RECONHECIMENTO DE PLACAS DE TRÂNSITO EM CICLOVIAS POR MEIO DE REDES NEURAIS

KAIO GIOVANNI PEREIRA DOS SANTOS

TRABALHO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RECONHECIMENTO DE PLACAS DE TRÂNSITO EM CICLOVIAS POR MEIO DE REDES NEURAIS

KAIO GIOVANNI PEREIRA DOS SANTOS

ORIENTADOR: PROF. ADOLFO BAUCHSPIESS

TRABALHO DE GRADUAÇÃO EM
ENGENHARIA ELÉTRICA

BRASÍLIA/DF: JULHO - 2019

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RECONHECIMENTO DE PLACAS DE TRÂNSITO EM CICLOVIAS POR MEIO DE REDES NEURAIS

KAIO GIOVANNI PEREIRA DOS SANTOS

TRABALHO DE GRADUAÇÃO SUBMETIDO AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM ENGENHARIA ELÉTRICA.

APROVADO POR:

_____

Prof. Adolfo Bauchspiess – ENE/Universidade de Brasília
Orientador

_____

Prof. Francisco Assis de Oliveira Nascimento – Dep./Universidade
Membro Interno

_____

Prof. Daniel Chaves Café – Dep./Universidade
Membro Interno

BRASÍLIA, 10 DE JULHO DE 2019.

FICHA CATALOGRÁFICA

SANTOS, KAIO G. P.
Reconhecimento de Placas de Trânsito em Ciclovias por meio de Redes Neurais
[Distrito Federal] 2019.
xii, 48p., 210 x 297 mm (ENE/FT/UnB, Graduando, Engenharia Elétrica, 2019).
Trabalho de Graduação – Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica
1. Redes Neurais 2. Programação
I. ENE/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

(2019). Reconhecimento de Placas de Trânsito em Ciclovias por meio de Redes Neurais , Trabalho de Graduação, Publicação , Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 48p.

CESSÃO DE DIREITOS

AUTOR: Kaio Giovanni Pereira dos Santos
TÍTULO: Reconhecimento de Placas de Trânsito em Ciclovias por meio de Redes Neurais .
GRAU: Graduando ANO: 2019

É concedida à Universidade de Brasília permissão para reproduzir cópias deste trabalho de graduação e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desse trabalho de graduação pode ser reproduzido sem autorização por escrito do autor.

_____

Kaio Giovanni Pereira dos Santos

Departamento de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

RESUMO

Título: Reconhecimento de Placas de Trânsito em Ciclovias por meio de Redes Neurais
Autor: Kaio Giovanni Pereira dos Santos
Orientador: Prof. Adolfo Bauchspiess

Controle e Automação é uma parte vital de sistemas Eletroeletrônicos com o recente avanço da tecnologia. O presente trabalho apresenta uma contribuição para a proposta de automatização de veículos, focando mais no estudo de formas de filtragem e simplificação de dados para reconhecimento de placas de trânsito Brasileiras usando Redes Neurais de Aprendizagem Profundas para uma possível aplicação em automatização motora em ciclovias do DF. Para tanto, foram usado algoritmos, predominantemente feitos na linguagem de programação Python e fazendo uso de Servidores em Nuvem da Google, que fazem reconhecimento de placas alemãs, devido as suas semelhanças com as placas brasileiras, com testes voltados para o uso em ciclovias do Distrito Federal.

ABSTRACT

Title:

Author: Kaio Giovanni Pereira dos Santos
Supervisor: Prof. Adolfo Bauchspiess

Control and Automation is a vital part of the recent Technological Advance that happened in recent years. This paper presents a contribution to a proposal of vehicle automation, focusing in studies on filtering and data simplification for Brazilian traffic signs recognition using Deep Learning Neural Networks for a potential application on motor automation on bicycle paths at Distrito Federal. For this matter, Python based algorithms and Google cloud servers were used, performing recognition of German traffic signs, due to its similarities to Brazilian ones, with testing facing bicycle paths in Distrito Federal.

SUMÁRIO

1 INTRODUÇÃO 1

1.1 Motivação 1

1.2 Objetivos 3

1.2.1 Objetivo Geral 3

1.2.2 Objetivos Específicos 3

2 FUNDAMENTAÇÃO TEÓRICA E MATERIAIS 4

2.1 Redes Neurais 4

2.1.1 Redes Neurais Profundas 5

2.2 Dados 6

2.2.1 Ambiente Computacional 6

2.2.2 Datasets 7

2.2.3 Overfitting 8

2.3 Modelo Usado de Redes Neurais 9

2.3.1 VGGnet 10

2.4 Processamento de Images 10

2.4.1 Greyscaling 11

2.4.2 Normalização 11

2.4.3 Equalização de Histograma 12

2.4.4 Programa 14

2.5 Análise de Componentes Principais 14

2.5.1 Determinação das Componentes Principais 15

2.5.2 Autofaces 16

2.5.3 Importância 17

2.6 Regressão Logística 18

2.6.1 Função Custo 19

2.6.2 Função Sigmóide 19

3 PROCEDIMENTO E RESULTADOS 20

3.1 Procedimento 20

3.2 Acurácia 20

3.2.1 Rede Convolucional 21

3.2.2 PCA 21

3.2.3 PCA e VGG 22

3.3 Análise dos Resultados 23

3.4 Teste de Desempenho 24

4 CONCLUSÃO E TRABALHOS FUTUROS 29
4.1 TRABALHOS FUTUROS 29

REFERÊNCIAS BIBLIOGRÁFICAS 29

A APÊNDICE 34
A.1 PRIMEIRO APÊNDICE 34
A.2 SEGUNDO APÊNDICE 38
A.3 TERCEIRO APÊNDICE 45

LISTA DE FIGURAS

1.1 Protótipo do Carro Autônomo da Google. Fonte: [1]..... 1
1.2 Comparação entre um Neurônio e seu modelo computacional. Fonte: [2]. 2

2.1 Números escritos à mão. Fonte: [3]..... 4
2.2 Exemplos de Imagens em um banco de dados de números escritos à mão. Fonte: [3]. .. 4
2.3 Exemplo gráfico de, à esquerda, uma rede neural tradicional e, à direita, uma rede neural profunda, onde suas camadas escondidas podem ter camadas convolucionais. Fonte: [4]. 6
2.4 Comparação entre as Placas de Sinalização Brasileiras (à esquerda) e as Alemãs (à direita). Fonte: [5] (esquerda) e [6] (direita). 7
2.5 Exemplos de Imagens usadas no Banco de Dados Alemão de Placas de Trânsito. Fonte:[7]. 7
2.6 Exemplo de Overfitting em um conjunto de pontos em um gráfico. Fonte: [8]. 8
2.7 Overfitting no conjunto de dados. Fonte: [9]. 9
2.8 Arquitetura padrão de uma Rede Neural Convolucional. Fonte: [10]. 9
2.9 Exemplo gráfico do processo de redução da imagem de entrada em um modelo VGG-net. Fonte: [11]..... 10
2.10 Exemplo da Conversão de uma imagem em sua escala cinza (greyscaling). Fonte: [12]. 11
2.11 Normalização de uma imagem feita pelo MatLab. Fonte: [13]. 12
2.12 Exemplo do processo de Equalização de Histograma de uma imagem. (a) Imagem original, (b) histograma da imagem, (c) histograma equalizado, (d) imagem melhorada. Fonte: [14]. 13
2.13 Exemplo comum de um gráfico da porcentagem da energia da informação (eixo vertical) versus quantidade de componentes principais (eixo horizontal). Fonte: [15]. 14
2.14 Exemplo de Autoface. À esquerda vemos uma face média e à direita vemos uma face criada a partir de diferentes valores de componentes principais. Fonte: [16]. 17
2.15 Foto de uma Placa de Trânsito com área de interesse. Fonte: [17]. 18
2.16 Comparação entre função custo linear e sigmóide. Fonte: [18]. 19

3.1 Diagrama dos Procedimentos. Cada seta indica a cor dos diferentes testes feitos nos modelos. A cor verde clara indica pré-processamento da imagem e a cor azul clara indica as ferramentas de medição de acurácia. 20
3.2 Acurácia do Modelo Convolucional VGG..... 21
3.3 Acurácia do Modelo PCA - Regressão Logística. 22
3.4 Acurácia de ambos modelos juntos. 22
3.5 Todas as acurácias testadas..... 23
3.6 Teste detalhado do modelo VGG. 24

3.7 Exemplo de recorte e redimensionamento de uma placa contida no vídeo mencionado. Fonte: Estevon [19]. 25
3.8 Teste de Performance feito com placas da ciclovia do DF. 26

LISTA DE TABELAS

3.1 Tabela de Resultados das porcentagens de acurácia de todos os três testes. 23
3.2 Tabela de resultados do teste com a primeira placa. 27
3.3 Tabela de resultados do teste com a segunda placa. 27
3.4 Tabela de resultados do teste com a terceira placa. 27
3.5 Tabela de resultados do teste com a quarta placa. 27
3.6 Tabela de resultados do teste com a quinta placa. 28

LISTA DE ACRÔNIMOS

CalTech101 California Institute of Technology database, banco de dados do Instituto de Tecnologia da California. 6

CIFAR10 Canadian Institute For Advanced Research database, banco de dados do Instituto Canadense para Pesquisas Avançadas. 6

GPU Graphics Process Unit, Unidade de Processamento Gráfico. 20

MNIST Modified National Institute of Standards and Technology database, banco de dados modificado do Instituto Nacional de Padrões e Tecnologia. 6

PCA Principal Component Analysis, Análise de Componentes Principais. 14, 20–22, 29

VGGnet Visual Geometry Group database, banco de dados do Grupo de Geometria Visual. 10, 14, 21, 22, 25, 29

LISTA DE SÍMBOLOS

$f(x)$	Composição de funções de uma Rede Neural Profunda
$f_L(.)$	Função da camada L em uma Rede Neural Profunda
A'	Função de Normalização Min-Max sobre um conjunto de dados A
A	Tamanho dos dados de um conjunto
A_{min}	Menor valor de um conjunto de dados A
A_{max}	Maior valor de um conjunto de dados A
D	Intervalo superior desejado para definir Normalização Min-Max
C	Intervalo inferior desejado para definir Normalização Min-Max
$T()$	Transformada
$T^{-1}()$	Transformada inversa
$p_r(r)$	Função Densidade de Probabilidade dos niveis de cinza da imagem original
$p_s(s)$	Função Densidade de Probabilidade dos niveis de cinza da transformada
X	Matriz de dados usadas em PCA
S	Matriz de covariância da população em PCA
R	Matriz de correlação da população em PCA
λ	Autovalores da matriz S ou R
$\tilde{a}$	Autovetor correspondentes aos autovalores
Y_i	i-ésimas Componentes Principais
F	Nova face em PCA
F_m	Face média em PCA
F_i	Autoface em PCA
α_i	Multiplos escalares positivos ou negativos

1 INTRODUÇÃO

Em um mundo progressivamente conectado por meio remoto, a inteligência artificial é um passo esperado do desenvolvimento humano. Tendo isso em vista, torna-se de grande interesse que robôs inteligentes sejam mais autônomos para desempenhar atividades consideradas exclusivas do homem.

O uso da Inteligência Artificial em certas atividades é necessário, pois nem tudo pode ser controlado por meio de equações de controle dinâmico. Tarefas consideradas triviais para seres humanos, como a identificação de objetos, são trabalhosas para computadores, porém podem ser simplificadas pela utilização da Inteligência Artificial. Neste relatório, o objeto de estudo destaca como o uso da Inteligência Artificial pode desempenhar mais uma de várias atividades triviais humanas: o reconhecimento de placas de trânsito em ciclovias.

1.1 MOTIVAÇÃO

Desenvolver um veículo que não necessite de um motorista para andar pelas ruas não é algo novo e já tem sido apresentado e testado por várias empresas de tecnologia. Um grande exemplo disso é o carro autônomo do Google [1], que não possui volante nem acelerador ou pedal de freio.

Figura 1.1: Protótipo do Carro Autônomo da Google. Fonte: [1].

Programas e algoritmos são amplamente criados para simular de forma adequada o funcionamento de um carro como se este tivesse um condutor. Dentro desses algoritmos, conseguimos citar

sensores que indicam distâncias entre carros, câmeras identificadoras de faixas no chão para que o carro continue na pista, etc. A fim de realizar esse reconhecimento, é necessário um algoritmo que possa identificar e diferenciar placas de trânsito para que o carro saiba o que fazer em determinado trajeto. O recurso mais promissor para esse fim é o uso de Redes Neurais em Inteligência Artificial.

A Inteligência Artificial foi primeiramente proposta por John McCarthy em 1956. McCarthy apresentou a ideia de que máquinas poderiam ter a mesma capacidade do ser humano, pensar e aprender por si mesmas, ideia desenvolvida mais tarde pelo matemático Alan Turing no mesmo ano [20].

Redes Neurais Artificiais são os modelos para uma representação da conexão física entre neurônios. Elas têm sido utilizadas para resolver uma variedade de problemas por meio da construção matemática de modelos que imitem o funcionamento de atividades neurais naturais na perspectiva do cérebro. Com o uso desse algoritmo, a máquina foi capaz de identificar e resolver problemas da mesma forma que um ser humano faria.

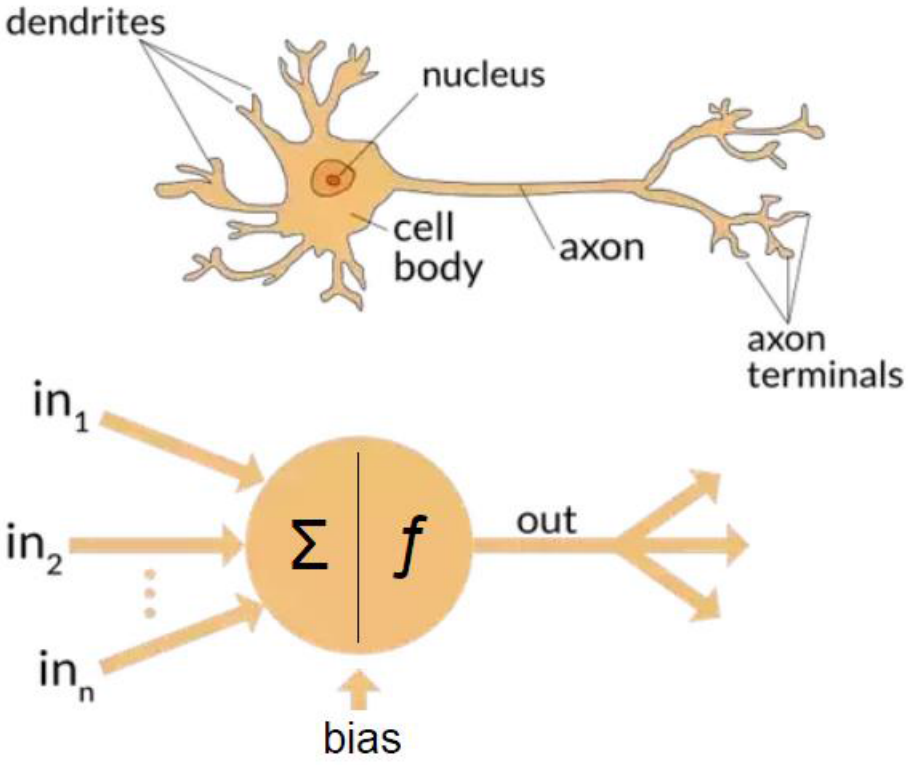

Figura 1.2: Comparação entre um Neurônio e seu modelo computacional. Fonte: [2].

A habilidade de conseguir se comportar como um ser humano é crucial para que o sistema de reconhecimento de placas seja possível. Isso somente pode ocorrer com a habilidade subjetiva inerente do ser humano, que dificilmente pode ser simulada por fórmulas e equações.

1.2 OBJETIVOS

Ciclovias brasileiras são frequentemente utilizadas para lazer ou como uma opção ao carro caso uma pessoa more perto de seu local de trabalho. Uma visão futurista deste trabalho é termos ciclovias capazes de servir como vias para máquinas que saibam navegar nesses espaços, e que isso possa incentivar a construção de melhores circuitos e a implementação de um sistema de sinalização de veículos mais complexo, a fim de poderem circular livremente em um espaço ciclístico, como por exemplo a movimentação de uma máquina em uma bicicleta sem precisar de um controle remoto.

1.2.1 Objetivo Geral

Temos carros em abundância nas rodovias e percebemos que sempre há um espaço livre e comumente inutilizado em ciclovias do Brasil, que pode ser muito útil para documentação e compartilhamento de melhores rotas para ciclistas do Distrito Federal, caso o reconhecimento de placas seja mais utilizado a fim da montagem de mapas das ciclovias do DF em tempo real. Para que esse objetivo seja alcançado, é preciso fazer com que o computador realize um processamento das imagens recebidas e saiba reconhecê-las por meio da Inteligência Artificial.

Sendo assim, o objetivo geral deste trabalho é criar uma ferramenta de reconhecimento de placas de trãnsito brasileiras para auxílio à programadores e possíveis auxílios à ciclistas.

1.2.2 Objetivos Específicos

Podemos então especificar os objetivos especificos como:

- Fazer um comparativo de métodos de simplificação de dados referentes a placas de trânsito brasileiras fazendo o uso de um banco de dados de placas alemãs;
- Averigurar o funcionamento destes métodos em imagens de ciclovias de Brasília através do design de uma inteligência computacional de reconhecimento de placas em ciclovias.

2 FUNDAMENTAÇÃO TEÓRICA E MATERIAIS

2.1 REDES NEURAIS

Usando o exemplo de [3], podemos considerar a seguinte sequência de números escritos à mão:

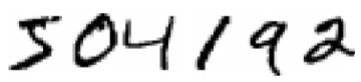

Figura 2.1: Números escritos à mão. Fonte: [3].

As pessoas facilmente conseguem identificar o número como 504192, porém, o processo por detrás desse reconhecimento consiste em uma conexão complexa contendo mais de 140 milhões de neurônios e mais de bilhões de conexões entre tais neurônios. E toda essa parte de processamento está contida em apenas uma parte do cerebro ao todo. Com isso, podemos perceber que o cérebro, diante de sua evolução desde o começo da humanidade, foi treinado para que este tipo de tarefa seja trivial.

Por outro lado, quando tentamos fazer com que um programa faça tal reconhecimento, precisamos de um grande poder de processamento, além de escrever um complexo programa para tentar realizar a mesma tarefa com uma porcentagem de sucesso que nem é 100%, o que é uma tarefa muito mais dificil.

O objetivo das redes neurais é criar um sistema que consiga aprender a partir de um banco de dados (Machine Learning), que neste caso é de vários números em diferentes escritas, como mostrado na imagem a seguir.

Figura 2.2: Exemplos de Imagens em um banco de dados de números escritos à mão. Fonte: [3].

Em outras palavras, o que a rede faz é criar suas próprias regras para tal reconhecimento de dados a partir de um banco de dados, ao qual terá seu sucesso mais garantido quanto maior for tal banco, ou seja, mais exemplos lhe forem apresentados, de tal forma que, quando uma nova imagem for apresentada, o programa terá uma chance melhor de reconhecê-la.

2.1.1 Redes Neurais Profundas

Pode ser entendido sobre o assunto que:

> O método que utiliza Redes Neurais Profundas, ou Deep Learning, é aquele que tenta descobrir um modelo utilizando um conjunto de dados e um método para guiar o aprendizado do modelo a partir desses exemplos, criando assim uma função que recebe dados brutos como entrada e fornece uma saída com a representação adequada do problema em questão [21].

Uma definição objetiva de uma rede neural profunda seria de que ela é uma rede neural comum com mais de uma camada, porém, tem vários aspectos por trás de uma definição como essa, como foi detalhado por John Patterson e Adam Gibson [22].

- Contém mais neurônios que redes neurais comuns;
- Maneiras mais complexas de se conectar as camadas/neurônios;
- Aumento drástico na quantidade de poder de processamento disponível para treino;
- Extração automática de características.

Ainda mais, aprendizado profundo pode ser definido como redes neurais divididas em quatro arquiteturas de rede fundamentais [22]:

- Redes Neurais Não-supervisionadas;
- Redes Neurais Convolucionais;
- Redes Neurais Recorrentes;
- Redes Neurais Recursivas.

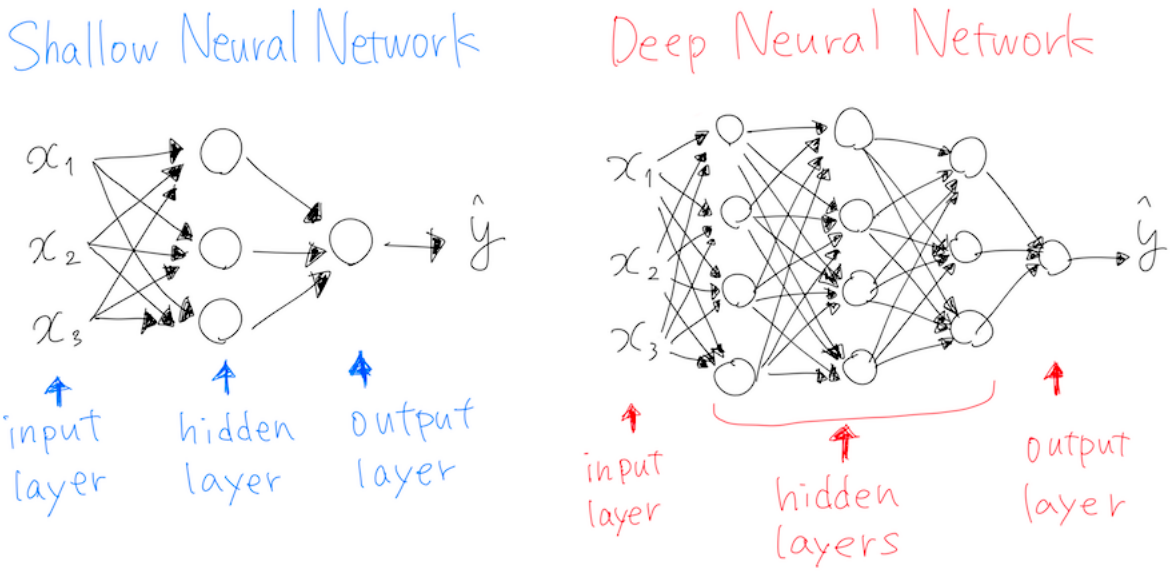

Figura 2.3: Exemplo gráfico de, à esquerda, uma rede neural tradicional e, à direita, uma rede neural profunda, onde suas camadas escondidas podem ter camadas convolucionais. Fonte: [4].

2.2 DADOS

2.2.1 Ambiente Computacional

Para o processamento das imagens utilizadas no reconhecimento de placas de trânsito, será necessário um ambiente em que isso seja facilmente dedutível. A linguagem de programação escolhida para fazer o projeto é a Python [23], escolhida por vários motivos:

- Python pode ser facilmente usada até mesmo por um programador inexperiente;
- A comunidade Python organiza eventos e encontros para colaboração de códigos etc.;
- A biblioteca em Python armazena mais de milhares de módulos e pacotes para diferentes aplicações não padronizadas;
- Python foi desenvolvida em uma licença Open-Source, o que significa que seus códigos são facilmente distribuídos e até abertos para uso comercial.

Na linguagem Python, utilizamos o chamado Tensorflow, que é uma biblioteca de códigos abertos para aprendizado de máquina, especifica para o uso em Inteligência Artificial. Para que um conjunto de dados possa ser utilizado pelo Tensorflow, são necessários dados devidamente separados e etiquetados, e isso é feito pelos bancos de dados específicos para reconhecimento de objetos mais conhecidos na área de Inteligência Artificial, como CIFAR10, MNIST e CalTech101. Infelizmente, foi notado,

durante o trabalho, que um banco de dados específicos das placas de trânsito brasileiras é inexistente. Sendo assim, como solução, foi feito uso do banco de imagens de placas de trânsito alemãs, que compartilham características visuais semelhantes às placas de trânsito brasileiras.

Figura 2.4: Comparação entre as Placas de Sinalização Brasileiras (à esquerda) e as Alemãs (à direita). Fonte: [5] (esquerda) e [6] (direita).

O banco de dados já existente usado para implementar este programa foi o German Traffic Sign Dataset [24], que é um banco de imagens únicas, com mais de 40 classes e acima de 50000 imagens no total.

Figura 2.5: Exemplos de Imagens usadas no Banco de Dados Alemão de Placas de Trânsito. Fonte:[7].

2.2.2 Datasets

Para fins de reconhecimento de características feitos em Redes Neurais Profundas, todos os dados de entrada devem ser separados em grupos distintos (Datasets) chamados Treinamento, Validação e

Teste. O conjunto de dados de treinamento é nome dado ao conjunto usado para treinar o modelo de rede neural. O modelo vê e aprende através dele. O conjunto de dados de validação é responsável pela avaliação imparcial do modelo ajustado aos dados de treinamento enquanto ajusta os seus parâmetros. Tais dados são usados para sintonizar os parâmetros do modelo, sendo assim, o modelo pode ver através deste conjunto, mas nunca aprende com ele. O conjunto de dados de teste é responsável pela avaliação imparcial do modelo final ajustado com os dados de treinamento e sintonizado com os dados de validação. Estes, por fim, são usados para testar o modelo com imagens minuciosamente escolhidas em diferentes situações em que o modelo pode ser apresentado [25].

2.2.3 Overfitting

Sobreajuste, ou Overfitting, é um termo estatístico que descreve um modelo estatístico que se ajusta muito bem ao conjunto de dados observado, incorporando os seus mínimos detalhes, como ruídos, e isso afeta sua performance quando apresentado a novos conjuntos de dados [26]. Visualmente, podemos identificar Overfitting quando a porcentagem de acurácia começa a diminuir após alcançar uma certa quantidade de épocas, que é a quantidade de vezes que os dados são ajustados no modelo.

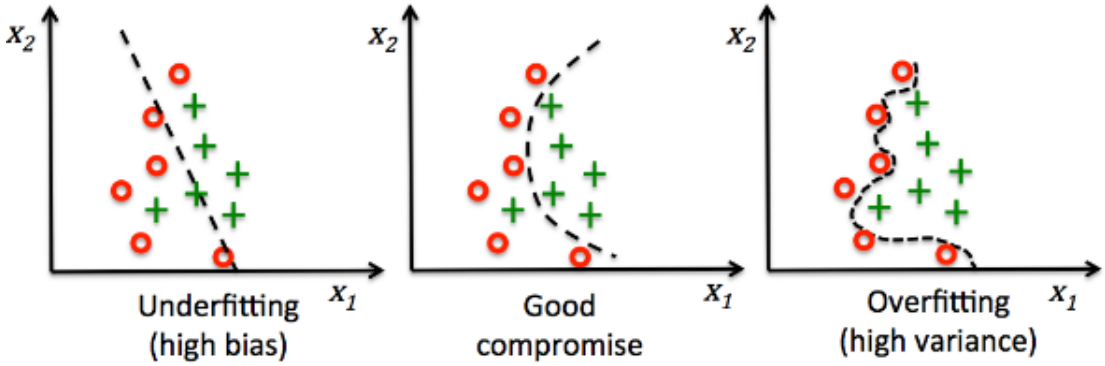

Figura 2.6: Exemplo de Overfitting em um conjunto de pontos em um gráfico. Fonte: [8].

O conjunto de validação é usado especialmente para evitar este problema, pois ele apresenta dados imparciais para manter o ajuste dos dados de treinamento o mais generalista possível, aumentando seu sucesso com outros dados a serem apresentados quando o sistema for completamente treinado.

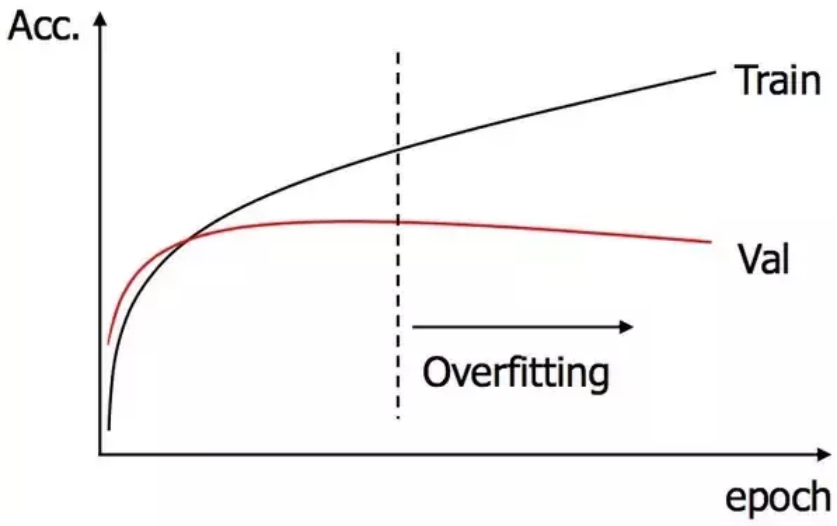

Figura 2.7: Overfitting no conjunto de dados. Fonte: [9].

2.3 MODELO USADO DE REDES NEURAIS

O reconhecimento de imagens será feito por um conjunto de algoritmos que simulam o comportamento de um neurônio humano, chamado de Redes Neurais Convolucionais. Esse mecanismo foi criado por Yann LeCun em 1998, no intuito de realizar o reconhecimento de letras feitas à mão e por máquinas de escrever.

Primeiramente, As Redes Neurais Convolucionais recebem os dados pré-processados, como será descrito na próxima seção, através de sua primeira camada, com cada camada à seguir recebendo dados da camada anterior. Ao fazer isso, a rede consegue extrair características dos dados, que nos casos das imagens são características visuais. Essas características são combinadas posteriormente, formando um Mapa de Características da imagem [10].

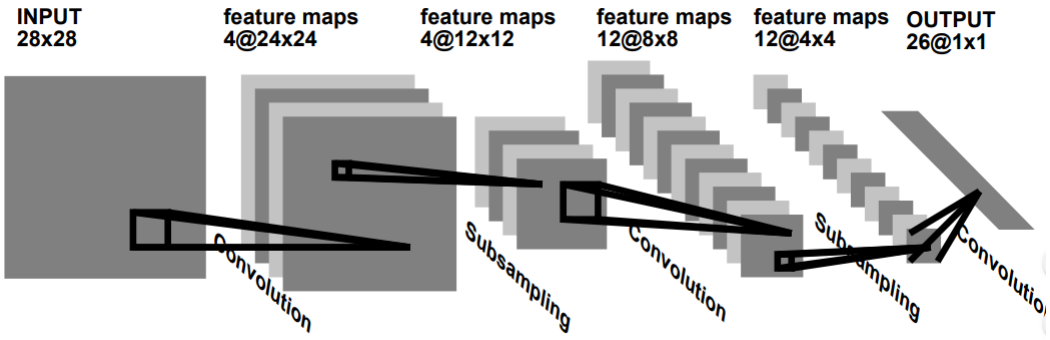

Figura 2.8: Arquitetura padrão de uma Rede Neural Convolucional. Fonte: [10].

Para que as características de cada imagem sejam extraídas de modo eficaz, precisamos trabalhar

em uma forma de melhorar tais imagens, pois, quando tais imagens são convertidas, sua qualidade pode ser inferior à da imagem original (Como pode ser visto, uma imagem precisa ser reduzida a 28x28 para entrar na Rede Neural). Sendo assim, há vários métodos disponíveis para melhorar a qualidade dessas imagens e aumentar a eficiência da própria rede Neural Utilizada.

2.3.1 VGGnet

O modelo usado neste trabalho foi VGGnet, que é um uma rede neural profunda de reconhecimento de objetos criada e treinada pelo Visual Geometry Group, do Instituto de Robótica da universidade de Oxford [27].

O VGGnet Foi inicialmente proposta por Karen Simonyan e Andrew Zisserman e conseguiu uma das maiores acurácias de teste usando um dos maiores bancos de dados de imagens existente, o ImageNet.

Em comparação com os primeiros modelos propostos, o VGGnet fazia o uso de várias camadas de tamanho reduzido, ao invés de uma grande camada. Múltiplas camadas faziam com que o modelo fosse mais profundo e aprendesse características mais complexas [28].

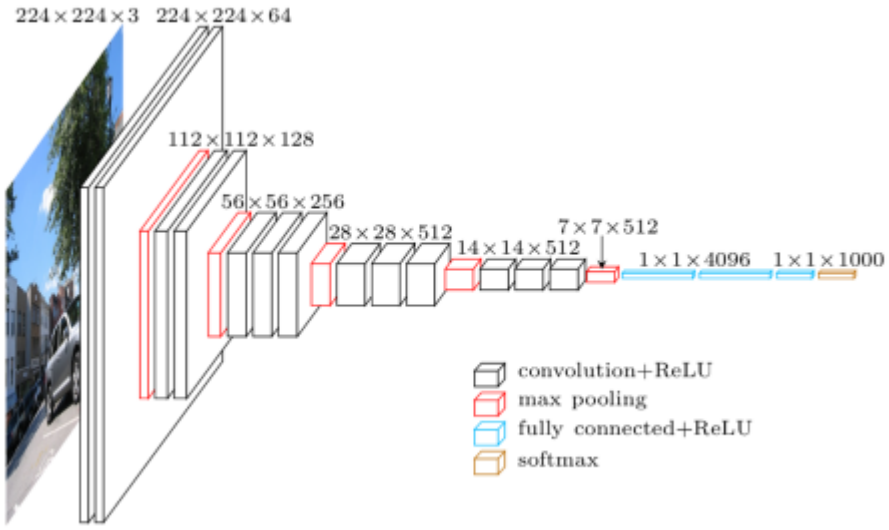

Figura 2.9: Exemplo gráfico do processo de redução da imagem de entrada em um modelo VGGnet. Fonte: [11].

2.4 PROCESSAMENTO DE IMAGES

Depois de as imagens serem corretamente absorvidas para uso dentro do programa e etiquetadas corretamente, o próximo passo é o processamento. Na parte de pré-processamento de imagens,

que corresponde ao desenvolvimento necessário à simplificação delas para uso mais eficiente, foram utilizadas as seguintes técnicas: Greyscaling, Normalização e Equalização de Histograma.

2.4.1 Greyscaling

O processo de converter uma imagem para sua representação na escala do cinza é feito para fins de redimensionamento [29]. No seu processo mais comum, é adquirida a informação da luminosidade, que é então convertida como no seguinte exemplo:

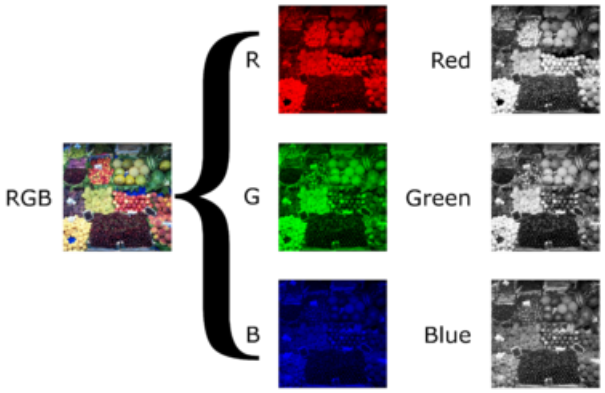

Figura 2.10: Exemplo da Conversão de uma imagem em sua escala cinza (greyscaling). Fonte: [12].

Ao assumir que R, G e B sejam sinais lineares da luminosidade das diferentes dimensões de cores de uma imagem (R para Vermelho, G para Verde e B para Azul), adquirir apenas a luminosidade de uma imagem provoca uma grande perda de informação. No entanto, essa informação é desnecessária no objetivo de se reconhecer padrões de formas e características [30].

2.4.2 Normalização

O processo de normalização usado neste trabalho, chamado de Normalização Min-Max, é uma técnica simples onde ela consegue encaixar dados em um intervalo pré-determinado, que neste caso, é conveniente que consigamos encaixar todos os valores em um intervalo especifico [0:1] [31]. Podemos definir a equação de normalização como

$$A' = (\frac{A - A_{min}}{A_{max} - A_{min}})(D - C) + C, \tag{2.1}$$

onde A' são os dados normalizados entre as fronteiras $[C, D]$ com A sendo o tamanho original dos dados, A_{min} sendo o menor valor de A e A_{max} o maior valor de A [31].

O propósito de normalizar as coordenadas de uma cor é ajustar os sensores ou emissores do sistema físico para criar uma escala de correção de cor para cada coordenada. Isso melhora a reprodução

das cores neutrais (cinzas), que é um passo adiante de uma renderização confiável das cores de uma imagem.

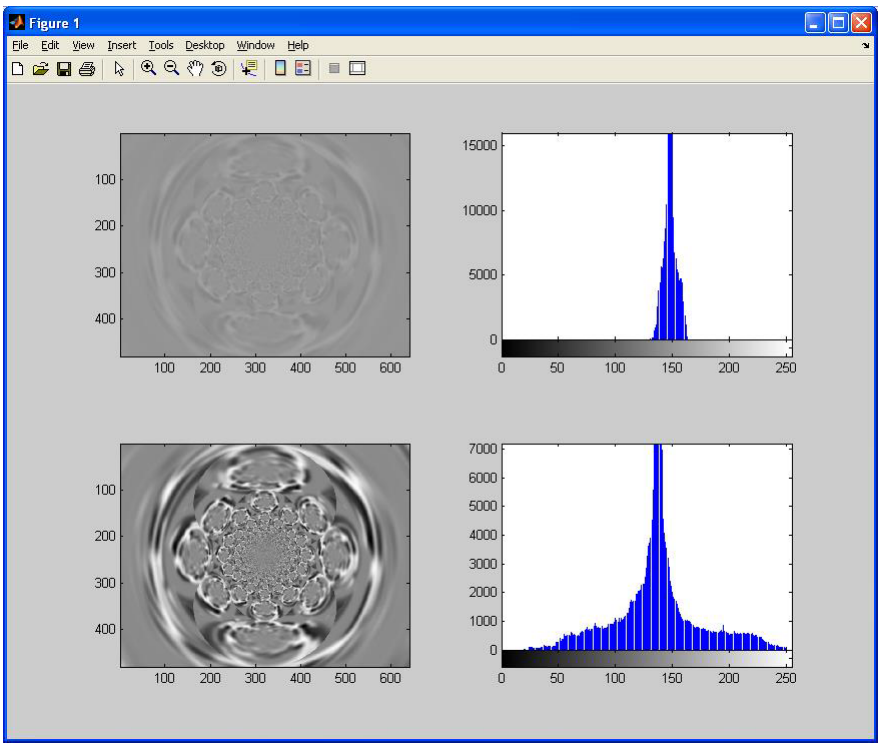

Figura 2.11: Normalização de uma imagem feita pelo MatLab. Fonte: [13].

2.4.3 Equalização de Histograma

Equalização de Histograma é uma técnica que consiste em ajustar a escala cinza de uma imagem para que o nível de histograma do cinza da imagem seja mapeado em um histograma uniforme, com o uso da variável r como um valor de cinza em uma imagem [14]. Inicialmente, assumimos que esse valor é contínuo e esteja em um intervalo fechado [0:1], em que r = 0 seja o preto e r = 1, o branco. Para cada r em um intervalo específico, teremos a transformação

$$s = T(r), \tag{2.2}$$

Produzindo um nível s para cada valor em pixel r na imagem original. Sendo assim, a transformada inversa seria

$$r = T^{-1}(s). \tag{2.3}$$

Agora, os níveis de cinza da imagem original e da transformada são caracterizados pelas suas funções densidade de probabilidade $p_r(r)$ e $p_s(s)$,respectivamente. Sendo assim, se temos uma trans-

formada conhecida e uma função $T(r)$ singular, que cresce no intervalo [0:1], então há a seguinte função densidade de probabilidade do nível de cinza transformado

$$P_s(s) = [P_r(r)\frac{dr}{ds}]_{r-T^-1(s)}. \tag{2.4}$$

Se a transformação é dada por:

$$s = T(r) = \int_0^r P_r(w)dw, \tag{2.5}$$

Então, substituindo $\frac{dr}{ds}$ = $\frac{1}{p_r(r)}$ na equação (2.4), obtemos $P_s(s)$ = 1, desse modo, é possível adquirir um histograma distribuído uniformemente de uma imagem por meio da transformação $T(r)$.

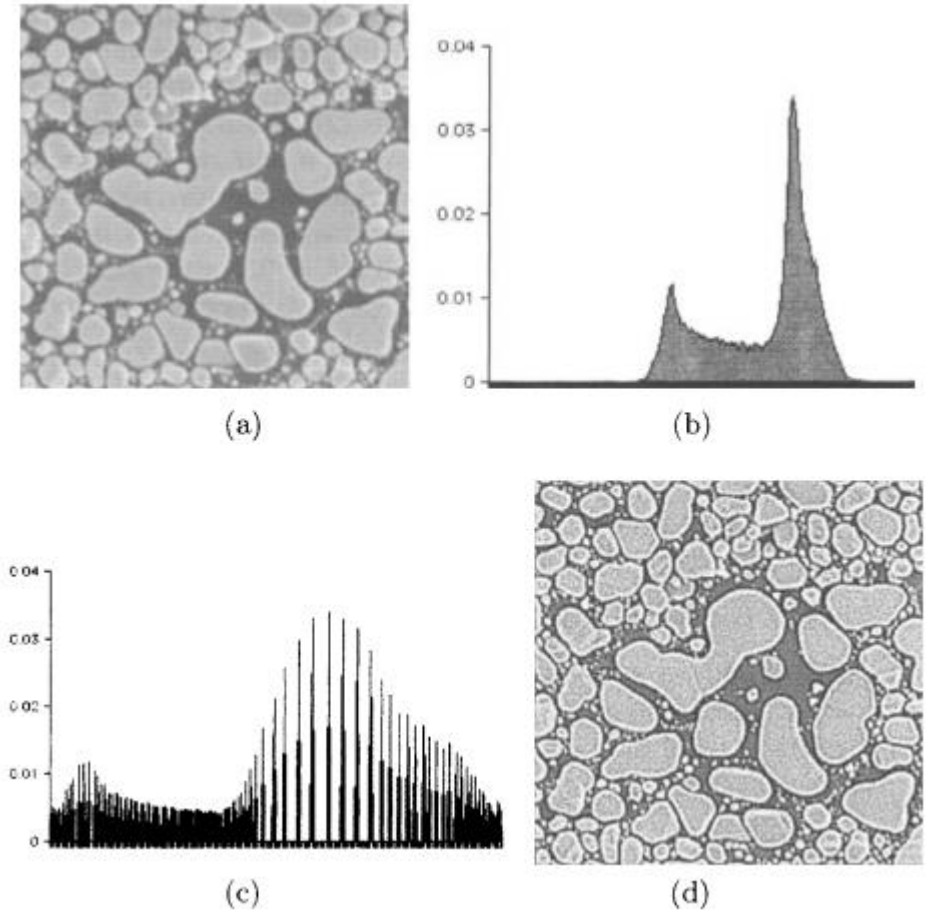

Figura 2.12: Exemplo do processo de Equalização de Histograma de uma imagem. (a) Imagem original, (b) histograma da imagem, (c) histograma equalizado, (d) imagem melhorada. Fonte: [14].

Como pode ser visto, este processo torna a forma da imagem muito mais nítida para ser trabalhada nas redes neurais mais para frente, portanto, esta técnica se faz necessária para que seja alcançada uma

boa eficiência.

2.4.4 Programa

Os algoritmos utilizados foram adaptados de versões compartilhadas no site de repositório de códigos em Python, Github. O algoritmo do modelo VGGnet foi feito por Aviv Shamsian[32] e o algoritmo do modelo PCA foi feito por Michael Galarnyk [33]. Os códigos em Python serão disponibilizados no Anexo A.

2.5 ANÁLISE DE COMPONENTES PRINCIPAIS

PCA ou Análise de Componentes Principais, é um método de redução de dimensionalidade de um grande banco de dados, transformando um grande conjunto de variáveis em um conjunto menor contendo a maior parte da informação do original.

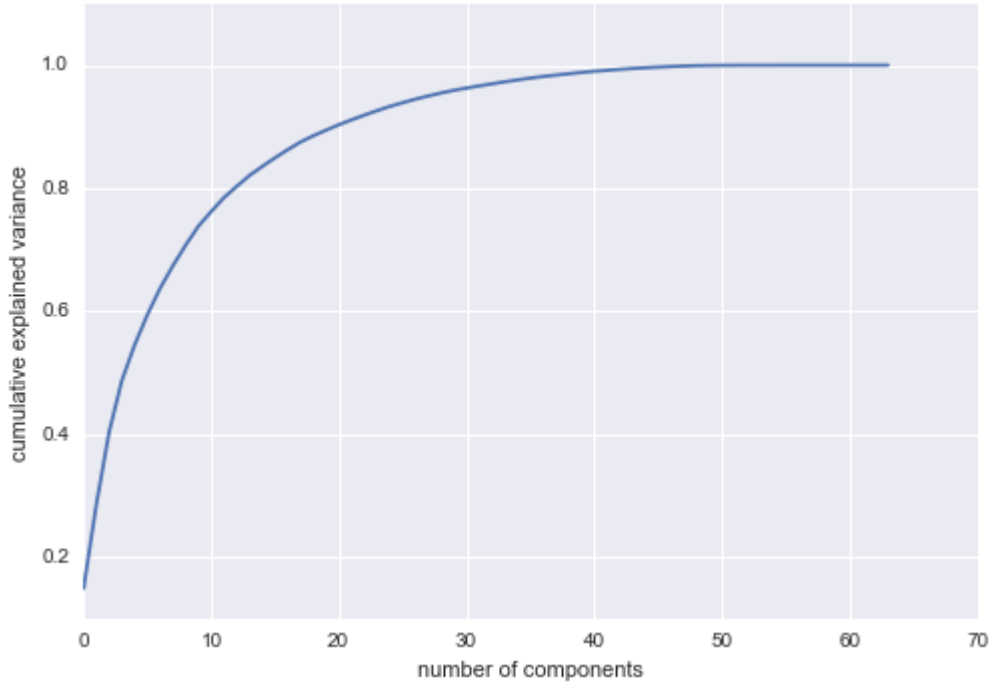

Figura 2.13: Exemplo comum de um gráfico da porcentagem da energia da informação (eixo vertical) versus quantidade de componentes principais (eixo horizontal). Fonte: [15].

> Considere o seguinte: temos um vetor x de dimensões m e desejemos transmiti-lo usando l números, onde l<m. Se simplesmente truncarmos o vetor x, causaremos um erro médio quadrado igual a soma das variâncias dos elementos eliminados de x. Assim, fazemos a seguinte pergunta: existe transformação linear inversiva T tal

que o truncamento de Tx seja ótimo no sentido do erro médio quadrado? Claramente, a transformação T deve ter propriedade que alguns de seus componentes tenham baixa variância [34].

2.5.1 Determinação das Componentes Principais

Seguindo o exemplo e explicação do professor Carlos Alberto Alves Varella [35], considere a situação em que observamos 'p' características de 'n' indivíduos de uma população. Tais características serão representadas pelas variáveis X_1, X_2, X_3, $\cdots$,X$_p$. A matriz de dados é de ordem 'n x p' denominada matriz 'X'.

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1p} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2p} \\ x_{31} & x_{32} & x_{33} & \cdots & x_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \cdots & x_{np} \end{bmatrix} \tag{2.6}$$

A estrutura de interdependência entre as variáveis da matriz de dados é representada pela matriz de covariância S ou pela matriz de correlação R. O objetivo da Análise de Componentes Principais é de transformar essa estrutura complicada em uma outra estrutura de variáveis Y_1, Y_2, Y_3, $\cdots$,Y$_p$ não correlacionadas e com variâncias ordenadas, para ser possível a comparação de indivíduos usando apenas as variáveis Y_{is} que apresentam maior variância.

A estimativa feita da matriz de covariância da população pode ser representada por S, que é simétrica e de ordem 'p x p'

$$S = \begin{bmatrix} \hat{V}ar(x_1) & \hat{C}ov(x_1x_2) & \hat{C}ov(x_1x_3) & \cdots & \hat{C}ov(x_1x_p) \\ \hat{C}ov(x_2x_1) & \hat{V}ar(x_2) & \hat{C}ov(x_2x_3) & \cdots & \hat{C}ov(x_2x_p) \\ \hat{C}ov(x_3x_3) & \hat{C}ov(x_3x_2) & \hat{V}ar(x_3) & \cdots & \hat{C}ov(x_3x_p) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{C}ov(x_px_p) & \hat{C}ov(x_px_2) & \hat{C}ov(x_px_3) & \cdots & \hat{V}ar(p) \end{bmatrix} \tag{2.7}$$

Já a matriz de correlação pode ser entendida como

$$R = \begin{bmatrix} 1 & r(x_1x_2) & r(x_1x_3) & \cdots & r(x_1x_p) \\ r(x_2x_1) & 1 & r(x_2x_3) & \cdots & r(x_2x_p) \\ r(x_3x_1) & r(x_3x_2) & 1 & \cdots & r(x_3x_p) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r(x_px_1) & r(x_px_2) & r(x_px_3) & \cdots & 1 \end{bmatrix} \tag{2.8}$$

Os componentes principais são determinados resolvendo-se a equação característica da matriz S ou R, ou seja:

$$det[S - \lambda I] = 0; \tag{2.9}$$

ou

$$det[R - \lambda I] = 0. \tag{2.10}$$

Se a matriz R for de posto completo igual a 'p', a equação $det[R - \lambda I] = 0$ terá 'p' raizes chamadas de autovalores. Sejam λ_1, λ_2, λ_3, $\cdots$,λ_p as raízes da equação característica da matriz R ou S, então:

$$\lambda_1 > \lambda_2 > \lambda_3 \cdots, \lambda_p. \tag{2.11}$$

Para cada autovalor λ_i existe um autovetor $\tilde{a_i}$:

$$\tilde{a_i} = \begin{bmatrix} a_{i1} \\ a_{i2} \\ \vdots \\ a_{ip} \end{bmatrix} \tag{2.12}$$

Que são autovetores normalizados e ortogonais entre si.

Sendo $\tilde{a_i}$ o autovetor correspodente ao autovalor λ_i, então o i-ésimo componente principal é dado por:

$$Y_i = a_{i1}X_1 + a_{i2}X_2 + \cdots + a_{ip}X_p \tag{2.13}$$

Sabendo disso, para reduzirmos a dimensionalidade, devemos calcular autovalores e autovetores da matriz de correlação dos dados de entrada e, depois, projetando os dados ortogonalmente sobre os autovetores pertencentes aos autovalores dominantes [34].

2.5.2 Autofaces

Uma aplicabilidade da Análise de Componentes Principais no campo de reconhecimento facial é chamada de Autofaces, que são adicionadas a faces médias para criar novas faces, que pode ser escrito matematicamente como uma série

$$F = F_m + \sum_{i=1}^{N} \alpha_i F_i, \tag{2.14}$$

Onde F é a nova face, F_m é a face média, que é o valor fixo da aproximação da série, F_i é a autoface e $alpha_i$ são múltiplos escalares positivos ou negativos. Autofaces são calculadas estimando os componentes principais de um banco de dados de faces [36].

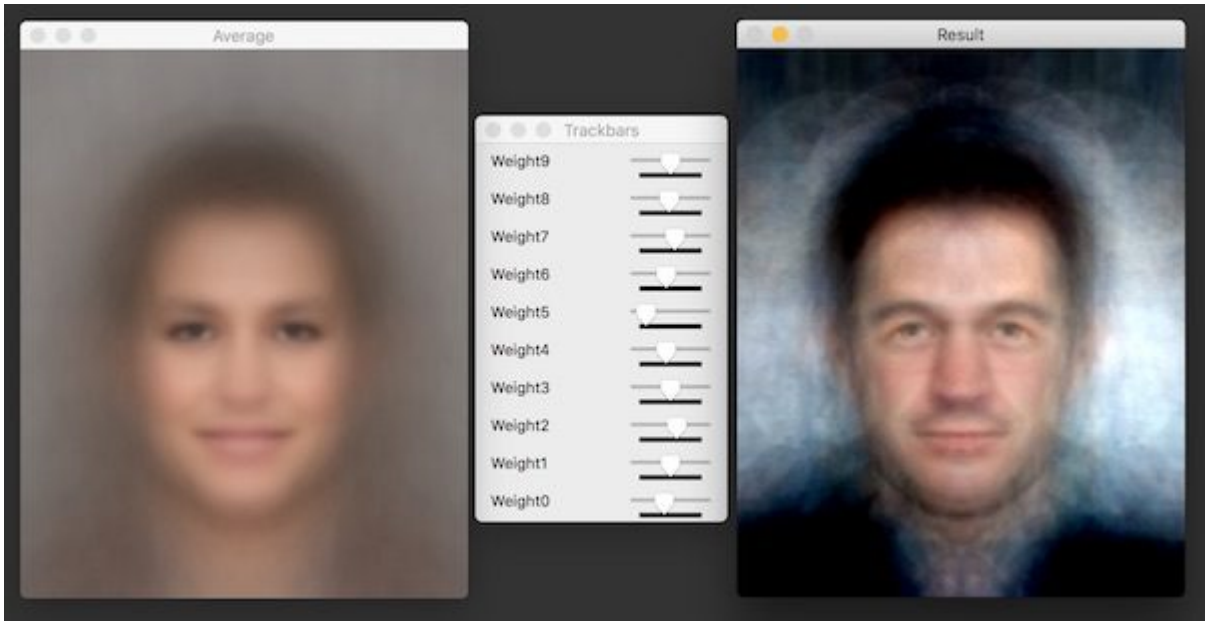

Figura 2.14: Exemplo de Autoface. À esquerda vemos uma face média e à direita vemos uma face criada a partir de diferentes valores de componentes principais. Fonte: [16].

2.5.3 Importância

O nosso projeto, de modo geral, visa fazer o reconhecimento de placas de trânsito. A inteligência artificial irá usar todos os dados coletados dentro das imagens para fazer o reconhecimento em si, porém nós sabemos que nem todos os dados contidos em uma imagem de uma placa de trânsito são relevantes para o propósito de reconhecimento. Tenha a imagem 11 como exemplo.

Figura 2.15: Foto de uma Placa de Trânsito com área de interesse. Fonte: [17].

Para o objetivo de reconhecimento, podemos concluir que, nesta imagem, qualquer coisa que não está contida no contorno verde é irrelevante, sendo assim, é de suma importância a simplificação para melhor eficiência.

Tal simplificação já foi feita nos outros métodos apresentados anteriormente, porém a Análise de Componentes Principais pode ser vantajosa se levarmos em consideração o seu próprio funcionamento. Ela funciona de forma a separar as características principais de uma imagem, que no nosso caso pode ser interpretado como as cores do fundo e detalhes do asfalto. Tais variantes podem ser melhor controladas se limitarmos a quantidade de componentes principais a serem consideradas, descartando assim boa parte dos dados irrelevantes de cada imagem, aumentando a eficiência do reconhecimento do programa.

2.6 REGRESSÃO LOGÍSTICA

"Regressão logística é um algoritmo de classificação usado para atribuir observações em um conjunto discreto de classes" [37]. Exemplos binários podem ser tumores malignos/não malignos, se uma imagem representa um cachorro ou um gato, dentre outros. Outro tipo de exemplo é de multiclasse, em que um dos exemplos é o escopo deste projeto, de reconhecimento de múltiplas classes diferentes. Em Machine Learning, regressão logística será um algoritmo de classificação probabilística e fará o uso de uma determinada função custo.

Para entender tal função custo, primeiro se faz necessário entender o que exatamente é uma função custo.

2.6.1 Função Custo

Usando a analogia criada por McDonald(2017), podemos imaginar uma criança que brinca com uma fogueira. Quando ela coloca seu dedo no fogo, ela se queima, e com este aprendizado, ela percebe que deve se afastar do fogo, portanto se afastando do mesmo. Da próxima vez, a criança ainda se senta muito próxima ao fogo, ainda sentindo desconforto com o calor da fogueira, portanto se afastando ainda mais do mesmo. Através de sua experiência de se queimar, a criança aprende a distância correta para permanecer referente à fogueira para ela não se queimar sem sentir frio. Nesta analogia, o calor da fogueira age como uma função custo, que serve para ajustar o comportamento de um agente aprendendo para que seus erros possam ser minimizados [18].

Em machine learning, usando como exemplo nosso objetivo de reconhecimento de placas, uma função custo mede o quanto que um modelo está errado em conseguir estimar a relação entre a imagem de uma placa e seu rótulo.

2.6.2 Função Sigmóide

Diferente da mais comumente usada Regressão Linear, a função de uma função sigmóide é limitar a função custo entre 0 e 1, ao invés de extrapolar os valores.

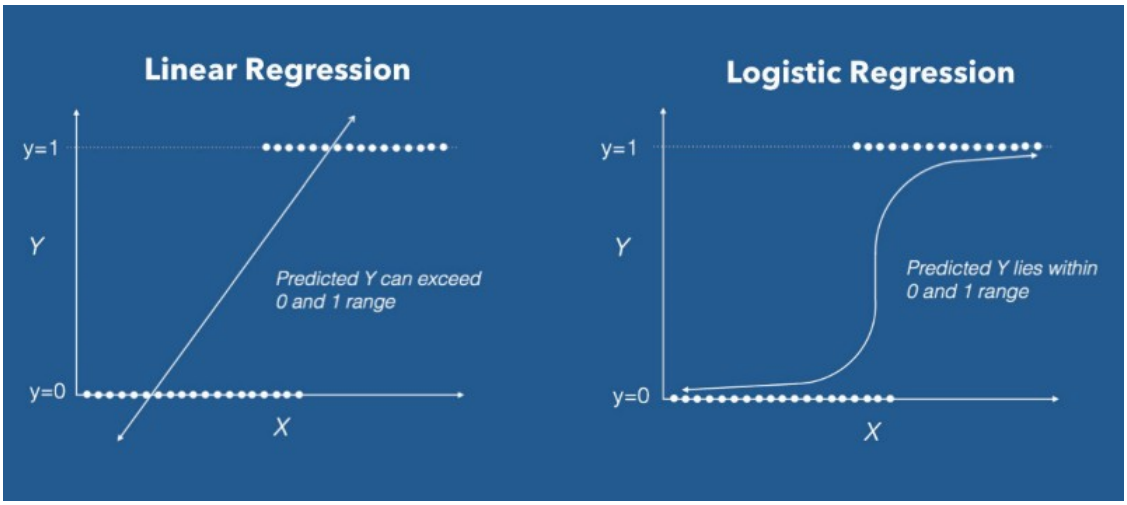

Figura 2.16: Comparação entre função custo linear e sigmóide. Fonte: [18].

Tal função também é particularmente boa para que haja um limiar de mínimo e máximo valor para que uma única classe seja decidida, por exemplo, se usarmos a função sigmóide para decidir se uma imagem é de um cachorro ou de um gato, teremos, por exemplo, um valor relativamente fixo perto de 0 para gatos e um valor perto de 1 para cachorros, reservando os valores perto de 0.5 para imagens ambíguas, coisa que não seria possível em uma função linear, onde duas imagens de gatos poderiam ter valores completamente diferentes.

3 PROCEDIMENTO E RESULTADOS

3.1 PROCEDIMENTO

Toda a programação foi feita usando Google Colaboratory, que é um ambiente de programação em Python que roda inteiramente na nuvem, onde foi usado os servidores da Google para diminuir o tempo de processamento de cada teste feito com os dados. O processamento em nuvem é feito com o auxilio de uma GPU (Graphics Process Unit), que é um componente eletrônico dedicado à renderização de gráficos em tempo real normalmente encontrado em uma placa de vídeo. O tempo de execução comum de um programa feito com GPU é de pelo menos 45 vezes mais rápido do que feito sem. Para que seja testada a correta funcionalidade do programa, foi feita uma comparação entre métodos de reconhecimento usando apenas PCA, outro usando apenas a rede convolucional e, finalmente, usando ambos em conjunto. Todos os testes de acurácia dos métodos foram feitos com base no conjunto de dados de placas de trânsito alemãs de treinamento, incluindo alguns exemplos de placas brasileiras, juntamente com os dados de validação de placas também alemãs. Por fim, foi feito um teste prático usando o método que retornou melhor acurácia para aferir a aplicabilidade do programa em um ambiente cicloviário brasileiro na ciclovia de Brasília.

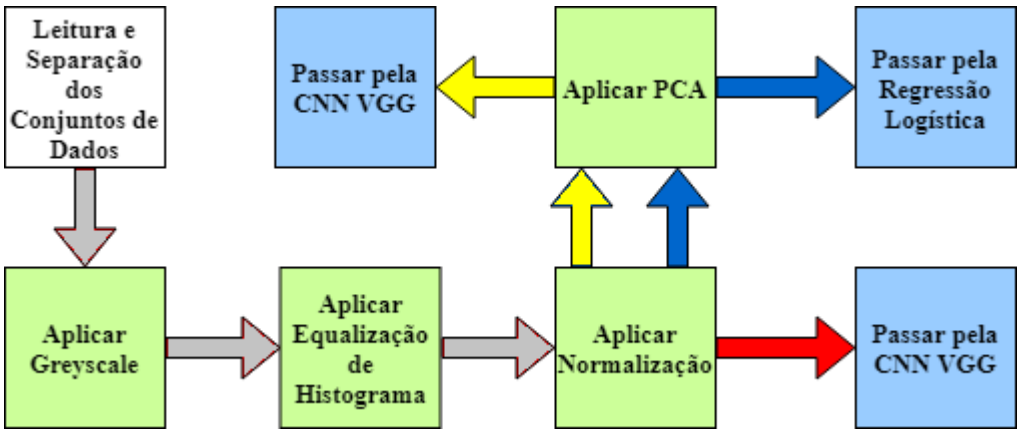

Figura 3.1: Diagrama dos Procedimentos. Cada seta indica a cor dos diferentes testes feitos nos modelos. A cor verde clara indica pré-processamento da imagem e a cor azul clara indica as ferramentas de medição de acurácia.

3.2 ACURÁCIA

Primeiramente, foi feito um teste de acurácia de validade ao comparar como os dados de treinamento foram adaptados pela rede neural para reconhecimento das características de cada placa. Tal teste foi feito comparando os dados de validação com os resultantes do treinamento.

3.2.1 Rede Convolucional

A rede convolucional usada foi a VGGnet, com a qual o teste de validação foi feito juntando ambos conjuntos de treinamento (90%) e validação (10%) e dividindo o conjunto resultante em novos conjuntos de treinamento e validação. Ambos os conjuntos de treinamento e validação foram convertidos em arquivos do tipo Pickle, que é uma extensão usada em Python para serialização e des-serialização de conjuntos de dados para facilitação da extração de componentes e de rótulos.

O resultado pode ser visto no gráfico a seguir:

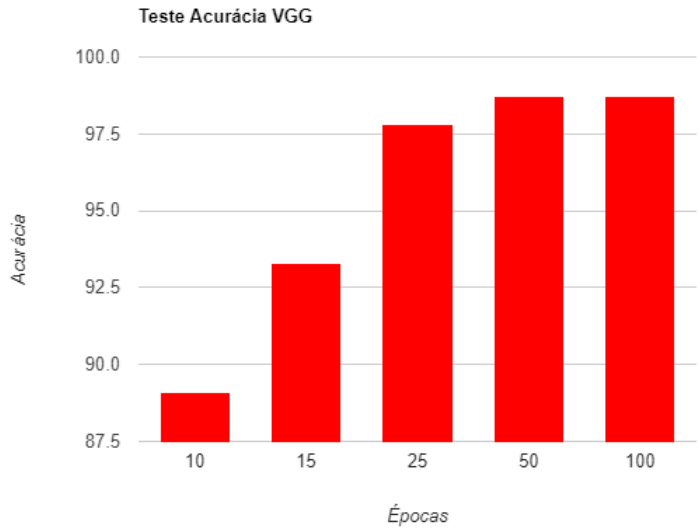

Figura 3.2: Acurácia do Modelo Convolucional VGG.

3.2.2 PCA

Em PCA, os dados de treinamento foram ortogonalmente transformados usando a ferramenta PCA do sklearn em Python, que faz parte do pacote Ski-kit Learn, específico para Machine Learning. Foi criado um conjunto de características (componentes principais) as quais contém 95% da *Explained Variance* [38] do conjunto, ao qual resultou em apenas 4 componentes principais. Após isso, foi feita uma regressão logística utilizando os dados de validação para aferir a acurácia do mesmo. O resultado pode ser visto no gráfico a seguir:

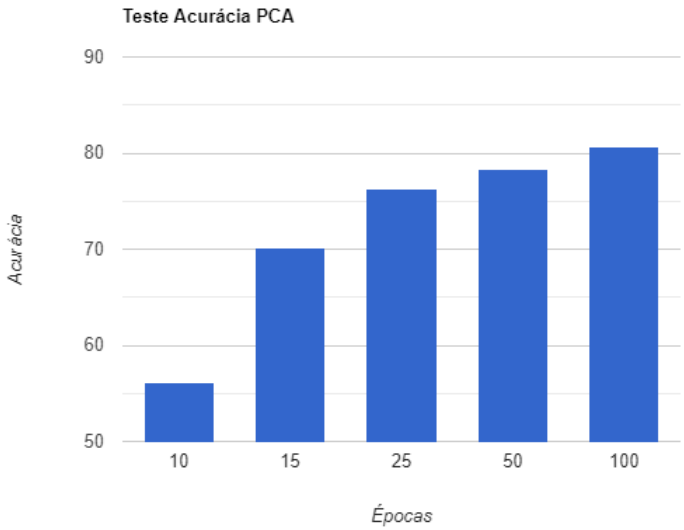

Figura 3.3: Acurácia do Modelo PCA - Regressão Logística.

3.2.3 PCA e VGG

Como teste final, foi incorporado à Rede Convolucional VGGnet o pré-processamento de PCA usado anteriormente, sem o uso da regressão logística para aferir a acurácia. O resultado foi o seguinte:

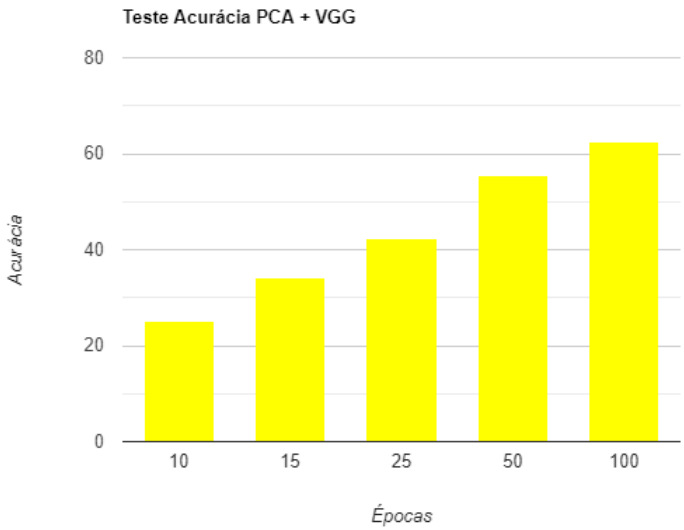

Figura 3.4: Acurácia de ambos modelos juntos.

3.3 ANÁLISE DOS RESULTADOS

Como podemos perceber pelos dados a seguir, obtivemos o melhor sucesso usando apenas a rede neural convolucional.

Tabela 3.1: Tabela de Resultados das porcentagens de acurácia de todos os três testes.

Épocas	**10**	**15**	**25**	**50**	**100**
PCA	56.1%	70.1%	76.3%	78.3%	80.6%
VGG	89.1%	93.3%	97.8%	98.7%	98.7%
PCA + VGG	25%	34.1%	42.2%	55.6%	62.6%

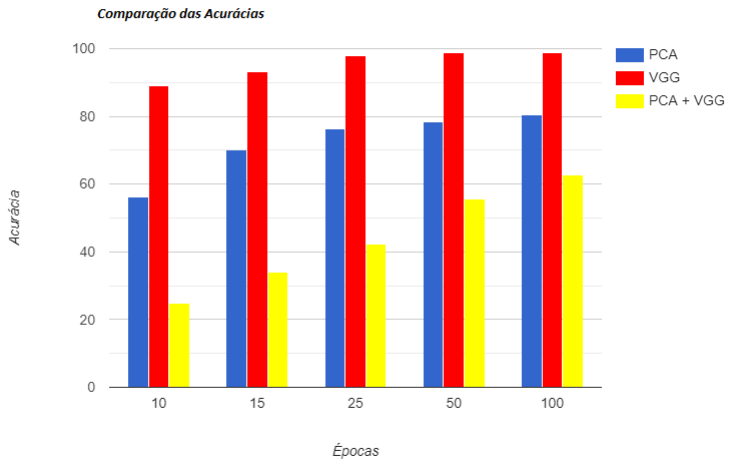

Figura 3.5: Todas as acurácias testadas.

E ao visualizar os dados da rede convolucional, podemos perceber que não houve um Overfitting, pois o valor da acurácia não diminuiu consideravelmente, apesar de já ter alcançado o seu pico em torno de 25 épocas.

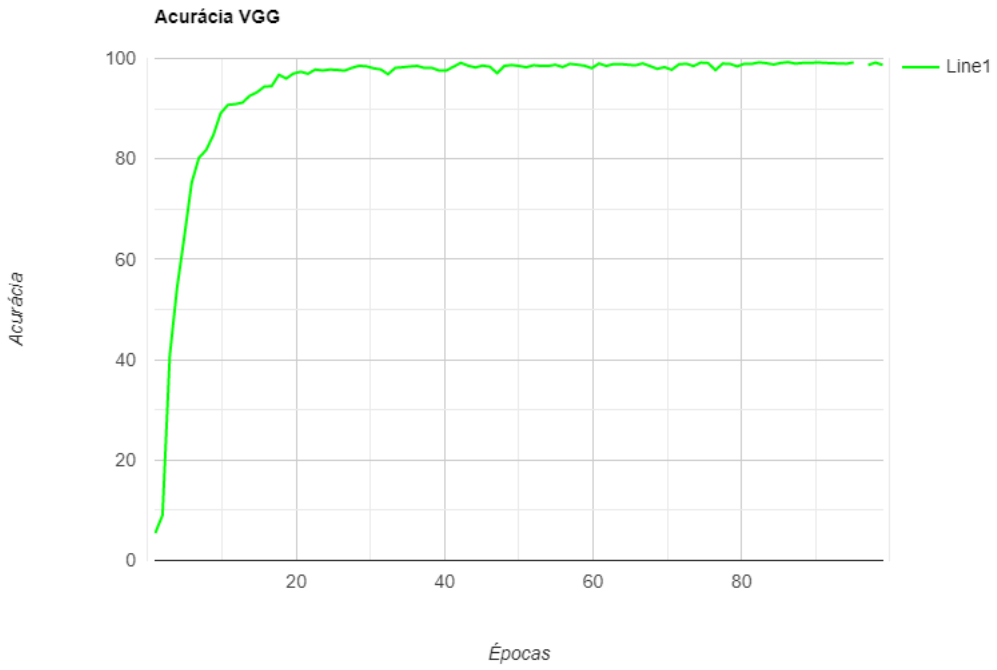

Figura 3.6: Teste detalhado do modelo VGG.

3.4 TESTE DE DESEMPENHO

Após serem vistas as acurácias de cada teste e decidindo o melhor método a ser usado, que no presente momento se mostrou ser apenas o da rede convolucional, alimentamos a rede com imagens, que a rede neural profunda ainda não viu, a fim de testar seu funcionamento.

As imagens de teste foram adquiridas através de um vídeo no Youtube de distribuição livre, onde Estevon Nagumo [19] gravou o trecho da ciclovia de Brasília. Após imagens serem recordadas do vídeo, foram objetivamente recortadas as imagens das placas e foram redimensionadas para encaixar na entrada da rede neural.

Figura 3.7: Exemplo de recorte e redimensionamento de uma placa contida no vídeo mencionado. Fonte: Estevon [19].

O método de reconhecimento empregado faz com que a rede neural leia uma imagem nova e faça uma estimativa de qual é o rótulo real da imagem, sendo a previsão final do modelo. A seguir, podemos ver a previsão do nosso modelo Convolucional VGGnet sobre 5 imagens adquiridas na ciclovia e ajustadas para encaixarem no programa. As imagens de teste de placas brasileiras foram colocadas como entrada do algoritmo da rede neural de placas alemãs, afim de atestar a acurácia da estimativa do programa. O programa irá fazer diversas estimativas e a que tiver o maior valor será a decisão final do programa, como pode ser visto à seguir.

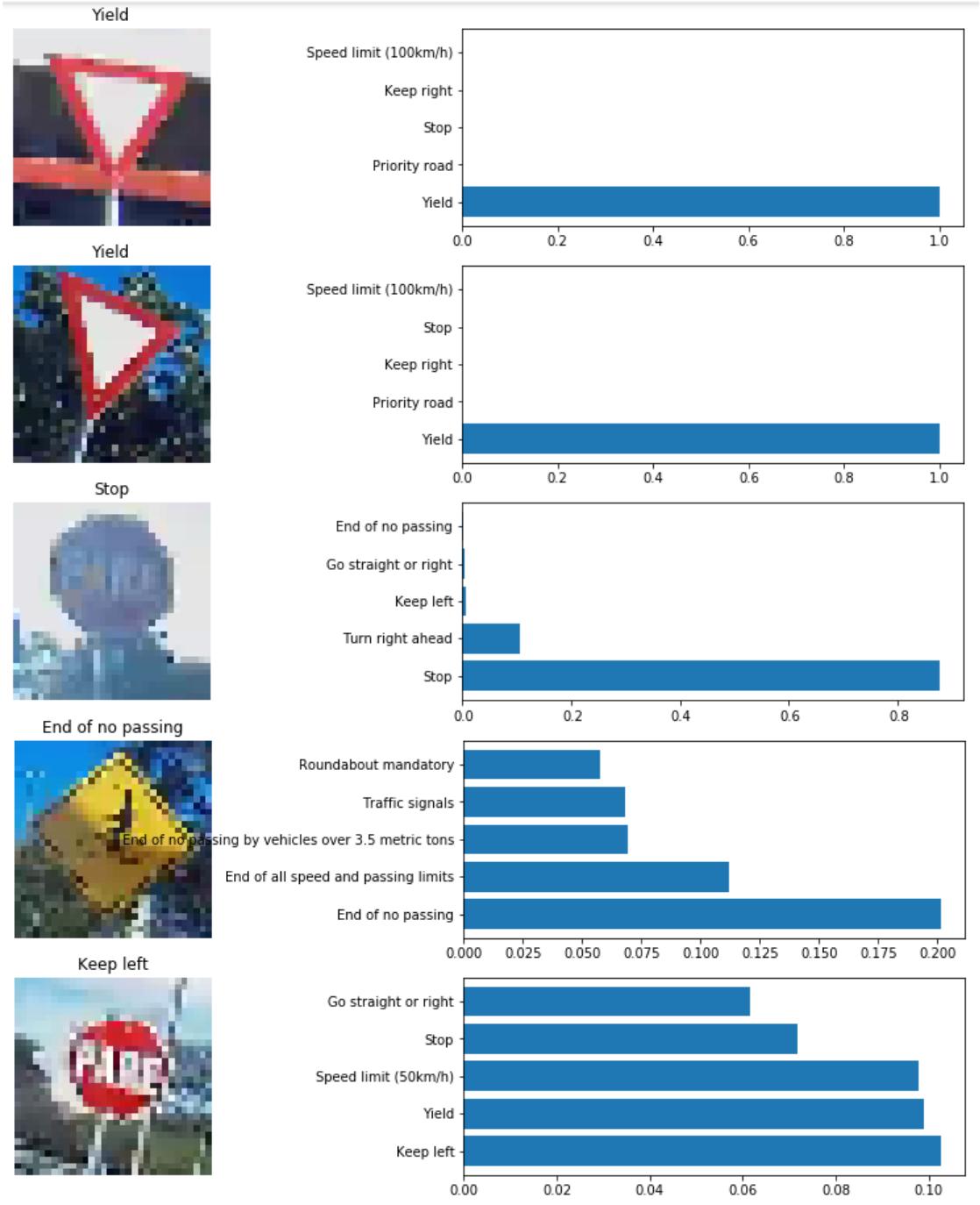

Figura 3.8: Teste de Performance feito com placas da ciclovia do DF.

Como podemos ver, a primeira placa teve 5 previsões:

Tabela 3.2: Tabela de resultados do teste com a primeira placa.

Previsões do Programa	**Maior estimativa do programa**	**Placa correta**	
Limite de Velocidade (100km/h)			Dê a Preferência
Mantenha-se à Direita			
Pare			
Rua Prioritária (especifica alemã)			
Dê a Preferência	100%	X	

A segunda placa teve as seguintes previsões:

Tabela 3.3: Tabela de resultados do teste com a segunda placa.

Previsões do Programa	**Maior estimativa do programa**	**Placa correta**	
Limite de Velocidade (100km/h)			Dê a Preferência
Pare			
Mantenha-se à Direita			
Rua Prioritária (especifica alemã)			
Dê a Preferência	100%	X	

A terceira placa teve as seguintes previsões:

Tabela 3.4: Tabela de resultados do teste com a terceira placa.

Previsões do Programa	**Maior estimativa do programa**	**Placa correta**	
Fim de Não-Ultrapassagem (específica alemã)			Pare
Siga Adiante ou à Direita			
Mantenha-se à Esquerda			
Vire à Direita			
Pare	90%	X	

A quarta placa teve as seguintes previsões:

Tabela 3.5: Tabela de resultados do teste com a quarta placa.

Previsões do Programa	**Maior estimativa do programa**	**Placa correta**	
Fim de Não-Ultrapassagem (específica alemã)			Travessia de Bicicletas
Sinais de Trânsito (específica alemã)			
Fim de Não-Ultrapassagem (específica alemã)			
Fim de limites de vel. e ultr. (específica alemã)			
Fim de Não-Ultrapassagem (específica alemã)	20%		

E, finalmente, a quinta placa teve as seguintes previsões:

Tabela 3.6: Tabela de resultados do teste com a quinta placa.

Previsões do Programa	**Maior estimativa do programa**	**Placa correta**	
Siga Adiante ou à Direita			Pare
Pare		X	
Limite de Velocidade (50km/h)			
Dê a Preferência			
Mantenha-se à Esquerda	10%		

O programa novamente fez muitas estimativas e acabou errando todas, fazendo com que em torno de 10% fosse o máximo com uma previsão incorreta.

O resultado final foi que o programa conseguiu identificar corretamente três das cinco placas apresentadas à ele, particularmente obtendo sucesso com a placa 'Dê a preferêcia', que se parece exatamente com a placa de mesma designação na Alemanha.

A primeira placa 'Pare' foi reconhecida, porém, as outras duas placas tiveram uma grande confusão de rótulos.

4 CONCLUSÃO E TRABALHOS FUTUROS

O presente trabalho foi um estudo de caso de uma rede convolucional (VGGnet) e como ela pode trabalhar em conjunto com uma forma de redução de dimensão (PCA), referente ao primeiro objetivo específico deste trabalho, e a tentativa de usar um conjunto de dados de placas de trânsito alemãs para serem aplicados em placas de trânsito brasileiras, objetivo este especificado como o segundo objetivo específico deste trabalho.

É indiscutível a evidência da escassez de regulações e sinalizações de trânsito em ciclovias, mas isso se dá à infraestrutura simplista das ciclovias do DF. Se as pessoas não fazem uso das ciclovias, não há por que torná-las algo mais do que lazer. Parte do objetivo deste trabalho foi criar uma expectativa de que, em um futuro próximo, ciclovias, com suas próprias regulações e sinalizações, possam fazer parte de um papel social maior, por exemplo, servir de transporte automatizado. O avanço da utilização de ciclovias culminará no interesse da própria população em usar tais ciclovias, pois servirão como uma melhor e mais segura alternativa do que dirigir carros.

Agora, quanto aos resultados do presente trabalho, mesmo que a acurácia de validação das placas de trânsito alemãs tenham sido consideravelmente boas, elas não serviram perfeitamente no reconhecimento de placas de trânsito brasileiras, o que já era esperado. É uma esperança notar, porém, que o modelo obteve algum sucesso em tal reconhecimento.

Mesmo que PCA seja uma boa ferramenta, neste exemplo, usado em conjunto com uma rede convolucional, resultou em um erro esperado, e uma alternativa mais simples e efetiva seria o uso de outras transformadas que funcionem bem em conjunto com redes convolucionais, como transformadas de wavelets.

4.1 TRABALHOS FUTUROS

Como forma de incorporar completamente a Engenharia Elétrica neste projeto, a discussão sobre trabalhos futuros pode ser embasada nos seguintes tópicos:

- Coleta e criação de um banco de dados brasileiro de placas de trânsito para maior sucesso do modelo neural;
- Construção rudimentar de um protótipo robótico que consiga se orientar em um ambiente cicloviário usando um melhor modelo neural;
- Proposta realista de um melhoramento das ciclovias do DF para incorporação robótica;

REFERÊNCIAS BIBLIOGRÁFICAS

1 HIGA, P. *Google mostra novo carro que dirige sozinho e não tem volante*. Acessado em 01/07/2019. <https://tecnoblog.net/157344/google-carro-autonomo-anuncio/>.

2 SHARMA, A. *What is the differences between artificial neural network (computer science) and biological neural network?* Acessado em 04/07/2019. <https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network>.

3 NIELSEN, M. A. *Neural Networks and Deep Learning*. 2015. 1-2 p.

4 TAKAHASHI, L. *What I learned in the 4-week Deep Learning course on Coursera*. Acessado em 04/07/2019. <https://becominghuman.ai/what-i-learned-in-the-4-week-deep-learning-course-on-coursera-1c142c2ce40a>.

5 LOPES, V. *Placas de trânsito e seus significados*. Acessado em 06/12/2018. <https://www.cursosdetransito.com.br/wp-content/uploads/2016/05/1394.jpg>.

6 MENEZES, T. *Peculiaridades do trânsito na Alemanha*. Acessado em 06/12/2018. <https://www.dw.com/image/19317946_303.jpg>.

7 AMEEN, M. *German Traffic Sign Classification Using TensorFlow. Benchmark*. Acessado em 06/12/2018. <https://github.com/mohamedameen93/German-Traffic-Sign-Classification-Using-TensorFlow>.

8 RASCHKA, S. *ARTIFICIAL NEURAL NETWORK (ANN) 7 - OVERFITTING REGULARIZATION*. <https://www.bogotobogo.com/python/scikit-learn/Artificial-Neural-Network-ANN-7-Overfitting-Regularization.php>.

9 CARTENET, R. *Which signals do indicate that the convolutional neural network is overfitted?* Acessado em 04/07/2019. <https://www.quora.com/Which-signals-do-indicate-that-the-convolutional-neural-network-is-overfitted>.

10 LECUN, Y.; BENGIO, Y. *Convolutional networks for images, speech, and time-series*. 1995. 5-8 p.

11 FROSSARD, D. *VGG in TensorFlow - Model and pre-trained parameters for VGG16 in TensorFlow*. Acessado em 04/07/2019. <https://www.cs.toronto.edu/~frossard/post/vgg16/>.

12 WIKIPEDIA. *Grayscale*. Acessado em 06/12/2018. <https://upload.wikimedia.org/wikipedia/commons/thumb/3/33/Beyoglu_4671_tricolor.png/400px-Beyoglu_4671_tricolor.png/>.

13 GIASSA. *II – Contrast Adjustment Image Normalization*. Acessado em 06/12/2018. <https://www.giassa.net/?page_id=472>.

14 ACHARYA, T.; RAY, A. K. *Image processing: Principles and Applications*. 2005. 111-113 p.

15 VANDERPLAS, J. *In Depth: Principal Component Analysis*. Acessado em 06/12/2018. <https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html>.

16 MALLICK, S. *EigenFace*. Acessado em 06/12/2018. <https://www.youtube.com/watch?v=J0arU2PAMls>.

17 GAZETATOLEDO. *Prefeitura investiga propaganda irregular em placas de sinalização*. Acessado em 06/12/2018. <https://www.gazetatoledo.com.br/NOTICIA/12659/PREFEITURA_INVESTIGA_PROPAGANDA_IRREGULAR_EM_PLACAS_DE_SINALIZACAO#.XRu_r-hKjIU>.

18 MCDONALD, C. *Machine learning fundamentals (I): Cost functions and gradient descent*. Acessado em 01/07/2019. <https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220>.

19 NAGUMO, E. *Ciclovia Asa Norte - SIG Brasília*. Acessado em 01/07/2019. <https://www.youtube.com/watch?v=m-RXb3Fh_fo>.

20 MARIAM, A. *Artificial Intelligence*. Acessado em 01/07/2019. <https://www.researchgate.net/publication/323498156_Artificial_Intelligence>.

21 PONTI, M. A.; COSTA, G. B. P. da. *Como funciona o Deep Learning*. Universidade de São Paulo, São Carlos, SP, Brasil: [s.n.].

22 PATTERSON, J.; GIBSON, A. *Deep Learning: A Practitioner's Approach*. 2017. 6 p.

23 PYTHON. *Learn more about Python*. Acessado em 01/07/2019. <https://www.python.org/about/>.

24 STALLKAMP, J. *Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition*. [S.l.]: Pergamon, Acessado em 01/07/2019. <http://www.sciencedirect.com/science/article/pii/S0893608012000457>.

25 SHAH, T. *About Train, Validation and Test Sets in Machine Learning*. Acessado em 01/07/2019. <https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7>.

26 GUPTA, T. *Deep Learning: Overfitting*. Acessado em 01/07/2019. <https://towardsdatascience.com/deep-learning-overfitting-846bf5b35e24>.

27 SAHU, A. *What is the VGG neural network?* Acessado em 01/07/2019. <https://www.quora.com/What-is-the-VGG-neural-network>.

28 SINHAL, K. *ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks*. Acessado em 01/07/2019. <https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/>.

29 BALA, R.; BRAUN, K. Color-to-grayscale conversion to maintain discriminability. 800 Phillips Rd, 128-27E, Webster, NY 14580, Acessado em 01/07/2019.

30 JONAS, G.; LUIZ, V. *Image Processing for Computer Graphics and Vision*. 1997. 61-62 p.

31 PATRO, S. G.; SAHU, K. K. Normalization: A preprocessing stage. *IARJSET*, 03 2015.

32 SHAMSIAN, A. *German-Traffic-Signs-Classification*. Acessado em 01/07/2019. <https://github.com/AvivSham/German-Traffic-Signs-Classification>.

33 GALARNYK, M. *PCA using Python (scikit-learn)*. Acessado em 01/07/2019. <https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60>.

34 HAYKIN, S. *Redes Neurais: princípios e prática*. Porto Alegre, Brasil: [s.n.], 2001. 429-433 p.

35 VARELLA, C. A. A. *Análise Multivariada Aplicada as Ciências Agrárias - Análise de Componentes Principais*. 2008.

36 MALLICK, S. *Eigenface using OpenCV (C /Python)*. Acessado em 01/07/2019. <https://www.learnopencv.com/eigenface-using-opencv-c-python/>.

37 PANT, A. *Introduction to Logistic Regression*. Acessado em 01/07/2019. <https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148>.

38 SCIKITLEARN. *sklearn.decomposition.PCA*. Acessado em 15/07/2019. <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>.

APÊNDICE

A APÊNDICE

A.1 PRIMEIRO APÊNDICE

Algoritmo de ordenação das imagens de placas de trânsito em ciclovias do DF. Essas imagens foram usadas para complementar o conjunto de dados de treinamento/validação.

```
# -*- coding: utf-8 -*-
"""Untitled38.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1zWC4B2OYJzCnLgvT6q
    Cvv3Gcq1fMeYnP
"""

import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
import pandas as pd

from google.colab import drive
drive.mount('/content/drive')
categories = ['Speed limit (20km/h)',
'Speed limit (30km/h)',
'Speed limit (50km/h)',
'Speed limit (60km/h)',
'Speed limit (70km/h)',
'Speed limit (80km/h)',
'End of speed limit (80km/h)',
'Speed limit (100km/h)',
'Speed limit (120km/h)',
'No passing',
'No passing for vehicles over 3.5 metric tons',
'Right-of-way at the next intersection',
'Priority road',
```

```
'Yield',
'Stop',
'No vehicles',
'Vehicles over 3.5 metric tons prohibited',
'No entry',
'General caution',
'Dangerous curve to the left',
'Dangerous curve to the right',
'Double curve',
'Bumpy road',
'Slippery road',
'Road narrows on the right',
'Road work',
'Traffic signals',
'Pedestrians',
'Children crossing',
'Bicycles crossing',
'Beware of ice/snow',
'Wild animals crossing',
'End of all speed and passing limits',
'Turn right ahead',
'Turn left ahead',
'Ahead only',
'Go straight or right',
'Go straight or left',
'Keep right',
'Keep left',
'Roundabout mandatory',
'End of no passing',
'End of no passing by vehicles over 3.5 metric tons']

datadir = '/content/drive/My Drive/traffic-signs-data/new_test_images/'

print(os.path.isdir(datadir))
for category in categories:
    path = os.path.join(datadir, category)
    if os.path.exists(path):
      for img in os.listdir(path):
          img_array = cv2.imread(os.path.join(path,img))
          print(img_array)
```

```
            plt.imshow(img_array, cmap="gray")
            plt.show()
            break
    else:
      break
    break

from tqdm import tqdm
training_data = []

IMG_SIZE = 32

def create_training_data():
    for category in categories:

        path = os.path.join(datadir,category)
        class_num = categories.index(category)

        if os.path.exists(path):
          for img in tqdm(os.listdir(path)):
            try:
                img_array = cv2.imread(os.path.join(path,img) ,
                cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(img_array, (IMG_SIZE,
                IMG_SIZE))
                training_data.append([new_array, class_num])
            except Exception as e:
                pass
        else:
          pass

create_training_data()

import random

random.shuffle(training_data)

x = []
y = []
for features,label in training_data:
```

```
    x.append(features)
    y.append(label)

print(len(x))
print(len(y))
print(np.array(x).shape)
x = np.array(x).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
y = np.array(y).reshape(-1)
print(x.shape)
print(y.shape)

print(y)
print(y[1])
print(len(y))
df = pd.DataFrame(index = ('0','1','2','3','4','5','6','7',
'8','9','10','11','12','13','14','15','16','17','18','19',
'20','21','22'), columns=('0','1','2','3','4','5','6','7','8',
'9','10','11','12','13','14','15','16',
'17','18','19','20','21','22','23','24',
'25','26','27','28','29','30','31','32',
'33','34','35','36','37','38','39','40',
'41','42'), dtype = float)
df.fillna(0, inplace=True)
for i in range(0, len(y)):
  df.iloc[i,y[i]] = 1.0
df

import pickle

pickle_out = open("/content/drive/My␣Drive/traffic-signs-data/
X_train_new.p","wb")
pickle.dump(x, pickle_out)
pickle_out.close()

pickle_out = open("/content/drive/My␣Drive/traffic-signs-data/
y_train_new.p","wb")
pickle.dump(df, pickle_out)
pickle_out.close()

pickle_in = open("/content/drive/My␣Drive/traffic-signs-data/
```

```
X_train_new.p","rb")
x = pickle.load(pickle_in)

pickle_in = open("/content/drive/My Drive/traffic-signs-data/
y_train_new.p","rb")
df = pickle.load(pickle_in)
```

A.2 SEGUNDO APÊNDICE

Este segundo algoritmo foi usado para aplicar PCA antes do modelo convolucional VGG, e também foi adaptado para aplicar apenas VGG ao banco de dados. O algoritmo pertence à Aviv Shamsian, incluindo algumas modificações para adaptação do PCA.

```
# -*- coding: utf-8 -*-
"""PCA_VGG New.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1CEsHVnFdPR_fyFAJyDAx
    __CttGLZcuGK

<a href="https://colab.research.google.com/github/AvivSham/
German-Traffic-Signs-Classification/blob/master/German_
TrafficSigns_Classification.ipynb" target="_parent"><img
src="https://colab.research.google.com/assets/colab-badge.svg"
alt="Open In Colab"/></a>
"""

!wget https://d17h27t6h515a5.cloudfront.net/topher/2017/
February/5898cd6f_traffic-signs-data/traffic-signs-data.zip
!wget https://github.com/AvivSham/German-Traffic-Signs-
Classification/blob/master/signnames.csv

!unzip traffic-signs-data.zip

# %matplotlib inline
import os, pickle, shutil
import numpy as np
```

```
from skimage.io import imread
import skimage.morphology as morp
from skimage.filters import rank
from sklearn.utils import shuffle, compute_class_weight
from sklearn.metrics import confusion_matrix
import csv
import cv2
import matplotlib.pyplot as plt
from keras.models import Input, Model
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten,
Dense
from keras.utils import to_categorical
from keras import optimizers
from keras.initializers import random_normal
from keras.callbacks import Callback, ReduceLROnPlateau,
ModelCheckpoint
import seaborn as sn
from sklearn.metrics import confusion_matrix

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

np.random.seed(seed=42)

training_file = '/content/drive/My␣Drive/traffic-signs-data/train.p'
validation_file = '/content/drive/My␣Drive/traffic-signs-data/valid.p'
testing_file = '/content/drive/My␣Drive/traffic-signs-data/test.p'
file_img = '/content/drive/My␣Drive/traffic-signs-data/X_train_new.p'
file_label = '/content/drive/My␣Drive/traffic-signs-data/y_train_new.p'

with open(training_file, mode='rb') as f:
  train = pickle.load(f)
with open(validation_file, mode='rb') as f:
  valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
  test = pickle.load(f)
with open(file_img, mode='rb') as f:
  new_file_img = pickle.load(f)
with open(file_label, mode='rb') as f:
  new_file_label = pickle.load(f)
```

```
signs_classes = []
with open('/content/drive/My␣Drive/traffic-signs-data/signnames.csv',
'r') as csvfile:
    signnames = csv.reader(csvfile, delimiter=',')
    next(signnames,None)
    for row in signnames:
        signs_classes.append(row[1])
    csvfile.close()

X_train, Y_train = train['features'], train['labels']
X_valid, Y_valid = valid['features'], valid['labels']
X_test, Y_test = test['features'], test['labels']
n_classes = len(np.unique(Y_train))

print("Number␣of␣train␣samples:␣", X_train.shape[0])
print("Number␣of␣validation␣samples:␣", X_valid.shape[0])
print("Number␣of␣test␣samples:␣", X_test.shape[0])
print("Number␣of␣classses:␣", n_classes)
print(X_train.shape)
print(new_file_img.shape)
print(new_file_label.shape)

X_train, Y_train = shuffle(X_train, Y_train)

def convert_to_gray(image):

  return cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

gray_images = list(map(convert_to_gray,X_train))
np.shape(gray_images)

def hist_equalization(image):
  kernel = morp.disk(30)
  return rank.equalize(image, selem=kernel)

def adapt_hist_equalization(image,clahe = cv2.createCLAHE(clipLimit=
2.0, tileGridSize=(2,2))):
  return clahe.apply(image)
```

```
equalizied_gray_images = list(map(hist_equalization,gray_images))

def norm_image(data):

  normalized_images = np.array(data,dtype = np.float32)/255
  return np.expand_dims(normalized_images, axis=-1)

def preprocess(x_data,y_data, n_classes = 43):
  gray_images = list(map(convert_to_gray,x_data))
  hist_equal_images = list(map(adapt_hist_equalization,gray_images))
  norm_images = norm_image(hist_equal_images)
  y_data = to_categorical(y_data, n_classes)
  return norm_images, y_data

def ConvBlock(model, pool, n_filters, mu, sigma):
  model = Conv2D(n_filters, kernel_size = 2, padding = 'same',
                 activation = 'relu',
                 kernel_initializer = random_normal(mean = mu,stddev =
                 sigma))(model)

  model = Conv2D(n_filters, kernel_size = 2, padding = 'same',
                 activation = 'relu',
                 kernel_initializer = random_normal(mean = mu,stddev =
                 sigma))(model)
  model = MaxPooling2D(pool_size = 2, strides = 2, padding = 'valid')(model)
  model = Dropout(0.5)(model)
  return model

def VGG_variation(input_shape,nf=32):

  inputs = x = Input(input_shape)
  for i in range(3):
    x = ConvBlock(x,pool = True, n_filters = nf * (i+2), mu = 0,
    sigma = 0.1)

  x = Flatten()(x)
  for _ in range(2):
    x = Dense(units = 128, activation = 'relu')(x)
  output = Dense(units = 43, activation = 'softmax')(x)
  VGG_var_model = Model(inputs = inputs, outputs = output)
```

```
    opti = optimizers.Adam(0.0001)
    VGG_var_model.compile(optimizer = opti, loss =
    'categorical_crossentropy',
                          metrics =
                          ['accuracy','categorical_crossentropy'])
    VGG_var_model.summary()
    return VGG_var_model

class My_Callback(Callback):
    def on_train_begin(self, logs={}):
        print("train␣begins!")
        return

    def on_train_end(self, logs={}):
        return

    def on_epoch_begin(self, epoch, logs={}):
        return

    def on_epoch_end(self, epoch, logs={}):
        print("-", end='')
        flag_val=True
        if epoch%5==0:
            train_acc = logs.get("acc")
            train_loss = logs.get("loss")
            try:
                val_acc = logs.get("val_acc")
                val_loss = logs.get("val_loss")
            except:
                flag_val=False
            if flag_val:
                print("\n%d"%epoch, "\ttrain_loss:␣", train_loss,
                      "\tval_loss:␣", val_loss, "\ttrain_acc:",

                train_acc,
                      "\tval_acc:", val_acc)
            else:
                print("\n%d"%epoch, "\ttrain_loss:␣",
                      train_loss, "\ttrain_acc:", train_acc)
        return
```

```
    def on_batch_begin(self, batch, logs={}):
        return

    def on_batch_end(self, batch, logs={}):
        return

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.3,
patience=10,
                              verbose=1, mode='auto', min_lr=1e-12)
my_callback = My_Callback()

checkpoint = ModelCheckpoint("VGG_GermanSigns_classification.h5",
monitor='loss',
                             verbose=0, save_best_only=True,
                             save_weights_only=True)

X_train_processed, Y_train_cat = preprocess(X_train,Y_train)
X_valid_processed, Y_valid_cat = preprocess(X_valid, Y_valid)

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd

X_train_processed = X_train_processed.reshape(34799,32,32)
nsamples, nx, ny = X_train_processed.shape
X_train_processed = X_train_processed.reshape((nsamples,nx*ny))

X_valid_processed = X_valid_processed.reshape(4410,32,32)
n2samples, n2x, n2y = X_valid_processed.shape
X_valid_processed = X_valid_processed.reshape((n2samples,n2x*n2y))

new_file_img = new_file_img.reshape(23,32,32)
n4samples, n4x, n4y = new_file_img.shape
new_file_img = new_file_img.reshape((n4samples,n4x*n4y))

pca = PCA(n_components = 144)

X_trainvalid = pd.concat([pd.DataFrame(X_train_processed),
pd.DataFrame(X_valid_processed)], axis=0)
```

```
X_trainvalid = pd.concat([X_trainvalid,pd.DataFrame(new_file_img)],
axis=0)

X_train_processed = pca.fit_transform(X_train_processed)
X_valid_processed = pca.fit_transform(X_valid_processed)
X_trainvalid = pca.fit_transform(X_trainvalid)

X_train_processed = X_train_processed.reshape(34799,12,12)
X_train_processed = X_train_processed[..., None]

X_trainvalid = X_trainvalid.reshape(39232,12,12)
X_trainvalid = X_trainvalid[..., None]

X_valid_processed = X_valid_processed.reshape(4410,12,12)
X_valid_processed = X_valid_processed[..., None]

y_trainvalid = pd.concat([pd.DataFrame(Y_train_cat)
, pd.DataFrame(Y_valid_cat)])

y_trainvalid.columns = [str(x) for x in y_trainvalid.columns]
new_file_label.columns = [str(x) for x in new_file_label.columns]
y_trainvalid2 = pd.concat([y_trainvalid, new_file_label],
ignore_index=True)



VGG_model = VGG_variation(X_trainvalid.shape[1:])
batch_size = 12
epochs = 100
weights = compute_class_weight('balanced',classes =
np.unique(Y_train),y = Y_train)

model_history = VGG_model.fit(X_trainvalid, y_trainvalid2,
batch_size=batch_size, epochs=epochs,
                    validation_split=0.1,shuffle=True,
                    callbacks = [my_callback, reduce_lr, checkpoint],
                    verbose=0,
                    class_weight = None)
```

A.3 TERCEIRO APÊNDICE

Finalmente o terceiro algoritmo foi usado para regressão logística do modelo usando apenas PCA. O algoritmo original pertence a Michael Galarnyk, feitas algumas alterações para importar os conjuntos de dados.

```
# -*- coding: utf-8 -*-
"""PCA.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/10DdPLODj
    -RgOzW07akFGtCJQ2C8gP9Fq
"""

import copy
import pickle
import numpy as np
import matplotlib.pyplot as plt
import random
import cv2
import skimage.morphology as morp
from skimage.filters import rank
from sklearn.utils import shuffle
import csv
import os
import tensorflow as tf
from tensorflow.contrib.layers import flatten
from sklearn.metrics import confusion_matrix

print(tf.test.gpu_device_name())

tf.__version__

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

import pandas as pd
```

```
import numpy as np
np.set_printoptions(suppress=True)

import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

# %matplotlib inline

training_file = '/content/drive/My␣Drive/traffic-signs-data/train.p'
validation_file = '/content/drive/My␣Drive/traffic-signs-data/valid.p'
testing_file = '/content/drive/My␣Drive/traffic-signs-data/test.p'
file_img = '/content/drive/My␣Drive/traffic-signs-data/X_train_new.p'
file_label = '/content/drive/My␣Drive/traffic-signs-data/y_train_new.p'

with open(training_file, mode='rb') as f:
  train = pickle.load(f)
with open(validation_file, mode='rb') as f:
  valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
  test = pickle.load(f)
with open(file_img, mode='rb') as f:
  new_file_img = pickle.load(f)
with open(file_label, mode='rb') as f:
  new_file_label = pickle.load(f)

signs = []
with open('drive/My␣Drive/traffic-signs-data/signnames.csv',
'r') as csvfile:
    signnames = csv.reader(csvfile, delimiter=',')
    next(signnames,None)
    for row in signnames:
        signs.append(row[1])
    csvfile.close()

X_train, train_lbl = train['features'], train['labels']
X_valid, valid_lbl = valid['features'], valid['labels']
X_test, test_lbl = test['features'], test['labels']
```

```
train_lbl2 = copy.deepcopy(train_lbl)
n_train = X_train.shape[0]

n_test = X_test.shape[0]

n_validation = X_valid.shape[0]

image_shape = X_train[0].shape

n_classes = len(np.unique(train_lbl))

print("Number␣of␣training␣examples:␣", n_train)
print("Number␣of␣testing␣examples:␣", n_test)
print("Number␣of␣validation␣examples:␣", n_validation)
print("Image␣data␣shape␣=", image_shape)
print("Number␣of␣classes␣=", n_classes)
label = [13, 14, 13, 29, 29, 13, 29, 14, 13, 14, 14, 13, 14,
29, 13, 29, 14, 14, 29, 14, 13, 13, 29,]

def gray_scale(image):
    return cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

def preprocess2(data):

    n_training = data.shape
    gray_images = np.zeros((n_training[0], n_training[1],
    n_training[2]))
    for i, img in enumerate(data):
        gray_images[i] = gray_scale(img)
    gray_images = gray_images[..., None]
    return gray_images

def local_histo_equalize(image):
    kernel = morp.disk(30)
    img_local = rank.equalize(image, selem=kernel)
    return img_local

def image_normalize(image):
    image = np.divide(image, 255)
    return image
```

```
def preprocess(data):

    gray_images = list(map(gray_scale, data))
    equalized_images = list(map(local_histo_equalize, gray_images))
    n_training = data.shape
    normalized_images = np.zeros((n_training[0], n_training[1],
    n_training[2]))
    for i, img in enumerate(equalized_images):
        normalized_images[i] = image_normalize(img)
    normalized_images = normalized_images[..., None]
    return normalized_images

train_img = preprocess(X_train)
train_img = train_img.reshape(34799,32,32)
nsamples, nx, ny = train_img.shape
train_img = train_img.reshape((nsamples,nx*ny))

new_img = new_file_img.reshape(23,32,32)
n4samples, n4x, n4y = new_img.shape
new_img = new_img.reshape((n4samples,n4x*n4y))

valid_img = preprocess(X_valid)
valid_img = valid_img.reshape(4410,32,32)
n2samples, n2x, n2y = valid_img.shape
valid_img = valid_img.reshape((n2samples,n2x*n2y))

pca_train = pd.concat([pd.DataFrame(new_img),
pd.DataFrame(train_img)], axis=0)
pca_label = pd.concat([pd.DataFrame(train_lbl),
pd.DataFrame(label)], ignore_index=True)

pca = PCA(.95)

pca.fit(pca_train)

from sklearn.linear_model import LogisticRegressionCV

logisticRegr = LogisticRegressionCV(cv=5,solver='saga',max_iter=50,
random_state=0,multi_class='multinomial')
```

```
logisticRegr.fit(pca_train, pca_label)

logisticRegr.predict(valid_img[0].reshape(1,-1))

logisticRegr.predict(valid_img[0:10])

score = logisticRegr.score(valid_img, valid_lbl)
print(score)
```