA 32 - SAÍDA....42

FIGURA 33 - IMAGEM ENTRADA 42
FIGURA 34 - *MAXPOOLING* 42
FIGURA 35 - *DROPOUT* 43
FIGURA 36 - REDES DROPOUT 44
FIGURA 37 - FLATTEN 45
FIGURA 38 - R-CNN VISÃO GERAL 46
FIGURA 39 – COMPARAÇÃO DE ALGORITMOS DE DETECÇÃO DE OBJETOS 47
FIGURA 40 - REDE SSD 48
FIGURA 41 - APLICAÇÕES DE MOBILENET 51
FIGURA 42 - SSD MOBILENETS 52
FIGURA 43 - ESTRUTURA REDE SSD MOBILENET 52
FIGURA 44 – TRANSFER LEARNING 55
FIGURA 45 - APRENDIZADO POR TRANSFERÊNCIA 56
FIGURA 46 – EXEMPLO *DATASET* COCO 57
FIGURA 47 - RGB 63
FIGURA 48 – *NOTEBOOK* COLAB 69
FIGURA 49 – REPRESENTAÇÃO CÉDULAS 70
FIGURA 50 - CONECTAR GOOGLE DRIVE 70
FIGURA 51 - EXEMPLO MODELO 71
FIGURA 52 – EXEMPLO TREINAMENTO MODELO 71
FIGURA 53 – EXEMPLO AVALIAÇÃO MODELO 72
FIGURA 54 – EXEMPLO DE PREDIÇÕES 73
FIGURA 55 - EXPORTAR MODELO TFLITE 73
FIGURA 56 - TENSORFLOW 74
FIGURA 57 – EXEMPLO XML 77
FIGURA 58 – IMAGEM ANOTADA 78
FIGURA 59 - FRAMEWORK SCRUM 79
FIGURA 60 – KANBAN - TRELLO 80
FIGURA 61 – KANBAN - TRELLO 83
FIGURA 62 – IMAGEM R$ 100 86
FIGURA 63 – IMAGEM R$ 100 – ROTAÇÃO 180º 86
FIGURA 64 – IMAGEM R$ 100 – ROTAÇÃO 90º 86
FIGURA 65 – IMAGEM R$ 100 – ROTAÇÃO 270º 86
FIGURA 66 – IMAGEM R$ 100 – ROTAÇÃO HORIZONTAL 86

FIGURA 67 – IMAGEM R$ 100 – ROTAÇÃO VERTICAL....86
FIGURA 68 – LABELIMG....87
FIGURA 69 – CÉDULAS E ANOTAÇÕES XML....87
FIGURA 70 – IMAGEM R$ 100 – FRENTE....88
FIGURA 71 – IMAGEM R$ 100 – VERSO....88
FIGURA 72 – ESTRUTURA MODELOS 1 E 2....89
FIGURA 73 - ESTRUTURA MODELO 3....90
FIGURA 74 - ARQUIVOS .MP3....92
FIGURA 75 - PROGRESSO DE TREINAMENTO DA REDE....97
FIGURA 76 – PERMISSÕES APLICATIVO....101
FIGURA 77 - ÍCONE APLICAÇÃO....102
FIGURA 78 - TELA INICIAL....102
FIGURA 79 - DETEÇÃO CÉDULA R$ 2,00....104
FIGURA 80 - DETEÇÃO CÉDULA R$ 5,00....105
FIGURA 81 - DETEÇÃO CÉDULA R$ 10,00....106
FIGURA 82 - DETEÇÃO CÉDULA R$ 20,00....107
FIGURA 83 - DETEÇÃO CÉDULA R$ 50,00....108
FIGURA 84 - DETEÇÃO CÉDULA R$ 100,00....109
FIGURA 85 - DIAGRAMA DE CASOS DE USO....127
FIGURA 86 - DIAGRAMA DE ATIVIDADES....134
FIGURA 87 – DIAGRAMA DE CLASSES....137
FIGURA 88 – DIAGRAMA DE CLASSES....138
FIGURA 89 – DIAGRAMA DE CLASSES....139
FIGURA 90 – DIAGRAMA DE CLASSES....140
FIGURA 91 – DIAGRAMA DE CLASSES....141
FIGURA 92 – DIAGRAMA DE CLASSES....142
FIGURA 93 – CLASSE MAINACTIVITY....143
FIGURA 94 – CLASSE DETECTORACTIVITY....144
FIGURA 95 - DIAGRAMA DE SEQUÊNCIA CLASSIFICAÇÃO....145
FIGURA 96 - DIAGRAMA DE SEQUÊNCIA REPRESENTAÇÃO CLASSES....146
FIGURA 97 - DIAGRAMA DE SEQUÊNCIA VOCALIZAÇÃO....147

LISTA DE GRÁFICOS

GRÁFICO 1 - CURVA ROC 62
GRÁFICO 2 - LOSS TOTAL.....98
GRÁFICO 3 - ROC.....100

LISTA DE QUADROS

QUADRO 1 – EXEMPLO MATRIZ CONFUSÃO 59
QUADRO 2 – ANOTAÇÃO XML PASCAL VOC 78
QUADRO 3 – EXEMPLO DE ANOTAÇÃO XML 88
QUADRO 4 – CÓDIGO PYTHON GERADOR DE ARQUIVO .MP3. 91
QUADRO 5 - MATRIZ DE CONFUSÃO 99
QUADRO 6 - MÉTRICAS 99
QUADRO 7 - RECONHECIMENTO DE CÉDULAS 129
QUADRO 8 - CLASSIFICAÇÃO MODELO REDE NEURAL 130
QUADRO 9 - REPRESENTAÇÃO DO RESULTADO NA TELA 131
QUADRO 10 - VOCALIZAÇÃO DO RESULTADO 132
QUADRO 11 - ARQUIVO LABELMAP1.TXT 135
QUADRO 12 – CÓDIGO ADAPTADO 136
QUADRO 13 - DETECÇÃO MODELO 136

LISTA DE TABELAS

TABELA 1 – CONVOLUÇÃO *STRIDE* 1X1 37
TABELA 2 - ESTRUTURA REDE SSD MOBILENET 53
TABELA 3 – DIVISÃO CLASSES TREINAMENTO E TESTES 85
TABELA 4 - RESUMO DAS SPRINTS 93
TABELA 5 – ESTRUTURA REDE MOBILENET 148

LISTA DE ABREVIATURAS OU SIGLAS

DA - data augmentation
CNN - Convolutional Neural Network
COCO - Common Objects in Context
CPU - Central Processing Unit
Fast-RCNN - Fast Rapid Convolutional Neural Network
FN - Falso negativo
FP - Falso positivo
GAN - Generative adversarial network
GPU - Graphics Processing Unit
gTTS - Google Text-to-Speech
JVM - Java Virtual Machine
ML - Machine Learning
mp3 - MPEG Layer 3
NPV - Negative Predictive Values
PPV - Positive Predictive Values
R-CNN - Rapid Convolutional Neural Network
RGB – Red Green Blue
ROC - receiver operator characteristic curve
ROI - Regiões de interesse
SO - Sistema operacional
SSD - Single Shot Detector
SURF - Speeded Up Robust Features
TCC – Trabalho de conclusão de curso
tflite - TensorFlow Lite
TPU - Unidade de Processamento de Tensor
VN - Verdadeiro negativo
VP - Verdadeiro positivo
VPN - valor preditivo negativo
VPP - valor preditivo positivo
XML - Extensible Markup Language

LISTA DE SÍMBOLOS

@ - arroba

® - marca registrada

º - graus

SUMÁRIO

1 INTRODUÇÃO 16
1.1 OBJETIVOS 17
1.1.1 Objetivo geral 17
1.1.2 Objetivos específicos 18
1.2 JUSTIFICATIVA 18
1.3 ORGANIZAÇÃO DO TRABALHO 19
2 REVISÃO DE LITERATURA 20
2.1 *MACHINE LEARNING* 22
2.2 REDES NEURAIS 25
2.2.1 Formas de aprendizado 25
2.2.2 Breve histórico 26
2.2.3 Arquitetura 27
2.2.3.1 Aprendizado 32
2.2.4 Função de perda 34
2.3 REDE NEURAL CONVOLUCIONAL (CNN) 36
2.3.1 R-CNN 45
2.3.2 Fast R-CNN 46
2.4 *SINGLE SHOT DETECTOR* (SSD) 47
2.5 MOBILENETS 50
2.6 SSD MOBILENETS 51
2.7 *TRANSFER LEARNING* 54
2.8 COMMON OBJECTS IN CONTEXT (COCO) 56
2.9 *DATA AUGMENTATION* 57
2.10 MEDIDAS DE AVALIAÇÃO 59
2.10.1 Matriz de confusão 59
2.10.2 Total de predições 60
2.10.3 Acurácia 60
2.10.4 Sensibilidade 60
2.10.5 Especificidade 60
2.10.6 PPV 61
2.10.7 VPN 61
2.10.8 Prevalência 61

2.10.9 ROC 61
2.11 IMAGEM COLORIDA 62
2.12 ANDROID 63
2.13 JAVA 64
2.14 PYTHON 65
2.14.1 Biblioteca Python gTTS 67
2.15 GOOGLE COLAB 67
2.16 TENSORFLOW 74
2.17 PASCAL VOC XML 76
2.18 SCRUM 78
2.19 TRELLO 80
3 MATERIAIS E MÉTODOS 81
3.1 MODELO DE ENGENHARIA DE SOFTWARE 81
3.2 MATERIAIS 81
3.2.1 RECURSOS DE *HARDWARE* 84
3.2.2 RECURSOS DE SOFTWARE 84
3.3 COLETA DE IMAGENS 85
3.4 MODELO 89
3.5 VOCALIZAÇÃO DAS CLASSES 91
3.6 PROJETO ANDROID BASE 92
3.7 DESENVOLVIMENTO DO PROJETO 92
3.7.1 Primeira *Sprint* 94
3.7.2 Segunda *Sprint* 94
3.7.3 Terceira *Sprint* 94
3.7.4 Quarta *Sprint* 95
3.7.5 Quinta *Sprint* 95
3.7.6 Sexta *Sprint* 95
3.7.7 Sétima *Sprint* 95
3.7.8 Oitava *Sprint* 96
3.7.9 Nona Sprint 96
4 APRESENTAÇÃO DOS RESULTADOS 97
4.1 APLICAÇÃO 100
4.1.1 Exemplos de predições 103
5 CONSIDERAÇÕES FINAIS 110

5.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS 111
REFERÊNCIAS 113
APÊNDICE A - LISTA DE REQUISITOS 126
APÊNDICE B – DIAGRAMA DE CASOS DE USO 127
APÊNDICE C – DIAGRAMA DE ATIVIDADES 133
APÊNDICE D – MAPEAMENTO *LABELS* 135
APÊNDICE E – DIAGRAMA DE CLASSES 136
APÊNDICE F - DIAGRAMAS DE SEQUÊNCIA 145
APÊNDICE G – ESTRUTURA REDE MOBILENET 148

1 INTRODUÇÃO

A prática de troca direta de valores deu origem a sistemas de pagamento, como o salário. Uma forma para se representar valores monetários é a moeda, sendo as primeiras criadas na Lídia (atual Turquia), no século VII A.C. (GONÇALVES, 1984). Para facilitar o seu armazenamento e transporte, negociantes de prata e ouro aceitaram a tarefa de armazenar moedas em cofres e passaram a dar recibos dos valores guardados, surgindo assim as primeiras cédulas de “papel moeda” (CASA DA MOEDA, 2020).

No Brasil a primeira moeda cunhada foi em 1694 na Casa da Moeda da Bahia. As primeiras cédulas foram os "réis", baseados no sistema monetário português (GARCIA, 2020).

Através da lei específica PL 8.880/1994 de 27 de maio de 1994 foi instituído no Brasil o sistema monetário Real, vigorando em 1º de julho de 1994 (DINIZ, 2019).

O dinheiro passou a fazer parte das relações de troca de serviços, mão de obra, produtos, aplicações financeiras, etc. Entre os meios de pagamento, cita-se o uso do dinheiro em espécie, cédulas e moedas de real, transações financeiras, como o DOC, TED, PIX, pagamento com cartões de crédito e débito, boletos bancários, *vouchers*, duplicatas, etc. De acordo com uma pesquisa feita em 2018 (BANCO CENTRAL DO BRASIL, 2020), 96% das pessoas entrevistadas usa como principal meio de pagamento o dinheiro.

Como a maior parte das pessoas utiliza o dinheiro como forma de pagamento, faz-se necessária a correta identificação do seu valor representativo pela população. Neste intuito, existem alguns elementos que auxiliam a identificação das cédulas de real, entre elas, podemos citar: faixa holográfica, número e valor da nota, cores, gramatura, faixa holográfica, tamanho, figura de animal, marcas de relevo, entre outros (BANCO CENTRAL DO BRASIL, 2020).

Visando a identificação das cédulas por pessoas portadoras de deficiência visual, os principais elementos utilizados no Brasil são: o tamanho das cédulas, braile e marcas de relevo. Um dos problemas do braile e das marcas de relevo é que desaparecem com o tempo e são pouco perceptíveis para os mais idosos. Outro problema do braile é que nem todos os portadores de deficiência visual sabem ler. Com relação ao tamanho das cédulas, o modelo adotado pelo Brasil foi baseado na Austrália, no qual cada cédula possui um tamanho diferente (LOPES, 2020). Como

adição a estes mecanismos de identificação, o Canadá imprime cada nota com uma tinta magnética invisível, reconhecida por um dispositivo que emite um sinal diferente para cada valor (LOPES, 2020).

Segundo dados do Censo de 2010 (IBGE, 2020), 23,9% da população brasileira declarou ter algum tipo de deficiência. Sendo a mais comum, a deficiência visual (3,5% da população). Deste censo, 582 mil pessoas informaram ser cegas e 6 milhões com baixa visão.

A crescente difusão e uso de smartphones (TOKARNIA, 2020) permite a aplicação e desenvolvimento de novas tecnologias de acessibilidade digital. No intuito de auxiliar no reconhecimento das cédulas de real por pessoas portadoras de deficiências visuais, neste trabalho aborda-se o uso do smartphone para reconhecimento de cédulas e vocalização da classe de reconhecimento em seis classes representativas: R$ 2,00, R$ 5,00, R$ 10,00, R$ 20,00, R$ 50,00 e R$ 100,00. O trabalho foca em aplicativos Android, porém a sua aplicação pode ser expandida para reconhecimento e classificação de imagens e ambientes computacionais.

O presente trabalho aborda o treinamento de uma rede neural utilizando SSD e MobileNets, voltado para o TensorFlow Lite para uma aplicação Android. Esta seção descreve os conceitos e técnicas empregadas no desenvolvimento deste trabalho.

1.1 OBJETIVOS

O presente trabalho tem por objetivo desenvolver uma aplicação para reconhecimento de cédulas de real para aplicação em dispositivos móveis Android.

1.1.1 Objetivo geral

Promover acessibilidade aos portadores de deficiência visual ao desenvolver uma aplicação capaz de reconhecer cédulas de real e vocalizar a classe identificada.

1.1.2 Objetivos específicos

Os objetivos específicos do trabalho são:

- Estudar a ferramenta Google Colab para treinamento de rede neural;
- Treinar um modelo de rede neural capaz identificar cédulas de real;
- Desenvolver um aplicativo para dispositivo móvel Android que utilize a câmera do celular para capturar as imagens das cédulas;
- Identificar 12 classes de cédulas;
- Gerar arquivos de áudio mp3 que vocalizem as classes;
- Vocalizar o resultado.

1.2 JUSTIFICATIVA

Dentre os principais elementos de identificação das cédulas de real, apenas três são relevantes para portadores de deficiência visual, sendo:

- Tamanho das cédulas: 2 reais – 12,1 cm x 6,5 cm; 5 reais – 12,8 cm x 6,5 cm; 10 reais – 13,5 cm x 6,5 cm; 20 reais – 14,2 cm x 6,5 cm; 50 reais – 14,9 cm x 7,0 cm; 100 reais – 15,6 cm x 7,0 cm; (VALOR CONSULTING, 2020)
- Marcas de relevo; e
- Braile.

Dos elementos citados, apenas o tamanho das cédulas é um diferencial prático de identificação das cédulas, já que o braile e as marcas de relevo se perdem com o tempo, idosos possuem dificuldade para perceber as marcas, além de que nem todos os portadores de deficiência visual e estrangeiros sabem ler em braile (LOPES, 2020).

O desenvolvimento de uma aplicação de reconhecimento de cédulas de real pode auxiliar na identificação das cédulas promovendo acessibilidade aos portadores de deficiência visual. De acordo com Almeida (2019, p. 12), “para a maioria das pessoas a tecnologia torna a vida mais fácil, para uma pessoa com necessidades especiais, a tecnologia torna as coisas possíveis”.

Diniz (2020) realizou uma pesquisa com o intuito de avaliar o uso de cédulas e moedas no uso cotidiano de deficientes visuais e identificar demandas de

desenvolvimento de novas tecnologias que auxiliem os portadores de deficiência visual. Um dos resultados foi a necessidade de tecnologias de identificação de cédulas de real.

Com o aumento do uso e popularidade dos smartphones (TOKARNIA, 2020), seu uso em conjunto com uma aplicação de smartphone com o objetivo de identificar das cédulas de real pode auxiliar portadores de deficiência visual nesta tarefa.

1.3 ORGANIZAÇÃO DO TRABALHO

O Capítulo 2 apresenta a revisão de literatura, conceitos, técnicas empregadas, tecnologias, métodos, ferramentas, formatos de documentos, entre outros utilizados neste trabalho.

Os materiais e métodos estão descritos no Capítulo 3. Esta seção descreve a plataforma utilizada para treinar a rede neural, método de coleta das imagens, anotações, classificação das cédulas, modelo treinado, vocalização das classes das cédulas, projeto Android base, o modelo de engenharia de software adotado e recurso de *hardware* e software.

Os resultados estão descritos no Capítulo 4. A seção apresenta as métricas de classificação, tempo de treinamento, função de perda (*loss function*), matriz de confusão, acurácia, sensibilidade, PPV, VPN, gráfico ROC, prevalência, especificidade e detecção de cédulas. Descreve sobre a aplicação Android e exibe alguns exemplos de predições.

O Capítulo 5 descreve as considerações finais, pontos-chave, observações, técnicas diferenciais no treinamento da rede neural e recomendações para trabalhos futuros.

2 REVISÃO DE LITERATURA

Existem diversos estudos voltados para o reconhecimento de cédulas monetárias de diversos países. Cada estudo utiliza uma técnica e abordagem diferente. Entre eles cita-se o trabalho de Suriya et al. (2014) voltado ao reconhecimento de rúpias indianas em dispositivos móveis com o objetivo de auxiliar portadores de deficiência visual. Foram utilizados os descritores de características SIFT[1], SURF[2] e ORB-FREAK[3] para reconhecimento das cédulas por meio da câmera do celular. Através da localização dos pontos-chaves da região de interesse das imagens (regiões das cédulas) obtém-se o histograma para construção do vocabulário visual. Através do modelo de representação é feita a classificação espacial, analisando os pontos-chaves de imagem de teste, verificando a consistência com o vocabulário e as classificando. O conjunto de dados é formado por notas de rupia indiana que possuem tamanhos semelhantes, diferem em cor e denominação impressa. As imagens foram capturadas usando câmeras de telefones celulares, com diferentes resoluções: 1,3 megapixels (MP), 2 MP e 5 MP. A coleta das imagens é proveniente de 13 ambientes diferentes: 6 internos e 7 externos. O conjunto de dados contém imagens de notas novas, gastas e com rabiscos. Os resultados em um conjunto de 2584 imagens foram de 96,7% de precisão e tempo de resposta médio de 3,28 segundos.

Lopes (2020) utiliza duas abordagens de classificação de cédulas de real: linear e não-linear. A abordagem não linear foi feita através de uma rede neural artificial do tipo multilayer perceptron (MLP), e a abordagem não linear, através de uma máquina de suporte de vetores (SVM), aplicado em um espaço vetorial de características. A técnica de análise de componentes principais (PCA) é utilizada para diminuir a dimensionalidade dos dados na etapa de pré-processamento das imagens. Para cada valor possível para as cédulas, são criadas duas classes: uma para a frente e outra para o verso da cédula O banco de dados formado possui 14 classes, sendo duas (frente e verso) para as cédulas de R$ 1,00, R$ 2,00, R$ 5,00, R$ 10,00, R$ 20,00, R$ 50,00 e R$ 100,00. As notas foram digitalizadas através das

---

[1] SIFT - scale-invariant feature transform - algoritmo de detecção de características locais em imagens.

[2] SURF - Speeded Up Robust Features - é um detector e descritor de recurso local. Ele é parcialmente inspirado no descritor de transformação de recurso invariável de escala (SIFT).

[3] ORB-FREAK - algoritmo de extração de recursos em imagens.

câmeras dos celulares Samsung modelos Galaxy Y GT S6102B e o Galaxy Pocket S5300, com 3 e 2 megapixels de resolução, respectivamente sobre um fundo branco, frente e verso. O total de imagens de cédulas capturadas foi de 500 imagens, sendo 261 do verso e 239 de frente. O conjunto de treinamento e testes foi separado em 90% e 10%. A abordagem não linear de redes neurais obteve uma taxa de 94% de acertos, enquanto a linear (SVM) de 92%.

Ulian (2016) preocupa-se com a identificação de cédulas de real voltadas para a segurança de informação. Neste intuito o número de série das cédulas pode ser usado para rastreamento e controle. A sua identificação é feita através do reconhecimento óptico de caracteres (OCR) através da ferramenta Tesseract. Para contornar problemas como a variação do ângulo, orientação e escala da imagem, imagens borradas, iluminação inadequada e caracteres distorcidos foi utilizado o algoritmo SURF para detectar pontos discriminantes nas cédulas. Para detectar a área de interesse dos números de série, utilizou-se a abordagem de componentes conexos de Regiões Extremas Maximamente Estáveis. Esta técnica procura por regiões semelhantes, denominadas *blobs*, em uma imagem de interesse. O banco de imagens foi criado com imagens de cédulas de R$ 2,00 e R$ 5,00, organizados em quatro classes, totalizando 980 números seriais. As imagens foram capturadas através da câmera de um celular Samsung Galaxy S3 GT-i9305T e divididas em quatro classes: A primeira classe é formada por 161 imagens horizontais e imagens com uma ou mais cédulas; A segunda classe é formada por 130 imagens rotacionadas em: 0º, 30º, 60º e 90º; A terceira classe contém 98 imagens com variações de escala, distanciando as cédulas da câmera em 7, 11 e 18 centímetros; As imagens da quarta classe não possuem números seriais, totalizando 34 imagens. Os resultados de *F-measure* foram de 0,93 de mensuração por número de série e 0,97 de mensuração por caractere.

Hasanuzzaman (2011) utiliza uma estrutura baseada em componentes em conjunto com SURF (*Speeded Up Robust Features*) no reconhecimento de cédulas de dólar voltada para portadores de deficiência visual. A estrutura de componentes é capaz de lidar com oclusão parcial e mudanças no ponto de vista, enquanto a SURF demonstra eficácia para tratar ruídos de fundo, rotações na imagem, escala e alterações de iluminação. As características das cédulas são extraídas por SURF e combinados com dados de SURF pré-calculadas de regiões de referência de cédulas da base de dados. Dois testes são realizados: o primeiro é o teste de ponto

de nível, para detectar pontos em comum entre as imagens; e o segundo é feito para comparar a região de teste. A base de dados é composta por 140 imagens de cédulas, sendo 20 de cada de cada classe: U$$ 1, U$$ 2, U$$ 5, U$$ 10, U$$ 20, U$$ 50 e U$$ 100 com variedade de condições, como oclusão, fundo diverso, rotação, alterações de iluminação, escala e pontos de vista. O algoritmo proposto obteve 100% de taxa de reconhecimento do conjunto de dados.

Toytman et al. (2011) desenvolveram uma aplicação Android utilizando descritores SURF para reconhecimento de cédulas de dólar, que possuem tamanho e cores semelhantes. Dada a imagem de entrada (768x432 pixels), é convertida em tons de cinza, é aplicado um filtro gaussiano de 3x3 para eliminar os ruídos. Em seguida aplica-se o detector de bordas Canny e aplica-se uma dilatação seguida de uma erosão utilizando um círculo de 3x3 como elemento estruturante. A partir da imagem resultante detectam-se os pontos de interesse e calculam-se os descritores SURF. Estes pontos de interesse são comparados com as imagens de referência das cédulas, resultando na classificação da nota. O algoritmo consegue identificar uma cédula até com 25% de oclusão. Notas dobradas apresentam uma limitação na classificação. Os testes foram feitos em um telefone Motorola Droid equipado com CPU de 800 MHz. O tempo médio de reconhecimento das cédulas foi de 30 segundos.

2.1 *MACHINE LEARNING*

Aprendizado de máquina (AM), ou *machine learning* (ML), é uma subárea da inteligência artificial (IA). Pensando de forma comportamental a IA é um conceito no qual mecanismos computacionais baseados na forma de pensamento humano são utilizados para resolver problemas e executar tarefas (RUSSEL e NORVIG, 2013). No aprendizado de máquina, o sistema consegue modificar o seu aprendizado com base em sua própria experiência, de forma a automatizar a construção de modelos analíticos, voltados para a aplicação em alguma área de conhecimento, resoluções de problemas, automatização de tarefas, reconhecimento de padrões, predições, entre outras aplicações. O processo de indução de uma hipótese (ou aproximação de função) a partir da experiência passada dá-se o nome Aprendizado de Máquina. De acordo com Faceli et al (2011), atividades como memorizar, observar e explorar situações para aprender fatos, melhorar habilidades motoras/cognitivas por meio de

práticas e organizar conhecimento novo em representações apropriadas podem ser atividades relacionadas ao aprendizado.

Através de um grande volume de dados o aprendizado de máquina pode reconhecer, extrair padrões e construir um modelo de aprendizado (FACELI et al, 2011). A partir deste modelo treinado e gerado, o sistema é capaz de realizar tarefas complexas, melhorar a sua previsão e atuar de forma mais precisa e inteligente sobre a tarefa a ser executada. A generalização da hipótese ocorre quando um algoritmo de AM consegue fornecer hipóteses válidas para um conjunto de dados diferente daqueles utilizados no treinamento. Caso a hipótese apresente uma baixa capacidade de generalização, pode ser que ela esteja superajustada aos dados (*overfitting*), desta forma diz-se que a hipótese memorizou e se especializou nos dados de treinamento. O caso contrário, no qual o modelo apresente uma baixa taxa de acerto mesmo no conjunto de treinamento é chamado de subajustamento (*underfitting*).

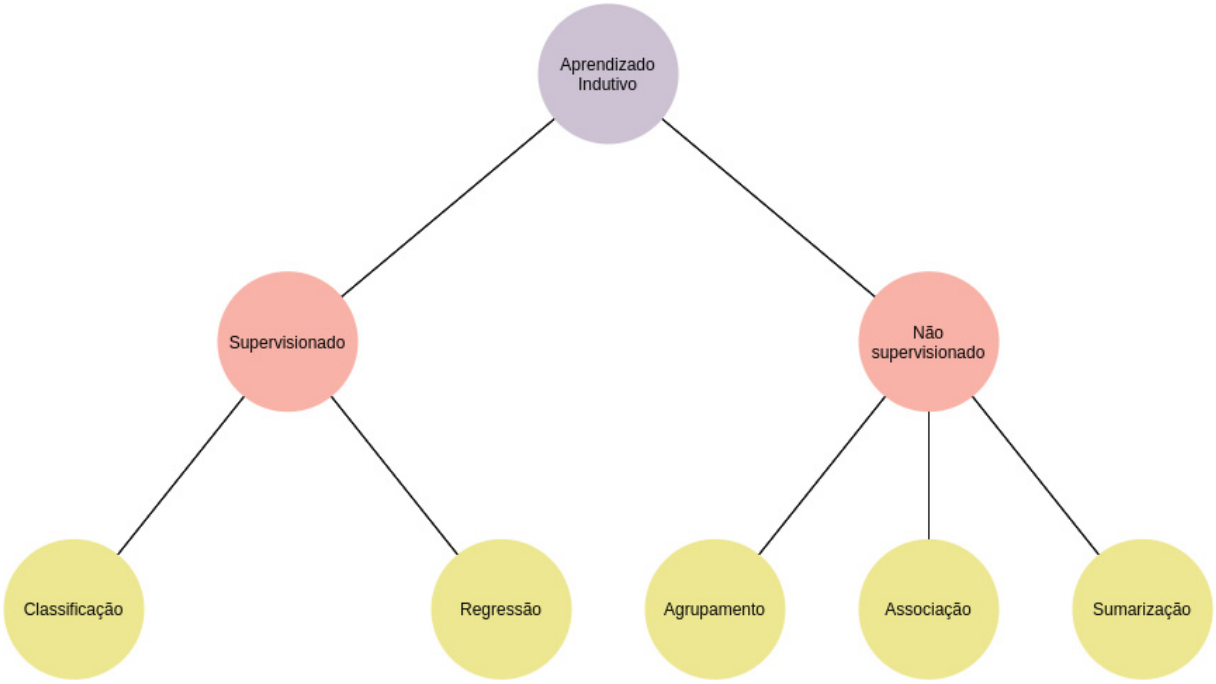
A representação utilizada pelo algoritmo de AM define a preferência ou viés indutivo de representação do algoritmo e pode restringir o conjunto de hipóteses que podem ser induzidas. Como exemplo, redes neurais representam uma hipótese por um conjunto de valores reais, associados aos pesos das conexões da rede, árvores de decisão utilizam uma estrutura de árvore (FACELI et al, 2011). O viés de busca está relacionado à forma como as hipóteses são pesquisadas. Sem viés não haveria aprendizado e generalização.

As tarefas de aprendizado podem ser preditivas (supervisionado) ou descritivas (não supervisionado). No aprendizado preditivo, cada objeto do conjunto de treinamento possui atributos de entrada e de saída. É chamado de supervisionado por causa da simulação de um “supervisor externo”, que conhece a saída de cada exemplo (entrada) e avalia a hipótese induzida. No aprendizado descritivo, os algoritmos de AM não utilizam o atributo de saída, seguindo o paradigma de aprendizado não supervisionado.

A FIGURA 1 apresenta uma hierarquia de aprendizado. O aprendizado indutivo é um processo onde são realizadas generalizações a partir dos dados. As tarefas supervisionadas (preditivas) são: classificação (rótulos discretos) e regressão (rótulos contínuos). As tarefas de aprendizado não supervisionado (descritivas) são: agrupamento (dados agrupados pela similaridade), sumarização (descrição simples

e compacta dos dados) e associação (padrões frequentes de associações entre os atributos) (FACELI et al, 2011).

FIGURA 1 - HIERARQUIA DE APRENDIZADO



FONTE: Adaptado de Faceli et al (2011)

No aprendizado por reforço a meta é reforçar uma ação positiva e punir uma ação negativa. Como exemplo, dada a tarefa de encontrar o melhor caminho entre dois pontos, pune-se a escolha de caminhos pouco promissores e recompensa-se a escolha de caminhos promissores (FACELI et al, 2011).

Principais abordagens de *machine learning* (ML): redes neurais, regressão linear, regressão logística, árvores de decisão, classificação naive bayes, *extreme gradient boosting* – XGBoost, SVM (*Support Vector Machine*), *ensemble methods*, algoritmos de agrupamento (*clustering*), decomposição em valores singulares, análise de componentes principais (PCA), análise de componentes independentes (ICA), KNN (K-vizinhos mais próximos), floresta aleatória ou *random forest*, algoritmos de redução dimensional (GOMES, 2019). Este trabalho foca no uso da abordagem de ML redes neurais, descrita na Seção 2.2.

2.2 REDES NEURAIS

Redes neurais são reproduções simplificadas da estrutura biológica neuronal. Nesta reprodução, os neurônios se interconectam, separados por camadas e sua aprendizagem se dá pela alteração em suas conexões (BERRY e LINOFF, 2004).

As redes neurais artificiais são modelos que buscam simular o processamento de informação do cérebro humano. São abstrações dos neurônios biológicos, que se conectam por meio de conexões sinápticas.

O processamento básico de informações nas redes neurais artificiais ocorre em diversas unidades simples chamadas de neurônios artificiais. As interconexões entre esses neurônios compõem uma rede neural artificial. Uma das propriedades mais importantes de uma rede neural artificial é a capacidade de aprender por intermédio de exemplos e fazer inferências sobre o que aprendeu, melhorando gradativamente o seu desempenho. Seu aprendizado se dá por algum algoritmo de aprendizagem cuja tarefa é ajustar os pesos de suas conexões (FACELI et al, 2011).

2.2.1 Formas de aprendizado

Existem duas formas básicas de aprendizado de redes neurais: aprendizado supervisionado e aprendizado não supervisionado. No aprendizado supervisionado, um agente externo apresenta à rede neural alguns conjuntos de padrões de entrada e seus correspondentes padrões de saída. Para cada entrada é indicado explicitamente se a resposta calculada é boa ou ruim.

Na aprendizagem não supervisionada não existe um agente externo para acompanhar o processo de aprendizado, somente os padrões de entrada estão disponíveis para a rede neural. A rede processa as entradas e tenta progressivamente estabelecer representações internas para codificar características e classificá-las automaticamente.

Embora existam diversas arquiteturas de redes neurais artificiais, nem todas são de *Deep Learning*, ou, como seu nome sugere, caracterizada por modelos de aprendizagem profunda, são redes neurais artificiais com muitas camadas ocultas (ou intermediárias). As arquiteturas de uma rede neural estão descritas na Seção 2.2.3.

2.2.2 Breve histórico

Existem 3 publicações iniciais mais importantes desenvolvidas por: McCulloch e Pitts (1943), Hebb (1949), e Rosenblatt (1958) que introduziram o primeiro modelo de redes neurais simulando “máquinas”, o modelo básico de rede de auto-organização, e o modelo Perceptron de aprendizado supervisionado, respectivamente. Em 1980 Kunihiko Fukushima propõe a Neoconitron, uma rede neural de hierarquia, multicamada, que foi utilizada para o reconhecimento de caligrafia e outros problemas de reconhecimento de padrões. Em 1989 surgem os primeiros algoritmos que usavam redes neurais profundas, porém devido ao tempo necessário para efetuar o treinamento o seu uso torna-se inviável em aplicações práticas (COOPER, 2019).

Já em 1992, Weng publica o Cresceptron, um método para realizar o reconhecimento de objetos 3-D automaticamente a partir de cenas desordenadas.

O termo “aprendizagem profunda” começa a ganhar popularidade após um artigo de Hinton e Salakhutdinov (2009) demonstra como uma rede neural de várias camadas poderia ser pré-treinada uma camada por vez.

Em 2009 acontece o *workshop* NIPS sobre Aprendizagem Profunda para Reconhecimento de Voz e descobre-se que com um conjunto de dados suficientemente grande, as redes neurais não precisam de pré-treinamento e as taxas de erro caem significativamente (NIPS, 2009).

Pesquisadores do Facebook, Taigman et al. (2020) desenvolvem a DeepFace, que utiliza aprendizado profundo capaz de identificar e marcar automaticamente usuários do Facebook em fotografias.

Em 2017 é desenvolvido o AlphaGo, algoritmo do Google que mapeia a arte do jogo de tabuleiro Go e vence o campeão mundial, Lee Sedol e um torneio em Seul (SILVER et al, 2017).

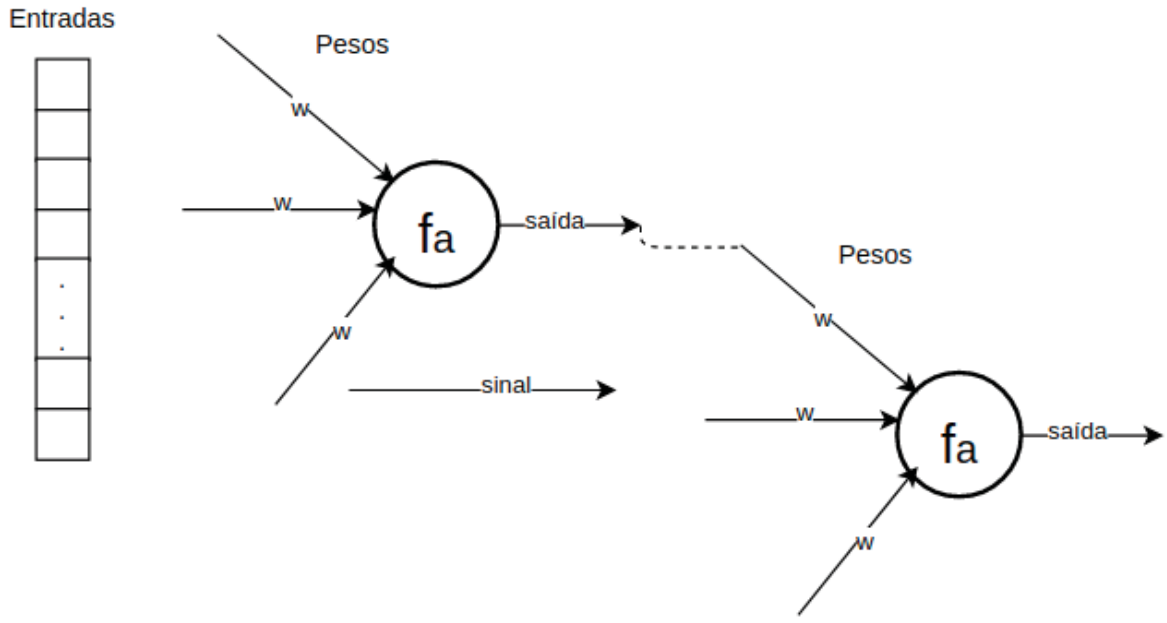
Em 2020 o AlphaFold, do Google, consegue determinar a forma 3D de uma proteína CASP (*Critical Assessment of Structure Prediction*) a partir de sua sequência de aminoácidos com 90% de precisão (CALLAWAY, 2020).

2.2.3 Arquitetura

A unidade fundamental de uma RNA é o neurônio (FIGURA 2). Cada terminal de entrada recebe um valor, que é ponderado e combinado por uma função matemática (denominada de função de ativação) resultando em uma saída. Como exemplo, suponha um objeto x com d atributos, representado na forma de vetor como $x = [x_1, x_2, \ldots, x_d]^t$, e um neurônio com d entradas com pesos $w = [w_1, w_2, \ldots, w_d]$. A entrada total recebida pelo neurônio (u) pode ser definida pela equação (1) (Faceli et al, 2011).

$$u = \sum_{j=1}^{d} x_i w_j \tag{1}$$

FIGURA 2 - NEURÔNIO ARTIFICIAL



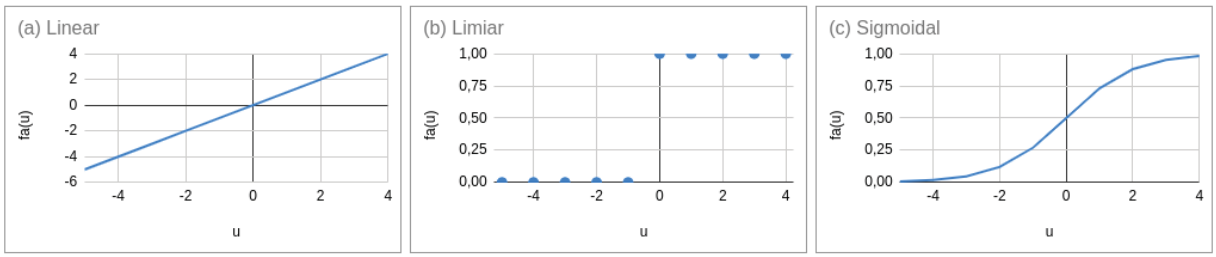
FONTE: Adaptado de Faceli et al (2011)

Os neurônios podem apresentar entradas negativas ($w_i < 0$) ou positivas ($w_i > 0$), caso seja 0, pode significar ausência de conexão associada.

Como representação de funções de ativação a FIGURA 3 demonstra: (a) linear; (b) limiar; e (c) sigmoidal. O uso da função linear identidade (a) implica em retornar como saída o valor da entrada (u). Na função limiar (b) o valor é 0 para valores negativos e 1 para os demais casos. Um neurônio é denominado ativo caso

sua saída seja 1, e inativo se 0. A função sigmoidal (c) apresenta diferentes inclinações, sua equação está representada em (2).

FIGURA 3 - FUNÇÕES DE ATIVAÇÃO



FONTE: Adaptado de Faceli et al (2011)

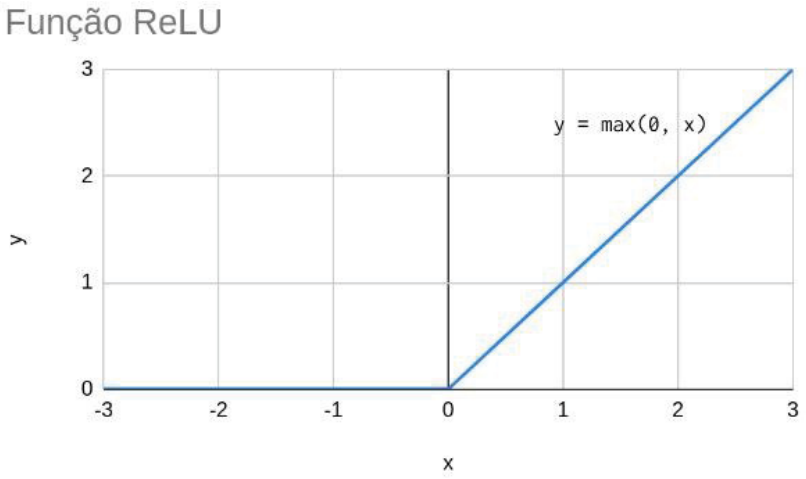
$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

Para problemas não-linearmente separáveis, pode-se utilizar como função de ativação a função ReLU. Esta função está definida na equação (3). Um exemplo está representado na FIGURA 4.

$$y = max(0, x) \tag{3}$$

FIGURA 4 - FUNÇÃO DE ATIVAÇÃO RELU

x	y
-3	0
-2	0
-1	0
0	0
1	1
2	2
3	3



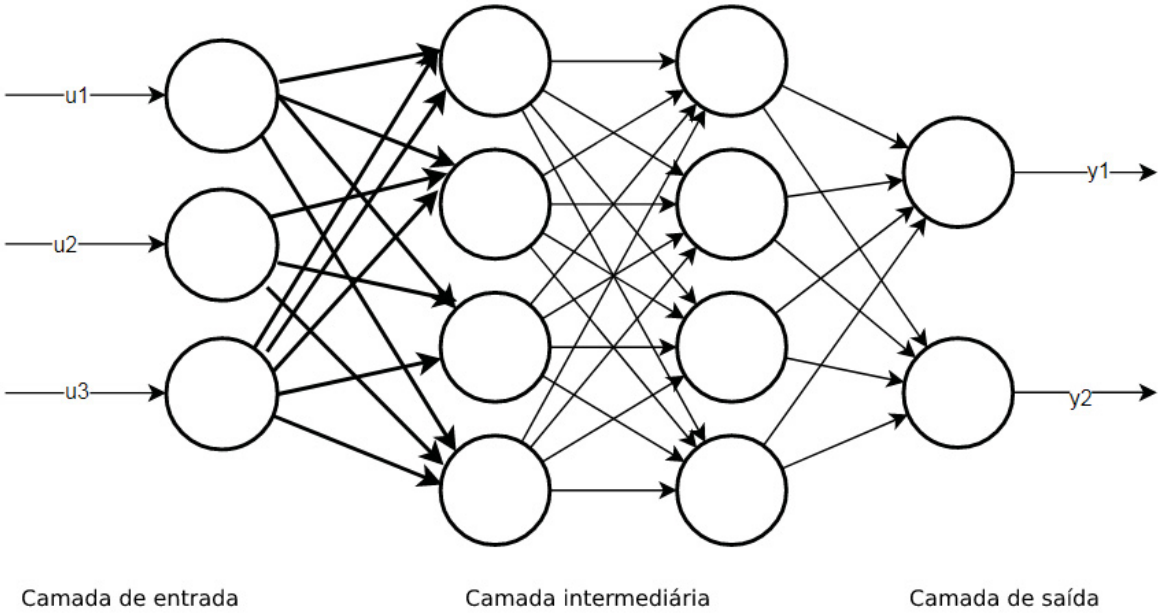
FONTE: Adaptado de Goodfellow et al (2016)

De acordo com GOODFELLOW et al (2016), ReLU comporta-se de forma linear para valores positivos e zero para os valores negativos, de forma que:

- Seu cálculo é barato. Permitindo que o modelo leve menos tempo para ser treinado;
- Devido a linearidade, possui uma conversão mais rápida. Se comparado a funções de ativação como a sigmoide e tanh, não possui o problema de gradiente de desaparecimento;
- Como ReLU é zero para entradas negativas, não ativa a saída em todos os casos.

As redes neurais podem ter uma ou várias camadas de neurônios. Redes com uma camada são indicadas para problemas que são linearmente separáveis. As redes multicamadas possuem uma ou mais camadas entre as camadas de entrada (*input layer*) e de saída (*output layer*), chamadas de camadas ocultas (*hidden layers*) (FACELI et al, 2011). A FIGURA 5 representa visualmente a camada de entrada, camada oculta (camada intermediária) e camada de saída.

FIGURA 5 – REDES NEURAIS MULTICAMADAS

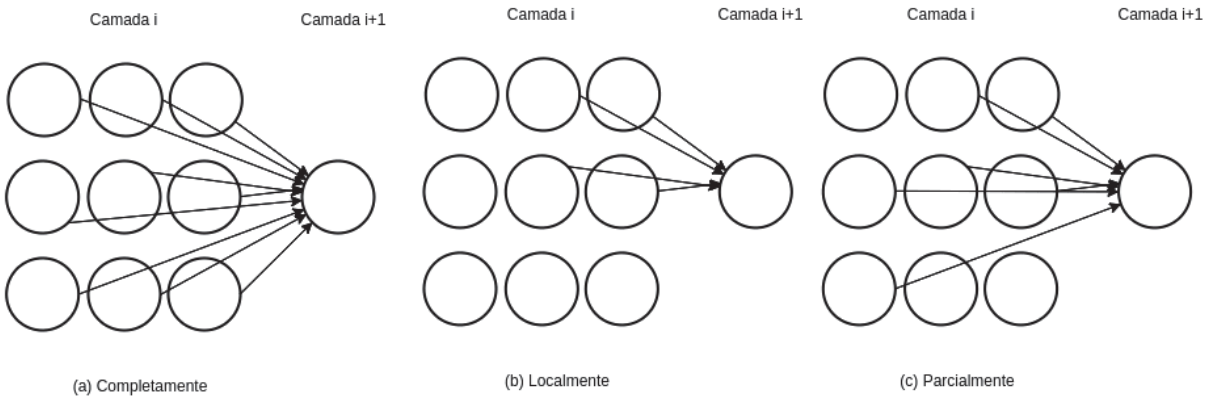


FONTE: Adaptado de Faceli et al (2011)

Em uma rede multicamadas, as conexões entre os neurônios podem ser classificadas em (FIGURA 6)(FACELI et al, 2011):

- Completamente conectada: neurônios da rede estão conectados a todos os neurônios da camada anterior e/ou seguinte;
- Parcialmente conectada: neurônios estão conectados a apenas alguns dos neurônios da camada anterior e/ou seguinte;
- Localmente conectada: redes parcialmente conectadas, na qual os neurônios conectados a um neurônio se encontram em uma região definida.

FIGURA 6 - PADRÕES DE CONEXÃO REDE MULTICAMADAS



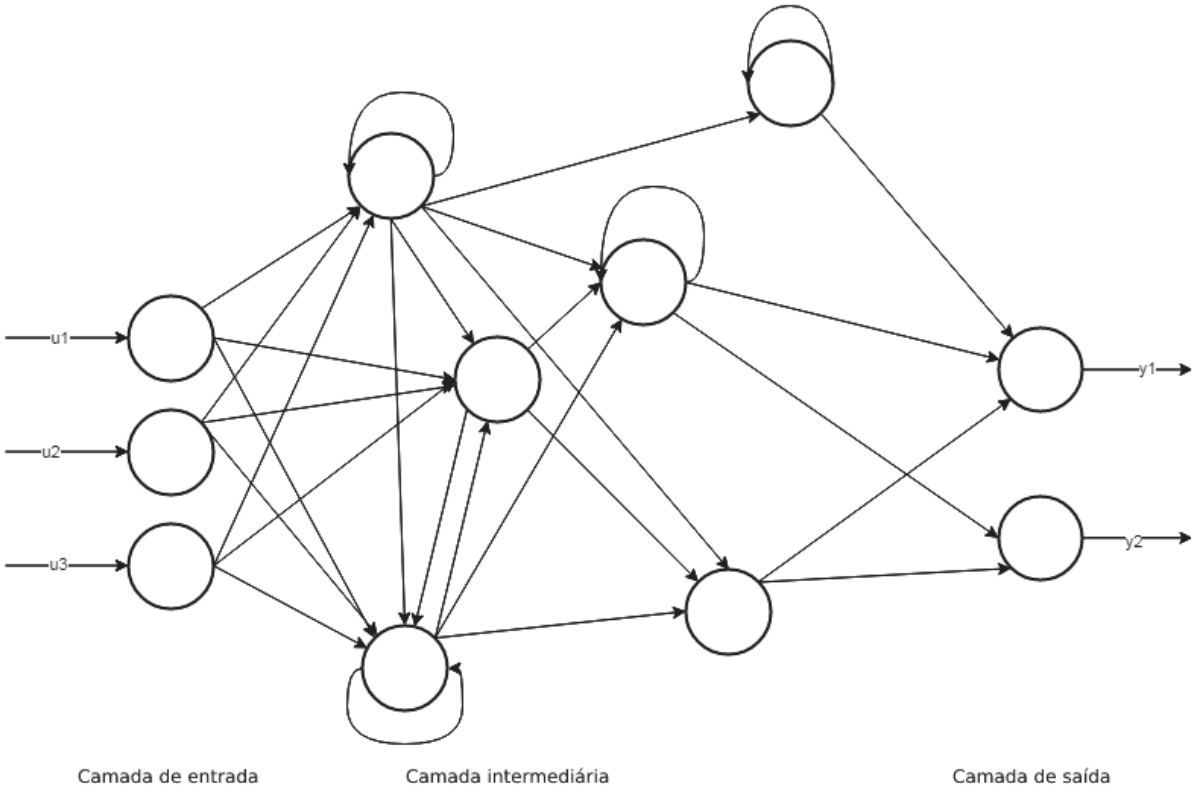
FONTE: Adaptado de Faceli et al (2011)

As arquiteturas de redes neurais podem ser descritas em 3 categorias específicas: rede neurais Feed-Forward, redes recorrentes e redes conectadas simetricamente.

Nas redes neurais Feed-Forward, a primeira camada é a entrada e a última camada é a saída. Se houver mais de uma camada oculta, são chamadas de redes neurais “profundas” (ou *deep learning*). De acordo com Cooper (2020), para aplicações práticas, estas são o tipo mais comum de rede neural.

As redes recorrentes têm pelo menos um ciclo direcionado em seu grafo de conexão, realimentando a rede, chamadas de redes com ciclos (*feedback*) (FACELI et al, 2011). A FIGURA 7 demonstra um exemplo de rede neural recorrente.

FIGURA 7 – REDES NEURAIS RECORRENTES



FONTE: Adaptado de Faceli et al (2011)

As redes conectadas simetricamente são como redes recorrentes, mas as conexões entre as unidades são simétricas (elas têm o mesmo peso em ambas as direções).

2.2.3.1 Aprendizado

Os algoritmos de treinamento podem ser divididos em quatro grupos (FACELI et al, 2011):

- Correção de erro: utilizado no aprendizado supervisionado, procura ajustar os pesos da RNA de forma a reduzir os erros;
- Hebbiano: utilizado no aprendizado não supervisionado, baseados na regra de Hebb, que reforça a conexão entre dois neurônios ativos;
- Competitivo: utilizado no aprendizado não supervisionado, promovem uma competição entre os neurônios para definir quais devem ter seus pesos ajustados;
- Termodinâmico (Boltzmann): utiliza algoritmos estocásticos, baseados em princípios da metalurgia.

A primeira RNA a ser implementada foi a rede perceptron, desenvolvida por Rosenblatt. Essa rede utiliza o modelo de McCulloch-Pitts como neurônio e introduziu o processo de treinamento supervisionado de correção de erro de RNAs. Os pesos são ajustados de acordo com a equação (4).

$$w_j(t+1) = w_j(t) + nx_i^j(y_i - \hat{f}(x_i)) \quad (4)$$

onde:

$w_j(t)$ = peso da j-ésima conexão de entrada, em um instante de tempo t.

n = taxa de aprendizado.

x_i^j = valor do j-ésimo atributo do vetor de entrada x_i.

$\hat{f}(x_i)$ = saída produzida pela rede, no instante de tempo t.

y_i = saída desejada pela rede (predição de x_i).

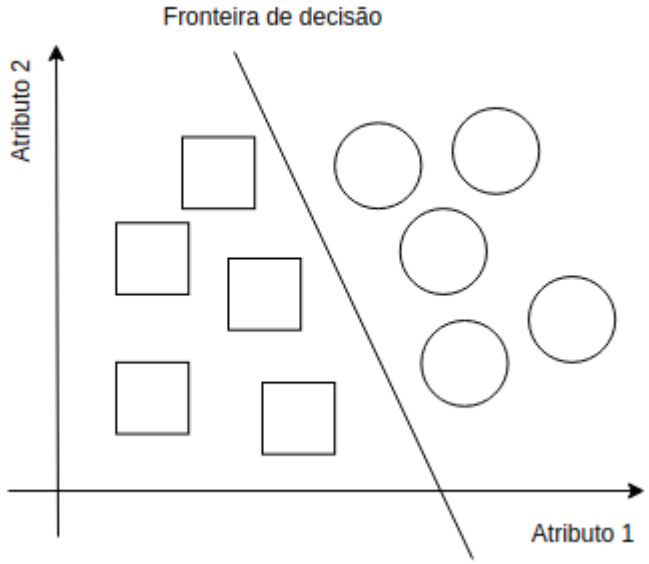
O valor da taxa de aprendizado (n) define a magnitude do ajuste feito no valor de cada peso, que define a velocidade de convergência da rede.

A rede adaline (*Adaptive Linear*) também é formada por apenas uma camada com função de ativação linear. Utiliza a regra delta (diferença entre os valores de saída desejada e produzida) para levar em consideração a magnitude do

erro no momento de ajustar os pesos da rede. Estas redes são utilizadas em problemas supervisionados de regressão.

Um dos problemas associados as redes perceptron e adaline é que elas classificam apenas problemas linearmente separáveis. Objetos de duas classes são linearmente separáveis se, em um espaço bidimensional, existir um hiperplano que os separe (FIGURA 8), caso não exista um hiperplano que os separe, diz-se que o problema não é linearmente separável (FACELI et al, 2011).

FIGURA 8 - OBJETOS LINEARMENTE SEPARÁVEIS



FONTE: Adaptado de Faceli et al (2011)

No intuito de resolver problemas não linearmente separáveis foram criadas redes perceptron multicamadas (MLP). Estas redes apresentam uma ou mais camadas intermediárias de neurônios e uma camada de saída. A arquitetura mais comum é a completamente conectada (neurônios de uma camada l estão todos conectados aos neurônios da camada l+1), a FIGURA 5 exemplifica uma rede MLP. Redes multicamadas utilizam funções de ativação não linear, como a função sigmoidal (FACELI et al, 2011).

Back-propagation é um algoritmo de descida de gradiente posposto para treinar redes multicamadas. Para que o algoritmo possa ser empregado, é preciso que a função de ativação seja contínua, diferenciável e preferencialmente não decrescente, com por exemplo, a função sigmoide (equação (2)). Baseia-se na regra delta (rede adaline), constitui-se na iteração de duas fases: uma para frente

(*forward*) e outra para trás (*backward*). Na primeira fase (*forward*) é calculado o delta, ou seja, a diferença entre os valores de saída produzidos e desejados para cada neurônio, indicando o erro. O valor de erro é utilizado na fase de *backward* para ajustar os pesos da entrada. O ajuste é feito da camada de saída até a primeira camada intermediária (FACELI et al, 2011).

2.2.4 Função de perda

Loss function, ou função de perda, é uma função utilizada para avaliar o quão bem o algoritmo modela o conjunto de dados. Quanto melhor for o modelo, melhor é a representação e menor será o valor da função de perda (*loss function*). Quanto pior for a representação, pior será o modelo e maior será o valor da função de perda. Conforme o algoritmo for sendo alterado e modificado com o objetivo de melhorá-lo, melhor será o modelo e menor será o valor da função de perda (ALGORITHMIA, 2018). A função informa o quanto estamos longe da predição ideal, ou seja, qual seria a “perda” ou “custo” a aceitarmos a predição gerada (PONTI e COSTA, 2017).

Pode-se citar como função de perda a função de entropia cruzada (*cross-entropy*). Seja um neurônio j, y o valor real e f(x) = ŷ sua predição, a soma das entropias cruzadas de cada classe j (neurônio) é dada pela equação (5) (PONTI e COSTA, 2017).

$$l^{(ce)} = -\sum_{j} y_i.log(\hat{y}_j + \varepsilon) \tag{5}$$

onde:

ε << 1 = é uma variável para evitar log(0). P.ex.: ε = 0,0001.

Como exemplo sejam os seguintes valores de y e ŷ:

y = [0,00 0,00 0,00 0,00 0,00 0,00 1,00 0,00 0,00 0,00]
ŷ = [0,18 0,00 0,00 0,02 0,00 0,00 0,65 0,05 0,10 0,00]

Então a entropia cruzada para esse exemplo seria:

$$l^{(ce)} = -(0+0+0+0+0+0-0,6214+0+0+0) = 0,6214$$

O valor desta função seria 0 apenas no caso ideal, onde y = ŷ.

Outra função é a do erro quadrático médio (EQM ou MSE, de *Mean Square Error*), sendo a soma das distâncias quadradas entre a variável-alvo (y) e os valores previstos (ŷ), representada na equação (6) (GROVER, 2021).

$$EQM = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{6}$$

O erro médio absoluto (MAE de *Mean Absolute Error*) é uma função de perda usada para modelos de regressão. MAE é a soma das diferenças absolutas entre as variáveis alvo (y) e previstas (ŷ), representada na equação (7) (GROVER, 2021).

$$MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n} \tag{7}$$

A perda de Huber é menos sensível a valores discrepantes nos dados do que a perda de erro quadrático. Caso o erro seja pequeno, torna-se um erro quadrático, caso contrário, um erro absoluto. O tamanho desse erro para torná-lo quadrático depende de um hiperparâmetro, δ (delta), que pode ser ajustado. Sua representação está na equação (8) (GROVER, 2021).

$$L_\delta(y,\hat{y}) = \begin{cases} \frac{1}{2}(y-\hat{y})^2, & \text{para } |y-\hat{y}| \leq \delta, \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2, & \text{caso contrário.} \end{cases} \tag{8}$$

Log-cosh é uma função usada em tarefas de regressão. Log-cosh é o logaritmo do cosseno hiperbólico do erro de previsão. Sua representação está na equação (9) (GROVER, 2021).

$$L(y, \hat{y}) = \sum_{i=1}^{n} log(cosh(\hat{y} - y_i)) \quad (9)$$

2.3 REDE NEURAL CONVOLUCIONAL (CNN)

A primeira aplicação eficiente de uma CNN foi feita por LeCun (1998). LeCun desenvolveu uma rede neural CNN totalmente conectada com sete camadas em convolução. Posteriormente as CNNs se tornaram mais profundas e complexas, como a AlexNet (KRIZHEVSKY, 2012), de oito camadas (três totalmente conectadas e cinco convoluções) e com sessenta milhões de parâmetros. Cita-se também a GoogleNet com quatro milhões de parâmetros e vinte e duas camadas (SZEGEDY, 2014).

A inspiração para CNNs veio de um experimento de Hubel e Wiesel (1962), que demonstraram que alguns neurônios são ativos juntos quando expostos a algumas curvas e linhas, reproduzindo o reconhecimento visual. A ideia principal das CNNs é filtrar linhas, bordas, curvas e em cada camada adicional transformar estes filtros em uma imagem mais complexa (ALVEZ, 2018). As convoluções funcionam como filtros que são passados em toda a imagem captando os traços mais relevantes. Quanto mais profundas as camadas de convolução (quantos mais filtros ou *kernel*), mais detalhes os traços irão possuir.

Uma CNN é uma rede neural utilizada para reconhecer padrões em dados, sendo composta por uma coleção de neurônios organizados em camadas, cada uma com seus próprios pesos e vieses de aprendizado. CNNs utilizam uma camada chamada de convolução, que possuem os *kernels* (filtros) que extraem características utilizadas para distinguir distintas imagens, ou seja, classificá-las (WANG, 2021). Os neurônios da camada de convolução são chamados de neurônios convolucionais. Eles realizam um produto escalar, elemento a elemento com o *kernel* e saída do neurônio da camada anterior.

A convolução é uma operação linear realizada por um filtro/*kernel* que é aplicada sobre uma imagem de entrada (GOODFELLOW et al, 2016). O *kernel* (FIGURA 10) é uma matriz utilizada na operação de matrizes. Esta operação é aplicada n vezes em diferentes regiões de uma imagem de entrada (FIGURA 9).

Aplicando o filtro (FIGURA 10) na imagem de entrada (FIGURA 9) com *stride* (tamanho do passo) de 1x1 obtemos as seguintes imagens de saída (denominadas de *feature map*) descritas na TABELA 1.

TABELA 1 – CONVOLUÇÃO *STRIDE* 1X1

Convolução	***Feature map***	**Operação**	**Resultado**
FIGURA 11	FIGURA 12	1x1 + 1x0 + 1x1 + 0x1 + 1x1 + 1x0 + 0x1 + 0x0 + 1x1	4
FIGURA 13	FIGURA 14	1x1 + 1x0 + 0x1 + 1x0 + 1x1 + 1x0 + 0x1 + 1x0 + 1x1	3
FIGURA 15	FIGURA 16	1x1 + 0x0 + 0x1 + 1x0 + 1x1 + 0x0 + 1x1 + 1x0 + 1x1	4
FIGURA 17	FIGURA 18	0x1 + 1x0 + 1x1 + 0x0 + 0x1 + 1x0 + 0x1 + 0x0 + 1x1	2
FIGURA 19	FIGURA 20	1x1 + 1x0 + 1x1 + 0x0 + 1x1 + 1x0 + 0x1 + 1x0 + 1x1	4
FIGURA 21	FIGURA 22	1x1 + 1x0 + 0x1 + 1x0 + 1x1 + 1x0 + 1x1 + 1x0 + 0x1	3
FIGURA 23	FIGURA 24	0x1 + 0x0 + 1x0 + 0x0 + 0x1 + 1x0 + 0x1 + 1x0 + 1x1	1
FIGURA 25	FIGURA 26	0x1 + 1x0 + 1x1 + 0x0 + 1x1 + 1x0 + 1x1 + 1x0 + 0x1	3
FIGURA 27	FIGURA 28	1x1 + 1x0 + 1x1 + 1x0 + 1x1 + 0x0 + 1x1 + 0x0 + 0x1	4

FONTE: O Autor (2021)

FIGURA 9 - IMAGEM DE ENTRADA

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 10 - *KERNEL* (FILTRO)

1	0	1
0	1	0
1	0	1

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 11 - *KERNEL* X IMAGEM ENTRADA 1

1x1	1x0	1x1	0	0
0x1	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 12 - MATRIZ RESULTANTE 1

4		

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 13 - *KERNEL* X IMAGEM ENTRADA 2

1	1x1	1x0	0x1	0
0	1x0	1x1	1x0	0
0	0x1	1x0	1x1	1
0	0	1	1	0
0	1	1	0	0

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 14 - MATRIZ RESULTANTE 2

4	3	

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 15 - *KERNEL* X IMAGEM ENTRADA 3

1	1	1x1	0x0	0x1
0	1	1x0	1x1	0x0
0	0	1x1	1x0	1x1
0	0	1	1	0
0	1	1	0	0

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 16 - MATRIZ RESULTANTE 3

4	3	4

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 17 - *KERNEL* X IMAGEM ENTRADA 4

1	1	1	0	0
0x1	1x0	1x1	1	0
0x0	0x1	1x0	1	1
0x1	0x0	1x1	1	0
0	1	1	0	0

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 18 MATRIZ RESULTANTE 4

4	3	4
2		

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 19 *KERNEL* X IMAGEM ENTRADA 5

1	1	1	0	0
0	1x1	1x0	1x1	0
0	0x0	1x1	1x0	1
0	0x1	1x0	1x1	0
0	1	1	0	0

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 20 - MATRIZ RESULTANTE 5

4	3	4
2	4	

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 21 - *KERNEL* X IMAGEM ENTRADA 6

1	1	1	0	0
0	1	1x1	1x0	0x1
0	0	1x0	1x1	1x0
0	0	1x1	1x0	0x1
0	1	1	0	0

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 22 - MATRIZ RESULTANTE 6

4	3	4
2	4	3

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 23 - *KERNEL* X IMAGEM ENTRADA 7

1	1	1	0	0
0	1	1	1	0
0x1	0x0	1x0	1	1
0x0	0x1	1x0	1	0
0x1	1x0	1x1	0	0

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 24 - MATRIZ RESULTANTE 7

4	3	4
2	4	3
1		

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 25 - *KERNEL* X IMAGEM ENTRADA 8

1	1	1	0	0
0	1	1	1	0
0	0x1	1x0	1x1	1
0	0x0	1x1	1x0	0
0	1x1	1x0	0x1	0

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 26 - MATRIZ RESULTANTE 8

4	3	4
2	4	3
1	3	

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 27 - *KERNEL* X IMAGEM ENTRADA 9

1	1	1	0	0
0	1	1	1	0
0	0	1x1	1x0	1x1
0	0	1x0	1x1	0x0
0	1	1x1	0x0	0x1

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 28 - MATRIZ RESULTANTE 9

4	3	4
2	4	3
1	3	4

FONTE: Adaptado de Goodfellow et al (2016)

Devido ao processo de convolução, a imagem de saída fica com uma dimensão menor do que a imagem original. Quanto maior o número de convoluções, menor será o tamanho da imagem resultante. No intuito de contornar este problema, pode-se utilizar a técnica de *Padding*, um processo no qual alguns pixels são adicionados ao redor da imagem antes da operação de convolução, de forma a manter a dimensionalidade da imagem resultante (GOODFELLOW et al, 2016). Como exemplo, ao aplicar um *padding* de mesmo valor (0) a imagem de entrada da FIGURA 29, obtém-se a imagem com *padding* da FIGURA 30.

FIGURA 29 - IMAGEM ENTRADA

1	0	0,5	0,5
0	0,5	1	0
0	1	0,5	1
1	0,5	0,5	1

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 30 - *PADDING*

0	0	0	0	0	0
0	1	0	0,5	0,5	0
0	0	0,5	1	0	0
0	0	1	0,5	1	0
0	1	0,5	0,5	1	0
0	0	0	0	0	0

FONTE: Adaptado de Goodfellow et al (2016)

Ao aplicar um filtro (FIGURA 31), obtém-se a imagem resultante (FIGURA 32) de mesma dimensão (4x4) da imagem de entrada (FIGURA 29).

FIGURA 31 - FILTRO / *KERNEL*

1	0
0	0,5

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 32 - SAÍDA

0,5	0	0,25	0,25
0	1,25	0,5	0,5
0	0,5	0,75	1,5
0,5	0,25	1,25	1

FONTE: Adaptado de Goodfellow et al (2016)

A operação de *Pooling* é um processo conhecido como *downsampling*, no qual é feito uma redução das dimensões dos mapas de atributos. Como exemplo pode-se citar a redução do tamanho da imagem (GOODFELLOW et al, 2016). Seu objetivo é diminuir a variância de pequenas alterações e reduzir a quantidade de parâmetros treinados pela rede neural. Existem três operações de *Pooling*: *MaxPooling*, *SumPooling* e *AveragePooling*. Estas operações seguem o mesmo princípio, diferindo na função aplicada (*Max*: maior valor; *Sum*: soma de valores; e *Average*: média dos valores). Como exemplo, a função *MaxPooling* seleciona o maior valor de uma região da matriz de entrada (dependendo do tamanho da *pool* aplicada). Ao aplicar a função de *MaxPooling* com filtro de 2x2 na imagem da FIGURA 33, obtém-se a imagem de saída da FIGURA 34.

FIGURA 33 - IMAGEM ENTRADA

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

FONTE: Adaptado de Goodfellow et al (2016)

FIGURA 34 - *MAXPOOLING*

6	8
3	4

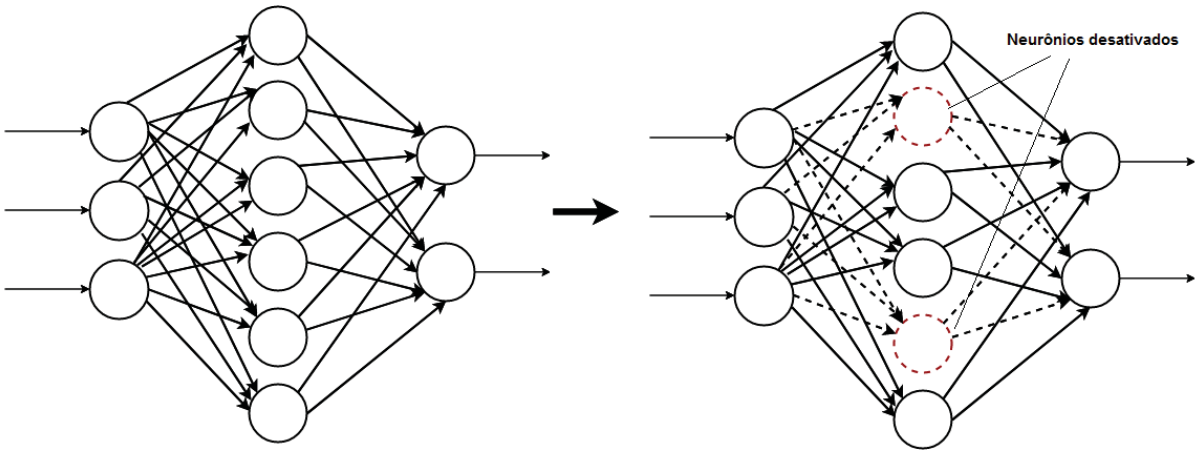
FONTE: Adaptado de Goodfellow et al (2016)

Dropout é uma técnica utilizada para diminuir o efeito de *overfitting* nas redes neurais multicamadas. Durante o treinamento descartam-se unidades aleatórias (junto com as conexões) do sistema neural com o intuito de evitar que as unidades se adaptem demais e fiquem muito dependentes dos neurônios ao seu redor. A camada de *dropout* permite desativar determinadas partes da rede neural

durante o treinamento. Tem por objetivo diminuir a sensibilidade da rede a pequenas variações. Esta técnica permite o aumento da capacidade de variação da rede neural (SRIVASTAVA et al, 2014).

O *dropout* é uma técnica que fornece uma maneira de combinar de forma exponencial e eficiente muitas arquiteturas de rede neural diferentes. Descartar uma unidade significa removê-la temporariamente da rede, junto com todas as suas conexões de entrada e saída, conforme mostrado na FIGURA 35. Uma das formas de escolha de quais unidades descartar é a forma aleatória (SRIVASTAVA et al, 2014).

FIGURA 35 - *DROPOUT*



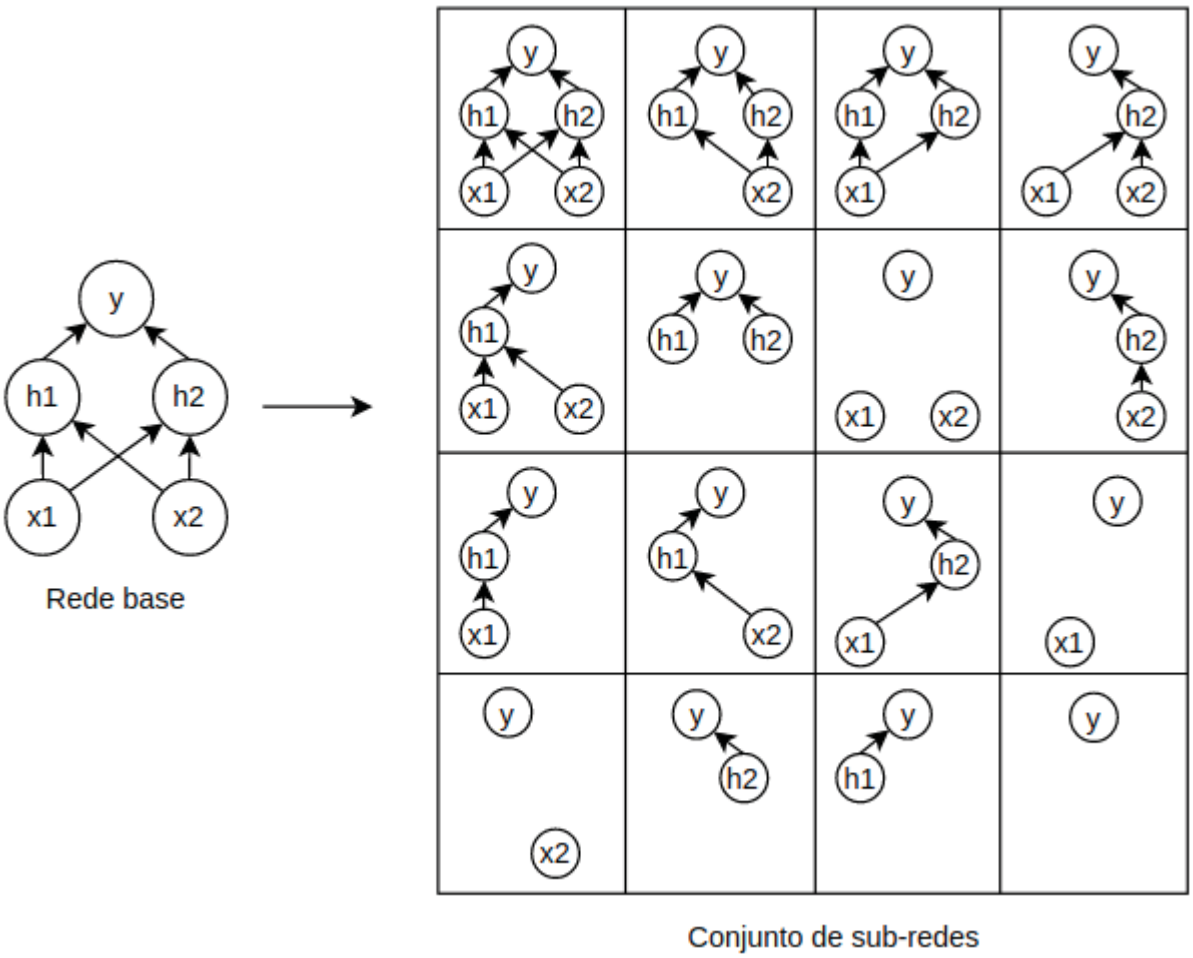
FONTE: O Autor (2021)

Treinar e avaliar diversos modelos e estruturas neurais é custoso em termos de tempo de execução e memória. De acordo com Goodfellow et al (2016) *dropout* fornece um método computacionalmente barato e poderoso para treinar e avaliar um conjunto de redes exponencialmente numerosas. Permite treinar o conjunto de todas as sub-redes que podem ser formadas removendo unidades intermediárias de uma rede básica subjacente (FIGURA 36). Uma unidade pode ser removida de uma rede multiplicando-se o seu valor por zero.

A FIGURA 36 demonstra um exemplo de redes formadas pelo *dropout* para uma rede básica com 2 entradas x1 e x2, duas camadas ocultas h1 e h2 e uma camada de saída y. Destas quatro unidades, existem 16 subconjuntos possíveis. Para o exemplo, algumas redes não possuem entradas, saída, ou nenhum caminho conectado até a saída. Este problema torna-se insignificante para redes largas, cuja

probabilidade de descarte de todas as redes possíveis torna-se menor (GOODFELLOW et al, 2016).

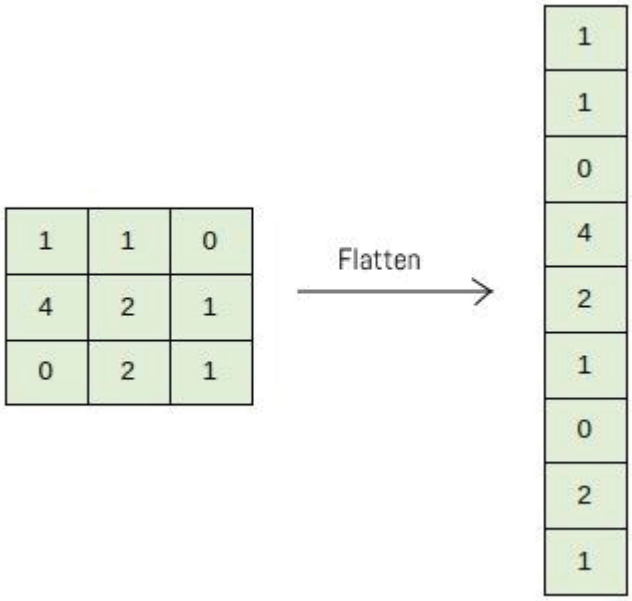
FIGURA 36 - REDES DROPOUT



FONTE: Adaptado de Goodfellow et al (2016)

A camada de *Flatten* é normalmente utilizada na extração de características de uma rede neural tradicional. Opera uma transformação na imagem, convertendo-a em um *array*. Como exemplo, a FIGURA 37 demonstra uma imagem 3x3 transformada em um *array* de 9 posições (GOODFELLOW et al, 2016).

FIGURA 37 - FLATTEN



FONTE: Adaptado de Goodfellow et al (2016)

2.3.1 R-CNN

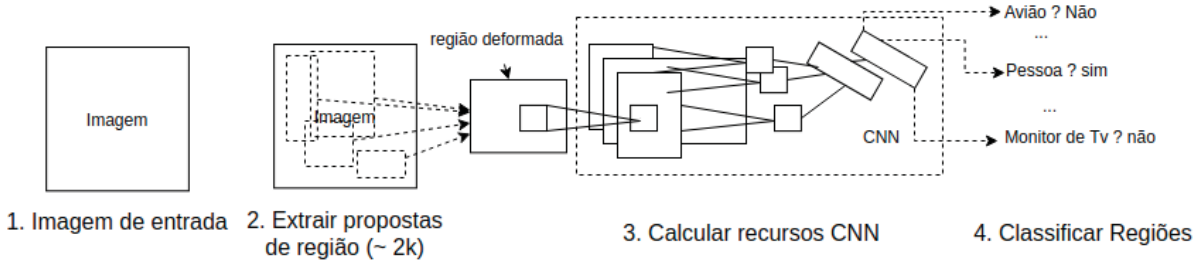
A diferença entre algoritmos de detecção de objetos e algoritmos de classificação é que nos algoritmos de detecção tenta-se desenhar uma caixa delimitadora em torno do objeto de interesse para localizá-lo na imagem (GIRSHICK et al, 2014). Podem existir n caixas delimitadoras em uma mesma imagem, representando diferentes objetos. Uma abordagem ingênua para este problema poderia ser pegar diferentes regiões de interesse da imagem e usar uma CNN para classificar o objeto naquela região. O problema nesta abordagem é que os objetos podem ter diferentes localizações dentro da imagem e diferentes proporções de tamanho. A seleção de inúmeras regiões torna-se proibitivo do ponto de vista computacional dado o seu custo.

O algoritmo R-CNN (Regiões com recursos de CNN) foi desenvolvido para encontrar as regiões de interesse (GIRSHICK et al, 2014). Para lidar com o problema de seleção de um grande número de regiões, Girshick et al. propuseram um método de busca seletiva (de regiões de interesse) que extrai 2.000 regiões da imagem, chamadas de propostas de região. A pesquisa seletiva consiste em 3 etapas:

1. Segmentação inicial, gerando regiões candidatas;
2. Aplica-se um algoritmo ganancioso, combinando as regiões semelhantes com outras maiores;
3. As regiões geradas são utilizadas para gerar as propostas finais de regiões candidatas.

A partir da imagem de entrada, extrai-se cerca de 2.000 propostas de região, estas são distorcidas em um quadrado e alimentadas em uma rede neural convolucional (FIGURA 38). Esta produz um vetor de características de saída de recursos com 4096 dimensões de saída. A CNN atua como extrator de recursos e sua saída é alimentada em um SVM (máquina de suporte de vetores) para classificar a presença do objeto dentro da proposta de região (GIRSHICK et al, 2014).

FIGURA 38 - R-CNN VISÃO GERAL



FONTE: Adaptado de GIRSHICK (2014)

O sistema de detecção de objetos consiste em três módulos:

1. O primeiro gera propostas de regiões. Estas definem o conjunto de detecções candidatas disponíveis;
2. O segundo módulo é composto por uma rede neural convolucional que extrai um vetor de características de comprimento fixo de cada região;
3. O terceiro módulo é um conjunto de SVMs utilizados para classificação.

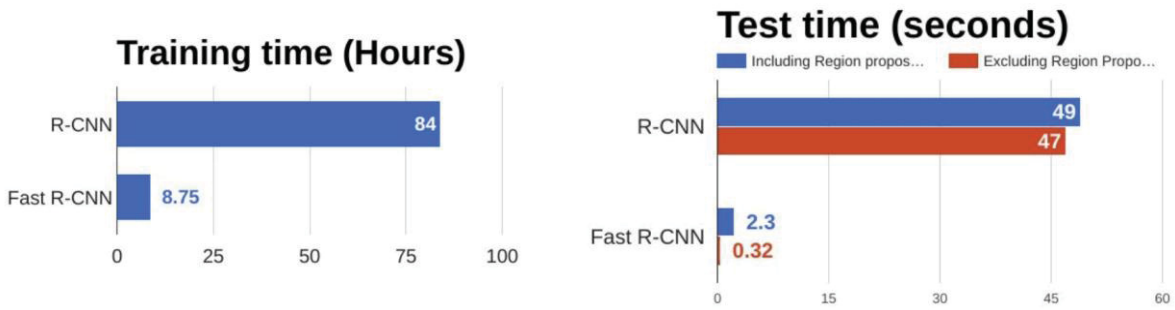
2.3.2 Fast R-CNN

A abordagem do Fast R-CNN é semelhante ao R-CNN, com a diferença de que a imagem de entrada é alimentada na CNN com o objetivo de gerar um mapa de recursos convolucionais. A partir deste mapa são identificadas a região das

propostas. Estas posteriormente são alimentadas em uma camada totalmente conectada (*fully connected layer*). A classe da região proposta é prevista, assim com os valores de deslocamento da caixa delimitadora. Esta abordagem é mais rápida do que a R-CNN porque não é alimentado com 2.000 propostas de região, mas a operação de convolução é feita apenas uma vez por imagem, gerando um mapa de características.

A FIGURA 39 exibe uma comparação entre os tempos de treinamento e detecção de uma rede R-CNN e Fast R-CNN. Sendo o tempo de treinamento e teste menor na rede Fast R-CNN do que na R-CNN.

FIGURA 39 – COMPARAÇÃO DE ALGORITMOS DE DETECÇÃO DE OBJETOS



FONTE: Adaptado de Girshick (2015)

2.4 *SINGLE SHOT DETECTOR* (SSD)

O *single shot detector* (SSD) é um método para detectar objetos em imagens usando uma rede neural profunda. Ele discretiza o espaço de saída das caixas delimitadoras em um conjunto de caixas padrão em diferentes proporções e escalas por localização do mapa de características (LIU et al, 2016). No momento da predição, a rede pontua a classe do objeto para cada caixa padrão e produz ajustes para melhor corresponder a forma do objeto. O algoritmo elimina a necessidade de geração de proposta de pixel subsequente e reamostragem, encapsulando todos os cálculos em apenas uma rede, o que facilita o seu treinamento.

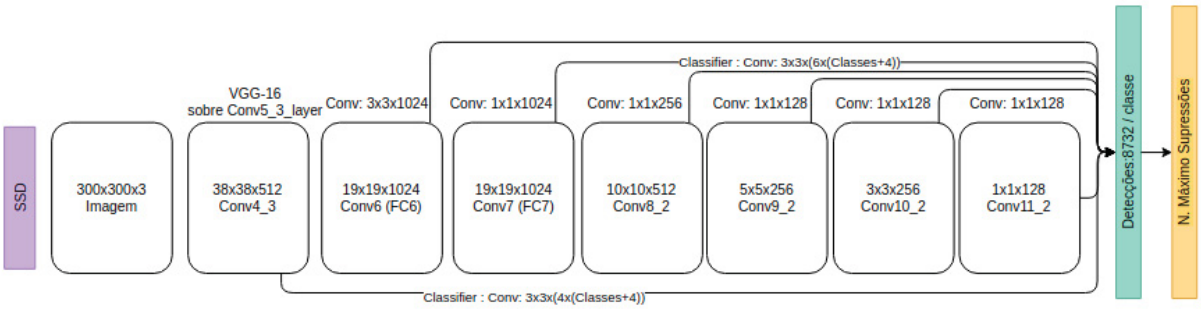
O modelo SSD de classificação de imagens possui um conjunto de mapas de características de tamanho e resolução decrescente, criando um conjunto de mapas voltado para detectar objetos em diferentes tamanhos. São definidos 4 a 6 caixas de detecção padrão para cada mapa de características, de diferentes tamanhos e proporções no intuito de ampliar a variedade de formas a serem

detectadas (JUNIOR, 2018). São produzidas previsões de diferentes escalas a partir dos mapas de recursos.

Os sistemas de detecção de objetos utilizam a seguinte abordagem: criar hipóteses de caixas delimitadoras e aplicar um classificador. O principal problema nesta abordagem é a velocidade de classificação para aplicações de tempo real (LIU et al, 2016). O detector de precisão mais rápido, Faster R-CNN, opera a uma taxa de 7 FPS (quadros por segundo), e as abordagens voltadas para diminuir este tempo resultam em uma redução significativa na precisão de detecção.

O SSD é baseado em rede profunda convolucional e não realiza uma nova amostragem de pixels ou hipóteses de caixa delimitadoras. A melhoria na velocidade vem da eliminação das propostas da caixa delimitadora e do pixel subsequente ou estágio de reamostragem de recursos. A rede utiliza um filtro convolucional para prever categorias de objetos e deslocamentos em locais de caixa delimitadora, utilizando filtros (preditores) em vários mapas de recursos para realizar a detecção em múltiplas escalas (LIU et al, 2016). A representação da rede está na FIGURA 40.

FIGURA 40 - REDE SSD



FONTE: Adaptado de Liu et al (2016)

O treinamento da SSD é feito para várias categorias de objetos. Seja x_{ij}^{p} = {1,0} um indicador para combinar a i-ésima caixa padrão com P j-ésima caixa de categoria, sendo $\sum_i x_{ij}^{p}$ >= 1. A função de perda geral é dada pela equação (10) (LIU et al, 2016).

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \quad (10)$$

onde:

N = número de caixas padrão correspondentes.

α = termo de ponderação, valor = 1.

l = caixa prevista.

g = caixa real.

c = centro da caixa prevista.

x = posição x.

L_{conf} = perda de confiança.

L_{loc} = perda de localização.

A função de perda de localização está representada na equação (11).

$$L_{loc}(x,l,g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{k} smooth_{L1}(l_i^m - \hat{g}_j^m)$$
$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \quad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h \tag{11}$$
$$\hat{g}_j^w = \log(\frac{g_j^w}{d_i^w}) \qquad \hat{g}_j^h = \log(\frac{g_j^h}{d_i^h})$$

onde:

c = centro.

d = caixa delimitadora padrão (default).

N = número de caixas padrão correspondentes.

l = caixa prevista.

g = caixa real.

ĝ = previsão da caixa delimitadora.

c = centro da caixa prevista.

x e y = posição.

w e h = largura e altura.

L_{loc} = perda de localização.

A função de perda de confiança está representada na equação (12).

$$L_{conf}(x,c) = -\sum_{i \in Pos}^{N} x_{ij}^{p} \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0) \quad onde \quad \hat{c}_i^p = \frac{exp(c_i^p)}{\sum_p exp(c_i^p)} \tag{12}$$

onde:

c = centro.

ĉ = centro previsto.

N = número de caixas padrão correspondentes.

l = caixa prevista.

g = caixa real.

ĝ = previsão da caixa delimitadora.

x = posição.

L_{conf} = perda de confiança.

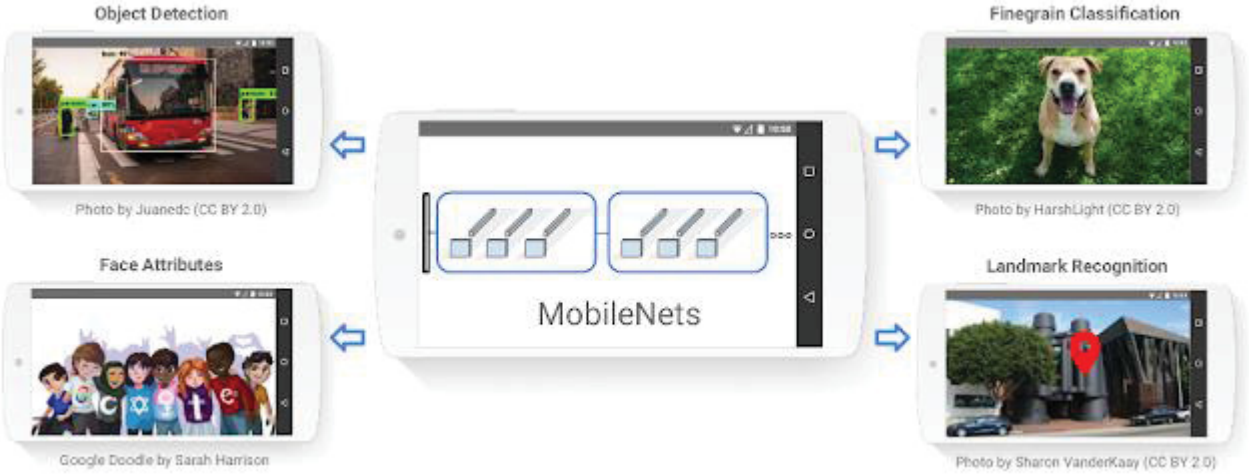
2.5 MOBILENETS

MobileNets é uma classe de modelos eficientes, pequenos, de baixa latência, parametrizados para aplicativos móveis. São baseadas em uma arquitetura de redes neurais convolucionais simplificada que utiliza convoluções separáveis em profundidade para criar redes neurais profundas e leves em termos de contagem de parâmetros e complexidade computacional. A arquitetura da rede possui dois hiperparâmetros globais: largura e o multiplicador de resolução que pode controlar o número de canais de entrada e saída das camadas de convolução e a resolução dos dados de entrada (altura, largura).

Podem ser construídos para classificação, detecção, incorporação, atributos de face, segmentação e localização geográfica em grande escala (HOWARD et al, 2017). O modelo utiliza um núcleo de tamanho 3x3 chamado de *depthwise separable convolution* que utiliza entre 8 e 9 vezes menos poder de processamento do que as convoluções comuns. O método divide a convolução em duas etapas: a primeira, *depthwise convolution* é uma convolução aplicada em um único canal da imagem (em vez de ser feita nos M canais). A segunda etapa, *pointwise convolution*, consiste numa convolução 1x1 que cria a combinação linear dos mapas de características gerados pela primeira etapa (HOWARD et al, 2017).

De acordo com Badawy (2021), mobilenet é uma arquitetura CNN computacionalmente eficiente, projetada para dispositivos móveis com capacidade de computação limitada. Pode ser usada para diferentes aplicações, incluindo: classificação finegrain, detecção de objetos, atributos faciais e geo-localização (FIGURA 41). Entre as áreas de aprendizado profundo móvel, a visão computacional continua sendo uma das mais desafiadoras (RODRIGUEZ, 2021).

FIGURA 41 - APLICAÇÕES DE MOBILENET



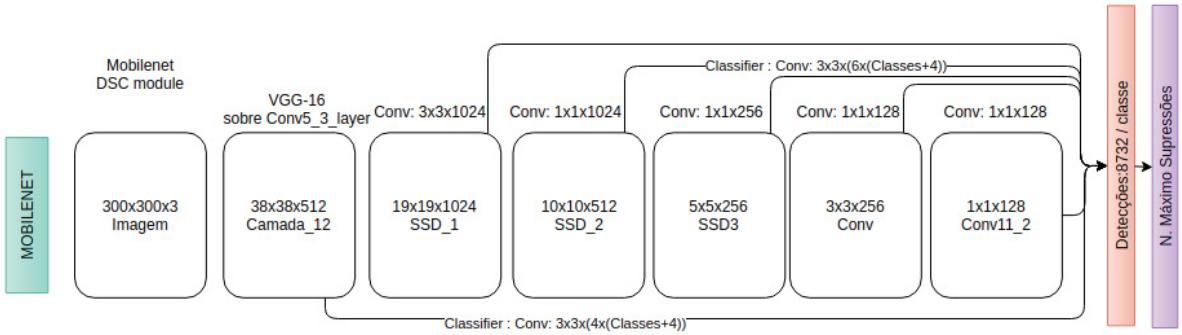
FONTE: Rodriguez (2021)

Em 2017 o Google desenvolveu uma implementação de mobilenet baseada numa série de modelos de visão computacional feitos no TensorFlow (Seção 2.16), com o objetivo de maximizar a precisão de forma a atender aos recursos restritos de um dispositivo móvel ou embarcado (RODRIGUEZ, 2021). Seus objetivos estão em criar um modelo menor (menos parâmetros) e menor complexidade (menos funções de adição e multiplicação). Esta abordagem gera modelos menores, de baixa latência e baixo consumo energético.

2.6 SSD MOBILENETS

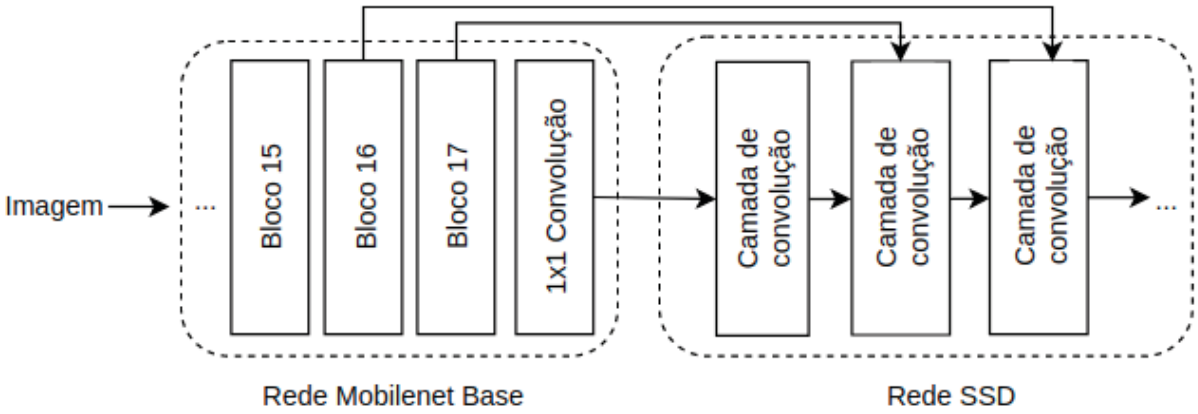
A SSD MobileNet é uma rede neural convolucional, que aprende a prever os locais onde objetos estão em uma imagem (SSD). Consiste em uma arquitetura de MobileNet seguida por várias camadas de convolução, conforme representação da FIGURA 42 (SINGHAL, 2020). As camadas da Rede Mobilenet servem de fundação para o modelo, realizando a extração de características para as camadas da rede convolucional SSD. São removidas as camadas finais de classificação da rede Mobilenet e a saída alimenta a rede de classificação SSD (FIGURA 43) (JUNIOR, 2018).

FIGURA 42 - SSD MOBILENETS



FONTE: Adaptado de Singhal (2019)

FIGURA 43 - ESTRUTURA REDE SSD MOBILENET



FONTE: Adaptado de Junior (2018)

Um exemplo de estrutura da rede está representado na TABELA 2. Sendo “Data” a camada de entrada que recebe a imagem.

TABELA 2 - ESTRUTURA REDE SSD MOBILENET

continua

#	Nome	Tamanho do filtro	Tamanho da saída	Número de caixas de detecção padrão
1	Data	-	300x300x3	-
2	Conv0	3x3x3x32	150x150x32	-
3	Conv1/dw	3x3x32x32	150x150x32	-
4	Conv1	1x1x32x64	150x150x64	-
5	Conv2/dw	3x3x64x64	75x75x64	-
6	Conv2	1x1x64x128	75x75x128	-
7	Conv3/dw	3x3x128x128	75x75x128	-
8	Conv3	1x1x128x128	75x75x128	-
9	Conv4/dw	3x3x128x128	38x38x128	-
10	Conv4	1x1x128x256	38x38x256	-
11	Conv5/dw	3x3x256x256	38x38x256	-
12	Conv5	1x1x256x256	38x38x256	-
13	Conv6/dw	3x3x256x256	19x19x256	-
14	Conv6	1x1x256x512	19x19x512	-
15	Conv7/dw	3x3x512x512	19x19x512	-
16	Conv7	1x1x512x512	19x19x512	-
17	Conv8/dw	3x3x512x512	19x19x512	-
18	Conv8	1x1x512x512	19x19x512	-
19	Conv9/dw	3x3x512x512	19x19x512	-
20	Conv9	1x1x512x512	19x19x512	-
21	Conv10/dw	3x3x512x512	19x19x512	-
22	Conv10	1x1x512x512	19x19x512	-
23	Conv11/dw	3x3x512x512	19x19x512	-
24	Conv11	1x1x512x512	19x19x512	3
25	Conv12/dw	3x3x512x512	10x10x512	-
26	Conv12	1x1x512x1024	10x10x1024	-
27	Conv13/dw	3x3x1024x1024	10x10x1024	-

TABELA 2 - ESTRUTURA REDE SSD MOBILENET

conclusão

#	Nome	Tamanho do filtro	Tamanho da saída	Número de caixas de detecção padrão
28	Conv13	1x1x1024x1024	10x10x1024	6
29	Conv14_1	1x1x1024x256	10x10x256	-
30	Conv14_2	3x3x256x512	5x5x512	6
31	Conv15_1	1x1x512x128	5x5x128	-
32	Conv15_2	3x3x128x256	3x3x256	6
33	Conv16_1	1x1x256x128	3x3x128	-
34	Conv16_2	3x3x128x256	2x2x256	6
35	Conv17_1	1x1x256x64	2x2x64	-
36	Conv17_2	3x3x64x128	1x1x128	6

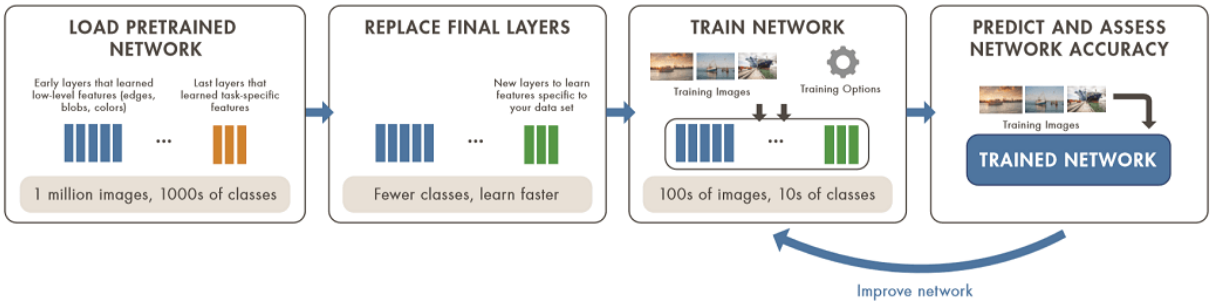
FONTE: Adaptado de Junior (2018)

As primeiras 28 camadas são do modelo treinado Mobilenet, as demais camadas são do modelo SSD. As camadas 24, 28, 30, 32, 34 e 36 representam os mapas de características que possuem as caixas de detecção padrão, e responsáveis por alimentar as camadas de convolução (SSD) extratoras de detecções (JUNIOR, 2018).

2.7 *TRANSFER LEARNING*

Transfer Learning (aprendizado por transferência) é uma abordagem de aprendizado profundo que utiliza modelos pré-treinados para gerar novos modelos poupando recursos computacionais de tempo e processamento no treino de redes neurais (GOODFELLOW et al, 2016). No aprendizado por transferência retiram-se as camadas de uma rede pronta, já treinada, e colocam-se novas camadas sobre a rede. O treinamento será feito sobre as novas camadas adicionadas (FIGURA 44).

FIGURA 44 – TRANSFER LEARNING



FONTE: Andrews (2019)

Conforme representação na FIGURA 44, após uma rede neural ser treinada, as camadas finais são removidas, e substituídas por uma nova rede neural. A saída das camadas iniciais é passada para uma nova rede neural, esta é treinada para reconhecer os padrões de interesse, no caso deste trabalho, o reconhecimento de cédulas de real.

O aprendizado por transferência é uma adaptação de domínio no qual o que foi aprendido em um ambiente (P1) é utilizado para melhorar a generalização de outro ambiente (P2). Por exemplo, o primeiro cenário (P1) envolve o aprendizado sobre figuras de gatos e cachorros e no segundo cenário (P2), aprender sobre formigas e vespas. Se houver significativamente mais amostras no primeiro cenário (P1) isso pode ajudar a aprender representações úteis para generalizar rapidamente a partir de poucos exemplos de P2. Categorias visuais compartilham noções de baixo nível de formas visuais e bordas, mudanças de iluminação, alterações geométricas, etc. Esta semântica de entrada pode ser compartilhada entre as redes (GOODFELLOW et al, 2016).

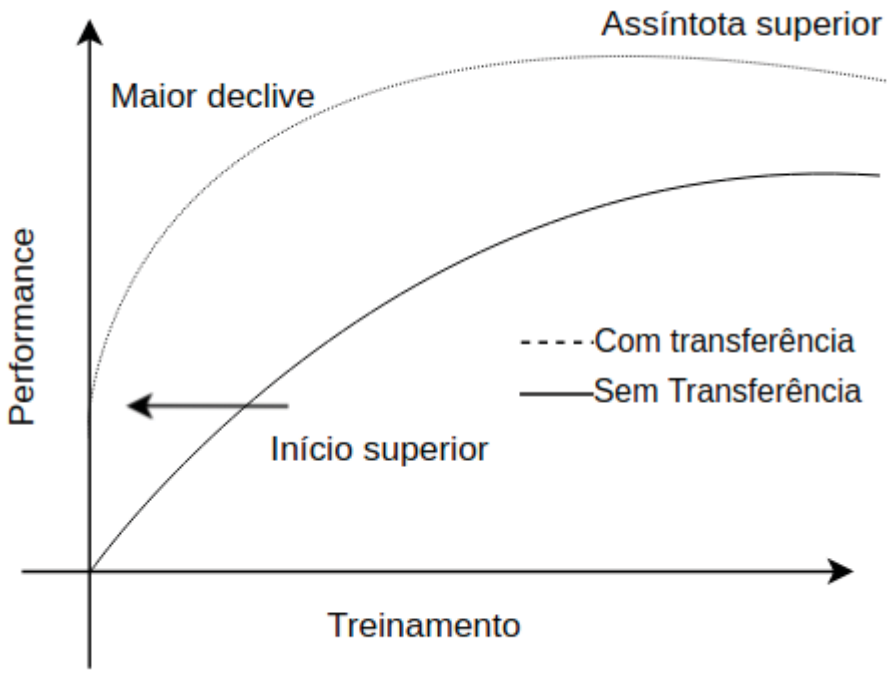
Na aprendizagem por transferência inicialmente treinamos uma rede de base em um conjunto de dados de base e em seguida redirecionamos os recursos aprendidos (transferimos) para uma segunda rede destino para ser treinada em um conjunto de dados destino. Este processo tende a funcionar caso os recursos forem adequados para as tarefas de base e alvo, ao invés de serem específicos para a rede destino (YOSINSKI, 2014).

O objetivo da aprendizagem por transferência é melhorar a aprendizagem na tarefa alvo, alavancando o conhecimento da tarefa de origem. A melhora do desempenho é chamada de transferência positiva. Caso um método de transferência

diminua o desempenho, ocorre transferência negativa. Olivas et al (2009) citam três benefícios do aprendizado por transferência (FIGURA 45):

1. Início superior (ou desempenho inicial): A habilidade inicial (antes de refinar o modelo) no modelo de origem é maior do que seria;
2. Maior declive: A taxa de melhoria de habilidade durante o treinamento do modelo de origem é mais acentuada;
3. Assíntota superior: Melhora a habilidade convergente do modelo treinado.

FIGURA 45 - APRENDIZADO POR TRANSFERÊNCIA



FONTE: Adaptado de Olivas et al (2009)

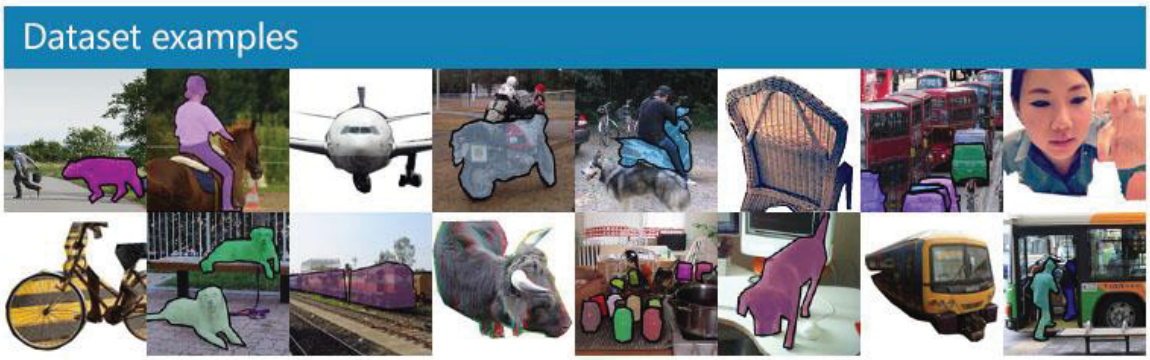
2.8 COMMON OBJECTS IN CONTEXT (COCO)

O *dataset* COCO foi criado a partir da segmentação de cenas complexas cotidianas e de objetos comuns, divididos em 91 classes, totalizando 2,5 milhões de instâncias rotuladas em imagens de 328 kb (COCO, 2020). possui diversos recursos, entre eles: Segmentação de objetos; Reconhecimento em contexto; Segmentação pan-óptica; 330 mil imagens (> 200 mil marcadas); 1,5 milhão de instâncias de

objetos; 80 categorias de objetos; 91 categorias de materiais; 5 legendas por imagem; 250.000 pessoas pontos-chave;

A FIGURA 46 demonstra um exemplo de imagens de reconhecimento definidas neste *dataset*.

FIGURA 46 – EXEMPLO *DATASET* COCO



FONTE: Coco (2020)

2.9 *DATA AUGMENTATION*

A finalidade da técnica d*ata augmentation* (DA) é aumentar o número de dados do conjunto de treinamento supervisionado. O principal objetivo é melhorar o treinamento da rede neural, obtendo-se melhores predições e melhorando assim seus resultados (OLIVEIRA, 2019).

Um classificador recebe uma entrada x e o identifica em uma classe y. Uma tarefa que ele encontra é ser invariante a ampla variedade de transformações que a entrada x pode ter. Desta forma, podemos gerar novos pares (x, y) transformando as entradas x do conjunto de treinamento. A técnica de aumento de dados tem se mostrado eficaz para problemas de classificação de reconhecimento de objetos. Operações como girar a imagem (*flips* horizontais, rotações de 180°, etc.) ou redimensioná-la podem ser utilizadas para o aumento de dados (GOODFELLOW et al, 2016).

Outra operação de aumento de dados é a injeção de ruído (*noise*) na entrada de uma rede neural. Neste intuito pode-se treinar a rede com um ruído aleatório aplicado na entrada ou às unidades ocultas da rede (aumento do conjunto de dados à nível de abstração) (GOODFELLOW et al, 2016).

Para uma base de imagens, pode-se utilizar a geração de imagens sintéticas através de redes neurais como a GAN. Goodfellow et al (2014) propuseram a criação de um framework baseado na geração de dados sintéticos, gerados através de duas redes neurais. Um modelo seria o gerador (G) e outro, discriminador (D). O modelo G (gerador) é responsável por gerar os dados, enquanto que o modelo D (discriminador) é responsável por avaliar a autenticidade das imagens geradas pelo modelo G. A ideia é que ambos os modelos trabalhem em conjunto, melhorando suas funções concomitantemente. Estes modelos podem ser implementados através de redes neurais, uma geradora e outra discriminadora.

Pode-se pensar no modelo G como análogo a uma equipe de falsificadores, tentando produzir uma nota falsa e usá-la sem detecção. O modelo D seria análogo à polícia, tentando detectar a moeda falsificada. O modelo G é treinado de forma a maximizar a probabilidade de D cometer um erro. A competição leva a ambos os modelos a melhorar seus métodos até que as falsificações sejam indistinguíveis das notas verdadeiras (GOODFELLOW et al, 2014).

A estrutura de modelagem adversarial é mais direta de aplicar quando os modelos são perceptrons multicamadas. Seja o gerador p_g sobre os dados x, são definidas variáveis de ruído de entrada $p_z(z)$. O mapeamento para o espaço de dados é definido como $G(z; \theta_g)$. Sendo G uma função diferenciável representada por uma rede perceptron multicamadas com parâmetros θ_g. Um segundo perceptron multicamadas é definido como $D(x, \theta_d)$ e produz um único escalar. D(x) representa a probabilidade de que x veio dos dados ao invés de p_g. D é treinada para maximizar a probabilidade de distinguir os rótulos corretos aos exemplos de treinamento e às amostras de G. G é treinada para minimizar log(1 - D(G(z))). Esta interação é representada por uma equação minmax (minimizar perdas e maximizar ganhos) em função do valor de V(D, G) (equação (13)) (GOODFELLOW et al, 2014).

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log(D(x))] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (13)$$

Um exemplo de aplicação da técnica de DA está na estrutura SSD. A principal desvantagem desta estrutura é o fato de que seu desempenho é diretamente proporcional aos tamanhos dos objetos. Comparando-se o desempenho de abordagens como a R-CNNs para classificação de objetos com tamanhos

pequenos, a SSD obtém um resultado inferior por não conseguir extrair informações muito úteis nas camadas superiores da rede. Desta forma técnicas de *data augmentation* são utilizadas para cortar e redimensionar aleatoriamente certas partes da imagem para ajudar a rede a identificar e aprender características destes conjuntos de dados (KUNAR, 2021).

2.10 MEDIDAS DE AVALIAÇÃO

Entre as medidas de avaliação de um modelo de rede neural, citam-se: matriz de confusão, acurácia, sensibilidade, especificidade, PPV (valor preditivo positivo), VPN (valor preditivo negativo), prevalência, ROC (*receiver operator characteristic curve*) e função de perda, descritas a seguir.

2.10.1 Matriz de confusão

A representação das frequências de classificação para cada classe do modelo pode ser feita através de uma matriz de confusão (BARANAUSKAS, 2020). Nesta, as linhas representam os valores preditos e as colunas os valores reais (QUADRO 1).

QUADRO 1 – EXEMPLO MATRIZ CONFUSÃO

		Valores Reais	
		Valor Positivo	Valor Negativo
Valores Preditos	Valor Positivo	VP Verdadeiro Positivo	FP Falso Positivo
	Valor Negativo	FN Falso Negativo	VN Verdadeiro Negativo

FONTE: Adaptado de Kawamura (2002)

O termo verdadeiro positivo (VP) é dado para casos em que o teste de um valor resulta em positivo e o valor observado é positivo. Verdadeiro negativo (VN) é quando o teste indica o valor negativo e o valor é negativo. São casos em que o teste avaliou corretamente o valor.

Os casos em que o teste falha, são chamados de falso positivo (FP) e falso negativo (FN). O primeiro, FP, ocorre quando o teste indica um valor positivo, porém

o valor é negativo. E o FN ocorre quando o teste indica um valor negativo e o valor é positivo.

2.10.2 Total de predições

O total de predições é dado pela fórmula representada na equação (14). (KAWAMURA, 2002). Sendo a somatória dos valores de VP, FP, VN e FN.

$$Total = VP + FP + VN + FN \tag{14}$$

2.10.3 Acurácia

A acurácia avalia a proporção de todas as predições corretas (verdadeiro positivo e verdadeiro negativo) sobre todos os resultados obtidos. Sua representação é dada pela fórmula descrita na equação (15) (KAWAMURA, 2002).

$$Acurácia = (VP + VN)/total \tag{15}$$

onde:

total = é o número total de todas as predições (Seção 2.10.2, equação (14)).

2.10.4 Sensibilidade

A sensibilidade é a probabilidade de se obter um resultado positivo entre os valores que são positivos. Sua fórmula está representada na equação (16) (KAWAMURA, 2002).

$$Sensibilidade = VP/(VP + FN) \tag{16}$$

2.10.5 Especificidade

A especificidade é a probabilidade de se obter um resultado negativo entre os valores que são negativos. Sua fórmula está representada na equação (17) (KAWAMURA, 2002).

$$Especificidade = VN/(VN + FP) \tag{17}$$

2.10.6 PPV

O valor preditivo positivo (VPP, ou PPV - *positive predictive value*) é a probabilidade de se obter um resultado positivo entre as predições positivas. Sua fórmula está representada na equação (18) (KAWAMURA, 2002).

$$VPP = VP/(VP + FP) \tag{18}$$

2.10.7 VPN

O valor preditivo negativo (VPN, ou NPV - *negative predictive value*), é a probabilidade de se obter um resultado negativo entre as predições negativas. Sua fórmula está representada na equação (19) (KAWAMURA, 2002).

$$VPN = VN/(VN + FN) \tag{19}$$

2.10.8 Prevalência

A prevalência representa a fração de valores positivos sobre todos os resultados obtidos. Sua representação é dada pela fórmula descrita na equação (20) (KAWAMURA, 2002).

$$Prevalência = (VP + FN)/total \tag{20}$$

onde:

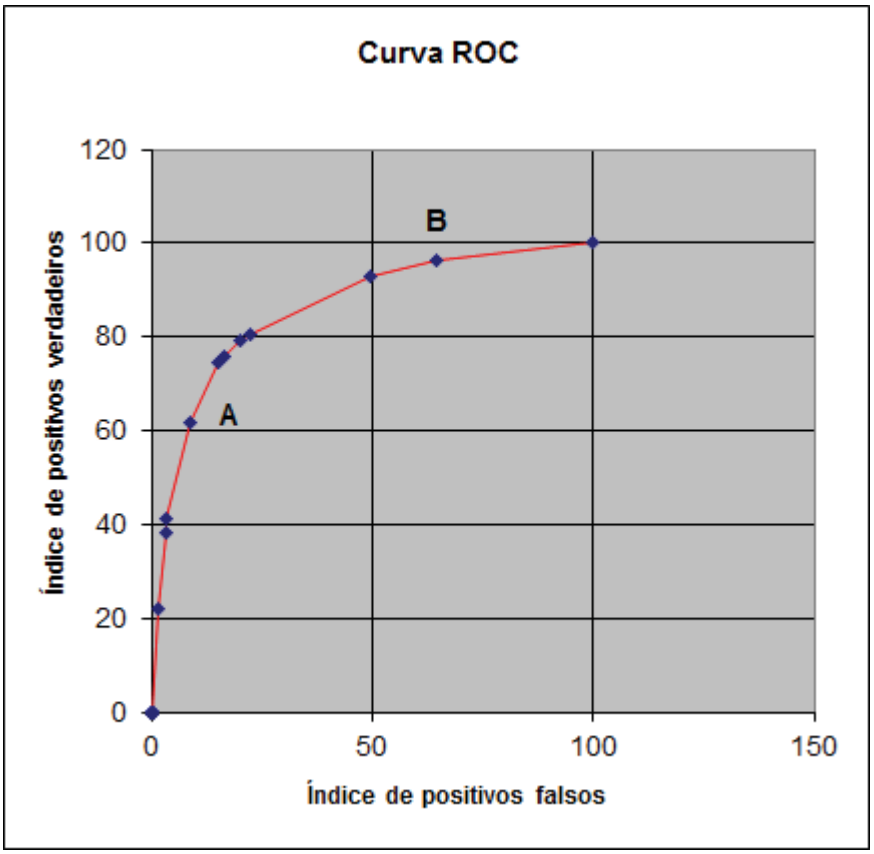
total = é o número total de todas as predições (Seção 2.10.2, equação (14)).

2.10.9 ROC

ROC (*receiver operator characteristic curve*) é uma forma gráfica de representar a relação entre a especificidade e a sensibilidade com o objetivo de ilustrar o desempenho de um modelo classificador. A Curva ROC é um gráfico de

sensibilidade (ou taxa de verdadeiros positivos) versus taxa de falsos positivos (GRÁFICO 1) (MARGOTTO, 2010). Os pontos que se encontram mais próximos ao canto superior esquerdo são os que possuem melhor otimização da sensibilidade em função da especificidade. Desta forma, quanto mais próximo os pontos estiverem desta área, melhor o modelo. Como exemplo, o ponto A possui menor sensibilidade e maior especificidade, enquanto o ponto B possui maior sensibilidade e menor especificidade (MARGOTTO, 2010).

GRÁFICO 1 - CURVA ROC



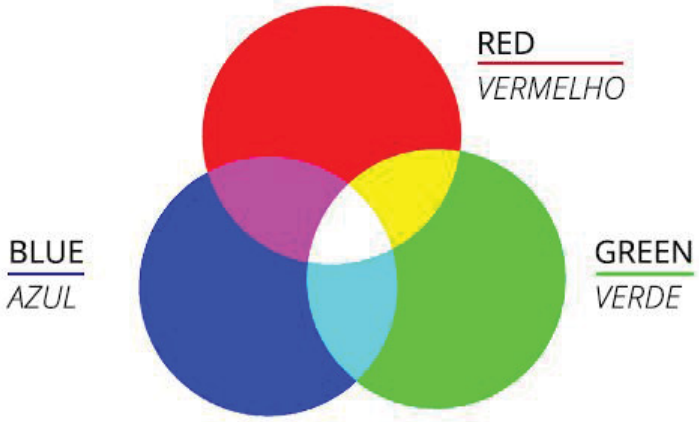
FONTE: Adaptado de Margotto (2010)

2.11 IMAGEM COLORIDA

Dentre o espectro de cores visíveis pelo ser humano, as cores podem ser representadas por uma combinação de três cores primárias: *Red* (R, vermelho), *Green* (G, verde) e *Blue* (B, azul). Desta forma a representação de uma imagem pode ser feita através de 3 camadas (*layers*) RGB com profundidade de 1 *byte* por pixel (MARTINS, 2020).

A percepção de cor é o resultante da incidência de luz nas células fotorreceptoras, denominadas cones (MARTINS, 2020). A intensidade e combinação de cada cor primária (R – vermelho, G – verde e B – azul) define a cor observada. A FIGURA 47 demonstra a representação das três cores primárias e algumas sobreposições simples, que resultam em novas cores, como a interseção das cores: vermelha e azul, formando o rosa; cor verde e azul, formando o ciano; vermelha e verde, formando o amarelo; vermelho, verde e azul formando o branco;

FIGURA 47 - RGB



FONTE: Queiroz (2020)

2.12 ANDROID

O sistema operacional Unix foi criado nos laboratórios da Bell AT&T por Ken Thompson em 1965. É um sistema multitarefa e compartilhado, sendo lançado em 1983 sob o nome de Unix System IV. Em 1985, o professor Andrew Tannenbaum, lança o sistema operacional Minix, de código fonte aberto baseado em Unix, para ensinar programação para os seus alunos (LOVE, 2013).

Em 1991, Linus Torvalds, desenvolve o sistema operacional Linux de código fonte aberto baseado em Unix. A criação era de uso pessoal com estilo de desenvolvimento coletivo. A primeira versão do sistema foi o Linux 0.02, lançado em 1991 (LOVE, 2013).

Android é um sistema operacional baseado no *kernel*[4] (núcleo do sistema operacional) Linux, desenvolvida pela empresa Android Inc. em 2003. O enfoque inicial do sistema operacional seriam as câmeras fotográficas, porém como este mercado não era tão grande na época, voltou-se para dispositivos móveis, rivalizando com o Windows Mobile e o Symbian (LECHETA, 2015). Em 2005 a empresa Google adquire a empresa Android Inc. e prossegue com o desenvolvimento do sistema.

Seu código fonte é disponibilizado pelo Google sob licença de código aberto, sendo comercializado com a premissa de ser um sistema flexível e atualizável.

O sistema possui como usabilidade a interface de tela sensível ao toque, baseada na manipulação direta (LECHETA, 2015), com aplicações em dispositivos móveis, televisores, carros, relógios inteligentes, consoles de videogames, câmeras digitais, computadores, dispositivos eletrônicos, entre outros. O sistema é capaz de lidar com *hardwares* como giroscópios, acelerômetros e sensores de proximidade.

2.13 JAVA

Java é uma plataforma e linguagem de programação orientada a objetos, multiplataforma, lançada pela empresa Sun Microsystems em 1995, posteriormente adquirida pela empresa Oracle em 2010 (DEITEL, 2017). É uma linguagem interpretada pela máquina virtual JVM (*Java Virtual Machine*). O código fonte é compilado para *bytecode*, e este, executado pela JVM.

A versão Java *Enterprise Edition* (Java EE) é voltada para o desenvolvimento de aplicativos de rede distribuídos em grande escala e aplicativos baseados na web (DEITEL, 2017). A versão Java *Micro Edition* (Java ME) - um subconjunto do Java SE - é voltado para o desenvolvimento de aplicativos para dispositivos embarcados com recursos restritos, como *smartwatches*, decodificadores de televisão, medidores inteligentes (para monitorar o uso de energia elétrica), smartphones, tablets Android, entre outros.

De acordo com Deitel (2017) o desenvolvimento dos microprocessadores permitiu o desenvolvimento dos computadores pessoais, dispositivos eletrônicos

---

[4] *Kernel*: kernel ou núcleo do sistema operacional (SO) é responsável por conectar o software (sistema operacional - SO) ao *hardware*. Estabelece uma comunicação entre os recursos do SO e administra suas funções (SACRAMENTO, 2014).

inteligentes e “Internet das Coisas”. Observando isto, a Sun Microsystems em 1991 financiou um projeto de pesquisa de uma linguagem de programação baseada em C++, chamada de Java. Com a ideia de "escreva uma vez, execute em qualquer lugar", Java chamou a atenção da comunidade empresarial, interessada no fenômeno da internet.

As classes Java são desenvolvidas em arquivos com a extensão .java e compiladas com o comando javac, produzindo um arquivo .class. O comando javac converte o código .java em *bytecode*, sendo executado pela JVM. A JVM é um software que simula um computador, ocultando o sistema operacional e o *hardware* dos programas interagem com ele. Isto permite que os programas escritos para a JVM executem em todas as distintas plataformas, sendo independentes da plataforma (DEITEL, 2017). Para executar o código, é chamado o comando java “nome arquivo”.class. Neste momento a JVM coloca os *bytecodes* do arquivo .class do programa na memória para ser executado. Conforme as classes são carregadas, o verificador de *bytecode* examina os *bytecodes* para verificar se são válidos e não violam as restrições de segurança do Java (como vírus e *worms*). Após a verificação, a JVM executa os *bytecodes* e realiza as ações do programa. A execução é feita pela tradução dos *bytecodes* para linguagem de máquina da plataforma sob o qual o programa executa.

2.14 PYTHON

Python é uma linguagem de programação interpretada de alto nível, orientada a objetos, tipagem dinâmica, de *script* (permite criar *scripts* para automatizar tarefas), multiplataforma, *open source* (código aberto), criada no final da década de 90 por Guido Van Rossum, pesquisador do Instituto Nacional de Matemática e Ciência da Computação da Holanda (MCKINNEY, 2018).

Como é uma linguagem de programação interpretada, em geral, a maior parte do código Python executará de forma mais lenta do que um código escrito em linguagem compilada, como C++. Outro desafio do Python está em criar aplicações multithreaded altamente concorrentes, limitadas por CPU (CPU-bound), devido ao GIL (Global Interpreter Lock, ou Trava Global do Interpretador), que é um mecanismo que evita que o interpretador execute mais de uma instrução Python por vez (MCKINNEY, 2018).

Por ser uma linguagem multiplataforma, *open source* (código aberto), e boa curva de aprendizagem, tem se tornado umas das principais linguagens para desenvolvimento de IA, *data science*, *machine learning* e *big data*. Possui facilidade de integração com códigos em C, C++ e FORTRAN, isto permite a utilização de códigos legados para álgebra linear, otimização, integração, transformadas rápidas de Fourier entre outros. Citam-se as bibliotecas (MCKINNEY, 2018):

- NumPy (*Numerical* Python) é uma biblioteca Python voltada para estrutura de dados, algoritmos e bibliotecas necessárias para aplicações científicas que envolvam dados numéricos em Python. Permite criar um contêiner para que os dados sejam passados entre os algoritmos e as bibliotecas;
- Pandas (*panel data* - dados em painel) contém estruturas de dados de alto nível e funções projetadas para otimizar e facilitar o trabalho com dados estruturados ou tabulares. Permite a manipulação, reformatação, agregações, seleção de subconjunto de dados. Alguns objetos do Pandas são o DataFrame (estrutura de dados tabular, orienta a colunas, com rótulos para colunas e linhas) e as Series (array unidimensional com rótulo);
- Matplotlib é uma biblioteca voltada para fazer plotagens e gerar visualizações de dados bidimensionais. Foi originalmente criada por John D. Hunter;
- SciPy é uma coleção de pacotes voltada para uma série de diversos domínios de problemas padrões no processamento científico.

IPython e Jupyter são interpretadores Python interativo. Ele incentiva um fluxo de trabalho de execução-exploração, em vez do fluxo de trabalho típico de edição-compilação-execução. Jupyter é o notebook web do IPython.

O scikit-learn é um kit de ferramentas de propósito geral para aprendizado de máquina, inclui submódulos para (MCKINNEY, 2018):

- Classificação: SVM, vizinhos mais próximos (*nearest neighbors*), floresta aleatória (*random forest*), regressão logística etc.
- Regressão: regressão de Lasso, regressão de ridge etc.
- Clustering: k-means, clustering espectral etc.
- Redução de dimensionalidade: PCA, seleção de atributos, fatoração de matrizes etc.

- Seleção de modelos: grid search (busca em grade), validação cruzada, métricas.
- Pré-processamento: extração de atributos, normalização.

O statsmodels é um pacote de análise estatística, contém algoritmos para estatística clássica e econometria. Inclui submódulos como (MCKINNEY, 2018):

- Modelos de regressão: regressão linear, modelos lineares generalizados, modelos lineares robustos, modelos lineares de efeitos mistos etc.;
- Análise de variância (ANOVA);
- Análise de séries temporais: AR, ARMA, ARIMA, VAR e outros modelos;
- Métodos não paramétricos: estimativa de densidade Kernel, regressão de kernel;
- Visualização de resultados de modelos estatísticos.

2.14.1 Biblioteca Python gTTS

A biblioteca gTTS (Google Text-to-Speech), desenvolvida na linguagem Python é uma ferramenta que faz uma interface para a API text-to-speech do Google Translate. Através desta ferramenta é possível criar um arquivo .mp3 (QUADRO 4) que reproduz o valor de um texto de interesse (DURETTE, 2014).

O MP3 (MPEG-1/2 Audio Layer 3) é um formato de compressão de dados de áudio. Em 1987 o Institut Integrierte Schaltungen (ISS) realizou estudos sobre uma forma de codificação perceptual (que utiliza somente as frequências sonoras captadas pelo ouvido humano) para transmissão digital de áudio. A pesquisa resultou no algoritmo de compressão de áudio sem perda de qualidade, o MPEG Audio Layer-3, conhecido como MP3 (MARTINS, 2008).

2.15 GOOGLE COLAB

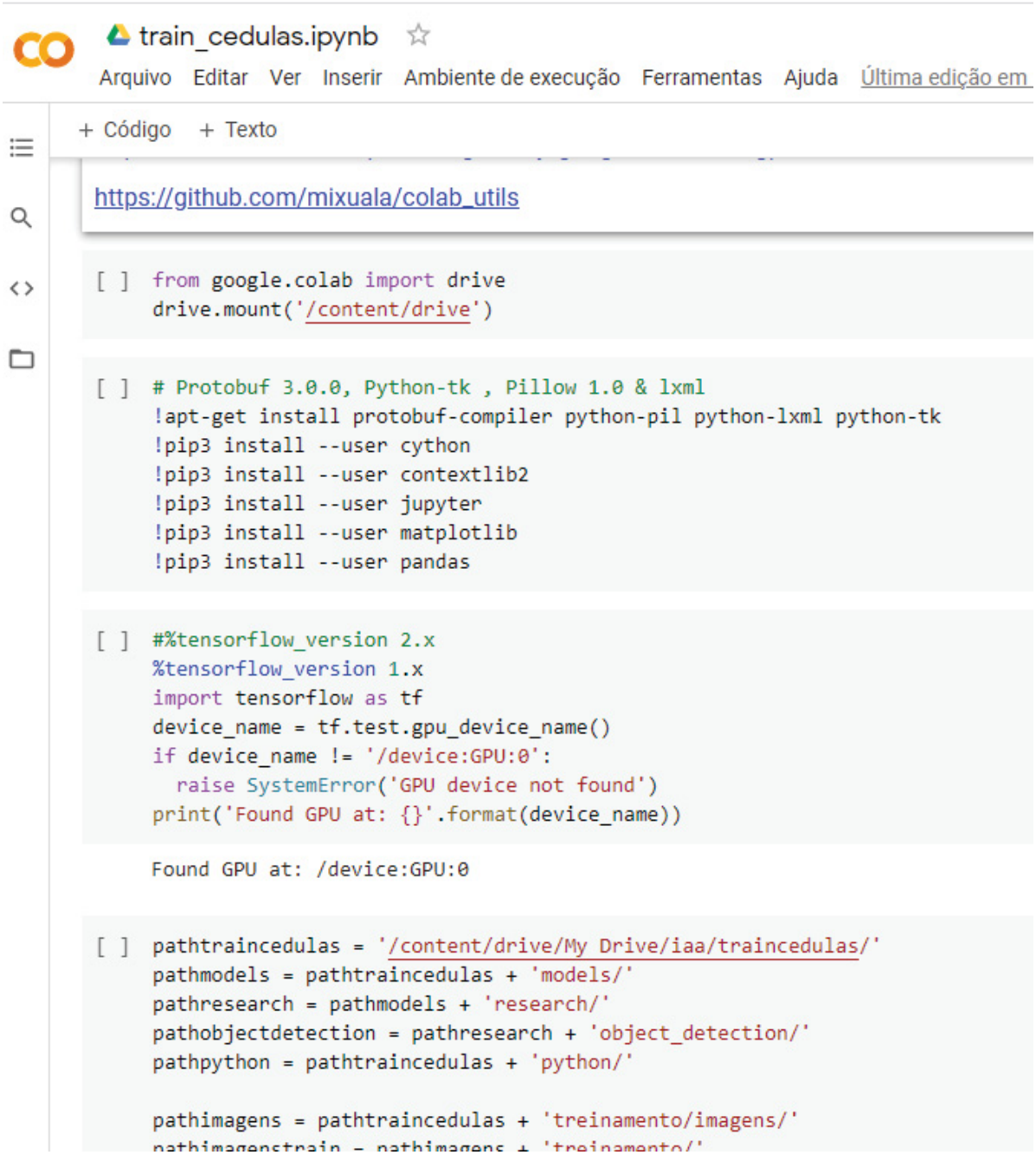
O serviço em nuvem gratuito Google Colaboratory (ou Colab) é hospedado pela empresa Google e tem o objetivo de incentivar a pesquisa de aprendizado de máquina e inteligência artificial (SANTOS, 2020). Através deste serviço, é possível criar *notebooks* que executam comandos da linguagem de programação Python

(Seção 2.14). O acesso as APIs do TensorFlow é feito através da linguagem Python. A plataforma do TensorFlow é descrita na Seção 2.16. Um exemplo de *notebook* do Google Colaboratory está representado na FIGURA 48.

Estas bibliotecas possuem diversos modelos pré-treinados de exemplo, bem como códigos e aplicações que podem ser utilizadas para criar aplicativos de smartphone (TENSORFLOW MODELS, 2020).

Os modelos de redes neurais voltados para dispositivos móveis possuem o formato do TensorFlow Lite (.tflite), descrito na Seção 2.16.

FIGURA 48 – *NOTEBOOK* COLAB



FONTE: O Autor (2020)

Através da ferramenta Google Colaboratory é possível utilizar comandos da linguagem Python (Seção 2.14) e realizar diversas atividades, como: carregar e exibir as imagens das cédulas (FIGURA 49); Conectar e acessar o Google Drive (FIGURA 50); definir e treinar modelos (FIGURA 51 e FIGURA 52); avaliar o modelo (FIGURA 53); realizar predições (FIGURA 54); exportar o modelo para o formato .tflite (FIGURA 55); entre outros.

FIGURA 49 – REPRESENTAÇÃO CÉDULAS

```
plt.figure(figsize=(10,10))
for i, (image, label) in enumerate(data.gen_dataset().unbatch().take(25)):
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(image.numpy(), cmap=plt.cm.gray)
  plt.xlabel(data.index_to_label[label.numpy()])
plt.show()
```

FONTE: O Autor (2020)

FIGURA 50 - CONECTAR GOOGLE DRIVE

Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

FONTE: O Autor (2020)

FIGURA 51 - EXEMPLO MODELO

```
num_classes = len(class_names) # 12 - cédulas

model = tf.keras.Sequential([
  layers.experimental.preprocessing.Rescaling(1./255),
  # CNN
  layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
  layers.MaxPooling2D((2, 2)),
  layers.Conv2D(64, (3, 3), activation='relu'),
  layers.MaxPooling2D((2, 2)),
  layers.Conv2D(64, (3, 3), activation='relu'),
  layers.Conv2D(32, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, activation='relu'),
  layers.MaxPooling2D(),

  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

FONTE: O Autor (2020)

FIGURA 52 – EXEMPLO TREINAMENTO MODELO

```
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=10
)
```

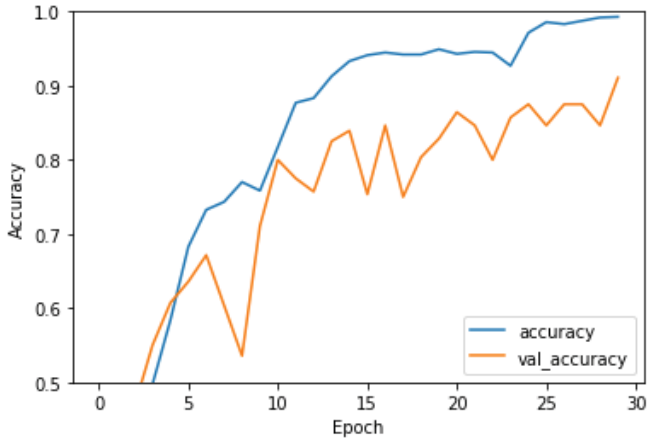
```
Epoch 1/10
36/36 [==============================] - 251s 7s/step - loss: 2.4494 - ac
Epoch 2/10
36/36 [==============================] - 60s 2s/step - loss: 2.0976 - acc
Epoch 3/10
36/36 [==============================] - 61s 2s/step - loss: 1.8446 - acc
Epoch 4/10
36/36 [==============================] - 61s 2s/step - loss: 1.6004 - acc
```

FONTE: O Autor (2020)

FIGURA 53 – EXEMPLO AVALIAÇÃO MODELO

```
[90] plt.plot(history.history['accuracy'], label='accuracy')
     plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
     plt.xlabel('Epoch')
     plt.ylabel('Accuracy')
     plt.ylim([0.5, 1])
     plt.legend(loc='lower right')


     loss, accuracy = model.evaluate(val_ds)
```

```
9/9 [==============================] - 0s 12ms/step - loss: 0.2239 - accuracy: 0.9357
```



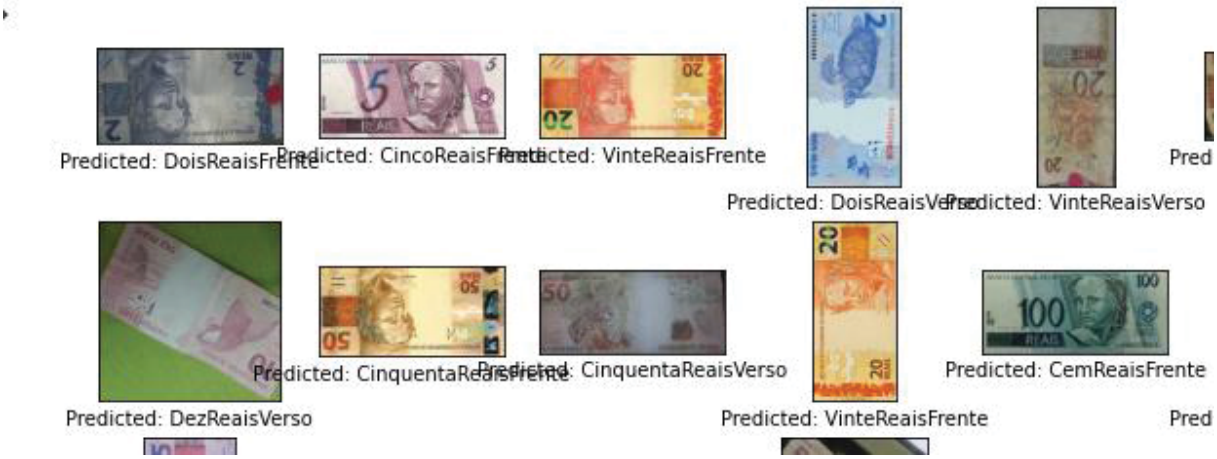

```
[91] print(accuracy)
```

```
0.9357143044471741
```

FONTE: O Autor (2020)

FIGURA 54 – EXEMPLO DE PREDIÇÕES

```
predicts = model.predict_top_k(test_data)
for i, (image, label) in enumerate(test_data.gen_dataset().unbatch().take(100)
  ax = plt.subplot(10, 10, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(image.numpy(), cmap=plt.cm.gray)

  predict_label = predicts[i][0][0]
  color = get_label_color(predict_label,
                          test_data.index_to_label[label.numpy()])
  ax.xaxis.label.set_color(color)
  plt.xlabel('Predicted: %s' % predict_label)
plt.show()
```



FONTE: O Autor (2020)

FIGURA 55 - EXPORTAR MODELO TFLITE

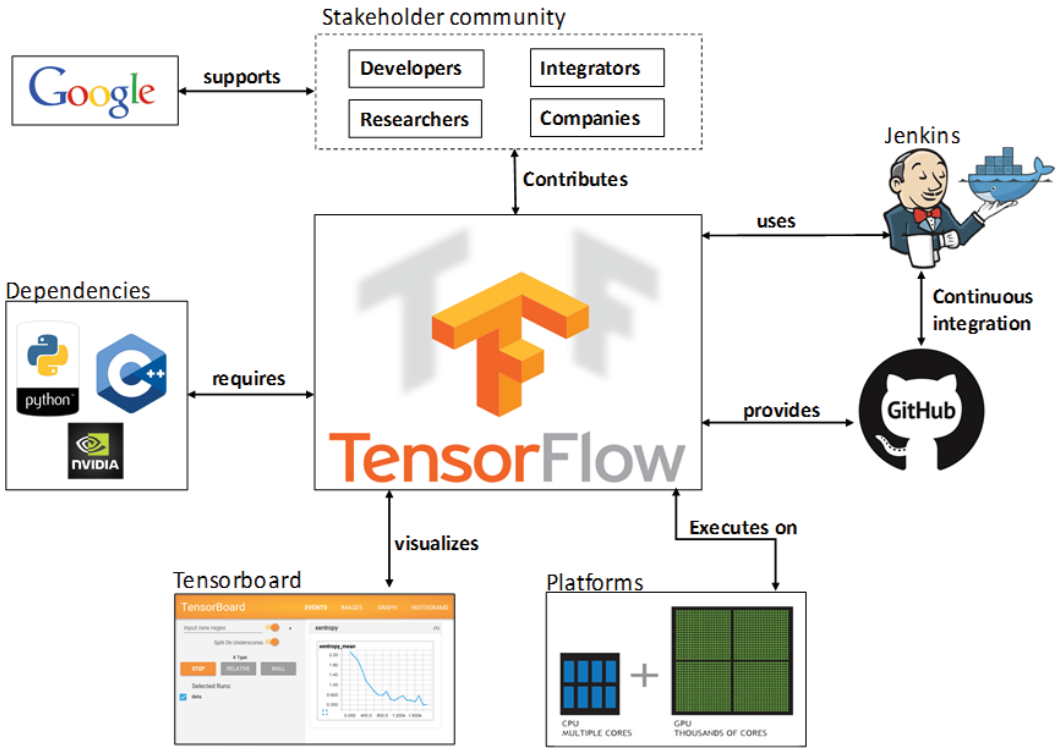
```
[ ]  model.export(export_dir = tflite_path, tflite_filename='cedulas2.tflite')
     %cd $tflite_path
     %ls

     /content/drive/My Drive/tflitetrain/modelotflite/cedulas2
     cedulas.tflite
```

FONTE: O Autor (2020)

2.16 TENSORFLOW

A plataforma de desenvolvimento para *machine learning*, TensorFlow, possui código aberto, um ecossistema de ferramentas flexível, APIs, bibliotecas e recursos criados pela comunidade de desenvolvimento (TENSORFLOW, 2020). Foi construído com o intuito de ser flexível, eficiente, extensível e portátil, possuindo a capacidade de gerar um produto ou serviço a partir de um modelo preditivo treinado, sem ter que reimplementar o modelo. Projetado para ser escalável em vários computadores, CPUs e GPUs dentro de máquinas individuais (FIGURA 56) (MATOS, 2020).

FIGURA 56 - TENSORFLOW



FONTE: Matos (2020)

TensorFlow é a API utilizada para definir modelos de aprendizado de máquina. A API é acessada através da linguagem Python que executa instruções da linguagem C++. O TensorBoard é um software de visualização de gráficos, fornecendo visualizações sobre o comportamento dos modelos (MATOS, 2020).

Permite a integração de sistemas como o GitHub para armazenar e versionar o código e o Jenkins, para publicar os artefatos no TensorFlow. Jenkins é

um servidor de automação de tarefas gratuito e de código aberto. Permite a integração e entrega contínuas, automatizada de construção, implantação e testes de sistemas (RITI, 2018). GitHub é um sistema de gerenciamento de versões de código de projetos. Possui também uma plataforma de rede social voltada para desenvolvedores. Permite o trabalho colaborativo de projetos feitos por desenvolvedores de todo o mundo (TSITOARA, 2020).

TensorFlow Lite é um conjunto de ferramentas voltado para criação de modelos do TensorFlow em dispositivos móveis, incorporados e de IoT (TENSORFLOW LITE, 2020). Os modelos gerados possuem a extensão .tflite. Possui dois componentes principais:

- Interpretador: executa modelos otimizados para diferentes tipos de *hardware*, como smartphones, micro controladores e dispositivos Linux incorporados;
- Conversor: converte modelos do TensorFlow em modelos no formato .tflite, e, realiza otimizações para diminuir o tamanho do modelo gerado e melhorar o desempenho de sua execução.

O modelo .tflite é voltado para a execução no dispositivo móvel, ao invés de enviar e receber dados de um servidor remoto (TENSORFLOW LITE, 2020). Citam-se algumas vantagens deste modelo:

- Baixa latência: sem ida e volta ao um servidor;
- Privacidade: nenhum dado sai do dispositivo;
- Conectividade: não exige conexão com a internet;
- Consumo de energia: comunicações em rede exigem maior consumo energético;
- Variabilidade: o formato é compatível com uma série de dispositivos, desde micro controladores até smartphones (TENSORFLOW LITE, 2020).

Observa-se que a execução de um modelo de *machine learning* em um dispositivo móvel fica limitado ao *hardware* e recursos computacionais do mesmo.

Ao implementar um modelo treinado, o mesmo é congelado, dando origem a um arquivo chamado *frozen graph*. Este pode ser percebido como um modelo cujos pesos estão fixos (congelados). O processo de congelamento de um modelo consiste em converter as variáveis em constantes, que são armazenadas

diretamente no grafo do modelo treinado (ACOSTA, 2020). Portanto, a versão congelada de um modelo não pode mais ser treinada.

De acordo com o Sachan (2021), redes neurais são computacionalmente muito caras, como exemplo a AlexNet, formada por mais de 60 milhões de parâmetros e um número semelhante de gradientes que são utilizados no treinamento. Os modelos de TensorFlow contém todos estes parâmetros e variáveis. Um modelo típico do Tensorflow contém 4 arquivos:

- model-ckpt.meta: contém o gráfico completo que descreve o fluxo de dados, anotações para variáveis, pipelines de entrada e outras informações relevantes;
- model-ckpt.data-0000-of-00001: valores das variáveis (pesos, vieses, espaços reservados, gradientes, hiperparâmetros etc.);
- model-ckpt.index: metadados;
- checkpoint: informações do checkpoint.

No momento de implantar um modelo treinado, todas essas informações são desnecessárias. O processo de congelar o modelo significa identificar os itens necessários (gráfico, pesos, etc.) e salvar em um arquivo com a extensão ".pb" - chamada de "definição gráfica congelada". Desta forma eliminando os metadados, gradientes e variáveis desnecessárias (SACHAN, 2021).

2.17 PASCAL VOC XML

Pascal VOC é um formato de anotação de dados, utilizado para anotar e rotular objetos em *datasets* utilizados para visão computacional (KHANDELWAL, 2019). A anotação rotula os objetos nas imagens de treinamento e testes. Serve para auxiliar a localização precisa dos objetos dentro das caixas delimitadoras, ou máscaras poligonais. O Pascal VOC utiliza arquivos no formato de marcação XML para definir os objetos de interesse nas imagens do *dataset*.

A extensão XML é uma sigla para *eXtensible Markup Language*, uma linguagem de marcação que define padrões e regras de formatação de documentos organizados de forma hierárquica (MAGALHÃES, 2020). A codificação dos textos XML é feita através de marcações ou *tags*. Um exemplo de marcação XML está representado na FIGURA 57. O exemplo trata da representação de uma nota (*tag*

note), informando o destinatário (*tag* to), remetente (*tag* from), cabeçalho (*tag* heading) e mensagem (*tag* body).

FIGURA 57 – EXEMPLO XML

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

FONTE: W3C XML (2020)

Um arquivo XML é criado para cada imagem do conjunto de dados. A caixa delimitadora de cada objeto de interesse é formada por quatro coordenadas: x superior esquerdo (xmin-top left), y superior esquerdo (ymin-top left), x inferior direito (xmax-bottom right), y inferior direito (ymax-bottom right). Um exemplo de imagem anotada com seu respectivo arquivo XML no formato PASCAL VOC está representado na FIGURA 58 e QUADRO 2. A classe desta imagem é definida na *tag* name, com valor “CemReaisFrente”.

FIGURA 58 – IMAGEM ANOTADA

FONTE: O Autor (2020)

QUADRO 2 – ANOTAÇÃO XML PASCAL VOC

```
<?xml version="1.0" encoding="UTF-8"?>
<annotation>
  <folder>treinamento</folder>
  <filename>283_90.jpg</filename>
  <path>/content/drive/My
Drive/iaa/traincedulas/treinamento/imagens/
treinamento/283_90.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>384</width>
    <height>800</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
<object>
<name>CemReaisFrente</name>
<pose>Unspecified</pose>
<truncated>0</truncated>
<difficult>0</difficult>
<bndbox>
<xmin>9</xmin>
<ymin>18</ymin>
<xmax>371</xmax>
<ymax>794</ymax>
</bndbox>
</object>
</annotation>
```

FONTE: O Autor (2020)

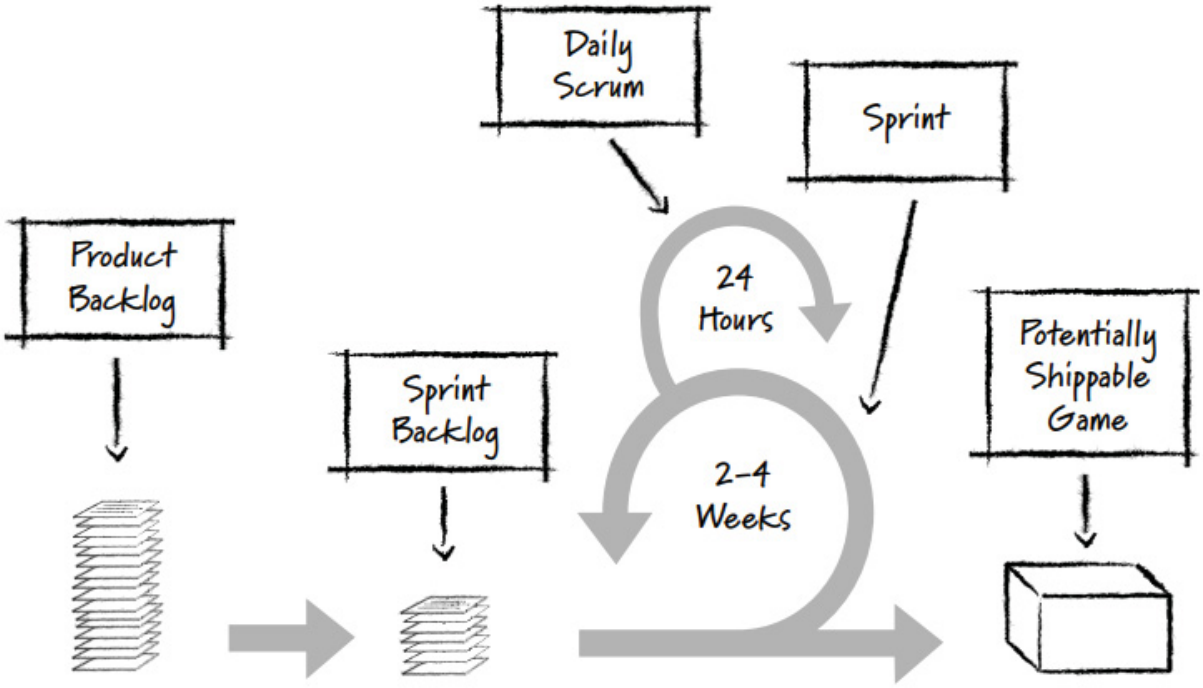
2.18 SCRUM

Scrum é um *framework* desenvolvido em 1990 que auxilia no desenvolvimento e sustentação de produtos complexos. A equipe do Scrum é formada pela figura do Scrum Master, o dono do produto (*product owner*) e a equipe de desenvolvedores. Dentro de uma equipe Scrum, não há subequipes ou hierarquias. Seu objetivo é o desenvolvimento do produto (COHN e KEITH, 2010).

O papel do Scrum Master é promover um ambiente onde (FIGURA 59):

1. O dono do produto ordena o trabalho para um *backlog* de produto (*product backlog*, que contém as funcionalidades esperadas do sistema);
2. O time Scrum agrega valor ao projeto durante uma *Sprint*;
3. O resultado da *Sprint* é avaliado pelo time Scrum e seus *stakeholders* (partes interessadas no produto);
4. Planeja a próxima *Sprint* com base nos resultados da *Sprint* anterior;
5. Repete o passo 1.

FIGURA 59 - FRAMEWORK SCRUM



FONTE: Cohn e Keith (2010)

A *Sprint* é um evento de duração fixa de um mês ou menos. Uma nova *Sprint* começa imediatamente após a conclusão do *Sprint* anterior. Todo o trabalho de planejamento das *Sprints*, Scrums Diários, revisão de *Sprint*, retrospectiva, acontece dentro de *Sprints* (COHN e KEITH, 2010). O progresso da *Sprint* é feito por todos os integrantes da equipe, durante a realização de reuniões diárias.

A filosofia da equipe Scrum é ser autogerenciada, de forma que a equipe decide internamente os papéis de atuação de cada integrante, quem atua em qual tarefa, de que forma e quando. O time é responsável por todas as atividades relacionadas ao produto: colaboração das partes interessadas, atividades,

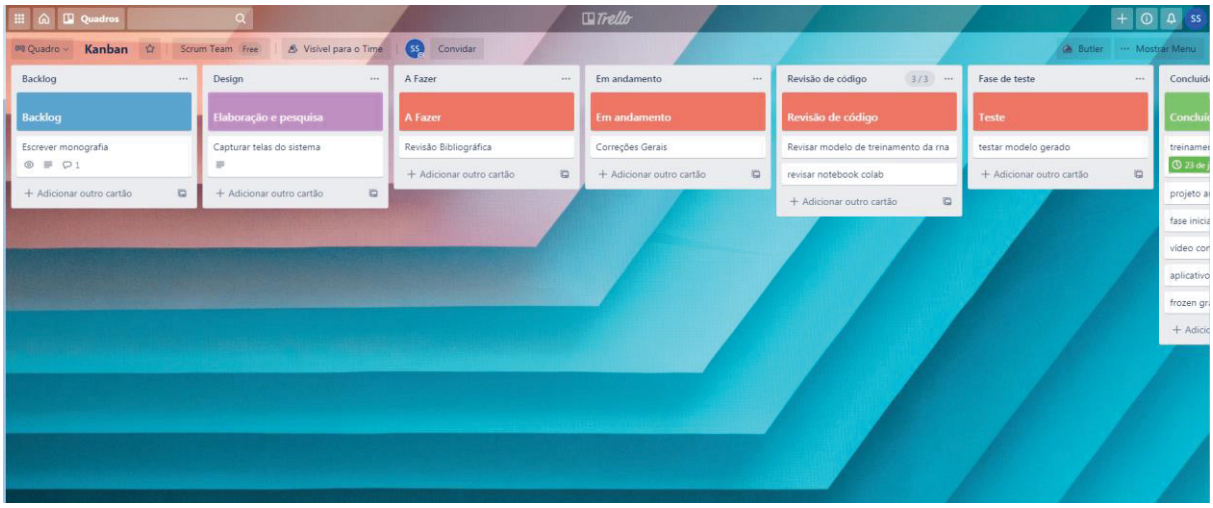
manutenção, verificação, pesquisa e desenvolvimento, agregar valor a cada *Sprint* (COHN e KEITH, 2010).

O tamanho do time deve ser pequeno, normalmente 10 ou menos pessoas. De acordo com Cohn e Keith (2010), equipes menores possuem melhor comunicação e possuem maior produtividade.

2.19 TRELLO

Trello (TRELLO, 2020) é uma ferramenta de gerenciamento de projetos gratuita online. Através dela é possível organizar as tarefas de trabalho, planos, criar quadros, Kanban, entre outros. Permite o uso compartilhado por equipes de desenvolvimento de projetos. O sistema pode ser acessado através do navegador ou através de aplicativo para smartphone (CASTELLI, 2020). Um exemplo de quadro Kanban utilizando a ferramenta Trello está representada na FIGURA 60.

FIGURA 60 – KANBAN - TRELLO



FONTE: O Autor (2020)

3 MATERIAIS E MÉTODOS

3.1 MODELO DE ENGENHARIA DE SOFTWARE

O modelo de engenharia de software utilizado no trabalho foi uma adaptação da metodologia ágil Scrum (Seção 2.18).

As *Sprints* (Seção 2.18) foram divididas em períodos de duas semanas (14 dias), observando-se os casos em que o início da próxima *Sprint* coincidia com feriados ou finais de semana. A divisão de tarefas e datas por *Sprint* está descrita na Seção 3.7, totalizando 9 *Sprints*. As reuniões diárias foram substituídas por uma avaliação e análise do andamento do projeto. Ao final de cada *Sprint* foi feita a retrospectiva e ajustes da *Sprint* anterior.

Para acompanhamento das tarefas a ferramenta Trello (Seção 2.19) foi utilizada para auxiliar na visualização das tarefas através do quadro Kanban. A FIGURA 61 representa o quadro Kanban criado nesta ferramenta.

3.2 MATERIAIS

Foi utilizado a plataforma de estudo e treinamento colaborativo do Google Colaboratory (GOOGLE Colab, 2020) para treinamento da Rede Neural em conjunto com o Google Drive (GOOGLE Drive, 2020) para armazenamento dos arquivos de configuração e treinamento. O uso do Google Colab em conjunto com o Google Drive possui diversas vantagens, entre elas:

- Ambientes de uso gratuito, sendo 15 GB gratuitos para o Google Drive;
- Facilidade em integrar o Google Colab com o Google drive para permanência dos dados de treinamento;
- Ambiente na nuvem, o que facilita o acesso, configuração e a execução;
- Ambiente Python pré-configurado voltado para IA. Desta forma poucas instalações foram necessárias;
- Suporte a execução em placas gráficas GPU;
- Configuração de *hardware* e memória compatíveis com as exigências para treinamento sistemas de IA;
- CPU: Intel® Xeon® CPU @ 2.30GHz;

- RAM: 13 GB;
- GPU: Tesla P100-PCIE-16GB;

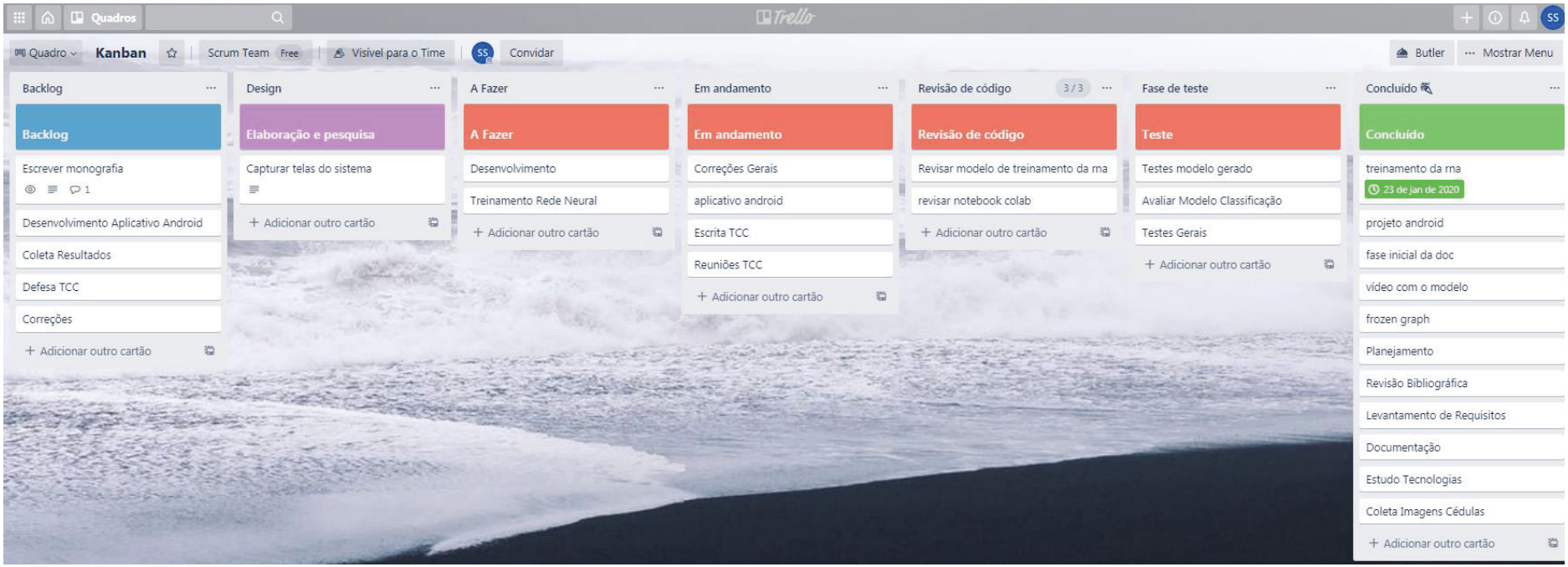
Desvantagens:

- É preciso manter uma sessão ativa com o Google Colab, caso contrário o treinamento é interrompido;
- É necessário possuir uma conexão com a internet, para acesso ao Google Colab e Google drive;
- Demora no processamento. Em média um passo de treinamento demora cerca de 30 segundos por execução;

Para contornar estes problemas, o Google recomenda o uso da computação paga em nuvem Google TPU (TPU - Unidade de Processamento de Tensor) (GOOGLE TPU, 2020) criado para acelerar e otimizar o processo de treinamento de aprendizado de máquina de rede neural.

Os recursos de *hardware* estão descritos na Seção 3.2.1, os recursos de software na Seção 3.2.2.

FIGURA 61 – KANBAN - TRELLO



FONTE: O Autor (2020)

3.2.1 RECURSOS DE *HARDWARE*

Para as tarefas de organização do trabalho, planejamento, documentação, estudo de tecnologias, coleta de imagens, operação de *Data Augmentation* (descrita na Seção 2.9), desenvolvimento e adaptação do aplicativo Android (Seção 3.6), escrita e correções do TCC, foi utilizada uma máquina com as características capacidade de processamento de 2 GB de memória RAM, processador Intel(R) Core(TM) 2 Quad Q6600 @ 2.40GHz.

Para o treinamento da rede neural, avaliação do modelo de classificação, testes e coleta de resultados, o Google Colaboratory (GOOGLE Colab, 2020) em conjunto com o Google Drive (GOOGLE Drive, 2020) para armazenamento dos arquivos de configuração e treinamento possuem as características de: 15 GB, 13 GB RAM, GPU Tesla P100-PCIE-16GB e CPU Intel® Xeon® CPU @ 2.30GHz.

Para as predições do modelo treinado em dispositivo Android, foi utilizado um dispositivo mobile Samsung Galaxy S10.

3.2.2 RECURSOS DE SOFTWARE

Para desenvolvimento do trabalho foi utilizado uma máquina com sistema operacional Windows.

As anotações gráficas das regiões de interesse (ROI) das imagens foram feitas através da ferramenta LabelImg (TZUTALIN, 2020).

O desenvolvimento da rede neural foi feito no ambiente do Google Colaboratory (GOOGLE Colab, 2020), que possui um ambiente com Python voltado para aprendizado de máquina.

Para desenvolvimento do aplicativo Android foi utilizado o software Android Studio (ANDROID STUDIO, 2020), utilizando a linguagem Java. O projeto Android base foi adaptado dos modelos de TensorFlow *examples* (TENSORFLOW *EXAMPLES*, 2020). A Máquina Virtual do Java (JVM – Java Virtual Machine) será necessária para desenvolvimento em Java.

A versão do sistema operacional Android do dispositivo mobile Samsung Galaxy S10 utilizado foi a versão 10.0.

3.3 COLETA DE IMAGENS

Inicialmente foram coletadas cerca de 500 imagens de cédulas de R$ 2,00, R$ 5,00, R$ 10,00, R$ 20,00, R$ 50,00 e R$ 100,00, frente e verso. Parte das imagens foi obtida através de uma câmera de celular, outras a partir do Google images. Posteriormente foi utilizada técnica de *Data Augmentation* (Seção 2.9) para incrementar o número de imagens para testes e treinamento. As imagens foram rotacionadas em ângulos de 90º, 180º e 270º. Como exemplo as figuras: FIGURA 62, FIGURA 63, FIGURA 64 e FIGURA 65 demonstram uma imagem de cédula de R$ 100,00 e suas rotações em ângulos de 180º, 90º e 270º respectivamente.

Exemplos de rotação horizontal e vertical estão representadas nas figuras: FIGURA 66 e FIGURA 67. Este tipo de rotação não foi utilizado neste trabalho porque não era objetivo a detecção de imagens espelhadas.

O número total de imagens obtida com a técnica de *Data Augmentation* foi de 1412, sendo 1136 imagens utilizadas para treinamento (80,45%) e 276 imagens para teste (19,55%). O foco do trabalho foram as cédulas novas de real (2019/2020), cédulas antigas, moedas, ou cédulas de outros sistemas monetários não são tratados neste trabalho.

A divisão entre as classes de treinamento e testes está descrita na TABELA 3. A classe “DoisReaisFrente” representa a imagem da cédula de R$ 2,00 da parte da frente, enquanto a classe “DoisReaisVerso”, representa a parte de trás.

TABELA 3 – DIVISÃO CLASSES TREINAMENTO E TESTES

	Classes	**Treinamento**	**Testes**	**Subtotais**	**Por classe**
1	DoisReaisFrente	83	25	108	208
2	DoisReaisVerso	78	22	100	
3	CincoReaisFrente	107	25	132	232
4	CincoReaisVerso	80	20	100	
5	DezReaisFrente	125	23	148	256
6	DezReaisVerso	87	21	108	
7	VinteReaisFrente	118	18	136	236
8	VinteReaisVerso	86	14	100	
9	CinquentaReaisFrente	122	28	150	248
10	CinquentaReaisVerso	74	24	98	
11	CemReaisFrente	128	29	157	232
12	CemReaisVerso	48	27	75	
	TOTAL	1136	276	1412	1412

FONTE: O Autor (2020)

FIGURA 62 – IMAGEM R$ 100

FONTE: O Autor (2020)

FIGURA 63 – IMAGEM R$ 100 – ROTAÇÃO 180º

FONTE: O Autor (2020)

FIGURA 64 – IMAGEM R$ 100 – ROTAÇÃO 90º

FONTE: O Autor (2020)

FIGURA 65 – IMAGEM R$ 100 – ROTAÇÃO 270º

FONTE: O Autor (2020)

FIGURA 66 – IMAGEM R$ 100 – ROTAÇÃO HORIZONTAL

FONTE: O Autor (2020)

FIGURA 67 – IMAGEM R$ 100 – ROTAÇÃO VERTICAL

FONTE: O Autor (2020)

As anotações gráficas das regiões de interesse (ROI) das imagens foram feitas através da ferramenta LabelImg (TZUTALIN, 2020). Esta ferramenta permite criar anotações gráficas em formato xml PASCAL VOC (EVERINGHAM, 2020). Um exemplo de definição de áreas de interesse está representado na FIGURA 68.

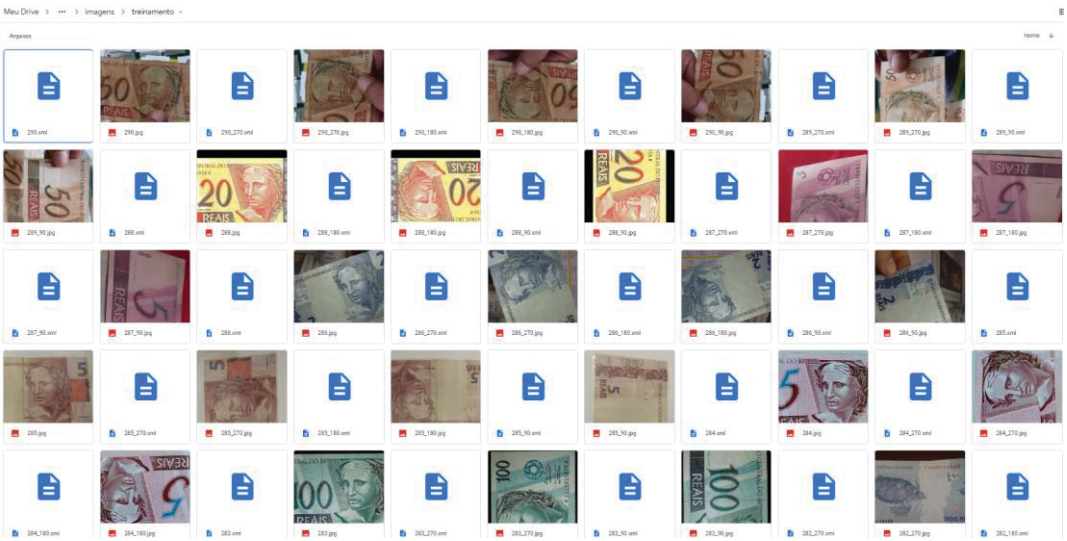
O formato adotado no presente trabalho foi o PASCAL VOC (Seção 2.17). A ROI, cédula de real, foi definida para cada imagem de treinamento e teste (FIGURA 69). O formato xml PASCAL VOC define uma caixa delimitadora para cada ROI,

sendo delimitada em x superior esquerdo (xmin-top left), y superior esquerdo (ymin-top left), x inferior direito (xmax-bottom right), y inferior direito (ymax-bottom right), conforme representação na FIGURA 58. Caso uma imagem possua mais de uma cédula de real, o arquivo xml resultante terá um conjunto de classes e boxes. Exemplos de anotações no formato xml PASCAL VOC estão representados nos quadros: QUADRO 2 e QUADRO 3.

FIGURA 68 – LABELIMG

FONTE: O Autor (2020)

FIGURA 69 – CÉDULAS E ANOTAÇÕES XML



FONTE: O Autor (2020)

QUADRO 3 – EXEMPLO DE ANOTAÇÃO XML

```
<?xml version="1.0" encoding="UTF-8"?>
<annotation>
   <folder>treinamento</folder>
   <filename>290.jpg</filename>
   <path>/content/drive/My
Drive/iaa/traincedulas/treinamento/imagens/treinamento/290.jpg</path>
   <source>
      <database>Unknown</database>
   </source>
   <size>
      <width>800</width>
      <height>470</height>
      <depth>3</depth>
   </size>
   <segmented>0</segmented>
   <object>
      <name>CinquentaReaisFrente</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
      <bndbox>
         <xmin>2</xmin>
         <ymin>55</ymin>
         <xmax>742</xmax>
         <ymax>406</ymax>
      </bndbox>
   </object>
</annotation>
```

FONTE: O Autor (2020)

As imagens foram classificadas em 12 classes: R$ 2,00 frente, R$ 2,00 verso, R$ 5,00 frente, R$ 5,00 verso, R$ 10,00 frente, R$ 10,00 verso, R$ 20,00 frente, R$ 20,00 verso, R$ 50,00 frente, R$ 50,00 verso, R$ 100,00 frente e R$ 100,00 verso. Exemplos da cédula de R$ 100,00 vista de frente e verso estão representadas nas figuras: FIGURA 70 e FIGURA 71, respectivamente.

FIGURA 70 – IMAGEM R$ 100 – FRENTE

FONTE: Verissimo (2020)

FIGURA 71 – IMAGEM R$ 100 – VERSO

FONTE: Verissimo (2020)

3.4 MODELO

O modelo da rede foi treinado utilizando o serviço em nuvem Google Colaboratory.

Como o objetivo do trabalho é gerar uma aplicação smartphone Android, o classificador foi treinado voltado para o TensorFlow Lite, utilizando SSD e MobileNets. Abordagem indicada pelos criadores do TensorFlow (DEREKJCHOW, 2020).

Alguns modelos de teste foram criados e treinados para comparar alguns itens estruturais da rede neural e sua relação com as funções de perda e acurácia. O primeiro e o segundo modelo foram criados com a seguinte estrutura da FIGURA 72. A primeira foi treinada durante 5 épocas e a segunda, 10 épocas. Os valores respectivos de *loss function* foram de 0,7502 e 0,6965; de acurácia, 0,7955 e 0,8766. Para esta estrutura e para o conjunto de cédulas de treinamento e validação, treinar por mais épocas ajudou a melhorar os valores de função de perda e acurácia.

FIGURA 72 – ESTRUTURA MODELOS 1 E 2

```
[24] model = image_classifier.create(train_data)

     INFO:tensorflow:Retraining the models...
     Model: "sequential"
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #
     =================================================================
     hub_keras_layer_v1v2 (HubKer (None, 1280)              3413024
     _________________________________________________________________
     dropout (Dropout)            (None, 1280)              0
     _________________________________________________________________
     dense (Dense)                (None, 12)                15372
     =================================================================
```

FONTE: O Autor (2020)

A estrutura do terceiro modelo está representada na FIGURA 73. Este foi treinado durante 10 épocas e obteve um valor de *loss function* de 0,6829 e acurácia de 0,9716.

FIGURA 73 - ESTRUTURA MODELO 3

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_2 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 178, 178, 32)      896
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 89, 89, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 87, 87, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 43, 43, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 41, 41, 64)        36928
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 39, 39, 32)        18464
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 19, 19, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 17, 17, 32)        9248
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 6, 6, 32)          9248
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 3, 3, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 288)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               36992
_________________________________________________________________
dense_3 (Dense)              (None, 12)                1548
=================================================================
```

FONTE: O Autor (2020)

A técnica de *Transfer Learning* (aprendizado de transferência) foi utilizada como ponto de partida para o classificador (Seção 2.7). A rede utilizada como base foi uma SSD MobileNet treinada sobre o *dataset* COCO (Seção 2.8). A representação da estrutura da rede SSD MobileNet está no APÊNDICE G. A estrutura do terceiro modelo (FIGURA 73) foi utilizada como em conjunto com esta técnica.

Após a geração do modelo, o mesmo foi convertido para um *frozen graph*, descrito na Seção 2.16. Basicamente um *frozen graph* é um modelo cujos pesos

estão fixos (congelados). Gerados os arquivos de *graph* para o tflite, os mesmos foram convertidos para o formato .tflite, utilizado no aplicativo mobile Android.

3.5 VOCALIZAÇÃO DAS CLASSES

A vocalização das classes das cédulas (Seção 3.3) foi feita através da reprodução de arquivos .mp3 gravados com o texto das classes. A geração dos arquivos de áudio .mp3 (Seção 2.14.1) foi criada com a biblioteca Python gTTS. A descrição desta biblioteca está na Seção 2.14.1. O QUADRO 4 demonstra o uso da biblioteca para gerar os arquivos .mp3 que reproduzem as classes reconhecidas.

QUADRO 4 – CÓDIGO PYTHON GERADOR DE ARQUIVO .MP3.

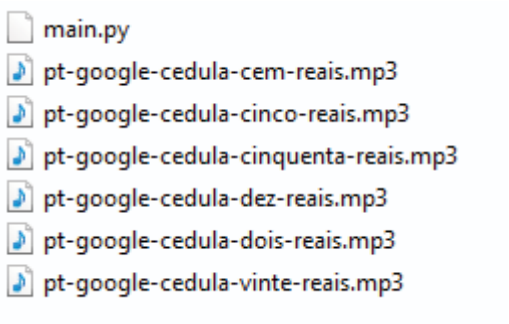
```
from gtts import gTTS

tts = gTTS('Dois reais', lang='pt', tld='com.br')
tts.save('pt-google-cedula-dois-reais.mp3')

tts = gTTS('Cinco reais', lang='pt', tld='com.br')
tts.save('pt-google-cedula-cinco-reais.mp3')

tts = gTTS('Dez reais', lang='pt', tld='com.br')
tts.save('pt-google-cedula-dez-reais.mp3')

tts = gTTS('Vinte reais', lang='pt', tld='com.br')
tts.save('pt-google-cedula-vinte-reais.mp3')

tts = gTTS('Cinquenta reais', lang='pt', tld='com.br')
tts.save('pt-google-cedula-cinquenta-reais.mp3')

tts = gTTS('Cem reais', lang='pt', tld='com.br')
tts.save('pt-google-cedula-cem-reais.mp3')
```

FONTE: Adaptado de DURETTE, 2014

Foram criados 6 arquivos de áudio no formato .mp3 (FIGURA 74), sendo reproduzidas as classes de: dois reais (R$ 2,00), cinco reais (R$ 5,00), dez reais (R$ 10,00), vinte reais (R$ 20,00), cinquenta reais (R$ 50,00) e cem reais (R$ 100,00).

Após a classificação pelo modelo, a classe detectada é vocalizada através da reprodução do respectivo arquivo .mp3.

FIGURA 74 - ARQUIVOS .MP3



FONTE: O Autor (2021)

3.6 PROJETO ANDROID BASE

O projeto Android (Seção 2.12) foi adaptado dos modelos de exemplos de projetos voltados à detecção de objetos do TensorFlow *examples* (TENSORFLOW *EXAMPLES*, 2020). O projeto base utiliza a câmera do dispositivo mobile, aplica o modelo .tflite e recupera um conjunto de predições de saída da rede neural. Este projeto foi alterado para detectar as classes e modelo treinado com o modelo quantificado de MobileNet SSD (Seção 2.6) de cédulas de real, e reproduzir o arquivo .mp3 correspondente a classe detectada. O projeto Android utilizado é feito na linguagem de programação Java (Seção 2.13).

3.7 DESENVOLVIMENTO DO PROJETO

O resumo das Sprints está descrito na TABELA 4. Sua descrição está nas seções: 3.7.1, 3.7.2, 3.7.3, 3.7.4, 3.7.5, 3.7.6, 3.7.7, 3.7.8 e 3.7.9.

TABELA 4 - RESUMO DAS SPRINTS

Sprint	Data	Atividades
1	30 de abril de 2020	Planejamento Cronograma Revisão Bibliográfica Levantamento de Requisitos
2	14 de maio de 2020	Documentação Diagrama de Classes Casos de Uso Diagrama de Atividades Escrita do resumo, introdução e objetivos da monografia
3	28 de maio de 2020	Justificativa da monografia Levantamento e estudo de tecnologias Testes e análise do levantamento de tecnologias
4	12 de junho de 2020	Coleta Imagens Cédulas Desenvolvimento Escrita TCC
5	26 de junho de 2020	Treinamento Rede Neural Avaliar Modelo Classificação Testes
6	10 de julho de 2020	Desenvolvimento Aplicativo Android Coleta Resultados
7	24 de julho de 2020	Ajustes no modelo Materiais e métodos da monografia
8	07 de agosto de 2020	Reuniões TCC Revisões Apresentação dos resultados da monografia Escrita da conclusão da monografia
9	01 de julho de 2021	Defesa TCC Correções e revisões

FONTE: O Autor (2021)

3.7.1 Primeira *Sprint*

A primeira *Sprint* foi iniciada no dia 30 de abril (TABELA 4). Nela foi realizado o planejamento, cronograma, início da revisão bibliográfica e levantamento de requisitos (APÊNDICE A).

O levantamento bibliográfico foi iniciado nesta *Sprint* e continuado na Segunda (3.7.2) e Terceira (3.7.3).

A lista de requisitos foi finalizada nesta *Sprint* e está descrita no APÊNDICE A.

3.7.2 Segunda *Sprint*

A segunda *Sprint* foi realizada no dia 14 de maio. Nesta foi iniciada a documentação do sistema, diagrama de classes e atividades, casos de uso, escrita do resumo, introdução e objetivos da monografia.

Foi continuado o levantamento bibliográfico, iniciado na primeira *Sprint* (3.7.1), sendo finalizado na próxima *Sprint* (3.7.3).

O Resumo foi iniciado e finalizado na terceira *Sprint* (3.7.3).

A Introdução e Objetivos foram iniciados e finalizados nesta *Sprint*, sendo descritos nas Seções 1 e 1.1, respectivamente.

O diagrama de casos de uso está descrito no APÊNDICE B (FIGURA 85), diagrama de atividade no APÊNDICE C e o diagrama de classe no APÊNDICE E, sendo finalizados nesta *Sprint*.

3.7.3 Terceira *Sprint*

A terceira *Sprint* foi realizada no dia 28 de maio. Foi iniciada a escrita da justificativa da monografia, o levantamento, estudo e testes de tecnologias para detecção e reconhecimento de objetos.

A justificativa e o levantamento foram finalizados nesta *Sprint* e estão descritos nas seções 1.2 e 2.

Nesta *Sprint* foi continuado e finalizado o resumo, iniciado na *Sprint* anterior (3.7.2).

O estudo das tecnologias e testes foi iniciado nesta *Sprint* e finalizado na próxima (3.7.4).

3.7.4 Quarta *Sprint*

A quarta *Sprint* foi realizada no dia 12 de junho. Foi feita a coleta das imagens da cédulas e o *Data Augmentation* (descrita na Seção 2.9), bem como as anotações no formato xml PASCAL VOC (Seção 3.4).

O estudo das tecnologias iniciada na terceira *Sprint* (3.7.3) foi continuado e finalizado nesta.

3.7.5 Quinta *Sprint*

A quinta *Sprint* foi realizada no dia 26 de junho. Foi iniciado o treinamento da rede neural, avaliação do modelo de classificação e testes. Estas tarefas foram finalizadas na sétima *Sprint*.

3.7.6 Sexta *Sprint*

A sexta *Sprint* foi realizada no dia 10 de julho. Foi feito a diagrama de sequências, iniciado o desenvolvimento, alterações no aplicativo Androide coleta de resultados.

O diagrama de sequências foi finalizado nesta Sprint e está descrito no APÊNDICE F.

O treinamento da rede neural, avaliação do modelo e testes iniciados na *Sprint* anterior foram continuados nesta e finalizados na sétima *Sprint*.

O desenvolvimento do aplicativo Android foi iniciado nesta *Sprint* e finalizado na sétima.

3.7.7 Sétima *Sprint*

A sétima *Sprint* foi realizada no dia 24 de julho. Foram feitos ajustes no modelo, forma de classificação das cédulas e início da escrita dos materiais e métodos da monografia.

O treinamento e avaliação da rede neural iniciada na *Sprint* anterior, foi finalizado nesta. A avaliação do modelo está descrita na Seção 4.

O aplicativo Android foi finalizado nesta *Sprint* e está descrito na Seção 4.1.

3.7.8 Oitava *Sprint*

A oitava *Sprint* foi realizada no dia 07 de agosto. Foram feitas reuniões do TCC, revisões no trabalho e monografia e início da escrita da conclusão.

A escrita da conclusão foi finalizada na nona *Sprint*.

3.7.9 Nona Sprint

A nona *Sprint* foi realizada no dia 21 de agosto. Esta *Sprint* contempla a defesa do TCC, correções e revisões.

A conclusão iniciada na *Sprint* anterior foi finalizada nesta, sendo descrita na Seção 5.

4 APRESENTAÇÃO DOS RESULTADOS

Os gráficos de treinamento podem ser visualizados através da ferramenta TensorBoard (TENSORBOARD, 2020). Este pode ser executado diretamente na ferramenta Google Colab.

Cada passo do treinamento demorou em média 30 segundos por execução, totalizando 13 dias e 200.00 passos, resultando em um *frozen graph* com *loss function* (Seção 2.2.4) de 0,7536 (GRÁFICO 2).

A FIGURA 75 demonstra uma parte de saída do treinamento da rede neural. As principais informações são: hora, número da etapa, *loss* e tempo gasto por etapa de treinamento.

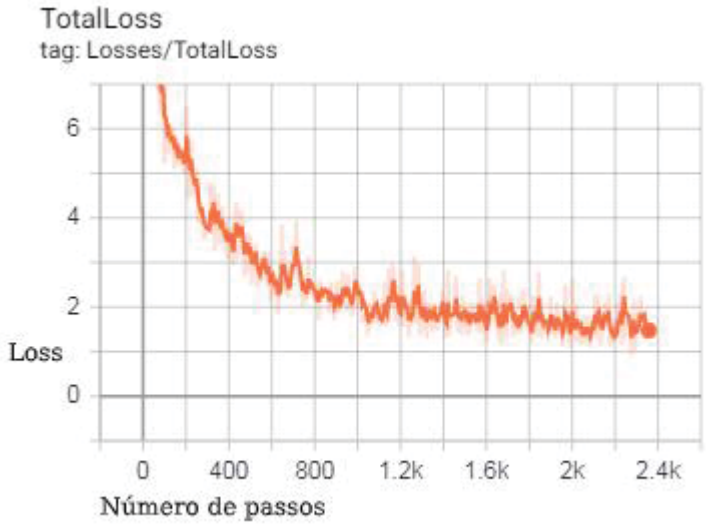
FIGURA 75 - PROGRESSO DE TREINAMENTO DA REDE

```
I0219 22:10:47.277540 140336507488000 supervisor.py:1099] global_step/sec: 0.0585059
INFO:tensorflow:global step 21376: loss = 1.6269 (18.925 sec/step)
I0219 22:11:00.156610 140338527532928 learning.py:507] global step 21376: loss = 1.6269 (18.925 sec/step)
INFO:tensorflow:global step 21377: loss = 0.7233 (16.404 sec/step)
I0219 22:11:16.561512 140338527532928 learning.py:507] global step 21377: loss = 0.7233 (16.404 sec/step)
INFO:tensorflow:global step 21378: loss = 0.7728 (16.370 sec/step)
I0219 22:11:32.932643 140338527532928 learning.py:507] global step 21378: loss = 0.7728 (16.370 sec/step)
INFO:tensorflow:global step 21379: loss = 0.7459 (16.573 sec/step)
I0219 22:11:49.507428 140338527532928 learning.py:507] global step 21379: loss = 0.7459 (16.573 sec/step)
INFO:tensorflow:global step 21380: loss = 0.5983 (16.481 sec/step)
I0219 22:12:05.989725 140338527532928 learning.py:507] global step 21380: loss = 0.5983 (16.481 sec/step)
```

FONTE: O Autor (2020)

O GRÁFICO 2 representa a saída da função de perda (*loss*) e o número de passos do treinamento. Após 200.000 passos a média dos valores da saída da função de perda sofreu pouca variação, permanecendo perto do valor 0,7536. Isto é um indicativo de que a rede atingiu uma estabilidade no treinamento. Neste caso, mais passos de treinamento não necessariamente resultam em um modelo melhor. Outro problema associado ao excesso de treinamento é o *overfitting*, no qual a rede neural fica “super-ajustada” aos dados de treinamento, perdendo sua capacidade de generalização e identificação de novos casos, como exemplo: os conjuntos de testes e aplicações reais.

GRÁFICO 2 - LOSS TOTAL



FONTE: O Autor (2020)

Os quadros QUADRO 5 e QUADRO 6 apresentam a matriz de confusão e as métricas de Acurácia, Sensibilidade, PPV, VPN, Prevalência e Especificidade. Observa-se que as notas de R$ 5,00, R$ 10,00 e R$ 100,00 verso e R$ 10,00 frente obtiveram os menores valores de sensibilidade. As anotações de “2Frente”, “2Verso”, representam as classes “DoisReaisFrente” e “DoisReaisVerso”, respectivamente, assim como: “50Frente” e “50Verso”, as classes: “CinquentaReaisFrente” e “CinquentaReaisVerso”.

As medidas de avaliação estão descritas na Seção 2.10, entre elas podemos citar: matriz de confusão (Seção 2.10.1), acurácia (Seção 2.10.3), sensibilidade (Seção 2.10.4), especificidade (Seção 2.10.5), PPV (Seção 2.10.6), VPN (Seção 2.10.7), prevalência (Seção 2.10.8), ROC (Seção 2.10.9) e função de perda (Seção 2.2.4).

As predições foram feitas em um dispositivo mobile Samsung Galaxy S10, com Android versão 10.0. Alguns resultados estão representados nas Figuras 8 e 9. Em algumas predições, a porcentagem de acerto foi baixa: cédulas de R$ 2,00 verso (50,00%), R$ 5,00 frente (65,23%) e R$ 10,00 frente (57,81%). Estes valores estão associados ao tempo de treinamento da rede, valor da função de perda, cédulas amassadas ou rasgadas, iluminação, posição da cédula, entre outros fatores. Estas porcentagens de acerto podem ser otimizadas com mais passos no treinamento, até um valor de *loss function* menor do que 0,7 (GRÁFICO 2). Neste intuito, a utilização de placas gráficas em processamento GPU pode acelerar o processo de

treinamento da rede e novas imagens podem ser adicionadas ao conjunto de testes e treinamento.

QUADRO 5 - MATRIZ DE CONFUSÃO

Classificado	2Frente	2Verso	5Frente	5Verso	10Frente	10Verso	20Frente	20Verso	50Frente	50Verso	100Frente	100Verso
2Frente	82	8	0	0	0	0	0	0	0	0	0	0
2Verso	3	67	0	0	0	0	0	0	0	0	0	0
5Frente	0	0	95	9	0	0	0	0	0	0	0	0
5Verso	0	0	4	68	0	0	0	0	0	0	0	0
10Frente	0	0	0	0	100	12	0	0	0	0	0	0
10Verso	0	0	0	0	13	79	0	0	0	0	0	0
20Frente	1	0	0	0	0	0	113	2	0	0	1	0
20Verso	0	0	0	0	0	0	2	81	0	1	1	0
50Frente	0	0	0	0	0	0	0	0	112	8	0	0
50Verso	0	0	0	0	0	0	0	0	5	68	0	1
100Frente	0	0	0	0	0	0	0	0	0	0	128	7
100Verso	0	0	0	0	0	0	0	0	0	0	5	30

FONTE: O Autor (2020)

As métricas (QUADRO 6) foram obtidas a partir dos dados da matriz de confusão, representada no QUADRO 5. As fórmulas utilizadas estão descritas na Seção 2.10.

QUADRO 6 - MÉTRICAS

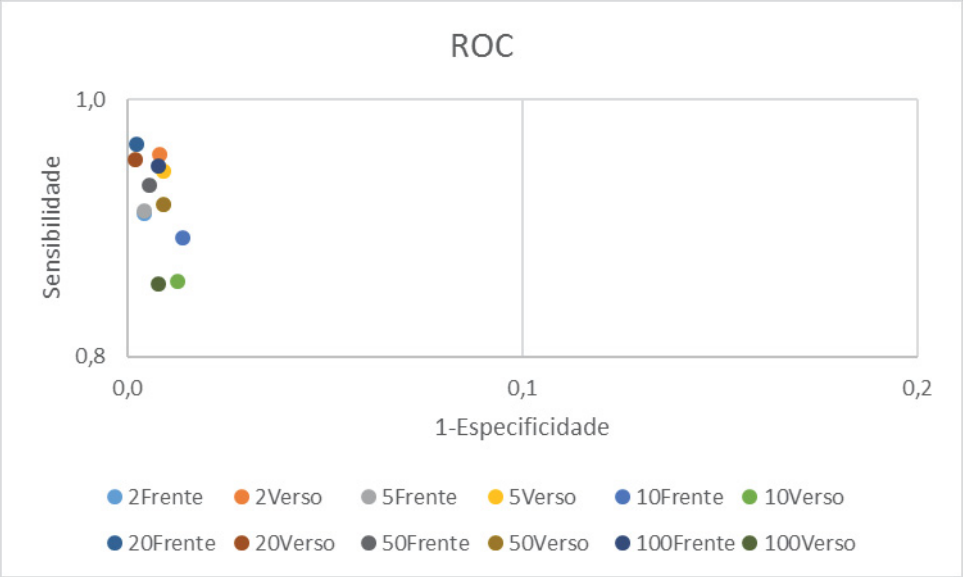
	Acurácia	Sensibilidade	PPV	VPN	Prevalência	Especificidade	1-Espec.
2Frente	0,9884	0,9111	0,9535	0,9916	0,0870	0,9958	0,0042
2Verso	0,9894	0,9571	0,8933	0,9969	0,0677	0,9917	0,0083
5Frente	0,9875	0,9135	0,9596	0,9904	0,1004	0,9957	0,0043
5Verso	0,9875	0,9444	0,8831	0,9958	0,0695	0,9907	0,0093
10Frente	0,9761	0,8929	0,885	0,9872	0,1069	0,9861	0,0139
10Verso	0,9761	0,8587	0,8681	0,9864	0,0878	0,9874	0,0126
20Frente	0,9942	0,9658	0,9826	0,9956	0,1137	0,9978	0,0022
20Verso	0,9942	0,9529	0,9759	0,9958	0,0826	0,9979	0,0021
50Frente	0,9875	0,9333	0,9573	0,9913	0,1158	0,9945	0,0055
50Verso	0,9855	0,9189	0,8831	0,9938	0,0713	0,9907	0,0093
100Frente	0,9865	0,9481	0,9481	0,9922	0,1302	0,9922	0,0078
100Verso	0,9875	0,8571	0,7895	0,995	0,0338	0,992	0,008
Média	0,9867	0,9212	0,9149	0,9927	0,0889	0,9927	0,0073

FONTE: O Autor (2020)

O gráfico ROC está representado no GRÁFICO 3. A partir das métricas QUADRO 6, o gráfico ROC foi criado com base nos dados de Sensibilidade e Especificidade. O modelo apresenta melhores resultados para as cédulas de R$

20,00, frente e verso, R$ 2,00 frente, R$ 100,00 frente, R$ 5,00 verso e R$ 50,00, cujos pontos ficaram mais próximos ao canto superior esquerdo (maior especificidade e sensibilidade).

GRÁFICO 3 - ROC



FONTE: O Autor (2020)

4.1 APLICAÇÃO

A aplicação Android base foi adaptada dos modelos de TensorFlow *examples* (TENSORFLOW *EXAMPLES*, 2020). O projeto Android base está descrito na Seção 3.6. O aplicativo precisa de permissões de acesso a câmera do celular para poder capturar as imagens das cédulas (FIGURA 76). O ícone adotado foi o padrão das aplicações de exemplo (FIGURA 77) e o nome dado foi ‘TFL Classify’. Para desenvolvimento do aplicativo Android foi utilizado o software Android Studio (ANDROID STUDIO, 2020), utilizando a linguagem Java (Seção 2.13).

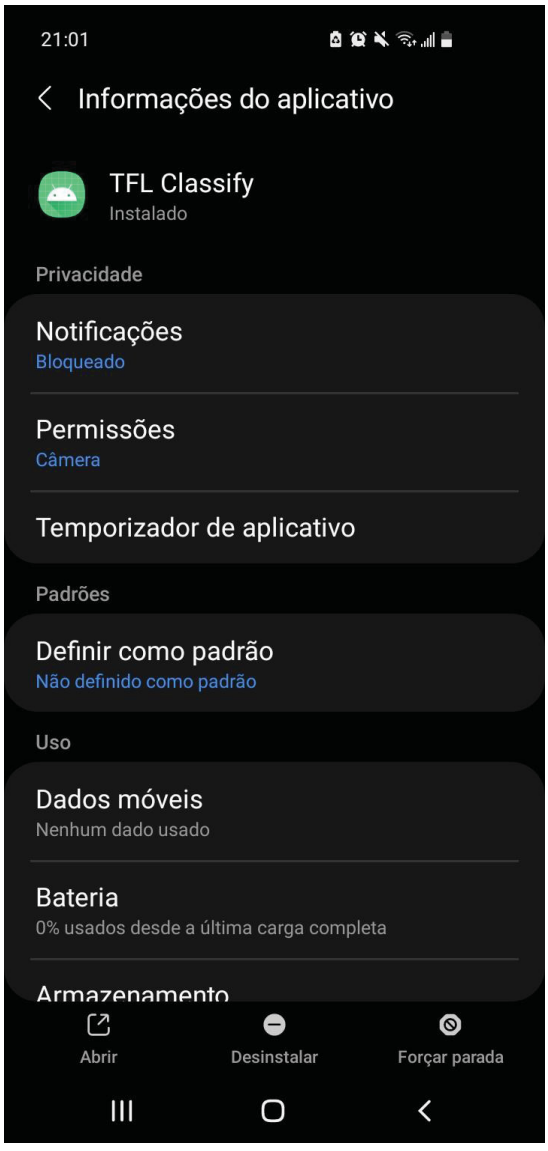
A aplicação Android apresenta apenas uma tela, representada na FIGURA 78. A ideia principal é tornar a aplicação o mais simples possível para ser utilizada por uma pessoa com deficiência visual. Sem telas adicionais, telas de configuração, botões, menus e ações extras que o usuário precise realizar para identificar as cédulas. A usabilidade da aplicação está em abrir o software, utilizar a câmera do celular para capturar a imagem da cédula, tocar na tela e aguardar a vocalização da classe identificada. Para uma nova detecção, basta manter o software executando,

capturar a imagem de uma nova cédula, tocar na tela e aguardar a vocalização da classe identificada, e assim sucessivamente. A aplicação captura de forma contínua as imagens da câmera, realizando as predições de forma ininterrupta.

Ao ser executada, abre diretamente a câmera do celular, representando as imagens capturadas nesta tela. Durante a execução, a aplicação captura as imagens e realiza a classificação através do modelo de rede neural treinado, exibe o resultado da classificação na tela e, após um toque na tela, vocaliza o resultado (Seção 3.5).

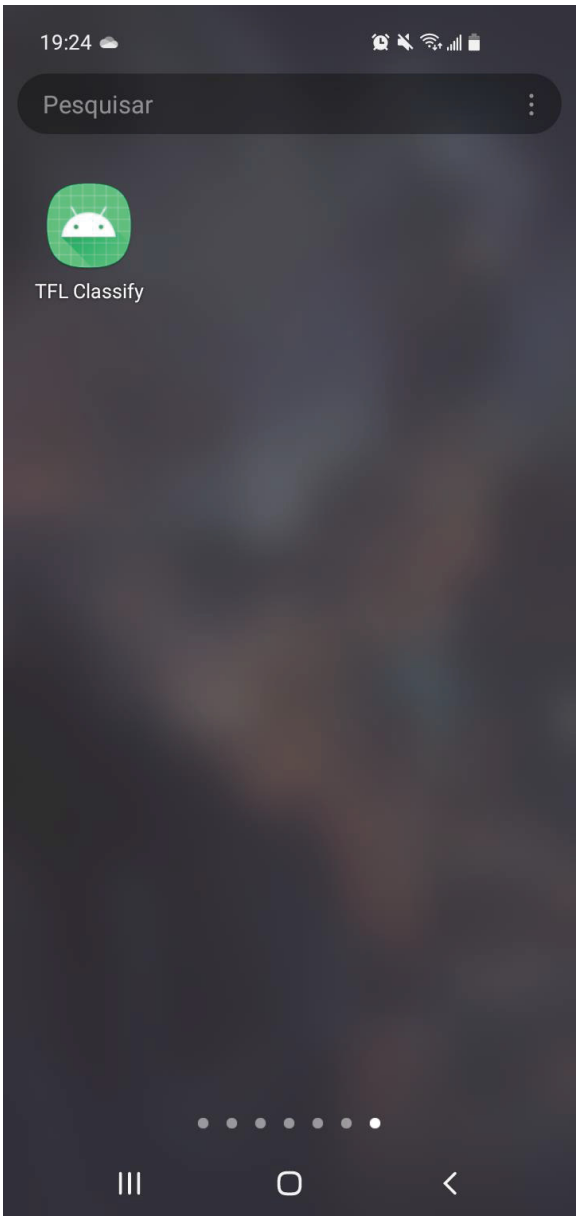
Alguns exemplos de predições estão descritos na Seção 4.1.1.

FIGURA 76 – PERMISSÕES APLICATIVO



FONTE: O Autor (2020)

FIGURA 77 - ÍCONE APLICAÇÃO



FONTE: O Autor (2020)

FIGURA 78 - TELA INICIAL

FONTE: O Autor (2020)

4.1.1 Exemplos de predições

A FIGURA 79 demonstra as predições da cédula de R$ 2,00 de frente e verso, respectivamente. O sistema apresenta as 3 primeiras classes e suas respectivas probabilidades. A classe com maior probabilidade é exibida por primeiro, seguida das demais classes. Para esta amostra, a rede neural obteve alguns falsos positivos, como a classificação para as classes de R$ 20,00 e R$ 100,00. Como, para ambos os casos, a classe de maior predição foi R$ 2,00 frente e verso, respectivamente, caso o usuário toque na tela, o sistema irá vocalizar a classe com de maior probabilidade, no caso R$ 2,00.

Seguindo a mesma lógica, a FIGURA 80 demonstra as predições para a cédula de R$ 5,00 frente e verso. Para esta amostra, a rede neural obteve os falsos positivos de classificação para as classes de R$ 10,00 e R$ 20,00. A classe de maior probabilidade foi a de R$ 5,00 (frente e verso).

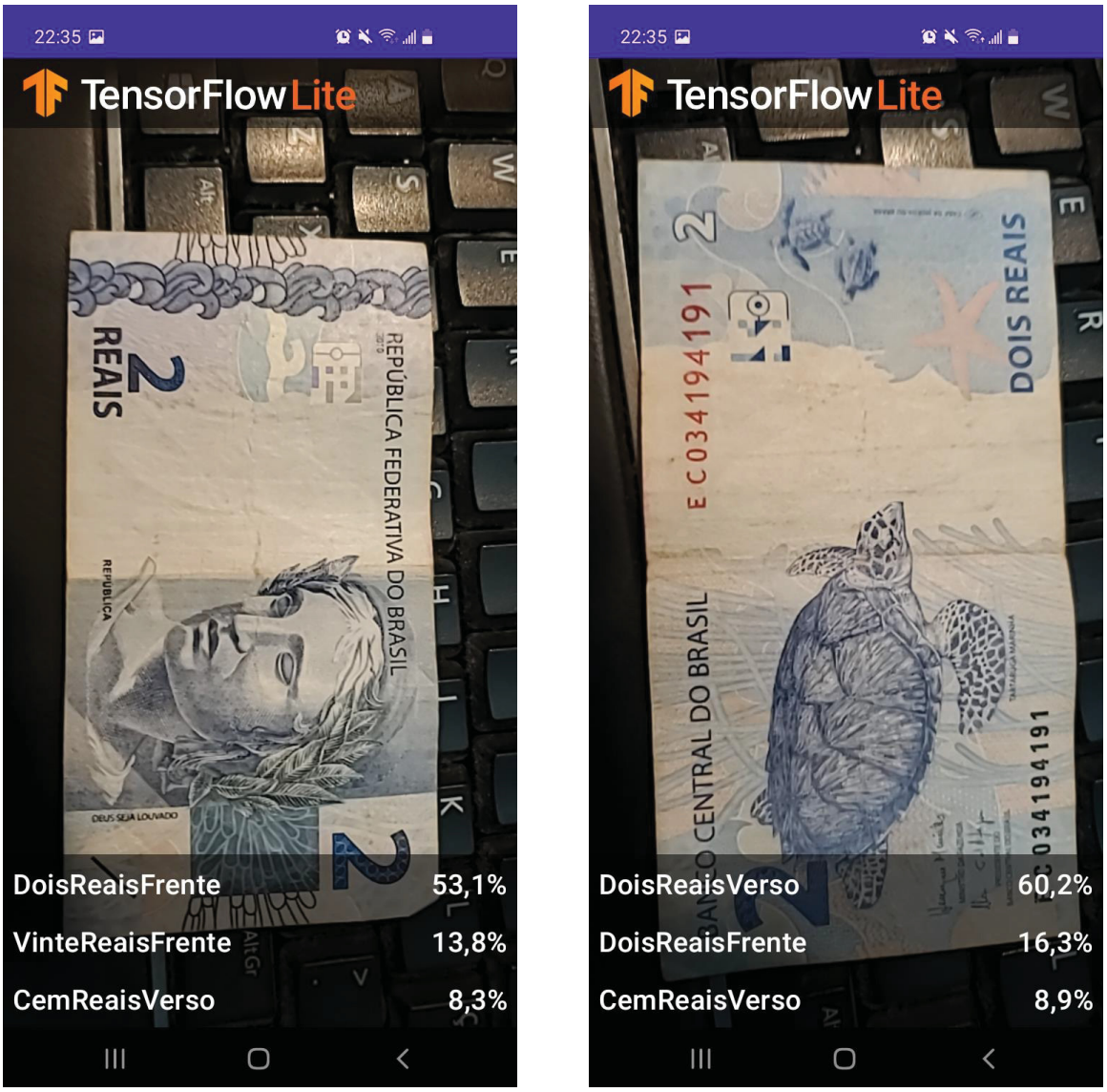
A FIGURA 81 demonstra as predições para a cédula de R$ 10,00 frente e verso. Para esta amostra, a rede neural obteve os falsos positivos de classificação para as classes de R$ 20,00 e R$ 50,00. A classe de maior probabilidade foi a de R$ 10,00 (frente e verso).

A FIGURA 82 demonstra as predições para a cédula de R$ 20,00 frente e verso. Para esta amostra, a rede neural obteve os falsos positivos de classificação para as classes de R$ 10,00, R$ 50,00 e R$ 100,00. A classe de maior probabilidade foi a de R$ 20,00 (frente e verso).

A FIGURA 83 demonstra as predições para a cédula de R$ 50,00 frente e verso. Para esta amostra, a rede neural obteve os falsos positivos de classificação para a classe de R$ 20,00. A classe de maior probabilidade foi a de R$ 50,00 (frente e verso).

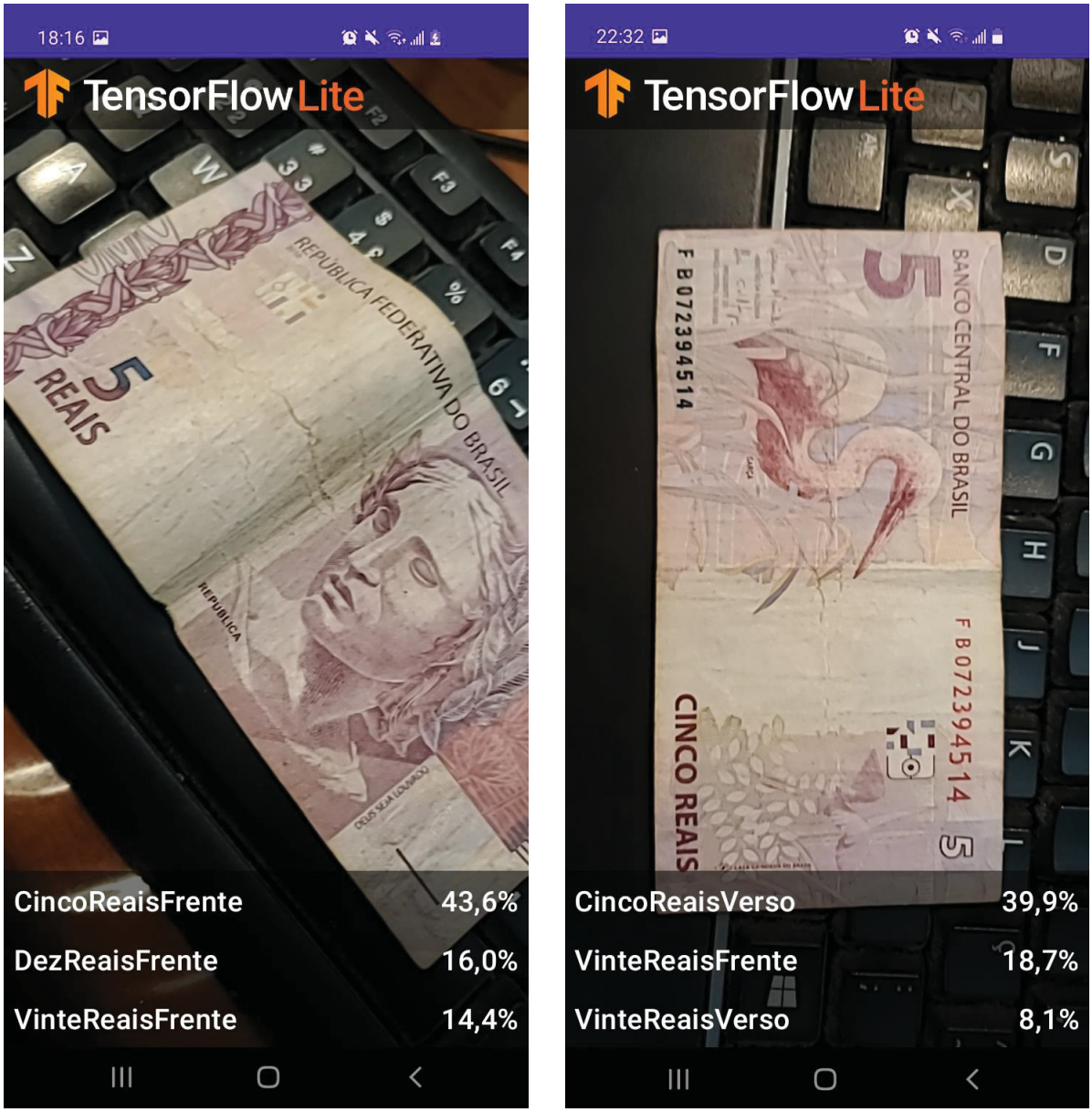
A FIGURA 84 demonstra as predições para a cédula de R$ 100,00 frente e verso. Para esta amostra, a rede neural obteve os falsos positivos de classificação para a classe de R$ 2,00. A classe de maior probabilidade foi a de R$ 100,00 (frente e verso).

FIGURA 79 - DETEÇÃO CÉDULA R$ 2,00



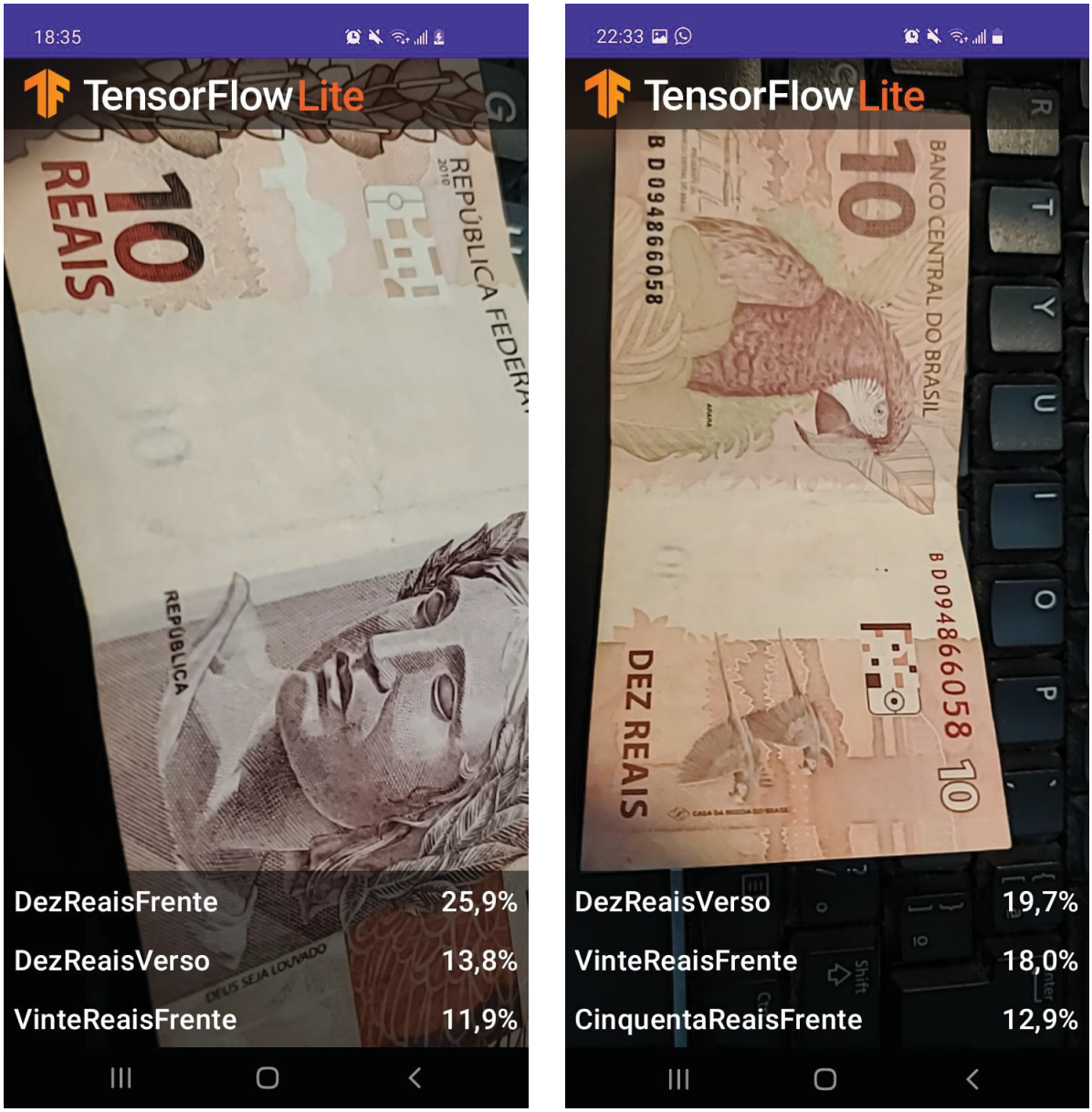
FONTE: O Autor (2020)

FIGURA 80 - DETEÇÃO CÉDULA R$ 5,00



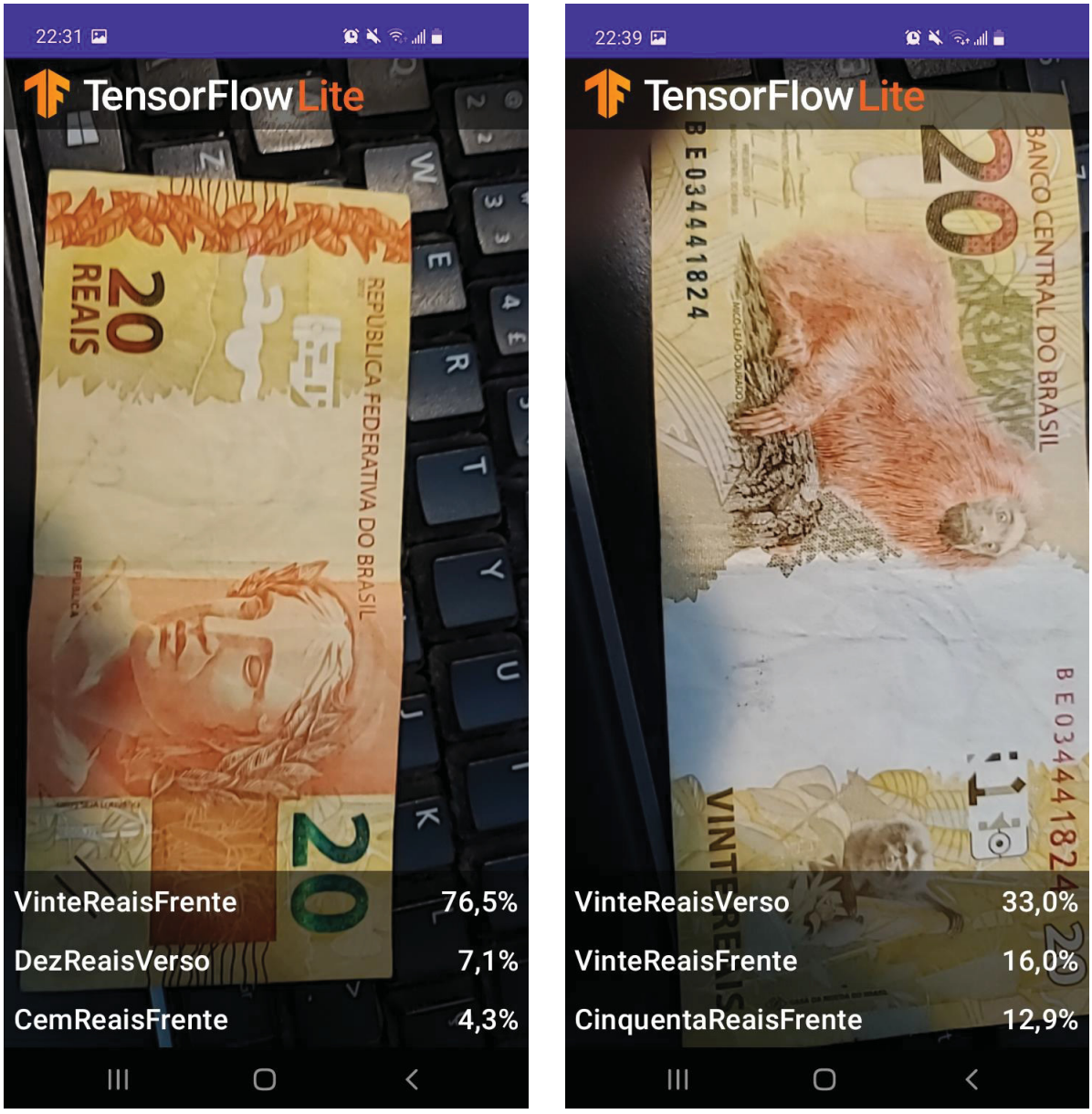
FONTE: O Autor (2020)

FIGURA 81 - DETEÇÃO CÉDULA R$ 10,00



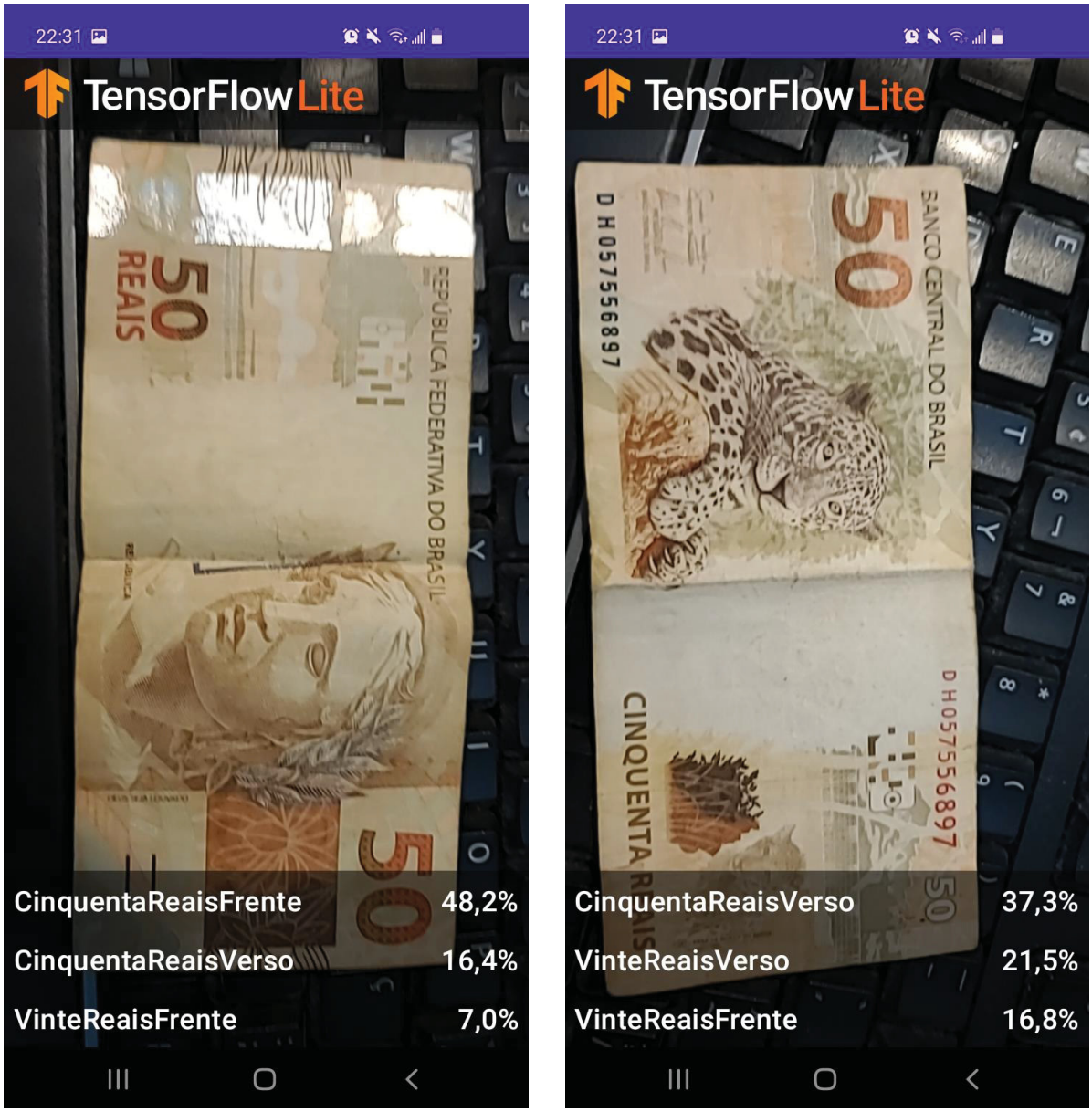
FONTE: O Autor (2020)

FIGURA 82 - DETEÇÃO CÉDULA R$ 20,00



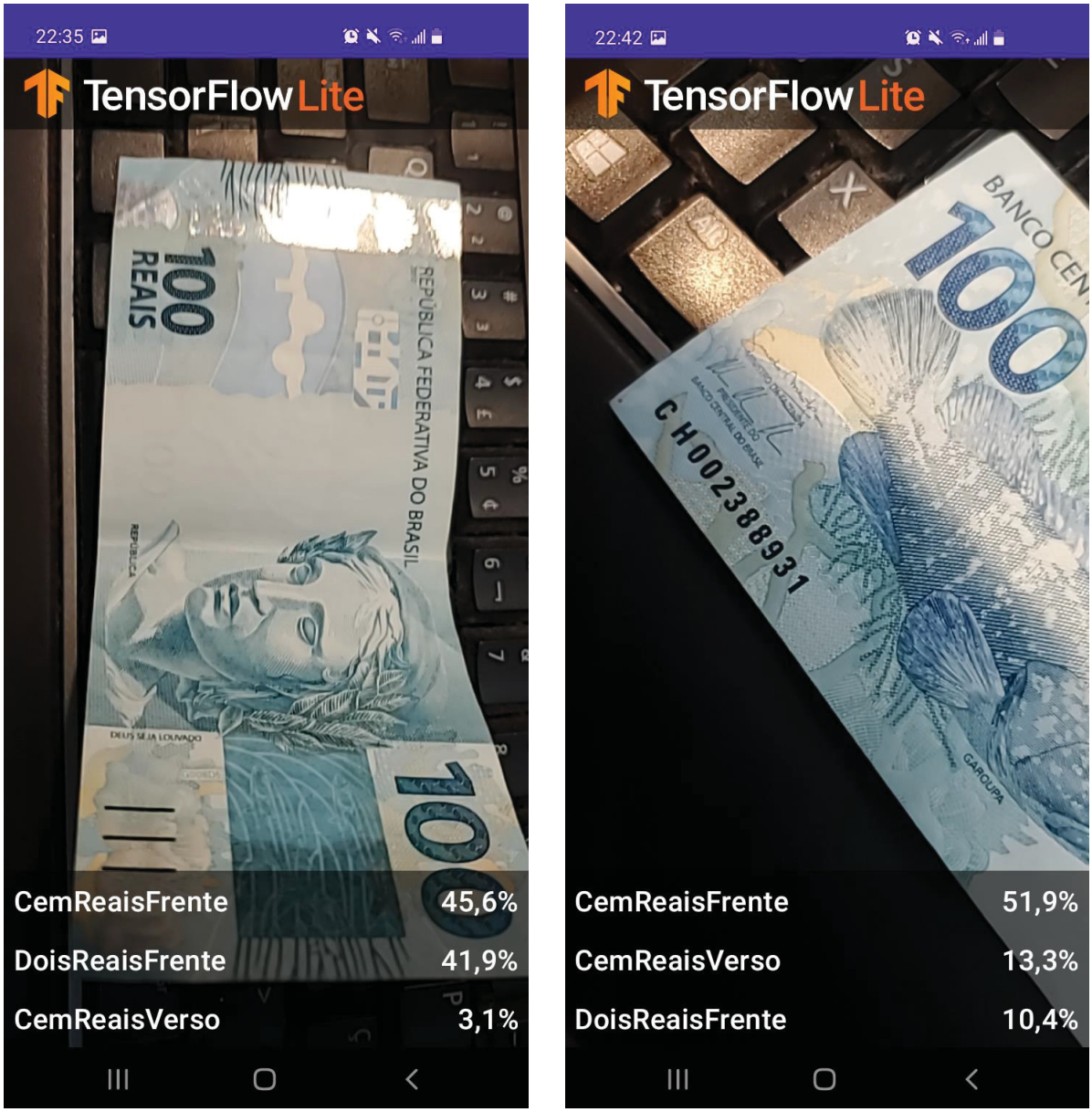
FONTE: O Autor (2020)

FIGURA 83 - DETEÇÃO CÉDULA R$ 50,00



FONTE: O Autor (2020)

FIGURA 84 - DETEÇÃO CÉDULA R$ 100,00



FONTE: O Autor (2020)

5 CONSIDERAÇÕES FINAIS

Por meio do presente trabalho foi possível desenvolver uma aplicação Android capaz de reconhecer cédulas de real em tempo real utilizando a câmera de um dispositivo móvel. Para cada cédula capturada pela câmera do celular, a aplicação reconhece uma das 12 classes de real e vocaliza a classe identificada, através da reprodução de arquivos .mp3. Os arquivos de áudio estão distribuídos em 6 áudios representativos de cada classe, observando que cada cédula possui frente e verso, sendo, por exemplo, para duas classes da nota de R$ 2,00 frente e verso, um arquivo de áudio que reproduz a vocalização do texto de dois reais.

O uso da ferramenta Google Colab (GOOGLE COLABORATORY, 2020) foi de grande auxílio para o desenvolvimento deste trabalho, pois possui um arcabouço de ferramentas, recursos e tecnologias que auxiliam no desenvolvimento, treinamento, coleta de dados, entre outros, disponíveis e de fácil acesso.

A fase de seleção das imagens das cédulas, definição e separação das classes de classificação com o software LabelImg é uma parte trabalhosa, mas crucial para o sucesso do treinamento da rede neuronal. A iluminação das fotos das cédulas, posição, dobraduras, amassados, rasgos, partes faltantes, posição dos dedos ao segurar as cédulas, entre outros, são fatores importantes para o treinamento. Cédulas antigas e novas/novos modelos, devem ser levadas em consideração, pois classificar uma cédula antiga de R$ 20,00 e uma cédula nova de R$ 20,00 em uma mesma classe pode prejudicar o resultado de classificação da rede.

Separar as cédulas em duas classes frente de verso foi um diferencial na classificação supervisionada, otimizando o tempo de treinamento. Em vista que a parte da frente de uma cédula é visivelmente diferente da parte de trás, treinar as duas imagens como se fossem uma mesma classe prejudicaria a qualidade de inferência da rede. Inicialmente o treinamento não levou em consideração essa peculiaridade de separação de classes, o que resultou em um modelo com muitos falsos positivos e baixa qualidade de classificação. Após esta separação, o modelo resultante apresentou melhoras no tempo de treinamento e qualidade de identificação das cédulas.

Após 200.000 passos no treinamento, observa-se uma estabilidade nos valores de saída da função de perda (*loss function*), representado no GRÁFICO 2,

permanecendo perto de 0,75. Isto é um indicativo de que o treinamento chegou em ponto de estabilidade, e mais passos no treinamento não resultam necessariamente em um modelo melhor. Neste caso, continuar a treinar a rede poderia levar a um problema de overfitting, ou “super-ajuste”, no qual a rede neural perde a sua capacidade de generalização e identificação das classes treinadas.

De acordo com a curva ROC (GRÁFICO 3), as classes com piores resultados de reconhecimento deste modelo foram as classes representativas das notas de R$ 10,00 e R$ 100,00, verso. Em algumas predições, a porcentagem de acerto foi baixa: cédulas de R$ 2,00 verso (50,00%), R$ 5,00 frente (65,23%) e R$ 10,00 frente (57,81%) (Seção 4). No intuito de melhorar a classificação das notas, diversos pontos podem ser usados para caracterizar as cédulas, como numeração (2, 5, 10, 20, 50 e 100), texto escrito (dois, cinco, dez, vinte, cinquenta e cem). Cada cédula possui um animal associado, sendo: R$ 2,00: Tartaruga-de-pente, R$ 5,00: Garça-branca-grande, R$ 10,00: Arara-vermelha, R$ 20,00: Mico-leão-dourado, R$ 50,00: Onça Pintada e R$ 100,00: Garoupa. Cada ponto de referência pode ser treinado em uma nova classe, de forma a auxiliar no processo de classificação das cédulas.

5.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

O sistema atual não prevê a detecção de cédulas falsas. Este sistema envolve fatores como detecção de padrões de autenticação de cédulas de real, como marca d'água, tamanho da cédula, fio de segurança, elementos em alto-relevo, valor da cédula que é exibido em determinadas posições e luminosidade, elementos fluorescentes, faixa holográfica, entre outros (BANCO CENTRAL DO BRASIL, 2020). Esta verificação está além do escopo deste trabalho.

Comparado a um servidor ou computador, o *hardware* de um dispositivo móvel possui diversas limitações de memória, processamento de CPU e qualidade de câmera. Deve-se levar em consideração estes fatores ao processar imagens para classificações em tempo real. Neste trabalho, o classificador apresentou algumas situações de falso-positivo, devido à qualidade do modelo treinado, iluminação, tempo de resposta do *hardware*, posição da cédula, qualidade de câmera, etc. Uma alternativa para o processamento em tempo real, seria treinar uma rede convolucional CNN, R-CNN ou Fast-RCNN, executando em um servidor com

comunicação do aplicativo. Neste caso, os maiores problemas seriam: a latência de rede, ou a falta de acesso à *internet* devido a áreas de sombra ou instabilidade da rede móvel, consumo de energia e tempo de resposta (tempo usado na classificação).

REFERÊNCIAS

ACOSTA, S. G. **How to export a TensorFlow 2.x Keras model to a frozen and optimized graph**. Disponível em: https://medium.com/@sebastingarcaacosta/how-to-export-a-tensorflow-2-x-keras-model-to-a-frozen-and-optimized-graph-39740846d9eb. Acesso em: 21 dez. 2020.

ALGORITHMIA. **Introduction to Loss Functions**. 2018 não paginado. Disponível em: https://algorithmia.com/blog/introduction-to-loss-functions. Acesso em: 28 dez. 2020.

ALMEIDA, F.; GODINHO, F.; COELHO, D. F.; TRANSMONTANA, M. **Internet para Necessidades Especiais**. UTAD / GUIA, 2019. E-book. Disponível em: http://www.acessibilidade.net/web/ine/livro.html. Acesso em: 16 dez. 2020.

ALVEZ, G. **Understanding ConvNets (CNN)**. 2018. Disponível em: https://medium.com/neuronio/understanding-convnets-cnn-712f2afe4dd3. Acesso: em: 18 dez. 2020.

ANDREWS, J. **Deep Learning Models With MATLAB And Cortex-A**. 2019. Disponível em: https://semiengineering.com/deep-learning-models-with-matlab-and-cortex-a/. Acesso em: 18 dez. 2020.

ANDROID STUDIO. **Software para desenvolvimento de aplicativos Android**. Disponível em: https://developer.android.com/. Acesso em: 24 dez. 2020.

BANCO CENTRAL DO BRASIL. Material de apoio. **Aprenda a identificar se o seu dinheiro é real**. 2 p. Disponível em: https://www.bcb.gov.br/novasnotas/assets/downloads/material-apoio/2e5/Folheto_br.pdf. Acesso em: 15 fev. 2020.

_____. **O brasileiro e sua relação com o dinheiro**. Disponível em: https://www.bcb.gov.br/content/cedulasemoedas/pesquisabrasileirodinheiro/Apresentacao_brasileiro_relacao_dinheiro_2018.pdf. Acesso em: 25 ago. 2020.

BARANAUSKAS, J. A. **Conceitos Adicionais e Métricas**. Departamento de Física e Matemática. FFCLRP. USP. Disponível em: http://dcm.ffclrp.usp.br/~augusto/teaching/ami/AM-I-Conceitos-Adicionais-Metricas.pdf. Acesso em: 29 dez. 2020.

BRASIL. **Lei nº 8.880, de 27 de maio de 1994**. Dispõe sobre o Programa de Estabilização Econômica e o Sistema Monetário Nacional, institui a Unidade Real de Valor (URV) e dá outras providências. Diário Oficial da União, Brasília, DF, Seção 1, edição extra, 28 mai. 1994. Seção 1, p. 7861.

BERRY, M. J. A.; LINOFF, G. **Data Mining Techniques: For Marketing, Sales, and Customer Support**. New York: Wiley Computer Publishing, 2004.

CALADO, F. A. R. **Machine Learning na Prática**: como escolher seus algoritmos?. Disponível em: http://igti.com.br/blog/machine-learning-na-pratica-como-escolher-seus-algoritmos/. Acesso em: 17 dez. 2020.

CALLAWAY, E. **'It will change everything': DeepMind's AI makes gigantic leap in solving protein structures**. Nature 588, 203-204, 2020. Disponível em: https://www.nature.com/articles/d41586-020-03348-4. Acesso em: 17 dez. 2020. https://doi.org/10.1038/d41586-020-03348-4.

CASA DA MOEDA. **Origem do Dinheiro**. Disponível em: https://www.casadamoeda.gov.br/portal/socioambiental/cultural/origem-do-dinheiro.html. Acesso em: 25 ago. 2020.

CASTELLI, I. **Trello**: como esta ferramenta pode ajudar você a organizar a sua vida. Disponível em: https://www.tecmundo.com.br/organizacao/75128-trello-ferramenta-ajudar-voce-organizar-vida.htm. Acesso em: 22 dez. 2020.

CARVALHO, A. P. L. F. **Redes Neurais Artificiais**. Disponível em: http://conteudo.icmc.usp.br/pessoas/andre/research/neural/. Acesso em: 06 dez. 2020.

COCO. **Common objects in context**. Disponível em: http://cocodataset.org/#home. Acesso em: 10 fev. 2020.

COHN, M.; KEITH C. **Agile Game Development with Scrum**. Addison-Wesley Professional: Boston, 2010.

COOPER, S. **Deep Learning for Beginners**: A comprehensive introduction of deep learning fundamentals for beginners to understanding frameworks, neural networks, large datasets, and creative applications with ease. Data Science; Illustrated, 2019.

DEREKJCHOW. **Tensorflow Object Detection API using Faster RCNN on Android devices**. Disponível em: https://github.com/tensorflow/models/issues/4848. Acesso em: 15 jan. 2020.

DEITEL, P. J.; DEITEL, H. M. **Java How to Program, Early Objects**, 11th edição. Nova Iorque: Pearson, 2017.

DINIZ, B. **DINHEIRO PARA CEGOS - Os padrões monetários e a comunidade cega brasileira**. Disponível em: https://www.diniznumismatica.com/2019/04/dinheiro-para-cegos.html. Acesso em: 16 dez. 2020.

DURETTE, P. N. **Google Translate's text-to-speech API - gTTS (Google Text-to-Speech)**, 2014. Disponível em: https://pypi.org/project/gTTS/. Acesso em: 19 fev. 2020.

EVERINGHAM, M. et al. The PASCAL **Visual Object Classes (VOC) Dataset and Challenge**. PASCAL - Pattern Analysis, Statistical modelling and Computational Learning. Disponível em: http://host.robots.ox.ac.uk/pascal/VOC/index.html. Acesso em: 2 fev. 2020.

FACELI, K et al. **Inteligência Artificial** - Uma Abordagem de Aprendizado de Máquina. 1ª edição. Rio de Janeiro, 2011.

FERNEDA, E. **Redes neurais e sua aplicação em sistemas de recuperação de informação**. Ribeirão Preto, USP, 2006. Disponível em: http://www.scielo.br/pdf/ci/v35n1/v35n1a03.pdf. Acesso em: 17 dez. 2020.

FUKUSHIMA, K. **Neocognitron**: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol. Cybernetics 36, 193–202 (1980). Disponível em: https://link.springer.com/article/10.1007/BF00344251#Bib1. Acesso em: 17 dez. 2020. https://doi.org/10.1007/BF00344251.

GARCIA, R. **A evolução da moeda brasileira** - Uma breve história do nosso dinheiro desde os tempos do Império. Disponível em: https://vejasp.abril.com.br/blog/memoria/evolucao-moeda-brasileira. Acesso em: 25 ago. 2020.

GIRSHICK, R. **Fast R-CNN**. 2015. Disponível em: https://arxiv.org/pdf/1504.08083.pdf. Acesso em: 09 jun. 2021.

_____; et al. **Rich feature hierarchies for accurate object detection and semantic segmentation**. 2014. Disponível em: https://arxiv.org/pdf/1311.2524.pdf7. Acesso em: 07 jun. 2021.

GOMES, P. C. T. **Principais algoritmos de machine learning**. 2019. Disponível em: https://www.datageeks.com.br/algoritmos-de-machine-learning/. Acesso em: 18 dez. 2020.

GONÇALVES, C. B. **Casa da Moeda do Brasil - 290 Anos de História**. Do Autor, 1984.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning (Adaptive Computation and Machine Learning series)**. The MIT Press, 2016. 800 p.

_____, et al. **Generative adversarial nets**. In: Advances in neural information processing systems. [S.l.: s.n.], 2014. p. 2672–2680.

GOOGLE COLABORATORY. **Ambiente Python na nuvem voltado para estudos de Inteligência Artificial**. Disponível em: https://colab.research.google.com/notebooks/intro.ipynb#scrollTo=5fCEDCU_qrC0. Acesso em: 13 jan. 2020.

GOOGLE DRIVE. **Serviço de armazenamento e sincronização de arquivos**. Disponível em: https://www.google.com/intl/pt-BR_ALL/drive/. Acesso em: 13 jan. 2020.

GOOGLE TPU. **Serviço em nuvem de processamento de Unidades de Processamento de Tensor - TPU**. Disponível em: https://cloud.google.com/tpu/. Acesso em: 14 jan. 2020.

GÖLGE, E. **Online Hard Example Mining on PyTorch**. Disponível em: http://www.erogol.com/online-hard-example-mining-pytorch/. Acesso em: 14 jan. 2020.

GROVER, P. **5 Regression Loss Functions All Machine Learners Should Know**. Disponível em: https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0. Acesso em: 05 jun. 2021.

HASANUZZAMAN, F. M. YANG, X. TIAN, Y. **Robust and effective component-based banknote recognition by SURF features**. 2011 20th Annual Wireless and Optical Communications Conference (WOCC), Newark, NJ, 2011, pp. 1-6, doi: 10.1109/WOCC.2011.5872294. Disponível em: https://ieeexplore.ieee.org/document/5872294. Acesso em: 25 mar. 2020.

HEBB, D. O. **The Organization of Behavior**: A Neuropsychological Theory. Psychology Press, 1949.

HINTON, G.; SALAKHUTDINOV, R. **Deep Boltzmann Machines**. Proceedings of the International Conference on Artificial Intelligence and Statistics, v. 5, 448-455, 2009. Disponível em: http://www.utstat.toronto.edu/~rsalakhu/papers/dbm.pdf. Acesso em: 17 dez. 2020.

HOWARD, G. A. et al. **MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications**. Cornell University, Google Inc, Computer Vision and Pattern Recognition, p. 1-9, 2017. Disponível em: https://arxiv.org/abs/1704.04861. Acesso em: 01 fev. 2020.

HUANG, J. **Accelerated Training and Inference with the Tensorflow Object Detection API**. Google AI Perception, 2018. Disponível em: https://ai.googleblog.com/2018/07/accelerated-training-and-inference-with.html. Acesso em: 18 jan. 2020.

HUBEL, D. H.; WIESEL, T. N. **Receptive fields, binocular interaction and functional architecture in the cat's visual cortex**. Journal of Physiology, 160, 106-154, 1962. Disponível em: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1359523/. Acesso em: 18 dez. 2020.

IBGE. **Pessoas com deficiência visual, total, percentual e coeficiente de variação, por grupos de idade e situação do domicílio**. Disponível em https://sidra.ibge.gov.br/tabela/5753. Acesso em: 25 ago. 2020.

JUNIOR, R. V. de S. **Estudo de métodos de inteligência computacional para detecção de humanos em imagens de webcam**. Orientador: Arthur Plinio de Souza Braga. 2018. 82 f. TCC (Graduação) – Curso de Engenharia Elétrica, Universidade Federal do Ceará, Fortaleza, 2018. Disponível em: http://www.repositorio.ufc.br/bitstream/riufc/45098/3/2018_tcc_rvsousajunior.pdf. Acesso em: 06 jun. 2021.

KAWAMURA, T. **Interpretação de um teste sob a visão epidemiológica**: eficiência de um teste. Arq. Bras. Cardiol., São Paulo, v. 79, n. 4, p. 437-441, out. 2002. Disponível em: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0066-782X2002001300015&lng=en&nrm=iso. Acesso em: 28 dez. 2020. https://doi.org/10.1590/S0066-782X2002001300015.

KHANDELWAL, R. **COCO and Pascal VOC data format for Object detection. Understanding annotation data formats for computer vision**. 2019. Disponível em: https://towardsdatascience.com/coco-data-format-for-object-detection-a4c5eaf518c5. Acesso em: 21 dez. 2020.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. **ImageNet classification with deep convolutional neural networks**. NIPS'12: Proceedings of the 25th International Conference on Neural Information Processing Systems, v. 1, p. 1097–1105, 2012. Disponível em: https://dl.acm.org/doi/10.5555/2999134.2999257. Acesso em: 18 dez. 2020.

KUNAR, A. **Object Detection with SSD and MobileNet**. Disponível em: https://aditya-kunar-52859.medium.com/object-detection-with-ssd-and-mobilenet-aeedc5917ad0. Acesso em: 06 jun. 2021.

RITI, P. **Pro DevOps with Google Cloud Platform**: With Docker, Jenkins, and Kubernetes. Apress: Nova Iorque, 2018.

LOVE, R. **Linux system programming**: talking directly to the kernel and C library. O'Reilly Media: Califórnia, 2013.

LECHETA, R. **Google Android**. Aprenda a criar aplicações para dispositivos móveis com o Android SDK. Novatec: São Paulo, 2015.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. **Gradient-based learning applied to document recognition**. Proceedings of the IEEE, v. 86, no. 11, pp. 2278-2324, 1998. Disponível em: https://ieeexplore.ieee.org/document/726791. Acesso em: 18 dez. 2020. https://doi.org/10.1109/5.726791.

LIU, W. et al. **SSD: Single Shot MultiBox Detector**. University of Michigan, Computer Vision and Pattern Recognition, p. 1-17, 2016. Disponível em: https://arxiv.org/abs/1512.02325. Acesso em: 1 fev. 2020.

LOPES, L. **Como os cegos diferenciam as notas de dinheiro ?**. Disponível em: http://revistaepoca.globo.com/Revista/Epoca/0,,EMI103120-15223,00-COMO+OS+CEGOS+DIFERENCIAM+AS+NOTAS+DE+DINHEIRO.html. Acesso em: 25 ago. 2020.

LOPES, L. C. R. S. **Classificador de cédulas de Real: Duas abordagens, linear e não-linear**. Instituto de Informática – Universidade Federal de Goiás (UFG), p. 1-6, 2014. Disponível em: http://ww2.inf.ufg.br/~gustavo/courses/grad/pra/final_projetcts/2014.2/Classificador%20de%20c%20%CC%81edulas%20de%20Real:%20Duas%20abordagens,%20linear%20e%20nao-linear.pdf. Acesso em: 25 mar. 2020

LUTZ, M; ASCHER, D. **Aprendendo Python**. 2ª edição. Porto Alegre: Bookman, 2007.

MARGOTTO, P. R. **CURVA ROC**. Como fazer e interpretar no SPSS. Curso de Medicina da Escola Superior de Ciências da Saúde. ESCS. SES. DF. 2010.

MARTINS, E. **O que é MP3?**. 2008. Disponível em: https://www.tecmundo.com.br/musica/214-o-que-e-mp3-.htm. Acesso em: 21 de dez. 2020.

MARTINS, S. B. **Introdução ao Processamento Digital de Imagens**. Universidade Estadual de Campinas, Instituto de Computação. Disponível em: https://www.ic.unicamp.br/~ra144681/misc/files/ApostilaProcDeImagesParteI.pdf. Acesso em: 21 dez. 2020.

MATOS, D. **Big Data + Deep Learning = Google TensorFlow**. Disponível em: https://www.cienciaedados.com/big-data-deep-learning-google-tensorflow/. Acesso em: 19 dez. 2020.

MCKINNEY, W. **Python Para Análise de Dados**: Tratamento de Dados com Pandas, NumPy e IPython. São Paulo: Novatec, 2018.

MAGALHÃES, A. L. **O que é e para que serve o arquivo XML**. Disponível em: https://canaltech.com.br/software/xml-o-que-e/. Acesso em: 21 dez. 2020.

MCCULLOCH, W.S., PITTS, W. **A logical calculus of the ideas immanent in nervous activity**. Bulletin of Mathematical Biophysics 5, 115–133 (1943). Disponível em: https://link.springer.com/article/10.1007/BF02478259. Acesso em: 17 dez. 2020. https://doi.org/10.1007/BF02478259.

NIPS. Workshop: **Deep Learning for Speech Recognition and Related Applications**, Whistler, BC, Canadá, dez. 2009.

OLIVAS, E. S. et al. **Handbook of Research on Machine Learning Applications and Trends**: Algorithms, Methods, and Techniques. Nova Iorque: Information Science Reference, 2009.

OLIVEIRA, W. D. G. **Data Augmentation via Generative Adversarial Networks aplicado em classificação de imagens**. 113 f. Monografia de graduação (Bacharelado em Ciência da Computação) - Universidade Federal de São Paulo, São José dos Campos, 2019.

POKHREL, S. **Image Data Labelling and Annotation** — Everything you need to know. Learn about different types of annotations, annotation formats and annotation tools. Disponível em: https://towardsdatascience.com/image-data-labelling-and-annotation-everything-you-need-to-know-86ede6c684b1. Acesso em: 21 dez. 2020.

PONTI, M. A.; COSTA, G. B. P. da. **Como funciona o deep learning**. In: Tópicos em gerenciamento de dados e informações 2017. Disponível em: https://sbbd.org.br/2017/wp-content/uploads/sites/3/2017/10/topicos-em-gerenciamento-de-dados-e-informacoes-2017.pdf. Acesso em: 05 jun. 2021.

QUEIROZ, J. **O que significa RGB**. Disponível em: http://www.afixgraf.com.br/o-que-significa-rgb/. Acesso em: 21 dez. 2020.

ROSENBLATT, F. **The perceptron**: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386–408, 1958. Disponível em: https://psycnet.apa.org/record/1959-09865-001. Acesso em: 17 dez. 2020. https://doi.org/10.1037/h0042519.

RUSSEL, S; NORVIG, P. **Inteligência Artificial**. 3ª edição. Rio de Janeiro: Elsevier, 2013.

SACHAN, A. **Freeze Tensorflow models and serve on web**. Disponível em https://cv-tricks.com/how-to/freeze-tensorflow-models/. Acesso em: 19 mai. 2021.

SACRAMENTO, V. **O que é e como funciona o kernel**; o núcleo do seu computador. 2014. Disponível em: https://www.techtudo.com.br/noticias/noticia/2014/02/o-que-e-e-como-funciona-o-kernel-o-nucleo-do-seu-computador.html. Acesso em: 24 dez. 2020.

SANTOS, T. G. **Google Colab**: o que é e como usar?. Disponível em: https://www.alura.com.br/artigos/google-colab-o-que-e-e-como-usar. Acesso em: 22 dez. 2020.

SILVER, D.; SCHRITTWIESER, J.; SIMONYAN, K. ET AL. **Mastering the game of Go without human knowledge**. Nature 550, 354–359, 2017. Disponível em: https://www.nature.com/articles/nature24270. Acesso em: 17 dez. 2020. https://doi.org/10.1038/nature24270.

SINGHAL, M. **Object Detection using SSD Mobilenet and Tensorflow Object Detection API**: Can detect any single class from coco dataset. Disponível em: https://medium.com/@techmayank2000/object-detection-using-ssd-mobilenetv2-using-tensorflow-api-can-detect-any-single-class-from-31a31bbd069. Acesso em: 18 dez. 2020.

SOLYMÁR, Z. STUBENDEK, A. RADVÁNYI, M. Karacs, K. **Banknote recognition for visually impaired**. 2011 20th European Conference on Circuit Theory and Design (ECCTD), Linkoping, 2011, pp. 841-844, doi: 10.1109/ECCTD.2011.6043828. Disponível em: https://ieeexplore.ieee.org/document/6043828. Acesso em: 25 mar. 2020

SRIVASTAVA, N. et al. **Dropout**: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 2014. Disponível em: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf. Acesso em: 05 jun. 2021.

SURIYA S. SHUSHMAN C. KUMAR V. C.V. Jawahar. **Currency Recognition on Mobile Phones**. IIT Kharagpur, India, CVIT, IIIT Hyderabad, India, 2014. Disponível em: https://web.stanford.edu/~shushman/Suriya2014Currency.pdf. Acesso em: 25 mar. 2020

SZEGEDY, C. et al. **Going Deeper with Convolutions**. ILSVRC, 2014. Disponível em: https://arxiv.org/abs/1409.4842v1. Acesso em: 18 dez. 2020.

TAIGMAN, Y.; YANG, M.; RANZATO, M.; WOLF, L. **DeepFace: Closing the Gap to Human-Level Performance in Face Verification**. Conference on Computer Vision and Pattern Recognition (CVPR). Disponível em: https://research.fb.com/publications/deepface-closing-the-gap-to-human-level-performance-in-face-verification/. Acesso em: 17 dez. 2020.

TENSORBOARD. **TensorFlow's visualization toolkit**. Disponível em: https://www.tensorflow.org/tensorboard. Acesso em: 15 jan. 2020.

TENSORFLOW. **Plataforma de aprendizado de máquina**. Disponível em: https://www.tensorflow.org/. Acesso em: 15 jan. 2020.

_____. **Por que usar o TensorFlow**. Disponível em: https://www.tensorflow.org/?hl=pt. Acesso em: 19 dez. 2020.

TENSORFLOW *EXAMPLES*. **TensorFlow Lite Object Detection Android Demo**. Disponível em: https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android. Acesso em: 25 jan. 2020.

TENSORFLOW LITE. **Plataforma de aprendizado de máquina voltada para Dispositivos Móveis**. Disponível em: https://www.tensorflow.org/lite. Acesso em: 15 jan. 2020.

_____. **Guia do TensorFlow Lite**. Disponível em: https://www.tensorflow.org/lite/guide. Acesso em: 21 dez. 2020.

TENSORFLOW MODELS. **Coleção de modelos de exemplos da API TensorFlow**. Disponível em: https://github.com/tensorflow/models.git. Acesso em: 15 jan. 2020.

TOYTMAN, I. THAMBIDURAI, J. **Banknote recognition on Android platform**. Department of Electrical Engineering – Stanford University, 2011. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.208.3502. Acesso em: 25 mar. 2020.

TOURE, N. **Setup TensorFlow for Object Detection on Ubuntu 16.04**, 2019. Disponível em: https://medium.com/@teyou21/setup-tensorflow-for-object-detection-on-ubuntu-16-04-e2485b52e32a. Acesso em: 20 jan. 2020.

_____. **Object-Detection - TensorFlow lite**. Disponível em: https://github.com/Tourenathan-G5organisation/Object-Detection. Acesso em: 25 jan. 2020.

TOKARNIA, M. **Celular é o principal meio de acesso à internet no país**. Disponível em: https://agenciabrasil.ebc.com.br/economia/noticia/2020-04/celular-e-o-principal-meio-de-acesso-internet-no-pais#:~:text=Os%20dados%20mostram%20que%2079,88%2C5%25%20em%202018. Acesso em: 08 ago. 2020.

TRELLO. **Trello**. Ferramenta de gerenciamento de projetos. Disponível em: https://trello.com/home. Acesso em: 22 dez. 2020.

TSITOARA, M. **Beginning Git and GitHub**: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer. Apress: Nova Iorque, 2020.

TZUTALIN, D. **LabelImg - graphical image annotation tool**, 2015. Disponível em: https://github.com/tzutalin/labelImg. Acesso em: 20 jan. 2020.

ULIAN, I. A. **Detecção e reconhecimento de números seriais em cédulas da segunda família do real**. 2016. 74 f. Dissertação (Mestrado em Informática Forense e Segurança da Informação) – Setor de Engenharia Elétrica, Universidade de Brasília, Brasília (DF), 2016. Disponível em: https://repositorio.unb.br/bitstream/10482/23217/1/2016_IanAlvesUlian.pdf. Acesso em: 25 ago. 2020.

VALOR CONSULTING. **Segunda Família do Real: Dimensões das notas** (Área: 14. Cédulas e moedas). Disponível em: https://www.valor.srv.br/pergResps/pergRespsIndex.php?idPergResp=7107. - Acesso em: 16 dez. 2020.

VERISSIMO, M. **Imagens De Dinheiro**. 100 reais cédula. Disponível em: https://br.pinterest.com/pin/360147301422599246/. Acesso em: 22 dez. 2020.

W3C XML. **XML Tutorial**. World Wide Web Consortium. Disponível em: https://www.w3schools.com/xml/default.asp. Acesso em: 21 dez. 2020.

WANG, Z. J. et al. **CNN Explainer** - Learning Convolutional Neural Networks with Interactive Visualization. Disponível em: https://poloclub.github.io/cnn-explainer/. Acesso em: 15 mai. 2021.

WENG, J. J.; AHUJA, N.; HUANG, T. S. **Learning Recognition and Segmentation Using the Cresceptron**. International Journal of Computer Vision, 1997. Disponível em: https://dl.acm.org/doi/10.1023/A:1007967800668. Acesso em: 17 dez. 2020. https://doi.org/10.1023/A:1007967800668.

YOSINSKI, J. et al. **How transferable are features in deep neural networks?**. In Advances in Neural Information Processing Systems 27. 2014. Disponível em: https://arxiv.org/abs/1411.1792. Acesso em: 06 jun. 2021.

APÊNDICE A - LISTA DE REQUISITOS

Requisitos Funcionais

- A aplicação deve capturar imagens da câmera do celular
- Exibir as imagens na tela da aplicação durante a execução
- Detectar cédulas de reais em imagens
- Classificar as cédulas em classes de real: R$ 2,00, R$ 5,00, R$ 10,00, R$ 20,00, R$ 50,00 e R$ 100,00
- Detectar as cédulas de frente e verso, bem como em diversos ângulos
- Detectar as cédulas mesmo que um pouco dobradas, ou manipuladas pelo usuário
- Vocalizar o resultado
- Exibir o resultado na tela da aplicação

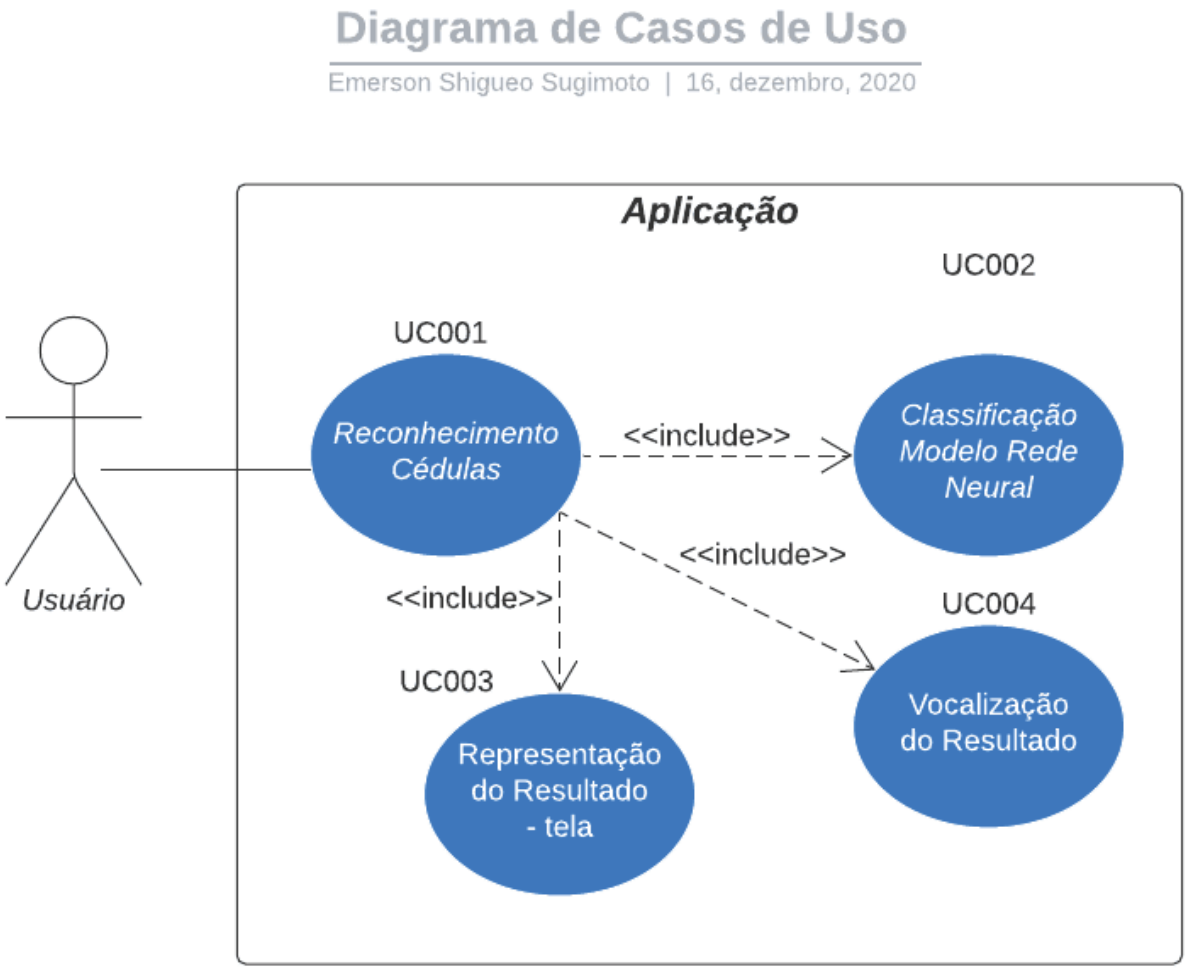
Requisitos Não-Funcionais

- Facilidade de uso
- Reconhecimento em tempo real
- Baixo número de falsos positivos
- Baixo número de falhas de execução
- Executar em um celular com sistema operacional Android

APÊNDICE B – DIAGRAMA DE CASOS DE USO

O diagrama de casos de uso está representado na FIGURA 85. Através da aplicação o usuário realiza o reconhecimento de cédulas através da classificação pela rede neural, obtendo resultado através da identificação da classe na tela do aplicativo e vocalização do resultado.

FIGURA 85 - DIAGRAMA DE CASOS DE USO



FONTE: O Autor (2020)

O fluxo básico do sistema é:

1. O usuário (ator) utiliza a câmera do celular para capturar imagens das cédulas de real;
2. O sistema utiliza o caso de uso UC001 - Reconhecimento de cédulas para iniciar o reconhecimento das cédulas;

3. O sistema utiliza o caso de uso UC002 - Classificação Modelo Rede Neural para identificar as classes;
4. O sistema ordena de forma decrescente as classes de acordo com a sua probabilidade;
5. O sistema utiliza o caso de uso UC003 - Representação do resultado na tela para representar os resultados na tela;
6. O usuário (ator) toca na tela do celular;
7. O sistema seleciona a classe de maior probabilidade;
8. O sistema utiliza o caso de uso UC004 - Vocalização do resultado para vocalizar a classe através da reprodução de arquivo .mp3 pré-gravado com o valor da classe identificada.

As especificações do caso de uso (FIGURA 85) estão descritas nos quadros: QUADRO 7, QUADRO 8, QUADRO 9 e QUADRO 10.

QUADRO 7 - RECONHECIMENTO DE CÉDULAS

Nome do caso de uso	Reconhecimento de cédulas
Identificador	UC001
Caso de Uso Geral	
Ator Principal	Usuário
Resumo	Este caso de uso descreve os passos necessários para a detecção das cédulas
Pré-condições	Imagens capturadas pela câmera do celular
Pós-condições	Classe da cédula identificada Representação da classe na tela do celular Vocalização do celular
Ações do Ator	Ações do Sistema
1. Utiliza a câmera do celular para capturar imagens das cédulas	
	2. Usa o caso de uso UC002 - Classificação Modelo Rede Neural para identificar as classes
	3. Ordena as classes de forma decrescente de acordo com a sua probabilidade
	4. Usa o caso de uso UC003 - Representação do resultado na tela
5. Toca na tela para vocalizar o resultado	
	6. Seleciona a classe com a maior probabilidade
	7. Usa o caso de uso UC004 - Vocalização do resultado
Restrições / Validações	1. Imagens capturadas pela câmera 2. Resultados da rede neural diferentes de nulo

FONTE: O Autor (2020)

QUADRO 8 - CLASSIFICAÇÃO MODELO REDE NEURAL

Nome do caso de uso	Classificação Modelo Rede Neural
Identificador	UC002
Caso de Uso Geral	UC001
Ator Principal	Sistema
Resumo	Este caso de uso descreve os passos necessários para a classificação das imagens pelo modelo de rede neural
Pré-condições	Imagens capturadas pela câmera do celular
Pós-condições	Classe da cédula identificada
Ações do Ator	Ações do Sistema
1. Disponibiliza as imagens capturadas pela câmera para o classificador de rede neural	
	2. Realiza a classificação das cédulas utilizando o Modelo Rede Neural
	3. Disponibiliza as classes identificadas em conjunto com suas probabilidades para o ator
4. Recebe as classes identificadas	
Restrições / Validações	1. Imagens capturadas pela câmera 2. Resultados da rede neural diferentes de nulo

FONTE: O Autor (2020)

QUADRO 9 - REPRESENTAÇÃO DO RESULTADO NA TELA

Nome do caso de uso	Representação do resultado na tela
Identificador	UC003
Caso de Uso Geral	UC001
Ator Principal	Sistema
Resumo	Este caso de uso descreve os passos necessários para a representação das classes na tela do celular
Pré-condições	Classes da imagem diferentes de nulo
Pós-condições	Representação na tela das classes
Ações do Ator	Ações do Sistema
1. Disponibiliza as classes identificadas para o sistema	
	2. Exibe as classes na tela do celular
Restrições / Validações	1. Classes da imagem diferentes de nulo

FONTE: O Autor (2020)

QUADRO 10 - VOCALIZAÇÃO DO RESULTADO

Nome do caso de uso	Vocalização do resultado
Identificador	UC004
Caso de Uso Geral	UC001
Ator Principal	Usuário
Resumo	Este caso de uso descreve os passos necessários para a vocalização das classes
Pré-condições	Classe identificada da imagem diferente de nulo
Pós-condições	Vocalização da classe
Ações do Ator	Ações do Sistema
1. Toca na tela do celular	
	2. Disponibiliza a classe identificada para o sistema
	3. Realiza a vocalização da classe através da reprodução de arquivo .mp3 pré-gravado com o valor da classe identificada
Restrições / Validações	1. Classe identificada da imagem diferente de nulo

FONTE: O Autor (2020)

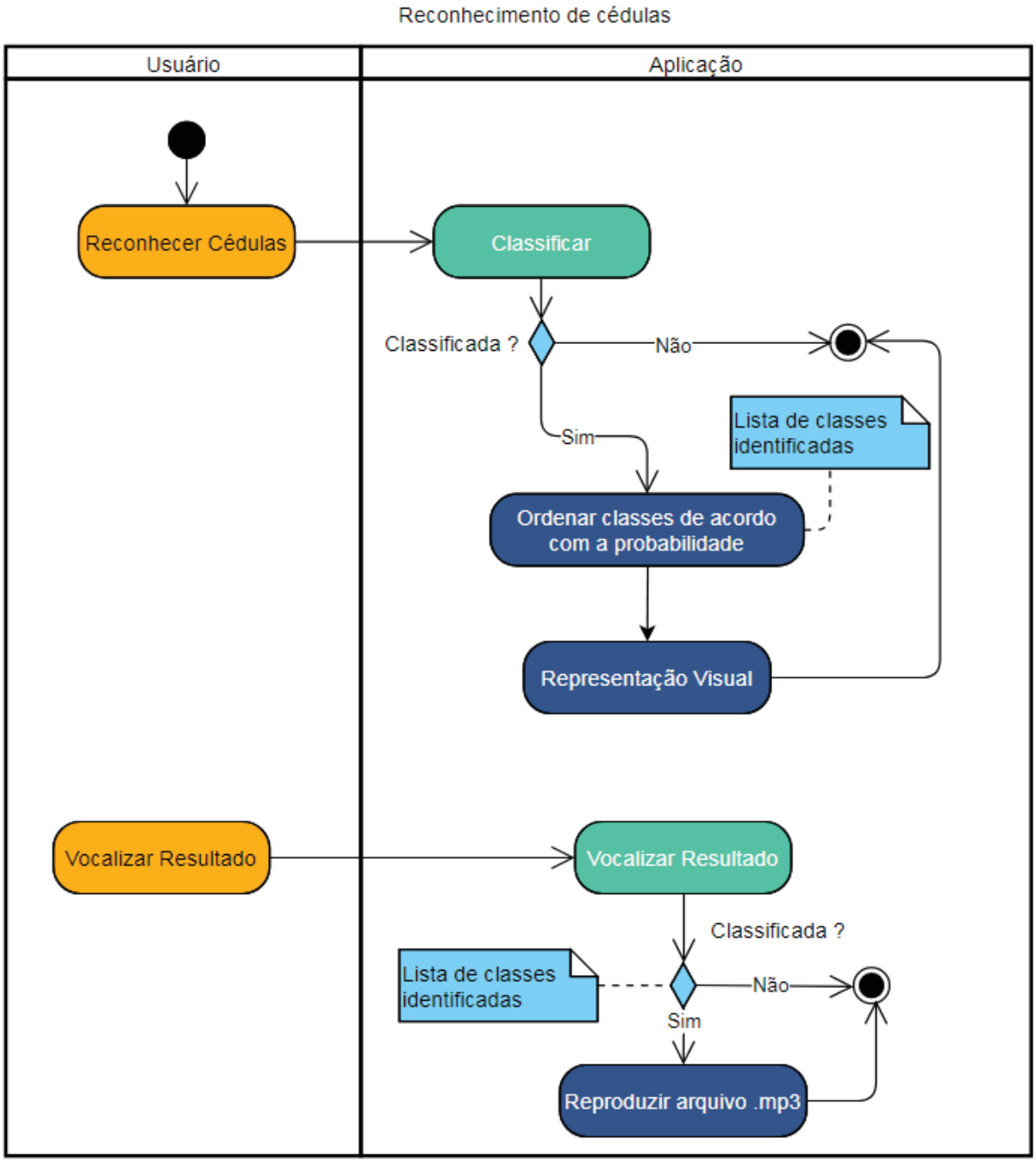
APÊNDICE C – DIAGRAMA DE ATIVIDADES

O diagrama de atividades está representado na FIGURA 86. Ao receber uma imagem da câmera do celular, o sistema utiliza o modelo da rede neural para realizar a classificação da cédula de real. Caso não encontre uma detecção, o algoritmo para.

O modelo de rede neural retorna um conjunto de classes e suas respectivas probabilidade. O sistema ordena a lista de forma decrescente de acordo com suas probabilidades. A representação visual é feita para as três classes mais promissoras.

Quando o usuário toca na tela do celular, o sistema realiza a classificação da imagem por meio da rede neural, recupera a classe mais promissora e reproduz o respectivo arquivo .mp3 com a classe de interesse.

FIGURA 86 - DIAGRAMA DE ATIVIDADES



FONTE: O Autor (2020)

APÊNDICE D – MAPEAMENTO *LABELS*

O mapeamento da saída da rede neural com as etiquetas (*labels*) treinadas na rede está representado no QUADRO 11. O arquivo de mapeamento é posicional de forma que a linha representa a posição da classe de saída da rede neural. A primeira linha é considerada a posição de saída zero. Por exemplo, a classe “DoisReaisFrente”, linha 1, representa a posição 0 (linha 1, posição subtraída de 1, resultando na posição 0); a classe “CinquentaReaisFrente”, linha 9, representa a posição 8 (linha 9, posição subtraída de 1, resultando na posição 8); e assim sucessivamente.

A saída da rede neural pode ser entendida como valores que variam de 0 a 11, mapeados para as classes através do arquivo de mapeamento “labelmap1.txt” (QUADRO 11).

QUADRO 11 - ARQUIVO LABELMAP1.TXT

DoisReaisFrente
DoisReaisVerso
CincoReaisFrente
CincoReaisVerso
DezReaisFrente
DezReaisVerso
VinteReaisFrente
VinteReaisVerso
CinquentaReaisFrente
CinquentaReaisVerso
CemReaisFrente
CemReaisVerso

FONTE: O Autor (2020)

APÊNDICE E – DIAGRAMA DE CLASSES

O diagrama de classes está representado nas imagens: FIGURA 87, FIGURA 88, FIGURA 89, FIGURA 90, FIGURA 91 e FIGURA 92.

As principais classes são a MainActivity.java (FIGURA 93) e DetectorActivity.java (FIGURA 94). O código adaptado está representado no QUADRO 12. A constante "`TF_OD_API_MODEL_FILE`" recebe o caminho do arquivo do modelo treinado da rede neural no formato tflite (Seção 2.16), no caso "`detectx.tflite`". A constante "`TF_OD_API_LABELS_FILE`" recebe o caminho do arquivo com as etiquetas (*labels*) das classes treinadas na rede neural, no caso "`labelmap1.txt`". O arquivo de mapeamento das classes está descrito no APÊNDICE D.

QUADRO 12 – CÓDIGO ADAPTADO

```
private static final String TF_OD_API_MODEL_FILE = "detectx.tflite";
private static final String TF_OD_API_LABELS_FILE =
"file:///android_asset/labelmap1.txt";
```

FONTE: O Autor (2020)

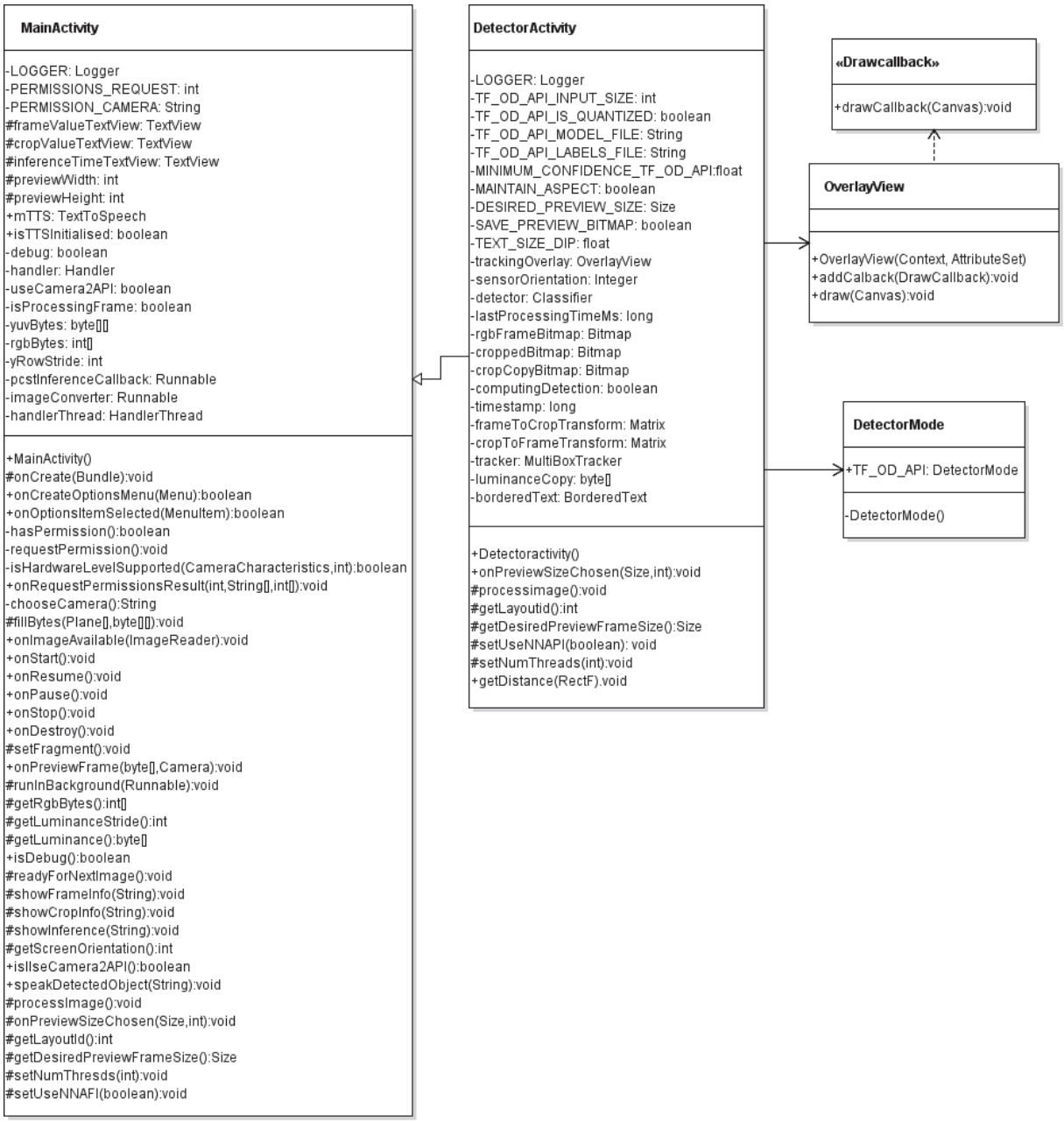
O principal método é protected void processImage(). As detecções do modelo são recuperadas através do código descrito no QUADRO 13.

QUADRO 13 - DETECÇÃO MODELO

```
final List<Classifier.Recognition> results =
detector.recognizeImage(croppedBitmap);
```

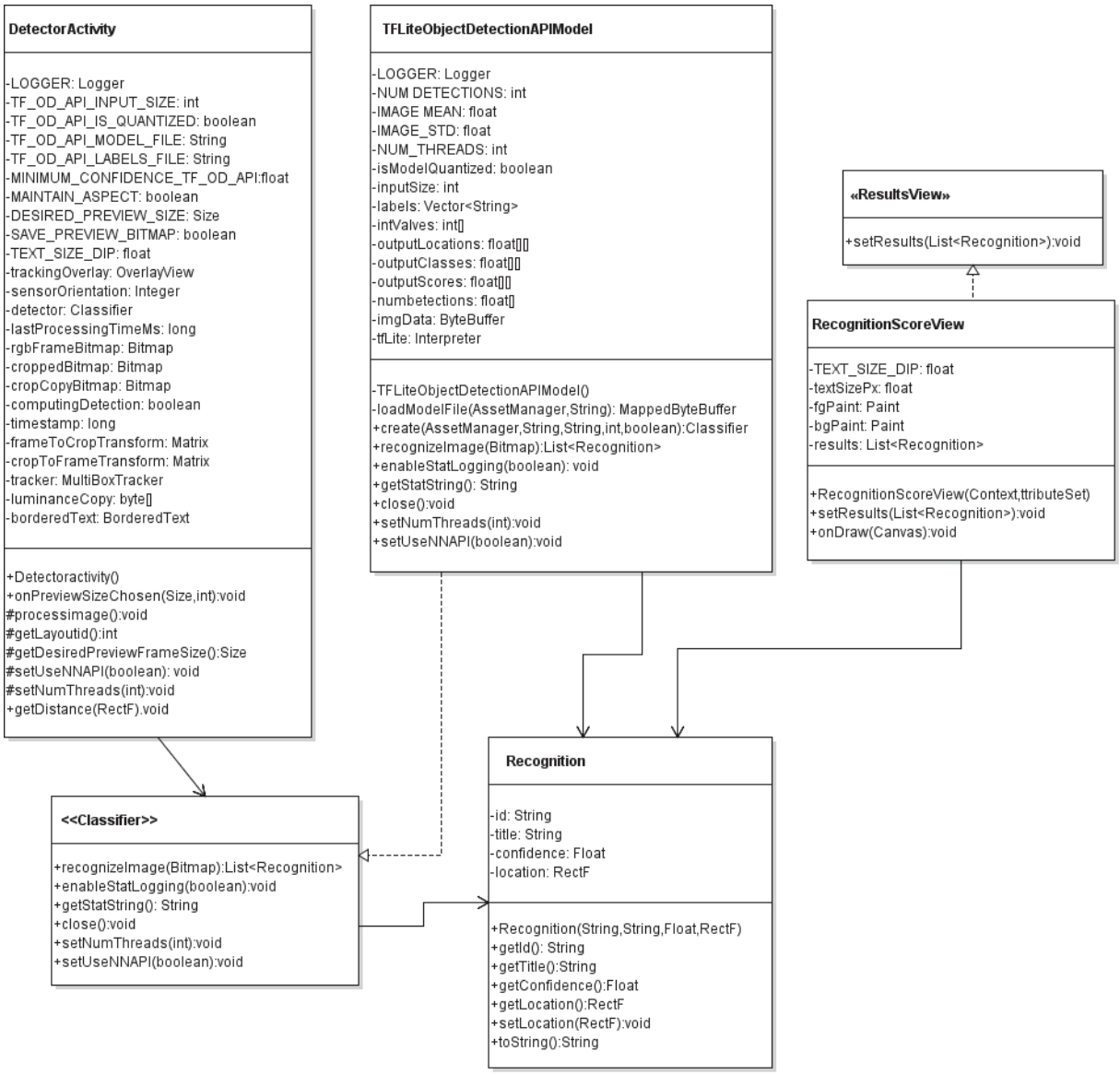
FONTE: O Autor (2020)

FIGURA 87 – DIAGRAMA DE CLASSES



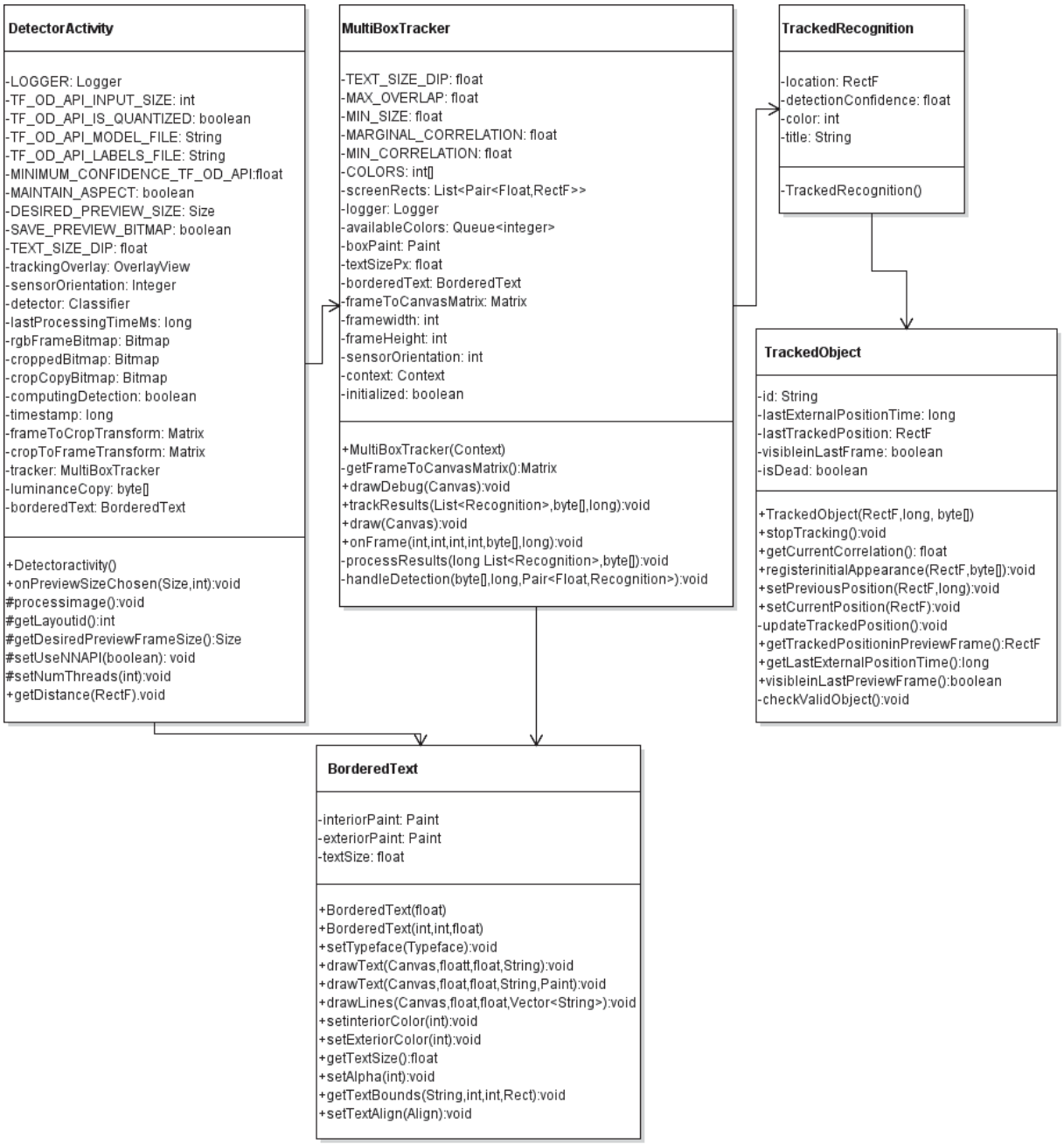
FONTE: Adaptado de TensorFlow *Examples* (2020)

FIGURA 88 – DIAGRAMA DE CLASSES



FONTE: Adaptado de TensorFlow *Examples* (2020)

FIGURA 89 – DIAGRAMA DE CLASSES

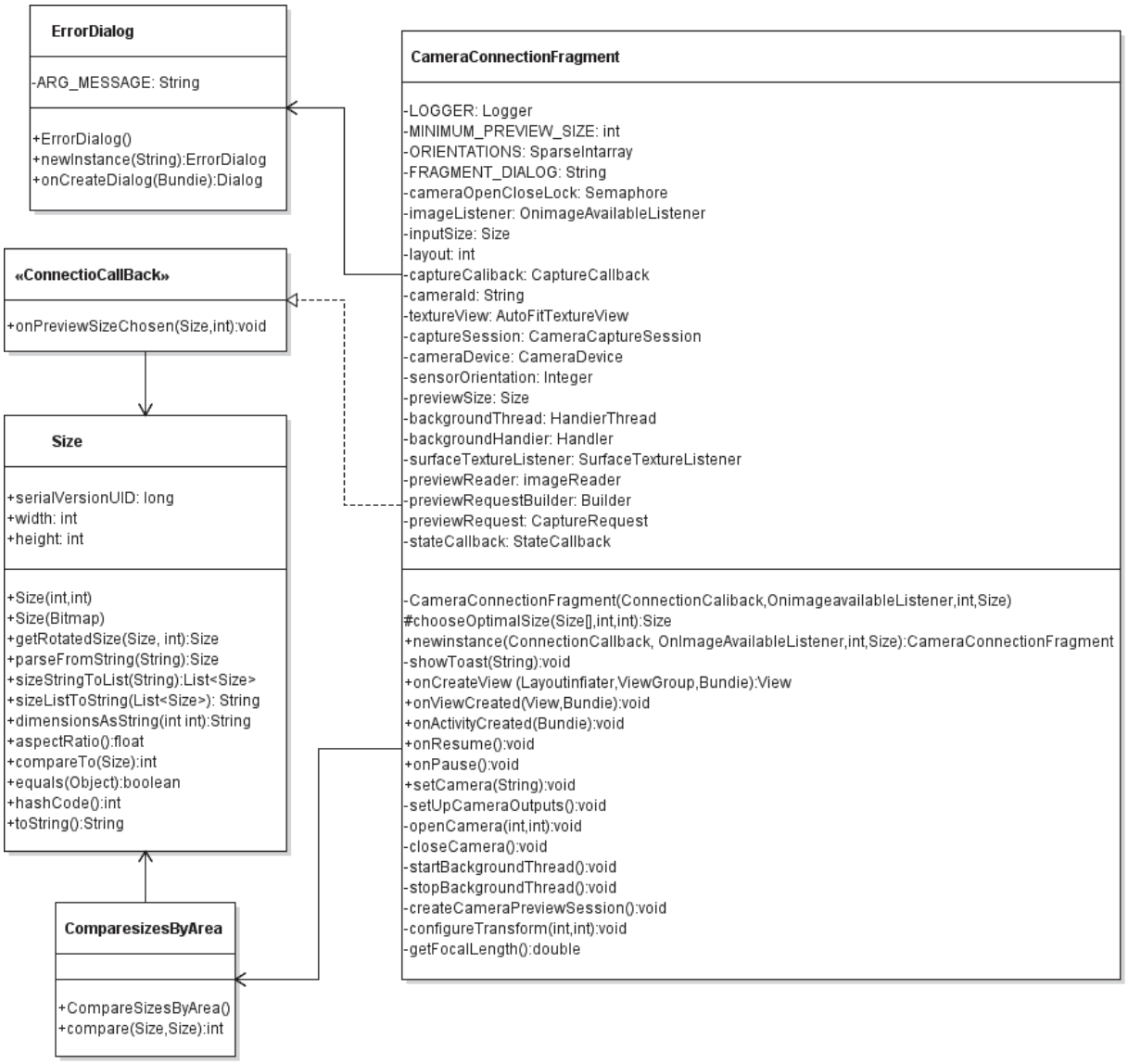


FONTE: Adaptado de TensorFlow *Examples* (2020)

FIGURA 90 – DIAGRAMA DE CLASSES

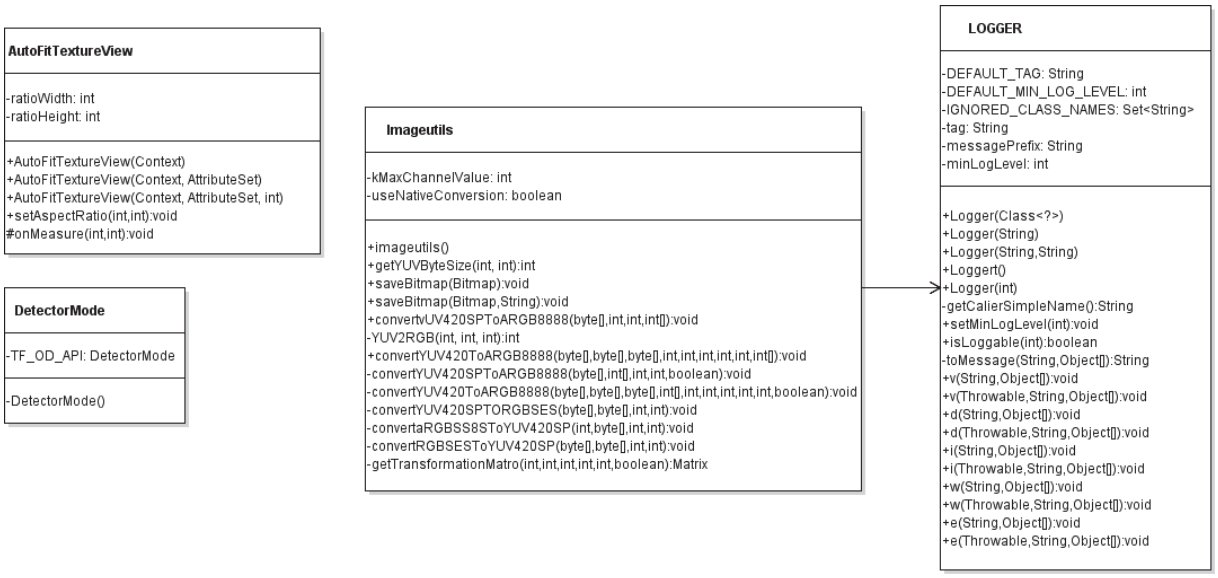
FONTE: Adaptado de TensorFlow *Examples* (2020)

FIGURA 91 – DIAGRAMA DE CLASSES



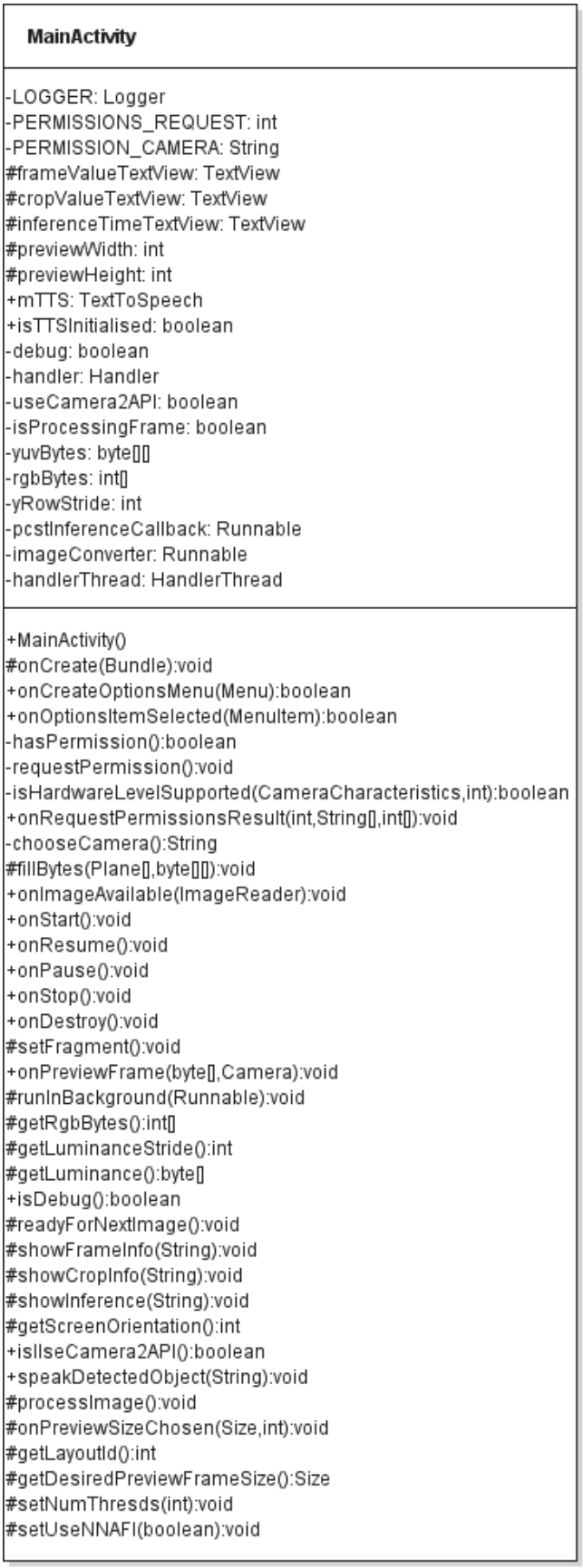
FONTE: Adaptado de TensorFlow *Examples* (2020)

FIGURA 92 – DIAGRAMA DE CLASSES



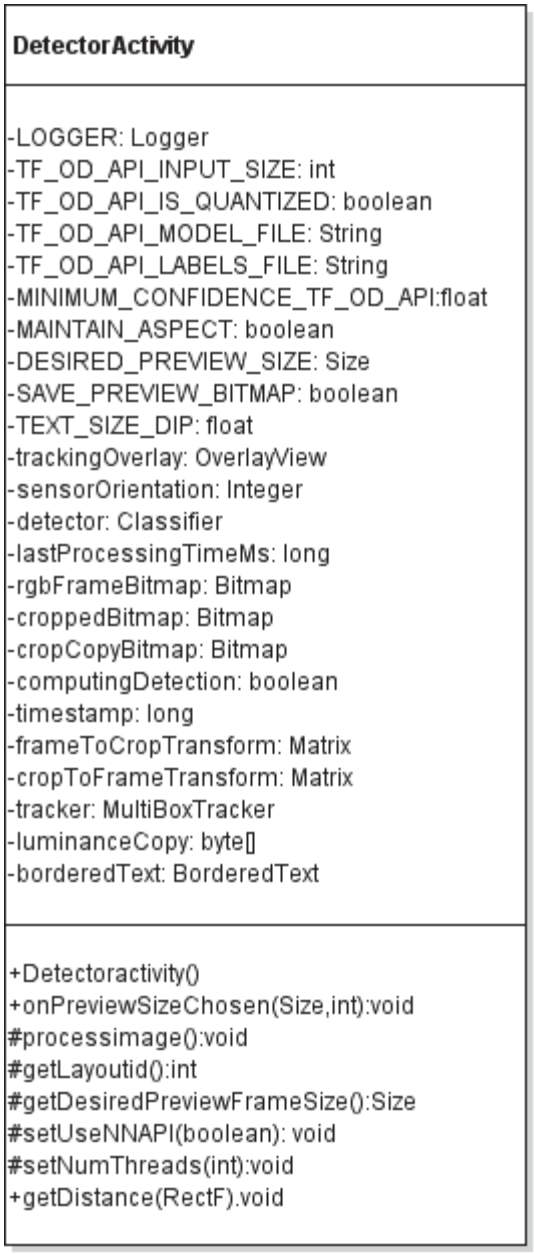
FONTE: Adaptado de TensorFlow *Examples* (2020)

FIGURA 93 – CLASSE MAINACTIVITY



FONTE: Adaptado de TensorFlow *Examples* (2020)

FIGURA 94 – CLASSE DETECTORACTIVITY



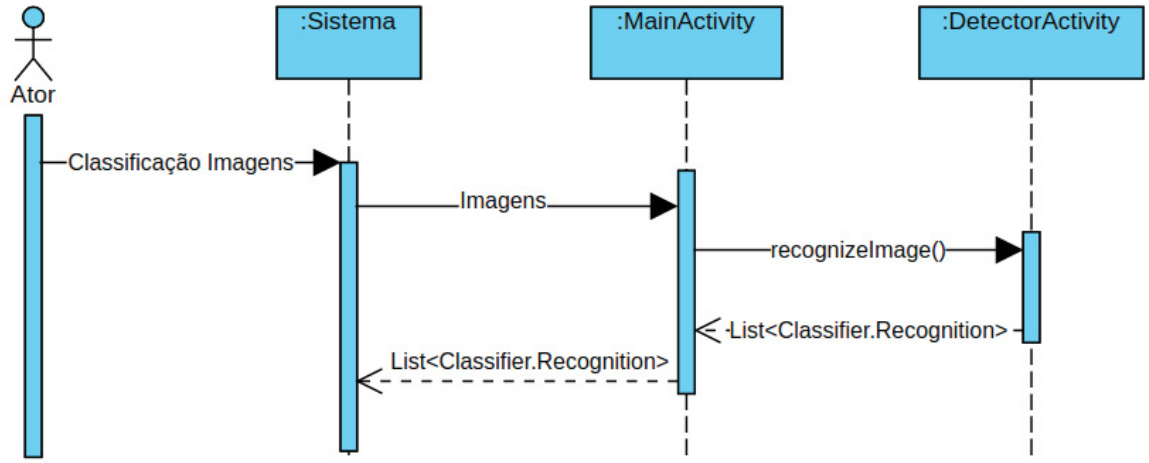
FONTE: Adaptado de TensorFlow *Examples* (2020)

APÊNDICE F - DIAGRAMAS DE SEQUÊNCIA

O Diagrama de sequência da classificação das imagens está representado na FIGURA 95. O fluxo básico do diagrama de sequência é:

1. O usuário (ator) utiliza a câmera do celular para capturar imagens das cédulas de real;
2. O sistema fornece as imagens para o modelo de rede neural (classe DetectorActivity, método recognizeImage());
3. O Modelo Rede Neural identifica as classes e retorna o conjunto de classes e probabilidade identificadas.

FIGURA 95 - DIAGRAMA DE SEQUÊNCIA CLASSIFICAÇÃO

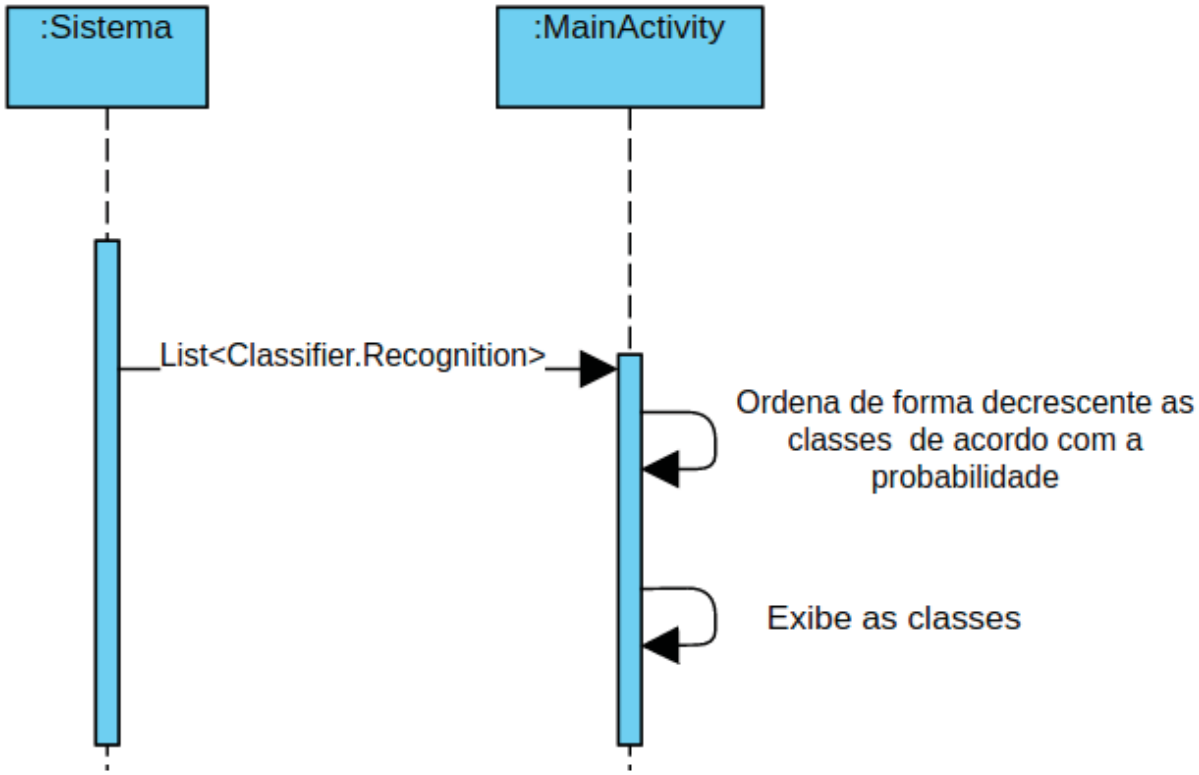


FONTE: O Autor (2021)

O Diagrama de sequência de representação das classes está representado na FIGURA 96. O fluxo básico é:

1. O sistema ordena de forma decrescente as classes de acordo com a sua probabilidade;
2. O sistema fornece as classes para representação do resultado visual das classes;
3. O sistema exibe as classes visualmente na tela do celular.

FIGURA 96 - DIAGRAMA DE SEQUÊNCIA REPRESENTAÇÃO CLASSES

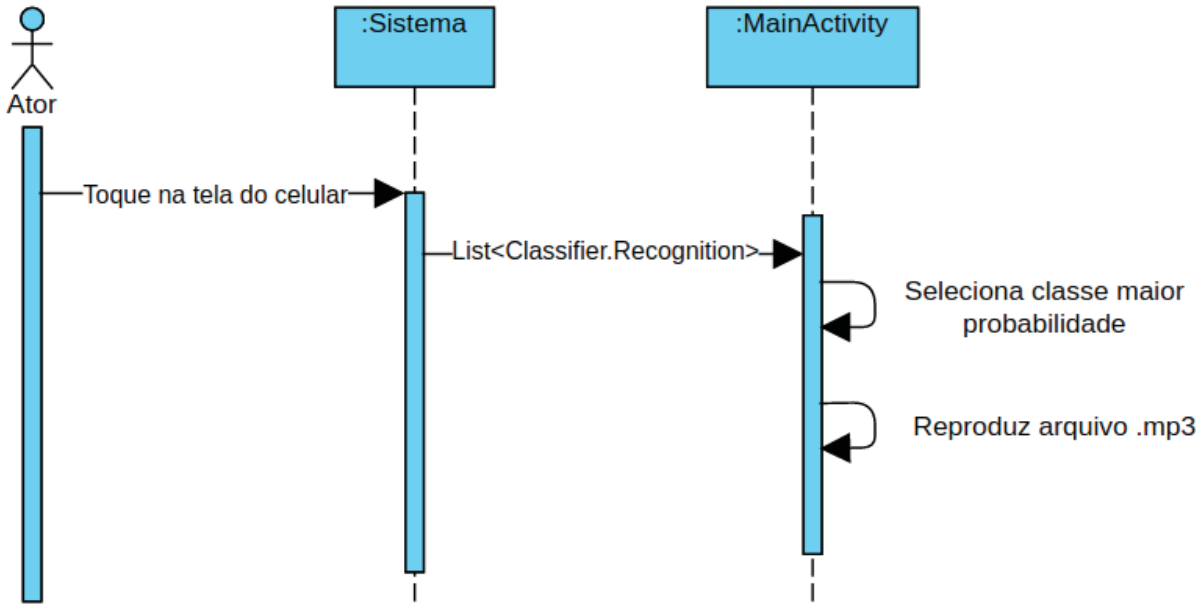


FONTE: O Autor (2021)

O Diagrama de sequência de vocalização está representado na FIGURA 97. O fluxo básico é:

1. O usuário (ator) toca na tela do celular;
2. O sistema seleciona a classe de maior probabilidade;
3. O sistema realiza a vocalização da classe através da reprodução de arquivo .mp3 pré-gravado com o valor da classe identificada.

FIGURA 97 - DIAGRAMA DE SEQUÊNCIA VOCALIZAÇÃO



FONTE: O Autor (2021)

APÊNDICE G – ESTRUTURA REDE MOBILENET

A estrutura da rede SSD MobileNet está representada na TABELA 5.

TABELA 5 – ESTRUTURA REDE MOBILENET

continua

Camada (tipo)	Formato saída	Param #	Conectado a
input_1 (InputLayer)	[(None, 224, 224, 3)	0	
Conv1 (Conv2D)	(None, 112, 112, 32)	864	input_1[0][0]
bn_Conv1 (BatchNormalization)	(None, 112, 112, 32)	128	Conv1[0][0]
Conv1_relu (ReLU)	(None, 112, 112, 32)	0	bn_Conv1[0][0]
expanded_conv_depthwise (Depthw	(None, 112, 112, 32)	288	Conv1_relu[0][0]
expanded_conv_depthwise_BN (Bat	(None, 112, 112, 32)	128	expanded_conv_depthwise[0][0]
expanded_conv_depthwise_relu (R	(None, 112, 112, 32)	0	expanded_conv_depthwise_BN[0][0]
expanded_conv_project (Conv2D)	(None, 112, 112, 16)	512	expanded_conv_depthwise_relu[0][0
expanded_conv_project_BN (Batch	(None, 112, 112, 16)	64	expanded_conv_project[0][0]
block_1_expand (Conv2D)	(None, 112, 112, 96)	1536	expanded_conv_project_BN[0][0]
block_1_expand_BN (BatchNormali	(None, 112, 112, 96)	384	block_1_expand[0][0]
block_1_expand_relu (ReLU)	(None, 112, 112, 96)	0	block_1_expand_BN[0][0]
block_1_pad (ZeroPadding2D)	(None, 113, 113, 96)	0	block_1_expand_relu[0][0]
block_1_depthwise (DepthwiseCon	(None, 56, 56, 96)	864	block_1_pad[0][0]
block_1_depthwise_BN (BatchNorm	(None, 56, 56, 96)	384	block_1_depthwise[0][0]
block_1_depthwise_relu (ReLU)	(None, 56, 56, 96)	0	block_1_depthwise_BN[0][0]
block_1_project (Conv2D)	(None, 56, 56, 24)	2304	block_1_depthwise_relu[0][0]
block_1_project_BN (BatchNormal	(None, 56, 56, 24)	96	block_1_project[0][0]
block_2_expand (Conv2D)	(None, 56, 56, 144)	3456	block_1_project_BN[0][0]
block_2_expand_BN (BatchNormali	(None, 56, 56, 144)	576	block_2_expand[0][0]
block_2_expand_relu (ReLU)	(None, 56, 56, 144)	0	block_2_expand_BN[0][0]
block_2_depthwise (DepthwiseCon	(None, 56, 56, 144)	1296	block_2_expand_relu[0][0]
block_2_depthwise_BN (BatchNorm	(None, 56, 56, 144)	576	block_2_depthwise[0][0]
block_2_depthwise_relu (ReLU)	(None, 56, 56, 144)	0	block_2_depthwise_BN[0][0]
block_2_project (Conv2D)	(None, 56, 56, 24)	3456	block_2_depthwise_relu[0][0]

TABELA 5 – ESTRUTURA REDE MOBILENET

continuação

Camada (tipo)	Formato saída	Param #	Conectado a
block_2_project_BN (BatchNormal	(None, 56, 56, 24)	96	block_2_project[0][0]
block_2_add (Add)	(None, 56, 56, 24)	0	block_1_project_BN[0][0] block_2_project_BN[0][0]
block_3_expand (Conv2D)	(None, 56, 56, 144)	3456	block_2_add[0][0]
block_3_expand_BN (BatchNormali	(None, 56, 56, 144)	576	block_3_expand[0][0]
block_3_expand_relu (ReLU)	(None, 56, 56, 144)	0	block_3_expand_BN[0][0]
block_3_pad (ZeroPadding2D)	(None, 57, 57, 144)	0	block_3_expand_relu[0][0]
block_3_depthwise (DepthwiseCon	(None, 28, 28, 144)	1296	block_3_pad[0][0]
block_3_depthwise_BN (BatchNorm	(None, 28, 28, 144)	576	block_3_depthwise[0][0]
block_3_depthwise_relu (ReLU)	(None, 28, 28, 144)	0	block_3_depthwise_BN[0][0]
block_3_project (Conv2D)	(None, 28, 28, 32)	4608	block_3_depthwise_relu[0][0]
block_3_project_BN (BatchNormal	(None, 28, 28, 32)	128	block_3_project[0][0]
block_4_expand (Conv2D)	(None, 28, 28, 192)	6144	block_3_project_BN[0][0]
block_4_expand_BN (BatchNormali	(None, 28, 28, 192)	768	block_4_expand[0][0]
block_4_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_4_expand_BN[0][0]
block_4_depthwise (DepthwiseCon	(None, 28, 28, 192)	1728	block_4_expand_relu[0][0]
block_4_depthwise_BN (BatchNorm	(None, 28, 28, 192)	768	block_4_depthwise[0][0]
block_4_depthwise_relu (ReLU)	(None, 28, 28, 192)	0	block_4_depthwise_BN[0][0]
block_4_project (Conv2D)	(None, 28, 28, 32)	6144	block_4_depthwise_relu[0][0]
block_4_project_BN (BatchNormal	(None, 28, 28, 32)	128	block_4_project[0][0]
block_4_add (Add)	(None, 28, 28, 32)	0	block_3_project_BN[0][0] block_4_project_BN[0][0]
block_5_expand (Conv2D)	(None, 28, 28, 192)	6144	block_4_add[0][0]
block_5_expand_BN (BatchNormali	(None, 28, 28, 192)	768	block_5_expand[0][0]
block_5_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_5_expand_BN[0][0]
block_5_depthwise (DepthwiseCon	(None, 28, 28, 192)	1728	block_5_expand_relu[0][0]
block_5_depthwise_BN (BatchNorm	(None, 28, 28, 192)	768	block_5_depthwise[0][0]
block_5_depthwise_relu (ReLU)	(None, 28, 28, 192)	0	block_5_depthwise_BN[0][0]
block_5_project (Conv2D)	(None, 28, 28, 32)	6144	block_5_depthwise_relu[0][0]
block_5_project_BN (BatchNormal	(None, 28, 28, 32)	128	block_5_project[0][0]

TABELA 5 – ESTRUTURA REDE MOBILENET

continuação

Camada (tipo)	Formato saída	Param #	Conectado a
block_5_add (Add)	(None, 28, 28, 32)	0	block_4_add[0][0] block_5_project_BN[0][0]
block_6_expand (Conv2D)	(None, 28, 28, 192)	6144	block_5_add[0][0]
block_6_expand_BN (BatchNormali	(None, 28, 28, 192)	768	block_6_expand[0][0]
block_6_expand_relu (ReLU)	(None, 28, 28, 192)	0	block_6_expand_BN[0][0]
block_6_pad (ZeroPadding2D)	(None, 29, 29, 192)	0	block_6_expand_relu[0][0]
block_6_depthwise (DepthwiseCon	(None, 14, 14, 192)	1728	block_6_pad[0][0]
block_6_depthwise_BN (BatchNorm	(None, 14, 14, 192)	768	block_6_depthwise[0][0]
block_6_depthwise_relu (ReLU)	(None, 14, 14, 192)	0	block_6_depthwise_BN[0][0]
block_6_project (Conv2D)	(None, 14, 14, 64)	12288	block_6_depthwise_relu[0][0]
block_6_project_BN (BatchNormal	(None, 14, 14, 64)	256	block_6_project[0][0]
block_7_expand (Conv2D)	(None, 14, 14, 384)	24576	block_6_project_BN[0][0]
block_7_expand_BN (BatchNormali	(None, 14, 14, 384)	1536	block_7_expand[0][0]
block_7_expand_relu (ReLU)	(None, 14, 14, 384)	0	block_7_expand_BN[0][0]
block_7_depthwise (DepthwiseCon	(None, 14, 14, 384)	3456	block_7_expand_relu[0][0]
block_7_depthwise_BN (BatchNorm	(None, 14, 14, 384)	1536	block_7_depthwise[0][0]
block_7_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	block_7_depthwise_BN[0][0]
block_7_project (Conv2D)	(None, 14, 14, 64)	24576	block_7_depthwise_relu[0][0]
block_7_project_BN (BatchNormal	(None, 14, 14, 64)	256	block_7_project[0][0]
block_7_add (Add)	(None, 14, 14, 64)	0	block_6_project_BN[0][0] block_7_project_BN[0][0]
block_8_expand (Conv2D)	(None, 14, 14, 384)	24576	block_7_add[0][0]
block_8_expand_BN (BatchNormali	(None, 14, 14, 384)	1536	block_8_expand[0][0]
block_8_expand_relu (ReLU)	(None, 14, 14, 384)	0	block_8_expand_BN[0][0]
block_8_depthwise (DepthwiseCon	(None, 14, 14, 384)	3456	block_8_expand_relu[0][0]
block_8_depthwise_BN (BatchNorm	(None, 14, 14, 384)	1536	block_8_depthwise[0][0]
block_8_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	block_8_depthwise_BN[0][0]
block_8_project (Conv2D)	(None, 14, 14, 64)	24576	block_8_depthwise_relu[0][0]
block_8_project_BN (BatchNormal	(None, 14, 14, 64)	256	block_8_project[0][0]

TABELA 5 – ESTRUTURA REDE MOBILENET

continuação

Camada (tipo)	Formato saída	Param #	Conectado a
block_8_add (Add)	(None, 14, 14, 64)	0	block_7_add[0][0] block_8_project_BN[0][0]
block_9_expand (Conv2D)	(None, 14, 14, 384)	24576	block_8_add[0][0]
block_9_expand_BN (BatchNormali	(None, 14, 14, 384)	1536	block_9_expand[0][0]
block_9_expand_relu (ReLU)	(None, 14, 14, 384)	0	block_9_expand_BN[0][0]
block_9_depthwise (DepthwiseCon	(None, 14, 14, 384)	3456	block_9_expand_relu[0][0]
block_9_depthwise_BN (BatchNorm	(None, 14, 14, 384)	1536	block_9_depthwise[0][0]
block_9_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	block_9_depthwise_BN[0][0]
block_9_project (Conv2D)	(None, 14, 14, 64)	24576	block_9_depthwise_relu[0][0]
block_9_project_BN (BatchNormal	(None, 14, 14, 64)	256	block_9_project[0][0]
block_9_add (Add)	(None, 14, 14, 64)	0	block_8_add[0][0] block_9_project_BN[0][0]
block_10_expand (Conv2D)	(None, 14, 14, 384)	24576	block_9_add[0][0]
block_10_expand_BN (BatchNormal	(None, 14, 14, 384)	1536	block_10_expand[0][0]
block_10_expand_relu (ReLU)	(None, 14, 14, 384)	0	block_10_expand_BN[0][0]
block_10_depthwise (DepthwiseCo	(None, 14, 14, 384)	3456	block_10_expand_relu[0][0]
block_10_depthwise_BN (BatchNor	(None, 14, 14, 384)	1536	block_10_depthwise[0][0]
block_10_depthwise_relu (ReLU)	(None, 14, 14, 384)	0	block_10_depthwise_BN[0][0]
block_10_project (Conv2D)	(None, 14, 14, 96)	36864	block_10_depthwise_relu[0][0]
block_10_project_BN (BatchNorma	(None, 14, 14, 96)	384	block_10_project[0][0]
block_11_expand (Conv2D)	(None, 14, 14, 576)	55296	block_10_project_BN[0][0]
block_11_expand_BN (BatchNormal	(None, 14, 14, 576)	2304	block_11_expand[0][0]
block_11_expand_relu (ReLU)	(None, 14, 14, 576)	0	block_11_expand_BN[0][0]
block_11_depthwise (DepthwiseCo	(None, 14, 14, 576)	5184	block_11_expand_relu[0][0]
block_11_depthwise_BN (BatchNor	(None, 14, 14, 576)	2304	block_11_depthwise[0][0]
block_11_depthwise_relu (ReLU)	(None, 14, 14, 576)	0	block_11_depthwise_BN[0][0]
block_11_project (Conv2D)	(None, 14, 14, 96)	55296	block_11_depthwise_relu[0][0]
block_11_project_BN (BatchNorma	(None, 14, 14, 96)	384	block_11_project[0][0]
block_11_add (Add)	(None, 14, 14, 96)	0	block_10_project_BN[0][0] block_11_project_BN[0][0]
block_12_expand (Conv2D)	(None, 14, 14, 576)	55296	block_11_add[0][0]

TABELA 5 – ESTRUTURA REDE MOBILENET

continuação

Camada (tipo)	Formato saída	Param #	Conectado a
block_12_expand_BN (BatchNormal	(None, 14, 14, 576)	2304	block_12_expand[0][0]
block_12_expand_relu (ReLU)	(None, 14, 14, 576)	0	block_12_expand_BN[0][0]
block_12_depthwise (DepthwiseCo	(None, 14, 14, 576)	5184	block_12_expand_relu[0][0]
block_12_depthwise_BN (BatchNor	(None, 14, 14, 576)	2304	block_12_depthwise[0][0]
block_12_depthwise_relu (ReLU)	(None, 14, 14, 576)	0	block_12_depthwise_BN[0][0]
block_12_project (Conv2D)	(None, 14, 14, 96)	55296	block_12_depthwise_relu[0][0]
block_12_project_BN (BatchNorma	(None, 14, 14, 96)	384	block_12_project[0][0]
block_12_add (Add)	(None, 14, 14, 96)	0	block_11_add[0][0] block_12_project_BN[0][0]
block_13_expand (Conv2D)	(None, 14, 14, 576)	55296	block_12_add[0][0]
block_13_expand_BN (BatchNormal	(None, 14, 14, 576)	2304	block_13_expand[0][0]
block_13_expand_relu (ReLU)	(None, 14, 14, 576)	0	block_13_expand_BN[0][0]
block_13_pad (ZeroPadding2D)	(None, 15, 15, 576)	0	block_13_expand_relu[0][0]
block_13_depthwise (DepthwiseCo	(None, 7, 7, 576)	5184	block_13_pad[0][0]
block_13_depthwise_BN (BatchNor	(None, 7, 7, 576)	2304	block_13_depthwise[0][0]
block_13_depthwise_relu (ReLU)	(None, 7, 7, 576)	0	block_13_depthwise_BN[0][0]
block_13_project (Conv2D)	(None, 7, 7, 160)	92160	block_13_depthwise_relu[0][0]
block_13_project_BN (BatchNorma	(None, 7, 7, 160)	640	block_13_project[0][0]
block_14_expand (Conv2D)	(None, 7, 7, 960)	153600	block_13_project_BN[0][0]
block_14_expand_BN (BatchNormal	(None, 7, 7, 960)	3840	block_14_expand[0][0]
block_14_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_14_expand_BN[0][0]
block_14_depthwise (DepthwiseCo	(None, 7, 7, 960)	8640	block_14_expand_relu[0][0]
block_14_depthwise_BN (BatchNor	(None, 7, 7, 960)	3840	block_14_depthwise[0][0]
block_14_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_14_depthwise_BN[0][0]
block_14_project (Conv2D)	(None, 7, 7, 160)	153600	block_14_depthwise_relu[0][0]
block_14_project_BN (BatchNorma	(None, 7, 7, 160)	640	block_14_project[0][0]
block_14_add (Add)	(None, 7, 7, 160)	0	block_13_project_BN[0][0] block_14_project_BN[0][0]
block_15_expand (Conv2D)	(None, 7, 7, 960)	153600	block_14_add[0][0]
block_15_expand_BN (BatchNormal	(None, 7, 7, 960)	3840	block_15_expand[0][0]

TABELA 5 – ESTRUTURA REDE MOBILENET

conclusão

Camada (tipo)	Formato saída	Param #	Conectado a
block_15_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_15_expand_BN[0][0]
block_15_depthwise (DepthwiseCo	(None, 7, 7, 960)	8640	block_15_expand_relu[0][0]
block_15_depthwise_BN (BatchNor	(None, 7, 7, 960)	3840	block_15_depthwise[0][0]
block_15_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_15_depthwise_BN[0][0]
block_15_project (Conv2D)	(None, 7, 7, 160)	153600	block_15_depthwise_relu[0][0]
block_15_project_BN (BatchNorma	(None, 7, 7, 160)	640	block_15_project[0][0]
block_15_add (Add)	(None, 7, 7, 160)	0	block_14_add[0][0] block_15_project_BN[0][0]
block_16_expand (Conv2D)	(None, 7, 7, 960)	153600	block_15_add[0][0]
block_16_expand_BN (BatchNormal	(None, 7, 7, 960)	3840	block_16_expand[0][0]
block_16_expand_relu (ReLU)	(None, 7, 7, 960)	0	block_16_expand_BN[0][0]
block_16_depthwise (DepthwiseCo	(None, 7, 7, 960)	8640	block_16_expand_relu[0][0]
block_16_depthwise_BN (BatchNor	(None, 7, 7, 960)	3840	block_16_depthwise[0][0]
block_16_depthwise_relu (ReLU)	(None, 7, 7, 960)	0	block_16_depthwise_BN[0][0]
block_16_project (Conv2D)	(None, 7, 7, 320)	307200	block_16_depthwise_relu[0][0]
block_16_project_BN (BatchNorma	(None, 7, 7, 320)	1280	block_16_project[0][0]
Conv_1 (Conv2D)	(None, 7, 7, 1280)	409600	block_16_project_BN[0][0]
Conv_1_bn (BatchNormalization)	(None, 7, 7, 1280)	5120	Conv_1[0][0]
out_relu (ReLU)	(None, 7, 7, 1280)	0	Conv_1_bn[0][0]

FONTE: Adaptado de TensorFlow *Examples* (2020)