Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Relatório de Estágio Integrado
Vsoft Tecnologia

Leo de Lima Araújo

Campina Grande - PB
Janeiro de 2020

Leo de Lima Araújo

Relatório de Estágio Integrado
Vsoft Tecnologia

Relatório de Estágio Integrado submetido à Coordenação de Curso de Graduação de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Elétrica.

Área de Concentração: Processamento de Imagens.

Orientadora: Prof. Luciana Ribeiro Veloso, Dr. Sc

Campina Grande - PB

Janeiro de 2020

Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Departamento de Engenharia Elétrica - DEE
Projeto de Engenharia Elétrica

Leo de Lima Araújo

Relatório de Estágio Integrado
Vsoft Tecnologia

Trabalho aprovado em: Campina Grande - PB, / /

Leo de Lima Araújo
Aluno

Luciana Ribeiro Veloso, Dr. Sc
Professora Orientadora

Campina Grande - PB

Janeiro de 2020

Resumo

O presente relatório descreve as atividades realizadas pelo estagiário Leo de Lima Araújo, estudante de Engenharia Elétrica pela Universidade Federal de Campina Grande, durante o estágio na empresa Vsoft Tecnologia entre 09 de Setembro de 2019 e 03 de Janeiro de 2020. O estágio foi realizado no setor de pesquisa da empresa e dois projetos na área de Visão Computacional foram desenvolvidos, o primeiro foi a concepção de um sistema de inspeção veicular utilizando *Deep Learning* e o segundo foi o desenvolvimento de uma aplicação capaz de realizar a segmentação automática de fichas decadactilares utilizando técnicas clássicas de processamento digital de imagem. Ambos os projetos resultaram na produção de protótipos, sendo o primeiro em forma de aplicação *Web* e a segunda em forma de um aplicativo *Mobile* para *Android*.

Palavras-chave: Vsoft, Visão Computacional, *Deep Learning*, *MobileNet*, Processamento Digital de Imagem.

Abstract

This report describes the activities executed by the intern Leo de Lima Araújo, student of Electrical Engineering at Universidade Federal de Campina Grande, during the internship performed at the company Vsoft Tecnologia between Semptember 9th of 2019 and January 1st of 2020. The internship was carried out in the research sector of the company and two Computer Vision projects were developed, a vehicle inspection system utilizing Deep Learning and an application capable of performing the automatic segmentation of fingerprint files utilizing classic image processing techniques. Both projects were resolved with the production of a prototype, being respectively, in form of a Web aplication and a Mobile App designed for Android.

Keywords: Vsoft, Computer Vision, *Deep Learning*, *MobileNet*, Digital Image Processing.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ICAO	*International Civil Aviation Organisation*
RGB	*Red, Green, Blue*
HSV	*Hue, Saturation, Value*
CNN	*Convolutional Neural Network*
CPU	*Central Processing Unit*
GPU	*Graphics Processing Unit*
LReLU	*Leaky Rectified Linear Unit*
ReLU	*Rectified Linear Unit*
RNA	Rede Neural Artificial
VGG	*Visual Geometry Group*
IoU	*Intersection Over Union*

Sumário

1 Introdução

Este relatório tem por objetivo a descrição das atividades desempenhadas pelo aluno Leo de Lima Araújo durante o período de estágio integrado realizado no setor de Pesquisa da empresa Vsoft Tecnologia, sob supervisão do líder de pesquisa, Fábio Falcão de França. O estágio foi desempenhado entre os dias 09/09/2019 e 03/01/2020 com carga horária de 40 horas semanais, totalizando 668 horas de serviço.

1.1 Vsoft Tecnologia

A Vsoft Tecnologia é uma empresa que atua no segmento de software, especializada em Identificação Biométrica, e produz soluções utilizando processamento digital de imagens e inteligência artificial. A empresa foi fundada em 2000 e seus principais produtos lidam com o reconhecimento de faces e impressões digitais.

Além da busca pelo aprimoramento dos algoritmos atuais de reconhecimento biométrico, a equipe de pesquisa se divide para desenvolver demandas diversas encaminhadas pela empresa. Alguns dos projetos em andamento são a validação de face no padrão ICAO (*International Civil Aviation Organisation*), o monitoramento de pessoas em sala de aula, o monitoramento de pessoas no interior de veículos e a detecção de semáforos e placas.

A referida empresa utiliza como ferramenta de gerenciamento de projetos o Azure DevOps, que mantem a organização de todos os projetos. Tarefas são atribuídas através dessa plataforma e cabe aos pesquisadores a documentação do seu progresso. Reuniões diárias são realizadas para promover atualizações sobre o andamento dos projetos.

1.2 Objetivos

O foco principal do estágio foi o desenvolvimento de algoritmos de processamento digital de imagens utilizando visão computacional e inteligência artificial, a exemplo de *Deep Learning*. As principais atividades desempenhadas durante o estágio foram:

- Construção de bancos de imagem para o treinamento de redes convolucionais;
- Desenvolvimento de arquiteturas de redes convolucionais utilizando Keras;
- Formulação de algoritmos para processamento dos dados com OpenCV e Python;
- Desenvolvimento de um sistema de detecção e segmentação automática de fichas decadactilares implementado em C++ utilizando técnicas de visão computacional.

1.3 Estrutura do Relatório

O restante deste trabalho será organizado conforme o modelo descrito a seguir:

- Capítulo 2: Fundamentação teórica abordando conceitos importantes de *Deep Learning* e Visão Computacional.
- Capítulos 3 e 4: Descrição das atividades realizadas durante o estágio em conjunto com os resultados obtidos.
- Capítulo 5: Conclusões do trabalho em conjunto com as considerações finais.

2 Fundamentação Teórica

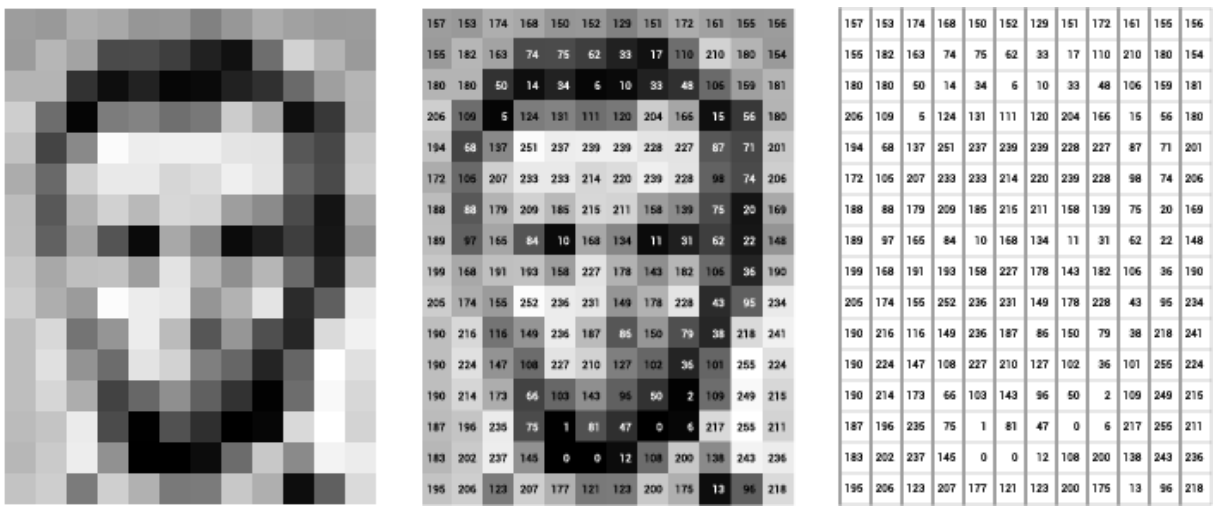
2.1 Espaços de Cores

Um espaço de cores é um modelo matemático capaz de descrever cores a partir de conjuntos de, em geral três ou quatro, variáveis, (STOKES et al., 1996), permitindo que as cores sejam mensuradas e especificadas quantitativamente, (TOET, 1992). Enquanto que para imagens coloridas mais canais podem ser necessários, imagens em escala de cinza só precisam de um único canal, que indica a intensidade de cada *pixel*.

Como pode ser visto na Figura 1, em uma imagem em escala de cinza de dimensões $H \times W$ cada um de seus *pixels* é representado por um único valor, usualmente um inteiro entre 0 e 255. Nessas imagens, a cor preta é comumente representada pelo valor 0 e a intensidade vai gradativamente se tornando mais clara à medida que o valor aumenta, se tornando branca ao atingir o valor máximo, 255. Em fotografias em escala de cinza a intensidade dos pixels corresponde à intensidade da luz mensurada pela câmera nos pixels em questão.

Os diferentes espaços de cores existentes servem para descrever esses conjuntos de valores, fornecendo uma interpretação para cada um deles. Alguns exemplos são os espaços RGB, empregado na maioria dos monitores, e HSV, que é uma alternativa interessante para a detecção de objetos com base em sua cor.

Figura 1 – Exemplo de uma imagem em nível de cinza e sua representação em uma matriz de pixels.



Fonte: `https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html`.

2.1.1 RGB

O sistema RGB (*Red, Green, Blue*) utiliza misturas de diferentes quantidades de vermelho, verde e azul para produzir diferentes tonalidades de cores, sendo a sigla derivada das iniciais das cores utilizadas, em inglês. Em uma imagem digital colorida no sistema RGB, um pixel pode ser entendido como um vetor cujas componentes representam as intensidades de vermelho, verde e azul de sua cor.

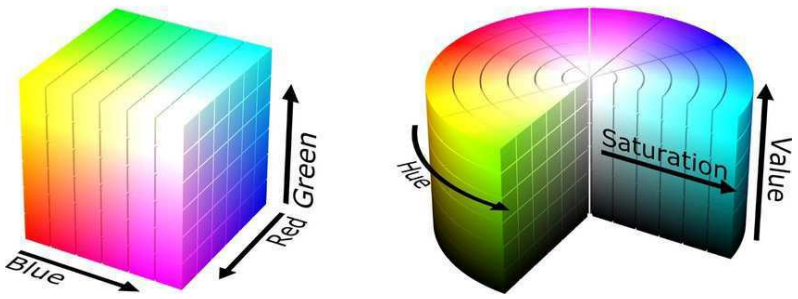
A mistura de cores no espaço RGB pode ser melhor compreendida ao pensar na interação entre luzes monocromáticas das cores referidas, uma vez que as luzes se misturam de maneira similar, o que torna o espaço RGB apropriado para o uso em dispositivos emissores de luz, como monitores e televisores. A porção à esquerda da Figura 2 mostra uma representação em um espaço tridimensional de como misturas de quantidades distintas de vermelho, azul e verde podem produzir as demais cores do espectro visível.

2.1.2 Espaço HSV

Um espaço de cores interessante para aplicações de segmentação e detecção é o HCV (tonalidade (*hue*), saturação (*chroma*), valor (*value*)). A tonalidade indica a matiz da cor, ou seja, sua coloração em seu estado puro, a saturação fornece uma indicação da pureza da cor, de modo que cores com baixa saturação são mais acinzentadas, e o valor, indica o brilho com relação a um ponto branco em condições semelhantes de iluminação.

Nesse espaço, descrito em coordenadas cilíndricas, a tonalidade é representada por um ângulo entre 0 e 360 graus, a saturação representa a distância entre o ponto em questão e o eixo de valor, que se extende na vertical partindo do preto em direção ao branco. Uma propriedade conveniente de se codificar cores desta forma é a possibilidade de localizar elementos de uma determinada cor simplesmente escolhendo uma faixa de valores de tonalidade, simplificando as tarefas de detecção.

Figura 2 – Representações dos Espaços RGB, à esquerda, e HSV, à direita.



Fonte: `https://medium.com/neurosapiens/segmentation-and-classification-with-hsv-8f2406c62b39`.

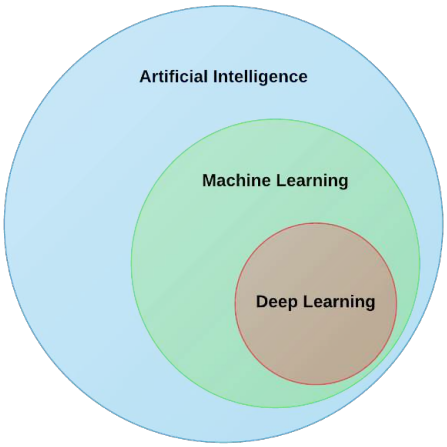
2.2 Deep Learning

Aprendizado de máquina (*Machine Learning*) é o campo de estudo que proporciona às máquinas a habilidade de aprender sem que elas sejam explicitamente programadas, (SAMUEL, 1959). Alguns dos métodos inclusos neste conjunto são *K Nearest Neighbors Classifier*, *Suport Vector Machine*, *Decision Tree Classifier* e *Random Forest Classifier*. Como pode ser visto na Figura 3, as técnicas de aprendizado de máquina são um dos métodos de Inteligência Artificial.

As técnicas de aprendizado profundo (*Deep Learning*) são um subcampo de aprendizado de máquina, onde o aprendizado se dá a partir de dados por meio de sucessivas camadas de níveis crescentes de abstração, (FRANCOIS, 2017). O uso de múltiplas camadas permite o aprendizado de funções que muito dificilmente poderiam ser descritas por programação explícita. Neste conjunto estão inclusas redes neurais artificiais, redes neurais convolucionais, redes neurais recorrentes, dentre outras arquiteturas.

Comumente, modelos de *Deep Learning* fazem uso de um tipo específico de aprendizado de máquina denominado aprendizado supervisionado. Neste tipo de aprendizado os dados de treinamento que são repassados ao algoritmo incluem as saídas desejadas, denominadas rótulos, (GÉRON, 2017). Utilizando esses dados, redes neurais e redes convolucionais conseguem regular seus parâmetros de modo a produzir as saídas desejadas a partir das entradas fornecidas.

Figura 3 – Hierarquia entre Deep Learning, Machine Learning e Inteligência artificial.



Fonte: `https://steemit.com/ai/@quantum-cyborg/what-is-deep-learning-how-does-it-differ-from-most-of-ai`

2.3 Redes Neurais Artificiais

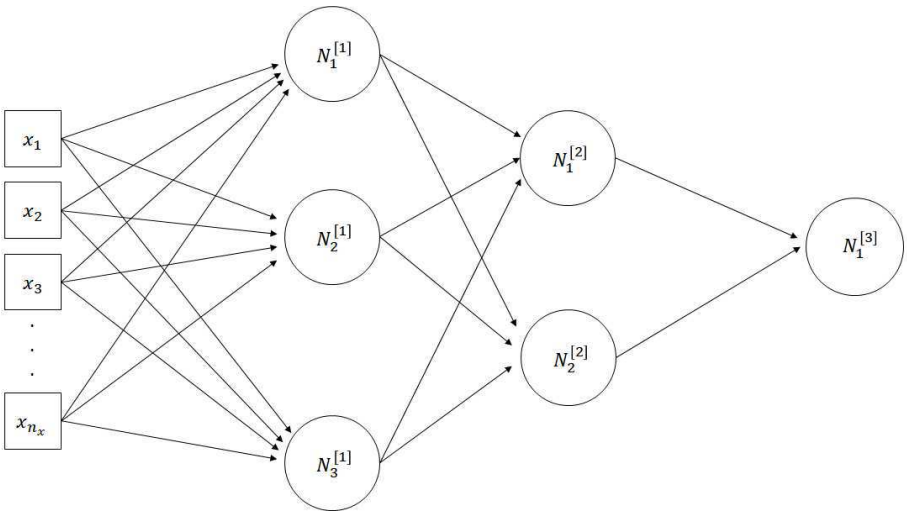
Redes Neurais Artificiais (RNA) podem ser definidas como um conjunto de unidades básicas de processamento, altamente conectadas, que processam a informação contida nas entradas fornecidas e produzem as saídas desejadas, (KHAN et al., 2018). As redes devem ser treinados a partir de grandes bancos de dados de modo a aprimorar a qualidade das saídas, podendo mapear de forma coerente funções não lineares de alta complexidade.

Um modelo de uma rede neural pode ser visto na Figura 4. Olhando da esquerda para a direita, a primeira camada é a camada de entrada, a última é a camada de saída e as demais são denominadas camadas ocultas. Comumente não se considera a camada de entrada quando se conta o número de camadas da rede, de modo que esta seria uma RNA de três camadas e, o que caracteriza uma rede neural "rasa", levando em consideração que atualmente já são treinados modelos com centenas de camadas.

As camadas ocultas têm esse nome porque não se tem controle sobre como elas processam a informação recebida. Seu conteúdo raramente é de interesse de quem utiliza a rede neural, elas representam apenas passos intermediários entre a entrada e a saída do sistema. Desta forma, na maioria das aplicações a rede neural é vista apenas como uma caixa preta que recebe um vetor de entradas X e produz um vetor de saídas Y.

Cada camada é composta de um determinado número de neurônios, representados pelos círculos na Figura 4, que são as unidades básicas do processamento da rede. Os neurônios de cada camada se conectam a todos os elementos da camada anterior de forma que a rede se torna densamente conectada. O número da camada de um determinado neurônio é determinado pelo expoente entre colchetes.

Figura 4 – Modelo de rede neural utilizando duas camadas ocultas.



Fonte: Próprio autor.

Todo neurônio interage com os elementos da camada anterior a partir de um conjunto de pesos, w, e um viés, b. A partir deles, se calcula um número real z, pela Equação 2.1, onde x representa as entradas do neurônio em questão, sendo a própria entrada da rede para as unidades da primeira camada e, para as demais camadas, é o conjunto de saídas, a, da camada anterior.

$$z = w^T.x + b, \tag{2.1}$$

$$a = g(z). \tag{2.2}$$

Em seguida é aplicada uma função de ativação $g(z)$, que produz a saída do neurônio. Essa função insere não linearidades no sistema, fazendo com que a RNA não fique restrita ao mapeamento de apenas funções lineares. Nota-se que funções de ativação distintas podem ser escolhidas para cada camada.

2.3.1 Atualização dos Parâmetros

Os pesos e o viés de cada neurônio são atualizados segundo um algoritmo denominado descida do gradiente. Para tal, é necessário definir duas funções importantes:

1. $L(y, \hat{y})$ - Função de Perdas: Mensura para o i-ésimo exemplo do conjunto o quão diferente do rótulo $y^{(i)}$ está a previsão $\hat{y}^{(i)}$ fornecida pela rede. A escolha dessa função é fundamental para que a rede seja capaz de efetivamente generalizar a função a ser mapeada e atualizar seus parâmetros. Mais informações sobre essa função podem ser encontrados em (FRANCOIS, 2017).

2. $J(w, b)$ - Função de Custo: Enquanto $L(y, \hat{y})$ mensura o desempenho da rede em um único exemplo, $J(w, b)$ o faz para todo um conjunto de dados. Minimizar o valor desta função para o conjunto de treino é, consequentemente, o objetivo do treinamento da rede.

A forma mais comum de implementar a função de custo é calcular a média da função de perdas para todos os m exemplos do conjunto, conforme a Equação 2.3.

$$J(w, b) = \frac{1}{m}.\sum_{i=1}^{m} L(y^{(i)}, \hat{y}^{(i)}) \tag{2.3}$$

Para implementar a descida do gradiente deve-se então calcular o gradiente de $J(w, b)$ com relação aos parâmetros w e b, de modo que a cada iteração do processo de treinamento eles serão atualizados segundo as equações 2.4 e 2.5. Os valores iniciais das variáveis usualmente são inicializados de forma aleatória.

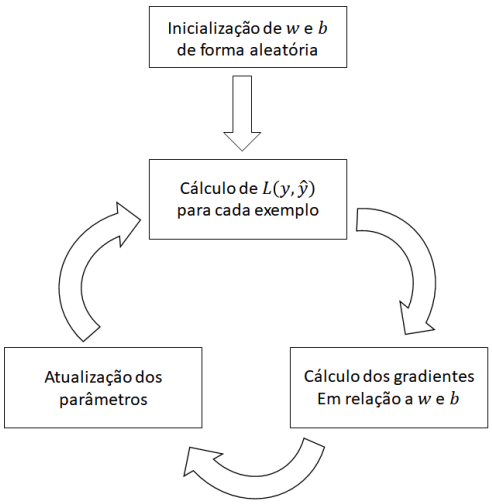
O termo α nas Equações 2.4 e 2.5 é denominado de taxa de aprendizagem e define a influência de cada iteração sobre os valores atuais dos parâmetros. Podendo fazer com que a cada iteração os parâmetros sofram grandes alterações, se α for alto, ou pequenas caso contrário. Nas equações, o termo k indica a iteração atual.

$$w_i(k+1) = w_i(k) - \alpha.\frac{\partial J(k)}{\partial w_i}, \tag{2.4}$$

$$b(k+1) = b(k) - \alpha.\frac{\partial J(k)}{\partial b}. \tag{2.5}$$

Para redes com mais de uma camada os gradientes de uma camada dependem dos gradientes da camada seguinte. Essa propagação no sentido inverso (da última camada para a primeira) é denominada retropropagação (*backpropagation*, em inglês) e pode ser vista mais detalhadamente em (FRANCOIS, 2017) e (ROSEBROCK, 2017). O treinamento segue o fluxograma da Figura 5 até que se atinja um critério de parada.

Figura 5 – Fluxograma do processo de treinamento.



Fonte: Próprio autor.

2.3.2 Hiperparâmetros

Em *Deep Learning* existem dois tipos de parâmetros, os que são treináveis e os que são definidos pelo projetista. Os primeiros correspondem aos pesos e viéses dos neurônios, enquanto os outros são características da rede como a taxa de aprendizagem, o número de camadas, o número de neurônios em cada camada, dentre outros.

Boas escolhas para os hiperparâmetros proporcionam melhores soluções e um treinamento mais veloz. Infelizmente, ainda não existe uma forma de determinar exatamente os melhores valores para cada hiperparâmetro, de modo que, para atingir uma performance ótima a rede deve ser treinada várias vezes até que se encontre um conjunto ótimo.

2.3.3 Funções de Perdas

Neste trabalho optou-se por abordar apenas as funções de perdas referentes a problemas de classificação, uma vez que os problemas abordados durante o estágio se enquadram nessa categoria. Nesse tipo de problema, o objetivo da rede é identificar para um conjunto de classes conhecido a qual delas pertence uma dada entrada.

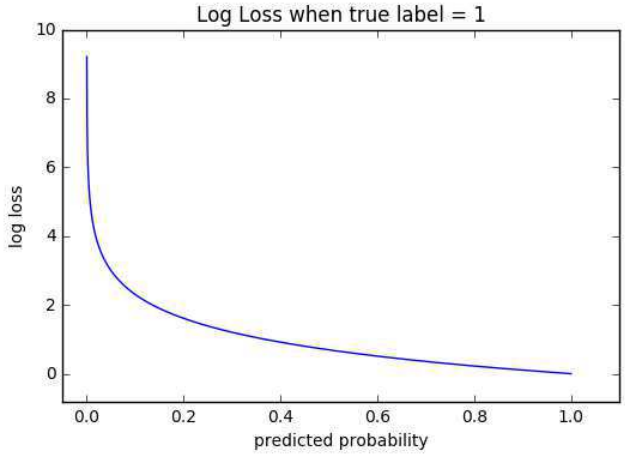
Para problemas de classificação binária, ou seja, quando só existem duas classes, a saída da rede, $\hat{y}$, será um número entre 0 e 1 que pode ser interpretado como a probabilidade de que a entrada pertença à primeira classe. Nesse caso, existem dois valores possíveis para a saída desejada, y, sendo 1 caso a entrada realmente pertença à primeira classe e 0 caso ela não pertença.

Consequentemente, espera-se que a função de perdas produza valores elevados se $y \neq \hat{y}$ e valores baixos para o contrário. Isso é alcançado pela função de entropia cruzada (Equação 2.6), também conhecida como função de perdas logarítmica (*Log Loss*), cujo gráfico pode ser visto na Figura 6 para $y = 1$. A Figura 6 ilustra um crescimento exponencial do valor de $L(y, \hat{y})$ à medida que a probabilidade prevista se afasta do valor da saída desejada. Caso $y = 0$ o gráfico seria espelhado com relação a $\hat{y} = 0,5$. Uma vez que y e $\hat{y} \in [0, 1]$, a função só está definida para este intervalo.

$$L(y, \hat{y}) = -(y.log\hat{y} + (1 - y).log(1 - \hat{y})) \tag{2.6}$$

Observa-se que todo problema de classificação pode ser reduzido a uma série de problemas de classificação binária. Tudo que deve ser feito é avaliar o caso para cada classe, de modo que para cada uma delas será avaliado a probabilidade de que a entrada pertença ou não à classe, permitindo a aplicação da função a cada uma das saídas.

Figura 6 – Função de perdas de entropia cruzada.



Fonte: `https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html`.

2.3.4 Funções de Ativação

Sigmóide

A função sigmóide (Equação 2.7) ilustrada na Figura 7 em conjunto com sua derivada, foi uma das primeiras escolhas dos pesquisadores para a função de ativação dos neurônios. Uma de suas qualidades notáveis é o fato dela ser derivável em toda a sua extensão, algo conveniente durante a descida do gradiente, pois simplifica a atualização dos parâmetros por meio da aplicação das Equações 2.4 e 2.5.

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}, \tag{2.7}$$

$$g'(z) = \sigma(z).(1 - \sigma(z)). \tag{2.8}$$

Outra característica interessante é o fato do resultado dessa função estar sempre contido entre 0 e 1. Isso a torna interessante para problemas de classificação binária. Porém, o fato de sua derivada rapidamente se aproximar de 0 conforme z se afasta da origem configura uma desvantagem. Como o aprendizado da rede depende do gradiente da função, derivadas proximas de 0 implicam em um aprendizado lento.

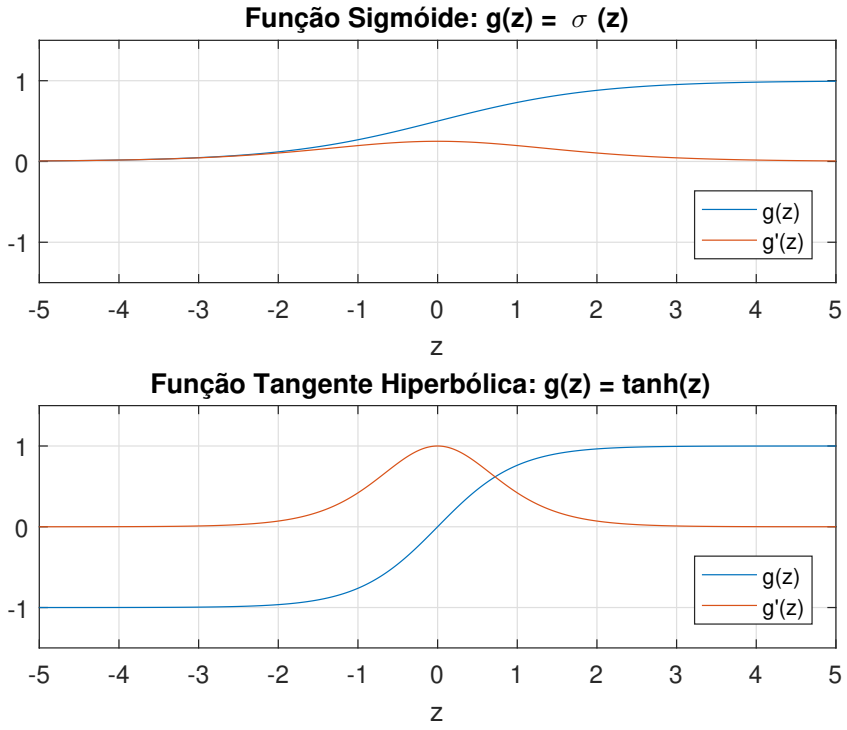


Figura 7 – Gráficos de $\sigma(z)$, $tanh(z)$ e suas derivadas.

Tangente Hiperbólica

A função tangente hiperbólica (Equação 2.9) é um tanto similar à sigmóide, como pode ser visto na Figura 7. No entanto, difere no sentido de que $tanh(z)$ produz resultados contidos entre -1 e 1.

$$g(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{2.9}$$

$$g'(z) = 1 - tanh^2(z). \tag{2.10}$$

Percebe-se também que esta função apresenta uma derivada mais expressiva que $\sigma(z)$, mesmo embora isso se concentre nos entornos de $z = 0$. Assim como a sigmóide, seu gradiente tende a 0 conforme $|z|$ aumenta, o que desacelera a velocidade do aprendizado.

ReLU

A função ReLU (Equação 2.11), cuja sigla significa Unidade Linear Retificadora (*Rectified Linear Unit*, em inglês), foi proposta para lidar com o problema descrito acima. Como pode ser visto na Figura 8, a derivada da função se mantém unitário para $z > 0$ e zero para $z < 0$, de modo que a ocorrência de valores altos não desacelera o aprendizado da RNA.

$$g(z) = ReLU(z) = max(0, z), \tag{2.11}$$

$$g'(z) = \begin{cases} 1 & \text{se } z \geq 0 \\ 0. & \text{caso contrário.} \end{cases} \tag{2.12}$$

Também deve se considerar que os cálculos proporcionados pela função (Equação 2.11) são consideravelmente simples. Como o treinamento da rede neural envolve múltiplas execuções destas operações, o uso de funções simples também implica em uma redução no tempo necessário para treinar a rede. Esses dois motivos fazem com que essa função seja bastante popular,

Todavia, como pode ser visto na Figura 8, a derivada da função apresenta uma descontinuidade em $z = 0$. Para contornar o problema, em geral se considera $g'(0) = 1$, o que não é matematicamente correto, mas não prejudica a execução do programa porque a probabilidade de que z seja exatamente igual a 0 é efetivamente nula, além de que, caso isso de fato ocorra, implicaria em um pequeno desvio do gradiente para a direção errada, algo que seria corrigido nas iterações seguintes.

Leaky ReLU

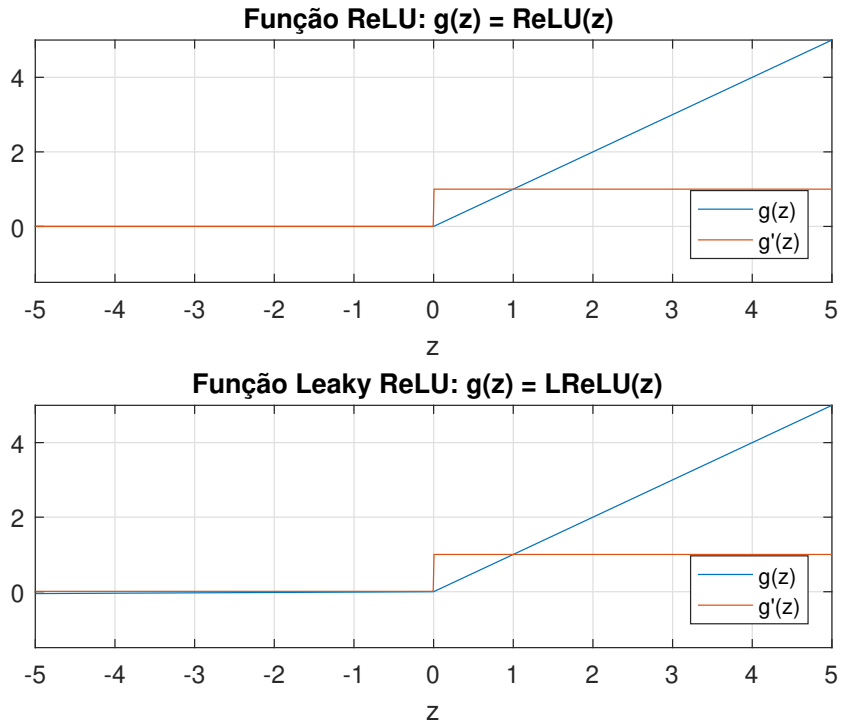
Embora a função ReLU não apresente um gradiente nulo para $|z| >> 0$, isso ocorre para todo $z < 0$, o que também pode desacelerar o aprendizado. Isso motivou o surgimento de uma versão modificada, a *Leaky* ReLU ($g(z) = LReLU(z)$), que conforme a Equação 2.13, possui derivada 0,01, e não zero, para $z < 0$. Isso faz com que os parâmetros da rede sejam ajustados mesmo quando z é negativo.

A LReLU está plotada na Figura 8 em conjunto com sua derivada. Em alguns casos a constante que multiplica z é tratada como um hiperparâmetro, permitindo que a rede ajuste o seu valor durante o treinamento.

$$g(z) = LReLU(z) = max(0,01.z, z), \tag{2.13}$$

$$g'(z) = \begin{cases} 1 & \text{se } z \geq 0. \\ 0,01 & \text{caso contrário.} \end{cases} \tag{2.14}$$

Figura 8 – Gráficos de $ReLU(z)$, $LReLU(z)$ e suas derivadas.



Fonte: Próprio autor.

Função *Softmax*

A função *softmax* é uma função especial utilizada na última camada de redes neurais classificadoras e tem a função de normalizar as probabilidades estimadas na saída da rede. Uma diferencial desta função de ativação é que a *softmax* é aplicada a uma matriz e produz uma outra matriz de mesmas dimensões, enquanto as demais funções de ativação apresentadas são aplicadas em um número real e produzem um outro número real.

Em problemas de classificação é comum dividir um problema com muitas classes em problemas de classificação binária, resultando em um número de neurônios igual ao número de classes na camada de saída. Caso seja utilizada a função sigmóide para obter suas ativações, cada um deles produzirá uma estimativa da probabilidade de que a entrada pertença à classe associada ao neurônio, porém, o somatório dessas probabilidades será diferente de 1, um problema que não ocorre caso seja utilizada a função *softmax*.

Outro problema da função sigmóide em problemas com muitas classes é que $\sigma(z) \approx 1$ para altos valores de z e $\sigma(z) \approx 0$ para baixos valores. Isso faz com que, após a ativação, uma mesma probabilidade possa ser associada a esses elementos ainda que uns sejam muito maiores ou menores que outros. Portanto, na saída desse tipo de RNA é necessário que se tenha um conjunto de probabilidades normalizado.

Para um vetor $Z = \begin{pmatrix} z_1 & z_2 & \dots & z_n \end{pmatrix}^T$, a função *softmax* é definida conforme a Equação 2.15.

$$softmax(Z) = \begin{pmatrix} \frac{e^{z_1}}{\sum_{j=1}^{n} e^{z_x}} \\ \frac{e^{z_2}}{\sum_{j=1}^{n} e^{z_x}} \\ \vdots \\ \frac{e^{z_n}}{\sum_{j=1}^{n} e^{z_x}} \end{pmatrix} \tag{2.15}$$

Como pode ser observado na Equação 2.15, tira-se a exponencial de cada valor de z_j e divide-se pelo somatório de todas as exponenciais, fazendo com que todos os elementos sejam positivos e seu somatório seja igual à unidade. Verifica-se que o valor da j-ésima saída cresce à medida que z_j cresce, evitando que dois valores de entrada distintos possam levar à mesma probabilidade de saída.

No contexto de redes neurais, o vetor Z referenciado na Equação seria composto pelos valores de z calculados em todos os n neurônios de uma determinada camada, de modo que $softmax(Z)$ produzirá as ativações de todos os neurônios da camada.

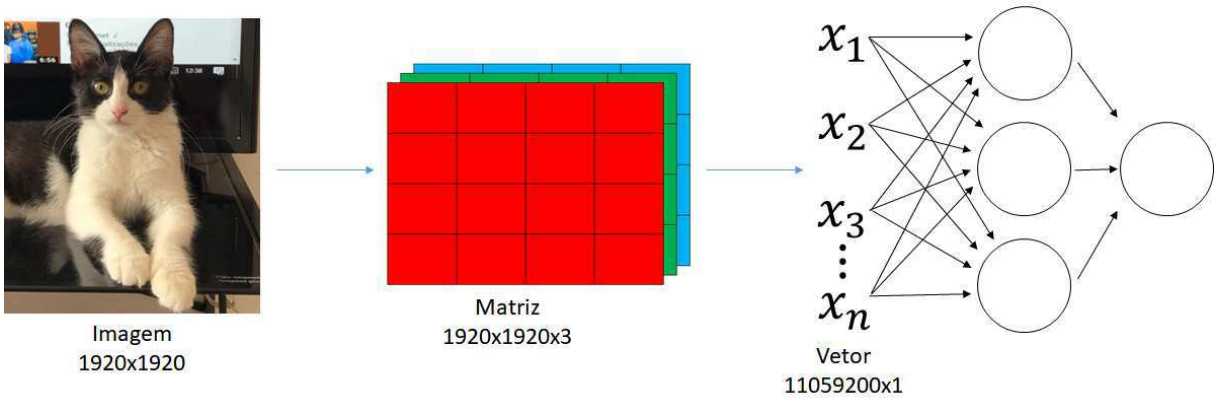
2.4 Redes Neurais Convolucionais

Um problema das RNAs quando utilizadas para o processamento de imagens é que a quantidade parâmetros treináveis se torna muito grande pois aumenta de acordo com as dimensões da imagem. Uma imagem colorida de $H \times W$ pixels é representada por um tensor de de $n = H.W.3$ elementos, e, para que sirva de entrada para uma RNA deve ser convertida em um vetor com mesmo número de elementos, conforme indicado na Figura 9. Para uma imagem de 1920×1920 pixels, esse vetor teria mais de 11 milhões de elementos, de modo que cada neurônio da primeira camada oculta teria um vetor de pesos contendo mais de 11 milhões de parâmetros para serem treinados.

As redes neurais convolucionais (CNN - *Convolutional Neural Networks*) surgiram no fim da década de 80, (FUKUSHIMA, 1988) e (LECUN et al., 1990), justamente como uma forma de lidar com esse problema, pois neste tipo de arquitetura, as multiplicações de matrizes que eram realizadas em cada camada são substituídas por convoluções entre imagens. Devido à natureza da operação de convolução, isso faz com que a quantidade de parâmetros não dependa das dimensões da imagem de entrada e consequentemente reduz o número de parâmetros.

Observa-se que nem toda camada de uma CNN realiza convoluções, as que o fazem são denominadas camadas convolucionais, outros tipos de camadas existentes são as camadas de ativação, de *pooling* e as camadas totalmente conectadas. Cada camada desempenha uma operação específica e, na estrutura da CNN, tipos diferentes de camadas são alternados com o objetivo de criar uma rede mais robusta. A escolha dos tipos de camada a serem utilizados não é trivial e tem forte impacto no resultado final, de modo que muitas vezes é necessário se tentar várias configurações diferentes antes que se obtenha o resultado desejado.

Figura 9 – Ilustração de uma rede neural cuja entrada é uma imagem colorida.



Fonte: Próprio autor.

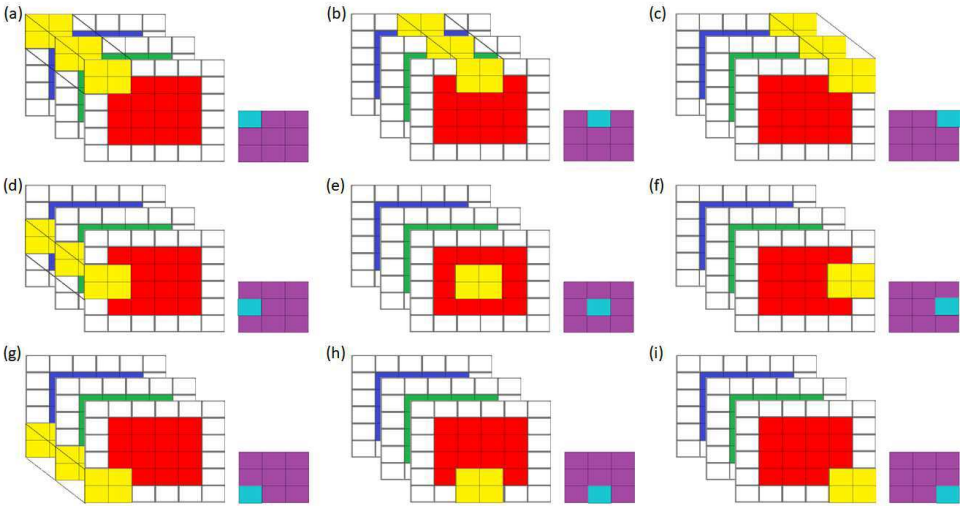
2.4.1 Camada Convolucional

Camadas convolucionais são os componentes principais de uma CNN e envolvem um conjunto de filtros que são convoluídos com sua entrada de modo a produzir sua saída, (KHAN et al., 2018). Além de reduzir o número de parâmetros da rede, o uso de camadas convolucionais torna o algoritmo mais robusto a transformações espaciais na imagem como a translação de objetos, devido ao fato de os seus filtros (ou *kernels*), que são os parâmetros da rede, atuarem por toda a extensão da imagem.

A convolução é uma operação esparsa, pois apenas algumas partes da entrada contribuem para uma determina porção da saída, e que reutiliza parâmetros, pois o mesmo conjunto de pesos é aplicado em múltiplas porções da entrada, (DUMOULIN; VISIN, 2016). Conforme a Figura 10, que ilustra uma convolução passo a passo, a operação pode ser vista como o deslizamento do filtro, representado em amarelo, por sobre a imagem, multiplicando os elementos que se sobrepõem e somando os resultados para produzir um elemento da saída, em azul. A operação continua até que todos os elementos da imagem de entrada tenham interagido com o filtro.

Existem três parâmetros relevantes para a operação, sendo estes o tamanho do filtro, o *padding*, e o *stride*. É comum o uso de filtros com mesma altura e largura, cujo valor designado por f é geralmente ímpar. O *padding* (p) se refere a *zero-padding* e representa o número de bordas com valor 0 adicionadas em torno da imagem a ser convoluída (borda branca na figura 10). Já o stride (s) indica de quanto será o deslizamento do filtro sobre a imagem de entrada com relação à imagem de saída em cada etapa da convolução. Manipulando os valores de f, p e s pode-se escolher as dimensões da saída da convolução.

Figura 10 – Operação de convolução tridimensional entre uma imagem colorida 4×4 e um filtro 2×2 com $p = 1$, $s = 2$.



Fonte: Próprio autor.

Percebe-se que as convoluções descritas produzem resultados bidimensionais. Porém, lembra-se que, em cada camada convolucional são empregados um número n_f de filtros, de modo que sua saída será a concatenação dos resultados das convoluções entre a sua entrada e cada um dos filtros, produzindo uma imagem de n_f canais, que será, então, enviada para a próxima camada da CNN. Naturalmente, cada camada pode ter um número diferente de filtros e este será um dos hiperparâmetros da rede.

Os parâmetros das camadas convolucionais são inicializados de forma aleatória e treinados de forma semelhante aos das RNAs, utilizando retropropagação. A quantidade de parâmetros depende apenas do número de filtros e de suas dimensões, de modo que para n_f filtros de tamanho f e n_c canais são empregados $(f^2.n_c+1).n_f$ parâmetros, sendo o termo $+1$ referente ao viés, onde 1 viés é adicionado ao resultado de cada uma das n_f convoluções. Se forem empregados 64 filtros $5 \times 5 \times 3$ em uma camada convolucional cuja entrada é uma imagem colorida de dimensões $1920 \times 1920 \times 3$, o número de valores a ser aprendidos será 4864, o que é bem menor que os 11 milhões necessários para uma RNA.

2.4.2 Camada de Ativação

Como o próprio nome sugere, esta camada é responsável por aplicar uma função de ativação sobre sua entrada. Geralmente esse tipo de camada é aplicado após as camadas convolucionais e não apresenta nenhum parâmetro treinável, levando a uma saída com as mesmas dimensões da entrada.

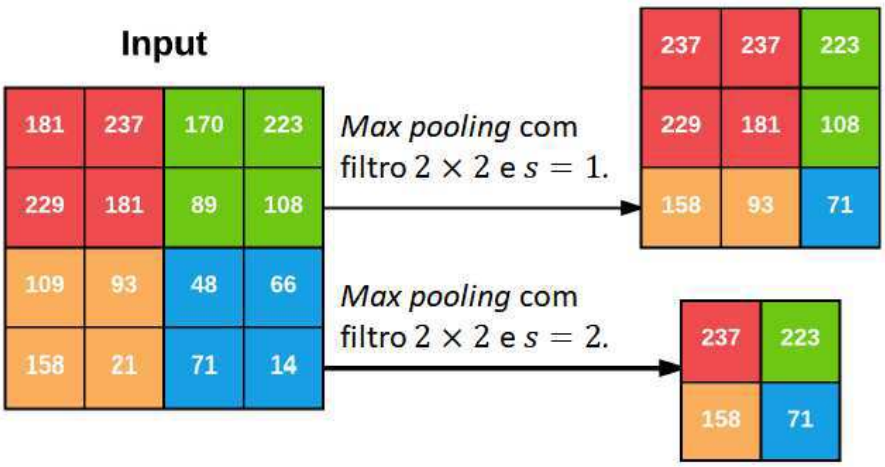
Nos diagramas de muitos trabalhos a camada de ativação é omitida, pois subentende-se que ela segue a camada convolucional, (ROSEBROCK, 2017). Também é comum chamá-la de camada ReLU, devido à ampla utilização dessa função de ativação em redes convolucionais. Neste trabalho a presença das camadas de ativação será indicada em conjunto com a sua localização em todos os modelos analisados.

2.4.3 Camada de Pooling

As camadas de *pooling* reduzem as dimensões da sua entrada utilizando uma função que condensa subregiões da entrada, como retirar a média (*average pooling*) ou o máximo (*max pooling*) dos valores da subregião, (DUMOULIN; VISIN, 2016). O uso destas camadas reduz a quantidade de parâmetros e de computações na rede, além de ajudá-la a aprender os padrões ao invés de decorar os exemplos do conjunto de treino, (ROSEBROCK, 2017).

A operação é similar à convolução, uma janela de tamanho f é deslizada pela entrada e, a cada passo, um valor é obtido segundo o tipo de *pooling* empregado. No caso de entradas com mais de um canal, a operação é realizada em cada canal de forma independente, produzindo uma saída com mesmo número de canais que a entrada.

Figura 11 – Aplicação de *max pooling* com diferentes *strides*.



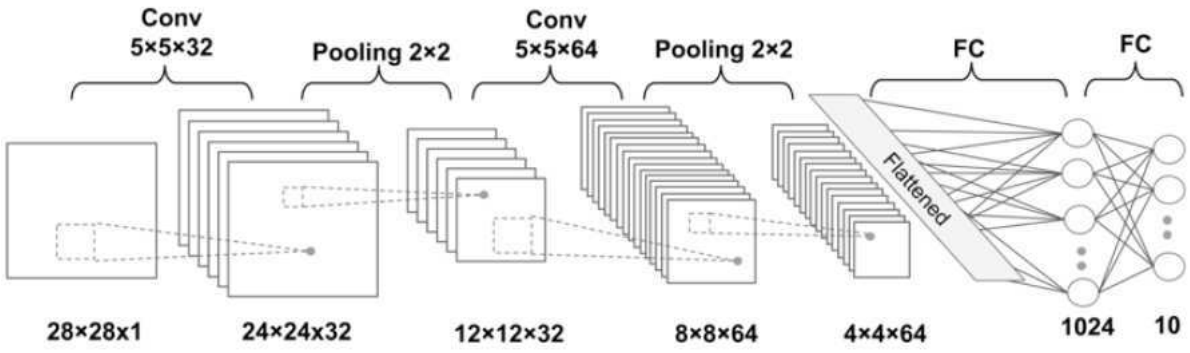
Fonte: Adaptado de (ROSEBROCK, 2017).

2.4.4 Camada Totalmente Conectada

As camadas totalmente conectadas são posicionadas ao final de CNNs, (ROSEBROCK, 2017). Antes que uma imagem possa servir de entrada para este tipo de camada, ele deve ser redimensionada para se transformar em um vetor coluna, de forma semelhante ao realizado em uma RNA. Deste ponto em diante, a rede convolucional funciona exatamente como as redes neurais artificiais mencionadas anteriormente.

Neste tipo de camada se localizam a maior parte dos parâmetros da rede. Por este motivo, CNNs tendem a reduzir as dimensões das imagens utilizando camadas convolucionais e *pooling* em suas camadas ocultas, minimizando assim o número de parâmetros nas camadas totalmente conectadas. A Figura 12 ilustra uma arquitetura típica de CNN que se utiliza das camadas vistas neste trabalho. Subentende-se que as ativações seguem as camadas convolucionais.

Figura 12 – CNN com camadas convolucionais, de *pooling* e totalmente conectadas.



Fonte: (RASCHKA; MIRJALILI, 2017).

2.5 *MobileNet*

Embora a comunidade de *Deep Learning* tenha realizado grandes avanços nos últimos anos, redes neurais ainda enfrentam grandes desafios com relação à sua aplicabilidade em sistemas reais que requisitam uma resposta em um intervalo de tempo limitado e, muitas vezes, dispondo de recursos computacionais também limitados. Trabalhos científicos atuais têm produzido redes neurais mais profundas e complexas visando obter maiores taxas de acurácia, entretanto, esses modelos nem sempre se preocupam em produzir redes mais eficientes em relação a seu tamanho e sua velocidade, (HOWARD et al., 2017).

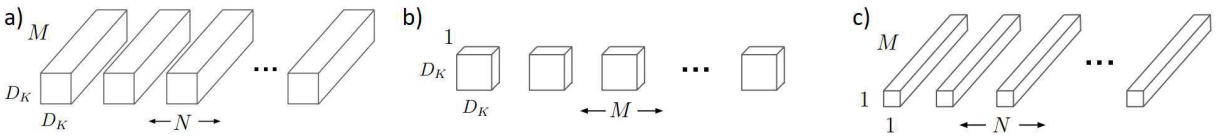
A necessidade de modelos que fossem mais rápidos e leves levou ao surgimento de novas arquiteturas como a *MobileNet*, proposta por (HOWARD et al., 2017). Essa arquitetura se baseia em convoluções separáveis (*Depthwise Separable Convolutions*) para produzir modelos leves e eficientes de redes convolucionais, sem que hajam grandes perdas na acurácia, permitindo o uso de *Deep Learning* em sistemas embarcados, aplicações para celular (*mobile*), dentre outras aplicações.

2.5.1 *Depthwise Separable Convolution*

De modo a reduzir o número de operações realizadas nas camadas convolucionais, os autores responsáveis pela *MobileNet* (HOWARD et al., 2017) optaram por separar as convoluções em duas etapas. Primeiro realiza-se uma convolução por profundidade em cada um dos canais da entrada com um filtro bidimensional (Figura 13 b)), em seguida realiza-se uma convolução pontual (Figura 13 c)), que corresponde a uma convolução tradicional com um filtro 1×1.

Enquanto a primeira convolução filtra os canais de entrada, a segunda é responsável por misturá-los de modo a extrair novas características. Desta forma, o papel desempenhado é bastante semelhante ao de uma convolução padrão, porém, o custo computacional dessa operação é significantemente menor. Conforme pode ser visto em (HOWARD et al., 2017), a razão entre o custo computacional de uma convolução separável e o de uma convolução tradicional para uma imagem dimensões $D_F \times D_F \times M$ e um conjunto de *kernels* de dimensões $D_K \times D_K \times M \times N$ é da ordem de $\frac{1}{N} + \frac{1}{D_K^2}$ operações.

Figura 13 – Filtros utilizados em uma convolução (a), em uma convolução por profundidade (b), e uma convolução pontual, (c).



Fonte: Adaptado de (HOWARD et al., 2017).

2.5.2 Arquitetura

Uma única camada convolucional, na primeira camada oculta, não se utiliza de convoluções separáveis, sendo a maior parte da *MobileNet* constituída de uma série de convoluções separáveis sucessivas (Tabela 1), de modo que o grande segredo para a performance obtida por essa arquitetura se deve às modificações implementadas nessas camadas. Após essa sequência, tem-se uma camada de *pooling*, seguida de uma camada totalmente conectada (FC) e, então, uma camada com ativação *softmax*. Observa-se que todas as camadas convolucionais são seguidas de uma camada de ativação ReLU. O termo "dw"na tabela indica convoluções por profundidade.

Uma outra estratégia utilizada nessa rede é a minimização das dimensões da imagem antes da camada totalmente conectada. Faz-se uso de uma camada de *pooling* para reduzir a imagem final de $7 \times 7 \times 1024$ para $1 \times 1 \times 1024$, o que diminui por 49 vezes o número de parâmetros nesta camada. Nota-se que, devido à baixa resolução da imagem e à grande quantidade de processamento já implementado pela rede até o momento torna-se preferível o uso do *average pooling* ao invés do *max pooling*, visto que o segundo descartaria grande parte da informação codificada até o momento.

Tabela 1 – Arquitetura da MobileNet

Tipo / Stride	**Dimensões do Filtro**	**Dimensões da Entrada**	**Repetições**
Conv / s = 2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$	x1
Conv dw / s = 1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$	x1
Conv / s = 1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$	x1
Conv dw / s = 2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$	x1
Conv / s = 1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$	x1
Conv dw / s = 1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$	x1
Conv / s = 1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$	x1
Conv dw / s = 2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$	x1
Conv / s = 1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$	x1
Conv dw / s = 1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$	x1
Conv / s = 1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$	x1
Conv dw / s = 2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$	x1
Conv / s = 1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$	x1
Conv dw / s = 1 Conv / s = 1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$	x5
Conv dw / s = 2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$	x1
Conv / s = 1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$	x1
Conv dw / s = 2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$	x1
Conv / s = 1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$	x1
Avg Pool / s = 1	Pool 7×7	$7 \times 7 \times 1024$	x1
FC / s = 1	1024×1000	$1 \times 1 \times 1024$	x1
Softmax / s = 1	Classifier	$1 \times 1 \times 1000$	x1

Fonte: Adaptado de (HOWARD et al., 2017).

3 Projeto de Inspeção Veícular

De modo a atender uma das demandas da empresa, uma parte da equipe de pesquisa foi designada para implementar, utilizando *Deep Learning*, um algoritmo capaz de identificar informações relevantes para o processo de inspeção de veículos a partir de fotografias tiradas com o celular. Dentre as exigências realizadas estavam a obtenção de:

- Propriedades do veículo: cor, fabricante e modelo;
- Localização de componentes: faróis dianteiros, lanternas traseiras, faróis de milha e limpadores;
- Situação de componentes[1]: faróis dianteiros, lanternas traseiras e luzes de freio.

As atividades realizadas nesse projeto estiveram compreendidas entre os dias 09/09/2019 e 25/11/2019, envolvendo a construção de bancos de dados e o projeto de redes neurais convolucionais aptas a realizar as tarefas relacionadas aos demais requisitos solicitados no projeto, modelados como problemas de classificação. As atividades foram encerradas com a produção de um protótipo em formato de aplicativo *web* capaz de realizar as operações solicitadas para vinte modelos diferentes de carros.

Uma lista dos modelos suportados pelo projeto, separados por fabricante, está disponível na Tabela 2. Foram identificadas as cores mais frequêntes de carros no Brasil, de modo que o sistema é capaz de identificar as 8 cores mais comuns em veículos: amarelo, azul, branco, cinza, prata, preto, verde e vermelho.

[1] Se o componente estava ou não acionado na foto recebida.

Tabela 2 – Modelos e fabricantes identificados pelo *webapp* desenvolvido.

Fabricante	Modelos
Chevrolet	Onix, Prisma, S10, Spin
Fiat	Mobi, Strada, Toro
Ford	Ka
Honda	Civic, Fit, HR-V
Hyundai	HB20, HB20S
Renault	Logan
Toyota	Corolla, Hilux
Volkswagen	Gol, Polo, Saveiro, Voyage

Fonte: Próprio autor.

O projeto já estava em andamento na data de início do estágio, de modo que um dos requisitos, a CNN responsável pela localização de componentes, já havia sido implementada. Esse requisito foi modelado como um problema de detecção e, para que se realizasse o treinamento da rede neural responsável por essa tarefa, um banco de dados foi montado pela equipe responsável. Embora tanto a rede quanto seu banco de dados tenham sido utilizados no decorrer do projeto, optou-se por não aprofundar a discussão sobre esta etapa dado que ela já estava concluída antes do início do estágio.

3.1 Banco de Dados

As primeiras atividades foram realizadas a partir do banco de dados utilizado no treino da rede de detecção, composto de imagens retiradas de sites brasileiros de vendas de veículos e contava com 17,3 mil imagens. Para definir quais modelos seriam identificados pela rede foi realizada uma pesquisa para determinar os modelos de carro mais comuns no Brasil, de modo que o banco foi filtrado e a quantidade de imagens caiu para 6,4 mil.

Devido às naturezas distintas das demais tarefas e o grande desbalanceamento observado nas classes do banco de dados, optou-se pela busca de mais imagens. A maior parte das fotos disponíveis se concentrava em alguns poucos modelos ou cores de carros, além disso eram muito pouco frequentes imagens nas quais os faróis, as luzes de freio ou as lanternas do veículo estivessem acesas. De modo que foi necessário montar novos bancos exclusivos para cada uma das tarefas restantes.

Cada banco de dados foi dividido em três conjuntos, um de treino, um de validação e outro de teste. Essa divisão é realizada porque durante o treinamento a rede pode aprender a decorar padrões específicos dos exemplos do conjunto de treino ao invés de padrões genéricos que efetivamente descrevem as classes, fenômeno conhecido como *overfitting*.

Dessa forma, a cada iteração do treinamento se verifica a performance da rede no conjunto de validação para garantir que os padrões aprendidos não sejam particulares da base de treino. O conjunto de testes serve para realizar uma verificação final com exemplos nunca vistos, garantindo assim que os padrões que a rede aprendeu a reconhecer são os genéricos e não particulares dos conjuntos de treino/validação.

Como uma precaução extra contra *overfitting*, a separação entre os conjuntos de treino e validação era feito durante o treinamento, sendo 20% das imagens separadas para validar o treino. Desta forma, garantia-se que a cada nova alteração na arquitetura das redes novos conjuntos de treino e validação eram produzidos, tornando mais difícil que as alterações favorecessem padrões particulares desses conjuntos.

Uma descrição mais detalhada da composição dos bancos de dados de cada tarefa é fornecida a seguir. Tem-se, na Tabela 3, o número de amostras nos conjuntos de treino e teste de cada tarefa.

3.1.1 Propriedades do Veículo

Começou-se pelas tarefas de classificação do modelo, do fabricante e da cor em decorrência da maior disponibilidade de dados. Os conjuntos de treino e validação foram produzidos a partir do banco de dados que já estava disponível e um banco de dados público, *CompCars*, (YANG et al., 2015), e o conjunto de testes foi produzido a partir de 1500 imagens coletadas a partir de sites de vendas de carros brasileiros.

O *CompCars* conta com 136,7 mil fotos de veículos com finalidade de identificação dos modelos em gravações de câmeras de segurança. Infelizmente a maior parte dos modelos presentes no *CompCars* eram estrangeiros e não existiam no Brasil, levando à necessidade de selecionar apenas os modelos relevantes para o projeto. Foram considerados os modelos mais vendidos nos anos de 2018 e 2019, além da disponibilidade de imagens no banco de dados para realizar a seleção. Deste modo, somente 1100 imagens desse banco foram utilizadas.

O número de modelos suportados foi determinado pela quantidade de imagens disponíveis para cada classe, de modo que foram utilizadas entre 150 e 500 imagens por classe e para a classificação de modelo. Para classificação de cor as classes continham entre 200 e 1000 amostras, tendo sido necessário eliminar imagens aleatoriamente pois algumas classes continham muito mais do que 1000 amostras.

3.1.2 Situação dos Componentes

Para a determinação da situação dos faróis, lanternas e luzes de freio foi necessário montar um banco de dados próprio. Esse banco foi composto majoritariamente de fotografias dos carros dos integrantes da equipe de pesquisa nos arredores da UFPB e dos veículos de diversas concessionárias da cidade de João Pessoa, com as devidas autorizações. Foram tiradas fotos dos veículos com os faróis dianteiros, lanternas traseiras e luzes de freio acionados e desligados.

Um problema a ser observado a respeito da montagem desse banco de dados é que em muitas das concessionárias os sistemas elétricos operavam em um modo especial de demonstração. Consequentemente, nem todas as funcionalidades do sistema elétrico dos mesmos estavam acessíveis de modo que em alguns veículos não era possível ligar as lanternas o freio, fazendo com que o banco de dados referente a essa tarefa fosse menor.

As 664 fotos produzidas foram utilizadas na composição dos conjuntos de treino e validação para as três tarefas. Como o número de imagens de cada tarefa foi relativamente

pequeno, entre 200 e 250 fotos/tarefa, algumas adaptações foram realizadas durante o treinamento para um melhor aproveitamento dos dados. Vasculhando o banco preexistente e o *CompCars* foi possível extrair cerca de 100 fotos de carros com farol dianteiro e lanterna traseira acionadas, permitindo a criação de um conjunto de testes para estas tarefas. Praticamente não foram encontradas imagens onde as luzes de freio dos veículos estavam acionadas nesses bancos.

A necessidade de um conjunto de testes para as luzes de freio fez com que fosse necessário recorrer a um outro banco de dados público, o *BDD100k*, (YU et al., 2018). Esse banco disponibiliza mais de 100 mil fotografias de vias públicas para o treinamento de algoritmos de carros autônomos. A partir das imagens desse banco foi possível a criação do conjunto de testes, pois algumas delas continham veículos com a luz de freio acionada. Optou-se por utilizar um mesmo número de imagens em todos os conjuntos de teste, de modo que foram separadas 100 imagens com o componente acionado e 100 imagens com o componente desligado.

Tabela 3 – Número de imagens em cada conjunto para todas as tarefas.

Classificador	**Treino**	**Teste**
Classificação de Modelo	7500	1500
Classificação de Cor	6400	400
Situação dos Faróis	250	200
Situação das Lanternas	250	200
Situação do Freio	200	100

3.2 Treinamento

Apesar dos esforços realizados na montagem dos *datasets*, o número de imagens para treinar as redes ainda era pequeno para o contexto de *Deep Learning*. Isso é prejudicial porque o uso de pequenas bases de dados facilita que a rede decore os exemplos do conjunto de treino (*overfitting*) ao invés de aprender a reconhecer os padrões que realmente caracterizam as classes a serem identificadas. Esse problema foi tratado durante o treinamento com o auxílio de uma técnica denominada *data augmentation*.

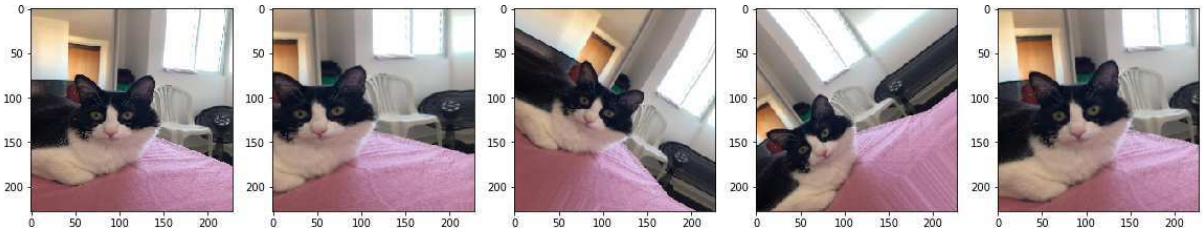
Outro problema dos bancos montados foi o desbalanceamento entre as classes. Um banco de dados desbalanceamento pode fazer com que as classes menos representadas sejam ignoradas pela rede, uma vez que ela pode atingir altos índices de acurácia ao focar apenas nas classes majoritárias. Esse problema também foi abordado durante o treinamento através do uso de *random oversampling* para balancear classes.

3.2.1 *Data Augmentation*

Data augmentation é uma forma de produzir um novo conjunto de dados a partir de um conjunto já existente por meio da aplicação de pequenas transformações, exemplificado na Figura 14. Exemplos de transformações podem ser cortes, rotações, translações ou espelhamentos aplicados de forma aleatória nas imagens. De modo que redes treinadas com o uso dessa técnica são mais robustas a essas transformações e capazes de aprender padrões mais generalistas.

Embora ainda fosse preferível a obtenção de imagens completamente novas, o uso de *data augmentation* é capaz de melhorar consideravelmente os resultados. Isso ocorre porque as transformações podem ser utilizadas para produzir um número arbitrário de novas imagens. Essa é uma das técnica mais comuns para tratar de *overfitting* e é utilizada quase de forma universal no treinamento de redes convolucionais, (FRANCOIS, 2017).

Figura 14 – Exemplo de aplicação de *data augmentation*, imagem original à esquerda.



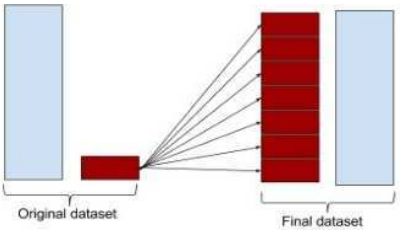
Fonte: Próprio autor.

3.2.2 Balanceamento das Classes

Em todos os modelos treinados aplicou-se *random oversampling* para balancear as classes durante o treinamento. Essa técnica, ilustrada na Figura 15, consiste na aplicação de *data augmentation* consecutivamente sobre as imagens das classes minoritárias até que elas contenham o mesmo número de exemplos que a classe majoritária, balanceando o conjunto de treino. Nota-se que não existe necessidade de balancear os conjuntos de validação e teste, embora seja interessante que eles possuam exemplos de todas as classes para que se possa avaliar corretamente o andamento do treino.

Como no caso da seção anterior, seria preferível dispor de novas imagens, no entanto, caso o número de exemplos da classe minoritária não seja muito pequeno, balancear as classes por meio desse algoritmo pode reduzir o viés causado pelo desbalanceamento na resposta da rede. Aplicar *random oversampling* em um conjunto com classes minoritárias muito menores que as classes majoritárias pode causar *overfitting* devido à repetição excessiva de imagens.

Figura 15 – Exemplo de balanceamento por *random oversampling*.



Fonte: `https://www.thelearningmachine.ai/imbalanced`.

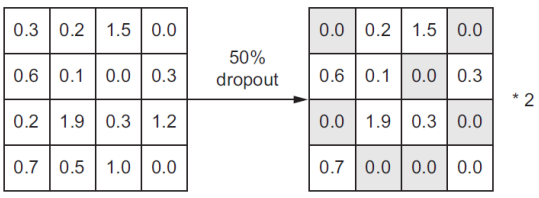
3.2.3 Regularização

Uma forma comum de mitigar *overfitting* é adicionar restrições à complexidade da rede ao forçar seus pesos a terem apenas valores baixos, forçando-os a assumir uma distribuição mais regular, (FRANCOIS, 2017). A esse processo se dá o nome de regularização por pesos (*weight regularization*) e corresponde a adicionar à função de custo um termo referente ao valor dos parâmetros da rede, fazendo com que para baixar o valor da função de custo a rede necessite empregar parâmetros pequenos. Fez-se uso de regularização L2, de modo que o custo adicionado é proporcional ao quadrado do valor dos parâmetros.

Outra forma de combate ao *overfitting* empregada foi a regularização por *dropout*. Esse tipo de regularização consiste no descarte aleatório de algumas das saídas da camada, conforme pode se ver na Figura 16, dificultando que os elementos da camada seguinte decorem padrões específicos e, consequentemente, reduzindo o *overfitting*. Essa técnica faz com que seja necessário que os elementos da camada seguinte utilizem o máximo de informação possível da camada anterior para definir sua saída, reduzindo casos em que essa saída é definida com base apenas em uma característica particular.

Faz-se ainda duas observações importantes sobre o *dropout*. A primeira é que ao aplicar sobre uma camada uma probabilidade de descarte $p \in [0, 1]$ deve-se multiplicar o resultado por $\frac{1}{p}$ de modo a manter a média dos valores da camada inalterados. Além disso, o *dropout* só é aplicado somente durante o treinamento, fazendo com que não ocorra nenhum descarte de informação na utilização da rede.

Figura 16 – Aplicação de *dropout* com 50% de probabilidade de descarte.



Fonte: (FRANCOIS, 2017).

3.3 Arquiteturas

Foram realizados testes com diversas arquiteturas distintas, indo desde modelos completamente autorais a modelos mais tradicionais como a VGG (SIMONYAN; ZISSERMAN, 2014), e a *ResNet*, (HE et al., 2016). Eventualmente ficou evidente que embora essas arquiteturas proporcionassem bons resultados (mais que 80% de acerto), elas exigiam muitos recursos computacionais e seu tempo de processamento era lento, demorando mais do que 1 minuto na aplicação final utilizando GPU. Desse modo, optou-se por utilizar uma arquitetura baseada na *MobileNet*, (HOWARD et al., 2017), devido às suas propriedades descritas na fundamentação teórica.

De modo a utilizar plenamente os avanços proporcionados pela *MobileNet*, optou-se por preservar a arquitetura de sua parte convolucional (Tabela 4) e realizar modificações apenas nas camadas totalmente conectadas, uma prática comum em *Deep Learning*, conforme (FRANCOIS, 2017). Assim, a parte convolucional da *MobileNet* (*Conv_MobileNet* deste ponto em diante) compôs as primeiras camadas das arquiteturas projetadas para classificação de cor e modelo, apelidadas respectivamente de ColorNet e ModelNet.

Essa estratégia não foi utilizada para treinar as demais redes. O pequeno número de imagens disponíveis para o treinamento dessas CNNs pode levar à ocorrência de *overfitting* caso fosse utilizado um modelo com muitos parâmetros, de modo que foi necessário desenvolver arquiteturas e estratégias próprias que fossem capazes de tratar os demais problemas de forma adequada. Seguindo o padrão, as demais redes projetadas foram apelidadas de FarolNet, LanternaNet e FreioNet, respectivamente.

Tabela 4 – Porção Convolucional da MobileNet.

Tipo / Stride	**Dimensões do Filtro**	**Dimensões da Entrada**	**Repetições**
Conv / s = 2	3 × 3 × 3 × 32	224 × 224 × 3	x1
Conv dw / s = 1	3 × 3 × 32 dw	112 × 112 × 32	x1
Conv / s = 1	1 × 1 × 32 × 64	112 × 112 × 32	x1
Conv dw / s = 2	3 × 3 × 64 dw	112 × 112 × 64	x1
Conv / s = 1	1 × 1 × 64 × 128	56 × 56 × 64	x1
Conv dw / s = 1	3 × 3 × 128 dw	56 × 56 × 128	x1
Conv / s = 1	1 × 1 × 128 × 128	56 × 56 × 128	x1
Conv dw / s = 2	3 × 3 × 128 dw	56 × 56 × 128	x1
Conv / s = 1	1 × 1 × 128 × 256	28 × 28 × 128	x1
Conv dw / s = 1	3 × 3 × 256 dw	28 × 28 × 256	x1
Conv / s = 1	1 × 1 × 256 × 256	28 × 28 × 256	x1
Conv dw / s = 2	3 × 3 × 256 dw	28 × 28 × 256	x1
Conv / s = 1	1 × 1 × 256 × 512	14 × 14 × 256	x1
Conv dw / s = 1 Conv / s = 1	3 × 3 × 512 dw 1 × 1 × 512 × 512	14 × 14 × 512 14 × 14 × 512	x5
Conv dw / s = 2	3 × 3 × 512 dw	14 × 14 × 512	x1
Conv / s = 1	1 × 1 × 512 × 1024	7 × 7 × 512	x1
Conv dw / s = 2	3 × 3 × 1024 dw	7 × 7 × 1024	x1
Conv / s = 1	1 × 1 × 1024 × 1024	7 × 7 × 1024	x1
Avg Pool / s = 1	Pool 7 × 7	7 × 7 × 1024	x1

Fonte: Adaptado de (HOWARD et al., 2017).

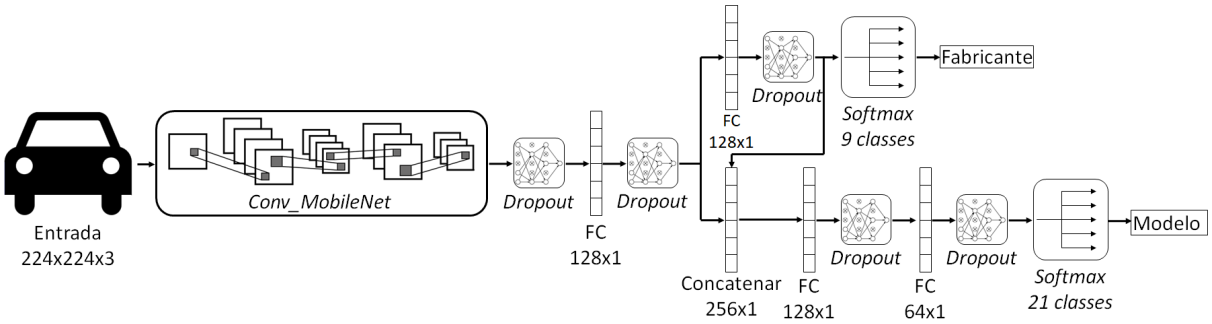
3.3.1 Modelo/Fabricante

A Figura 17 ilustra a arquitetura da ModelNet, a CNN projetada para classificar os carros quanto ao modelo do veículo. A entrada da rede passa pela *Conv_MobileNet* e produz um vetor coluna de 1024 neurônios que serve de entrada para as demais camadas da rede. As camadas identificadas por **FC** e ***Softmax*** são camadas totalmente conectadas seguidas pelas funções de ativação ReLU e *Softmax*, respectivamente. O número de elementos em cada camada está indicado abaixo das mesmas, salve em caso de camadas *Dropout*, que não contém elementos.

A rede recebe uma imagem de dimensões $224 \times 224 \times 3$, tamanho padrão de entrada para a *MobileNet*, e produz duas saídas, uma referente ao fabricante do veículo, e outra referente ao modelo do veículo, configurando essa CNN como uma rede neural multitarefa (*Multitask*). Como pode ser visto nos trabalhos de Ruder (2017) e Wu, Chan e Healthcare (2019), o treino de arquiteturas multitarefas contribui para a redução de *overfitting* e melhora a eficiência de modelos ao proporcionar o compartilhamento de parâmetros para diferentes tarefas.

Outra característica da ModelNet é a existência de uma conexão entre os ramos de suas saídas a partir de uma camada de concatenação que junta os elementos das suas entradas em um único vetor que é passado para a camada seguinte. Os elementos da camada totalmente conectada do ramo referente à classificação de fabricante estão conectados ao outro ramo, de modo que o fabricante identificado pela rede é uma informação utilizada diretamente para determinar o modelo do veículo, permitindo, por exemplo, que a rede dê mais atenção aos modelos produzidos por um determinado fabricante que ela tenha identificado.

Figura 17 – Esquemático da CNN projetada para identificar modelo e fabricante.



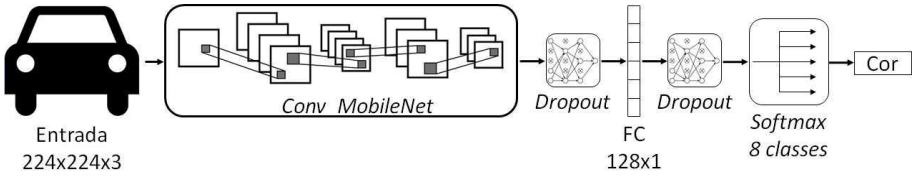
Fonte: Próprio autor.

3.3.2 Cor

Pode-se argumentar que é consideravelmente mais simples identificar as cores de um veículo do que seu modelo ou fabricante uma vez que os elementos das classes no primeiro problema apresentam variabilidade consideravelmente menor. Como consequência, a ColorNet, projetada para essa tarefa, é uma CNN consideravelmente mais simples que a ModelNet, como pode ser visto na Figura 18.

Sua entrada é uma imagem de dimensões 224×224×3 e, após a etapa convolucional, só se seguem duas camadas totalmente conectadas, uma com ativação ReLU e outra *softmax*, para determinar a saída. Sua saída é um número entre 0 e 7 que representa uma das classes entre amarelo, azul, branco, cinza, prata, preto, verde e vermelho, que foram as cores mais comuns nos carros do banco de dados.

Figura 18 – Esquemático da CNN projetada para identificar a cor do veículo.



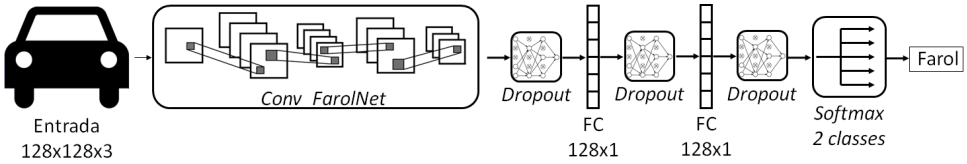
Fonte: Próprio autor.

3.3.3 Faróis e Lanternas

A grande dificuldade nessas tarefas de classificação binária foi projetar uma arquitetura que minimizasse o *overfitting* causado pelo pequeno número de imagens dos conjuntos de treino. Para reduzir a quantidade de parâmetros optou-se por utilizar uma sequência de camadas convolucionais e de *pooling* com incrementos sucessivos no número de canais de cor (Figuras 20 e 22) ao invés de utilizar a *Conv_MobileNet*, permitindo reduzir o número de parâmetros em cerca de 60% com relação aos que seriam utilizados com a base convolucional da *MobileNet*.

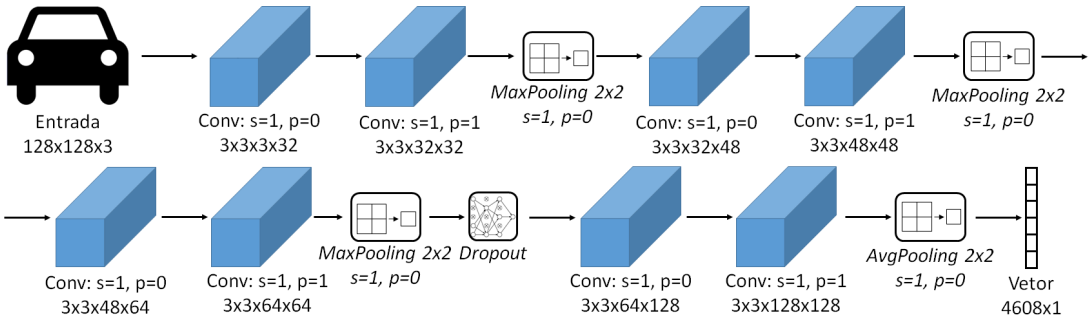
Como pode ser visto nas Figuras 19 e 21, ambas as arquiteturas foram idênticas após as respectivas porções convolucionais. As diferenças nas arquiteturas se deram no número de camadas convolucionais e no tamanho de seus filtros devido às particularidades de cada tarefa. A classificação das lanternas se mostrou mais desafiadora que a dos faróis, que apresentam mudanças bruscas na sua coloração quando estão acesos, enquanto as lanternas além de não apresentarem essa mudança de cor ao serem acionadas ainda podiam ser confundidas com outras partes dos veículos caso eles fossem de cor vermelha.

Figura 19 – Esquemático da FarolNet.



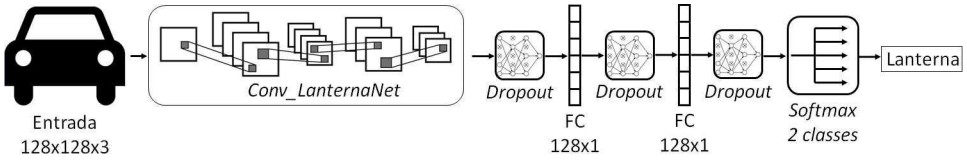
Fonte: Próprio autor.

Figura 20 – Esquemático da porção convolucional da FarolNet.



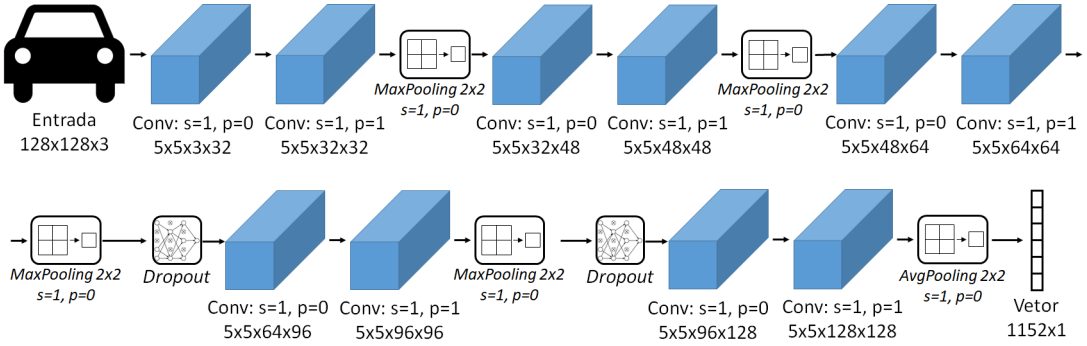
Fonte: Próprio autor.

Figura 21 – Esquemático da LanternaNet.



Fonte: Próprio autor.

Figura 22 – Esquemático da porção convolucional da LanternaNet.



Fonte: Próprio autor.

3.3.4 Luz de Freio

A FreioNet foi possivelmente a CNN mais difícil de se projetar e treinar. Não só possuía o menor de todos os bancos de dados como era muito fácil confundir um carro com as lanternas ligadas e um carro com o freio acionado, algo que pode ser notado a partir da Figura 23. Nota-se que as grandes diferenças vistas no exemplo da Figura 26 são uma iluminação mais intensa das lanternas quando o freio é acionado e a adição de novas fontes luminosas, duas na região das lanternas e outra entre o vidro e a traseira do veículo.

Figura 23 – Fotografias de um mesmo veículo nos estados desligado, à esquerda, lanterna ligada, ao centro, e freio acionado, à direita.

Fonte: Próprio autor.

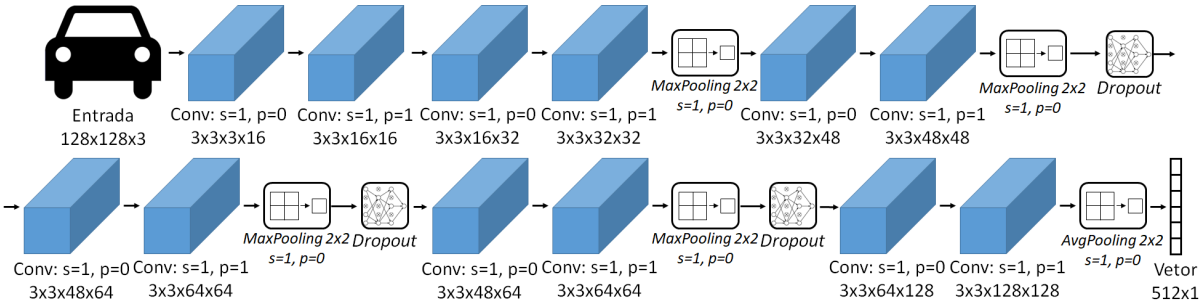
Um dos problemas da utilização dessa rede foi a ocorrência frequente de erros quando a lanterna estava acionada, fazendo com que a rede indicasse que o freio estava acionado. Apesar disso, a rede identificou o acionamento do freio corretamente quando a lanterna estava desligada, de modo que optou-se por especializar a rede para operar apenas na ausência da lanterna. Esse problema só foi tratado durante a implementação do *WebApp*.

Conforme as Figuras 24 e 25, vê-se que a FreioNet teve uma arquitetura semelhante às CNNs utilizadas para os faróis e lanternas, porém com um número maior de camadas. Já que não existia uma parte comum a todos os modelos de veículos que indicasse o acionamento do freio, fez-se uso da imagem inteira para que a própria rede neural identificasse a região de interesse para a tarefa em cada veículo. Consequentemente, foram necessárias mais camadas para que a rede fosse capaz de localizar a região de interesse e então realizar a classificação.

Diferentemente das demais redes, a FreioNet não se utilizou de *dropout* entre suas camadas totalmente conectadas. O não uso desse artifício foi determinado de forma empírica durante o treinamento da rede, porém, a obtenção de melhores resultados indica que as camadas da rede compactam a informação contida na imagem de tal for que os neurônios nas camadas finais retém uma quantidade significativa de informação, fazendo com que o desligamento de alguns neurônios em decorrência do *dropout* iria comprometer a capacidade da rede de determinar a saída com exatidão.

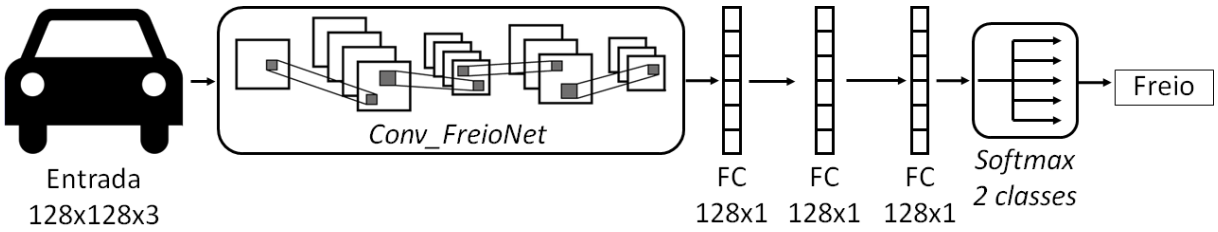
A explicação fornecida acima também pode ser utilizada para justificar o uso de *average pooling* ao invés de *max pooling* ao final das camadas convolucionais. Devido à grande compactação de informação o uso de *max pooling* promoveria o descarte de informações cruciais que foram detectadas pelos filtros da rede, de modo que o *average pooling* produz melhores resultados ao preservá-las. Observa-se que essa estratégia também é empregada ao final da parte convolucional da *MobileNet*, conforme visto na Tabela 4.

Figura 24 – Esquemático da porção convolucional da FreioNet projetada para identificar se o freio estava acionado ou não.



Fonte: Próprio autor.

Figura 25 – Esquemático da CNN projetada para identificar se o freio estava acionado ou não.



Fonte: Próprio autor.

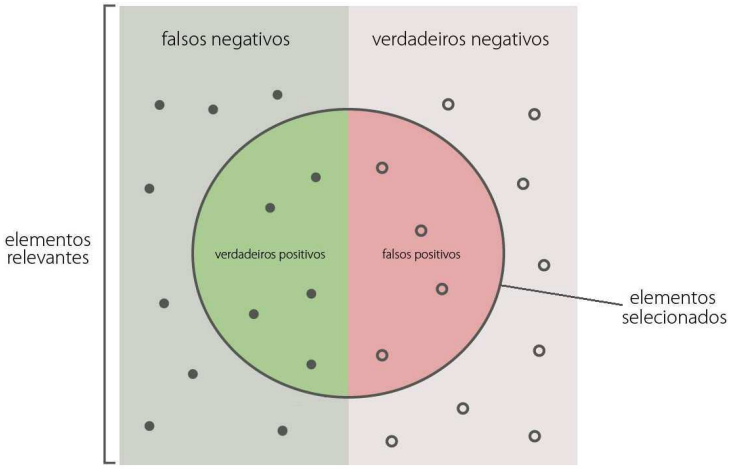
3.4 Resultados

Ao avaliar a qualidade das previsões realizadas em um problema de classificação geralmente são medidas duas quantidades, a de acertos, quando a classe prevista é igual ao gabarito, e o caso contrário, os erros. Isso basta para que se defina a acurácia, o índice que geralmente é monitorado durante o treinamento das redes e corresponde à razão entre os acertos e o total de classificações realizadas. Porém, é uma prática comum analisar os resultados para cada uma das classes e subdividir os erros e acertos em:

- Falsos positivos (FP): Elementos identificados como pertencentes à classe analisada sem que essa seja a classe à qual pertencem.
- Falsos negativos (FN): Elementos pertencentes à classe analisada, mas que não foram identificados como pertencentes a ela.
- Verdadeiros positivos (TP): Elementos pertencentes à classe analisada que foram reconhecidos como elementos dessa classe.
- Verdadeiros negativos (TN): Elementos que não pertencem à classe analisada que foram identificados como elementos de outras classes.

A Figura 26 ilustra os diferentes tipos de resultado. Na imagem os círculos com preenchimento representam elementos da classe analisada e os sem preenchimento representam elementos das demais classes, o círculo preto no centro da figura delimita a região dos elementos que foram identificados como pertencentes à classe analisada. Assim, as regiões da imagem são classificadas como verdadeiros positivos (verde claro), falsos positivos (vermelho), falsos negativos (verde escuro) e verdadeiros negativos (cinza).

Figura 26 – Representação dos diferentes tipos de resultado possíveis.



Fonte: Adaptada de `https://en.wikipedia.org/wiki/F1_score`.

Essa divisão possibilita uma melhor noção da performance do classificador, pois diferentes tipos de erro têm mais importância que outros em aplicações diferentes. Em um sistema de identificação biométrica falsos positivos são extremamente prejudiciais, pois podem conceder acesso a pessoas não autorizadas e falsos negativos não tem o mesmo impacto, pois basta realizar uma nova solicitação. Já em sistemas biomédicos, falsos negativos na detecção de doenças podem causar consequências terríveis para o paciente.

Consequentemente, faz sentido a utilização de índices capazes de penalizar mais severamente o classificador a partir do tipo de erro que seja mais prejudicial. Para avaliar a performance dos classificadores projetados fez-se uso da precisão e da revocação, descritos a seguir e ilustrados pela Figura 27.

3.4.1 Precisão

É uma medida que varia entre 0 e 1, sendo 1 correspondente a uma performance ideal. Indica a taxa de acerto do classificador com relação aos elementos que foram classificados como pertencentes à classe avaliada, penalizando-o pela ocorrência de falsos positivos. Pode ser calculada a partir da equação 3.1.

$$Pre = \frac{TP}{TP + FP} \tag{3.1}$$

3.4.2 Revocação

É uma medida que varia entre 0 e 1, sendo 1 correspondente a uma performance ideal. Indica a taxa de acerto do classificador com relação aos elementos que pertencem à classe avaliada, penalizando-o pela ocorrência de falsos negativos. Pode ser calculada a partir da equação 3.2.

$$Rev = \frac{TP}{TP + FN} \tag{3.2}$$

Figura 27 – Interpretação da precisão e da revocação a partir das regiões da Figura 26.

Fonte: Adaptada de `https://en.wikipedia.org/wiki/F1_score`.

3.4.3 Medida F

Esse índice também varia entre 0 e 1 e tem o valor 1 como indicação de um classificador ideal. A medida F é uma forma de se compactar os dois índices anteriores em um único, onde um número β real e positivo é escolhido de modo que o valor atribuído à revocação será β vezes maior que o atribuído à precisão.A Medida F pode ser calculado segundo a Equação 3.3.

$$F_\beta = (1 + \beta^2).\frac{Pre.Rev}{(\beta^2.Pre) + Rev} \tag{3.3}$$

Valores comuns de β são $\frac{1}{2}$, 1 e 2, produzindo as medidas $F_{\frac{1}{2}}$, F_1 e F_2. Neste trabalho optou-se por atribuir o mesmo peso aos falsos positivos e falsos negativos, de modo que utilizou-se a medida F_1. Observa-se, porém, que durante o treinamento foi crucial observar os valores de precisão e revocação de cada uma das classes para que se pudesse determinar em quais classes as CNNs estavam apresentando maior dificuldade e os problemas que deveriam ser tratados.

3.4.4 Síntese dos Resultados

A Tabela 5 fornece as médias de cada uma das métricas apresentadas para cada uma das CNNs implementadas. Observa-se que a tabela apresenta para cada classificador a média dos índices extraídos de cada uma de suas classes, de modo a compactar os resultados sem que haja significativa perda de informações.

Pela tabela, verifica-se que com exceção da ModelNet, os modelos apresentaram todos os índices acima de 90% de modo que foram julgados satisfatórios. A performance da ModelNet foi abaixo do desejado, de fato alguns problemas inatos da tarefa tornaram a abordagem utilizada insuficiente, levando à necessidade de se realizar alguns ajustes durante o desenvolvimento do *WebApp* que serão descritos na próxima seção. Ao final foi possível melhorar a classificação de modelo e obter uma medida F_1 de 87%.

Tabela 5 – Resultados das redes quando avaliadas no conjunto de testes.

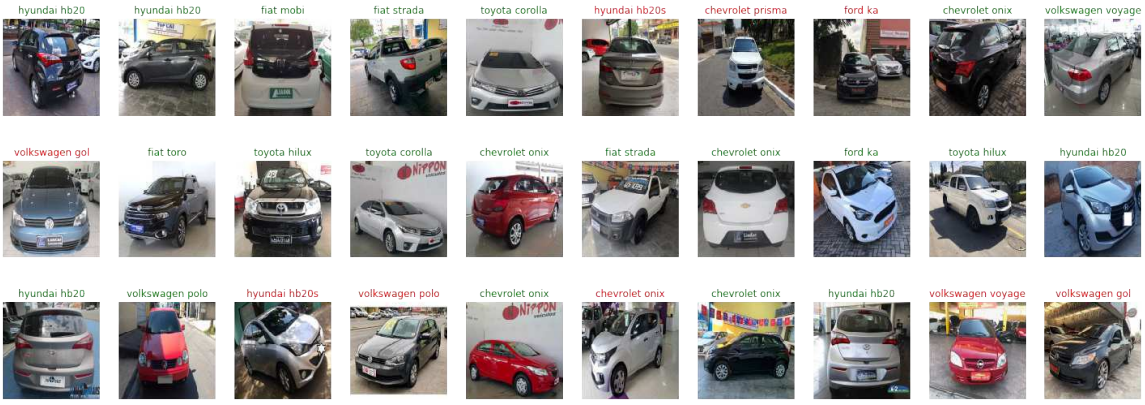
Classificador	**Acurácia**	**Precisão**	**Revocação**	**Medida F_1**
ModelNet	66,0%	69,0%	68,0%	66,0%
ColorNet	90,25%	90,0%	90,0%	90,0%
FarolNet	92,6%	93,0%	93,0%	93,0%
LanternaNet	93,9%	94,0%	93,0%	93,0%
FreioNet	92,0%	92,0%	92,0%	92,0%

Fonte: Próprio autor.

As Figuras 29 a 32 ilustram alguns exemplos de imagens pertencentes ao conjunto de testes selecionadas aleatoriamente e as saídas produzidas pelas CNNs. Imagens desse tipo eram produzidas com frequência durante o projeto, indicando acertos em verde e erros em vermelho, para que se pudesse visualizar os padrões de erro, identificados como:

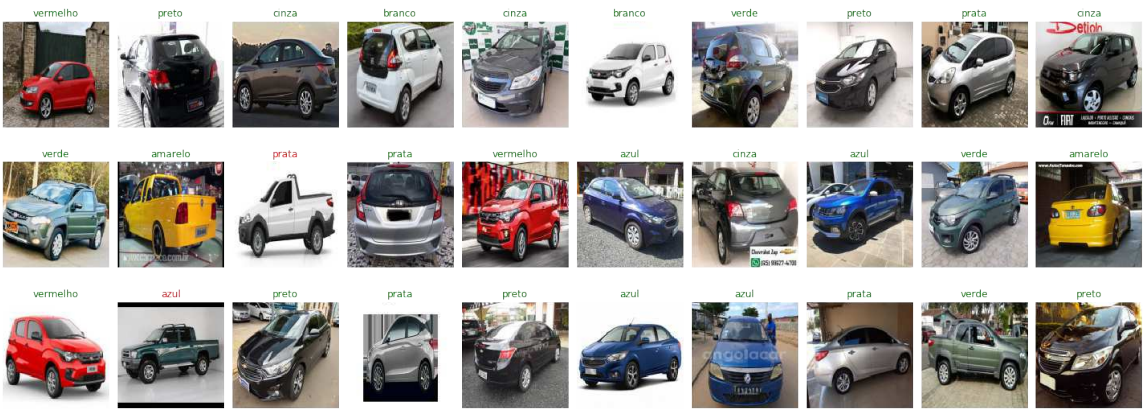
- Classificação de modelo: Confusões entre modelos semelhantes, como carros da Volkswagem ou modelos que também possuem versão sedan, como HB20 e HB20S.
- Classificação de cor: Erros devido a colorações semelhantes como branco e prata, preto e cinza em iluminações desfavoráveis.
- Classificação de farol: Frequentemente ocorriam erros devido ao reflexo da luz solar ou ofuscamento do farol aceso pela iluminação ambiente.
- Classificação de lanterna e freio: Problemas quando o carro era vermelho, pois a lanterna se confundia com o resto do veículo.

Figura 28 – Saída da ModelNet para imagens aleatórias.



Fonte: Próprio autor.

Figura 29 – Saída da ColorNet para imagens aleatórias.



Fonte: Próprio autor.

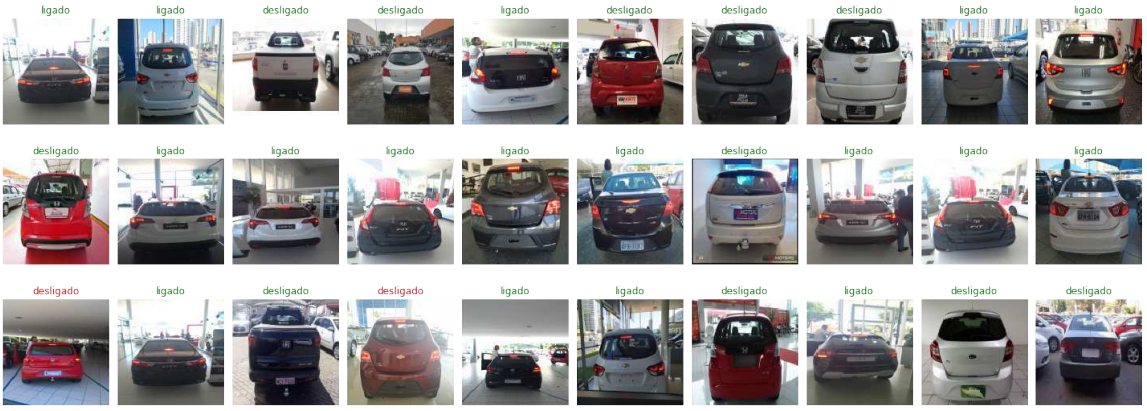
Figura 30 – Saída da FarolNet para imagens aleatórias.

Fonte: Próprio autor.

Figura 31 – Saída da LanternaNet para imagens aleatórias.

Fonte: Próprio autor.

Figura 32 – Saída da FreioNet para imagens aleatórias.



Fonte: Próprio autor.

3.4.5 Complexidade e Tempo de Resposta

As redes também foram avaliadas quanto ao tempo de resposta e sua complexidade. A complexidade foi medida com relação ao número de parâmetros treináveis, em milhões, e o tempo de resposta foi medido em milisegundos. Para minimizar o erro de medição ao determinar a latência das redes foi medido o tempo necessário para o processamento de todo o banco de testes e em seguida esse tempo foi dividido pelo número de amostras, fornecendo o tempo necessário para processar cada imagem.

Tanto o tempo de resposta quanto o número de parâmetros estão disponíveis na Tabela 6. Nas avaliações, a latência foi medida considerando o uso de uma placa gráfica (GPU) ou não, para o primeiro caso foi utilizada GPU NVIDIA GeForce GTX 1060 e para o segundo se utilizou de um processador (CPU) Dual core Intel Core i5-5200U 2.4GHz.

Comparando o tempo de resposta ao utilizar ou não placa gráfica, verifica-se que o uso de GPU produz tempos de resposta de 2 a 3 vezes mais rápidos. Isso ocorre porque esse tipo de componente é otimizado para realizar operações matemáticas de forma paralelizada, o que permite um cálculo mais veloz de operações como convoluções. O tempo necessário para passar uma imagem pelas 5 redes é 92,73ms utilizando GPU e 215,94ms utilizando CPU, permitindo que se realizasse o processo mais de 4 vezes em um segundo, de modo que se considerou a latência das redes satisfatória.

A ModelNet foi a rede que apresentou o maior número de parâmetros e maior tempo de resposta, conforme o esperado já que essa era a CNN mais complexa. Comparando a performance da ModelNet e da ColorNet, com as demais fica evidente o quanto a arquitetura da *MobileNet* é eficiente, pois redes apresentam um número muito maior de camadas e, apesar disso, não apresentam número de parâmetros e nem tempo de resposta consideravelmente maior.

Conforme a maior parte dos parâmetros se concentra nas camadas totalmente conectadas das redes convolucionais, podia-se esperar a grande diferença no número de parâmetros entre a FarolNet e a FreioNet, mesmo embora essa última seja mais profunda. A rede responsável pelos faróis produz um vetor de 4608 elementos ao fim de sua por-

Tabela 6 – Complexidade e tempo de resposta das CNNs projetadas.

Classificador	**Tempo de Resposta GPU (ms)**	**Tempo de Resposta CPU (ms)**	**Parâmetros** (10^6)
ModelNet	26,41	69,72	3,40
ColorNet	23,76	58,04	3,26
FarolNet	10,87	23,11	0,94
LanternaNet	26,30	49,02	1,57
FreioNet	5,39	16,05	0,62

Fonte: Próprio autor.

ção convolucional, fazendo com que a camada seguinte precise de muitos pesos em seus neurônios, enquanto FreioNet produz um vetor de apenas 512 elementos. A LanternaNet além de produzir um vetor de mais de mil elementos utilizou filtros 5×5, aumentando ainda mais seu número de parâmetros.

3.5 *Webapp*

Dois problemas ainda precisavam ser resolvidos antes de implementar o aplicativo: melhorar a classificação de modelo e lidar com a falsa detecção do acionamento do freio quando a lanterna estava acionada. Focando primeiro na classificação de modelo, observou-se que algumas classes apresentavam tal grau de similaridade que vistos de certos ângulos era muito difícil diferenciá-los, a exemplo de carros como HB20 e HB20S (Figura 33), em que a diferença se dá apenas na porção traseira do veículo. Consequentemente, julgou-se necessário que a rede pudesse visualizar o veículo por pelo menos três ângulos, vistas frontal, lateral e traseira, para poder realizar a classificação com exatidão.

Para que todas essas vistas fossem consideradas foi implementado um sistema de votação. O sistema receberia agora três entradas, fotos de um mesmo carro tiradas de ângulos distintos, e a partir de uma média ponderada das respostas fornecidas pela ModelNet se determinaria o modelo do veículo com exatidão. Ao realizar diversos testes conseguiu-se elevar a medida F_1 da classificação de modelo de 66,0% para 87,0% ao atribuir um maior peso ao resultado da vista traseira e um menor à vista frontal. Apesar do aumento do tempo de resposta do sistema considerou-se essa melhora suficiente para o que foi proposto, de modo que o modelo será aprimorado quando se der continuidade ao projeto.

Um sistema de votação também foi empregado para a classificação de cores, me-

Figura 33 – Fotografias de um HB20, à esquerda, e um HB20S, à direita.

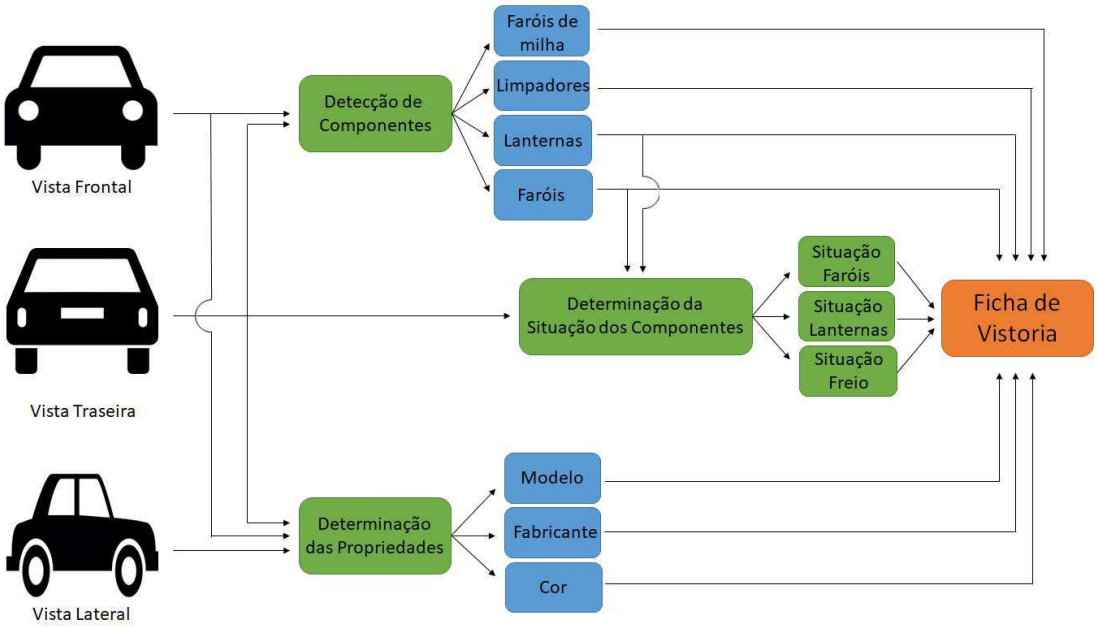
Fonte: `https://www.volanty.com`

lhorando sua medida F_1 para mais de 95%. Além disso, também foi possível tratar os problemas da FreioNet visto que faróis e lanternas não podem ser ligados de forma independente em todos os carros da base. Assim, quando os faróis estão desligados as lanternas também estão e, portanto, pode-se acionar a FreioNet apenas quando for identificado que o farol está desligado. O caso de o freio e a lanterna estarem acionados ao mesmo tempo foi desconsiderado, pois não haveria propósito para isso durante a vistoria.

Finalmente, tem-se um fluxograma do funcionamento do *WebApp* na Figura 34. Nele subentende-se que as imagens de entrada, de tamanho arbitrário, são redimensionadas para os tamanhos apropriados antes de passarem pelos blocos que foram descritos nas seções anteriores. Uma imagem da interface do aplicativo pode ser vista na Figura 35, nela pode-se realizar o *upload* de imagens referentes às três vistas do veículo através dos botões "Frente", "Lateral"e "Traseira", uma vez que as imagens são enviadas pode-se solicitar a vistoria ao clicar no botão "Vistoria".

O sistema demora aproximadamente 30 segundos para realizar a vistoria utilizando GPU (GTX 1060) e cerca de 3 minutos utilizando CPU. A maior parte desse tempo se deve à rede de detecção, que consome sozinha aproximadamente 80% do tempo. Uma vez que a vistoria é realizada o site produz uma ficha indicando as informações obtidas (Figura 36) e as imagens frontal e traseira com os componentes detectados (Figura 37). São fornecidas informações como a quantidade de cada componente detectado e as propriedades como modelo, cor e situação dos faróis, lanternas e freio em conjunto com a confiança do sistema na veracidade da detecção ou classificação.

Figura 34 – Fluxograma do *WebApp*.



Fonte: Próprio autor.

Figura 35 – Interface do *WebApp*.

Fonte: Próprio autor.

Figura 36 – Informações determinadas pelo aplicativo a partir das fotos de um prisma prata disponíveis na Figura 37.

Ficha de Vistoria	**Confiança**
Nome: chevrolet prisma	0.9904926
Cor: prata	0.8336894
Farol Acionado: sim	0.9999999
Lanterna Traseira Acionada: não	0.9931212
Luz de Freio Acionada: não	0.99851865
Quantidade de Farois: 2	0.9944085
Quantidade de Farois de Milha: 0	0
Quantidade de Retrovisores: 2	0.9496655
Quantidade de Limpadores: 0	0
Quantidade de Lanternas: 2	0.8197901

Fonte: Próprio autor.

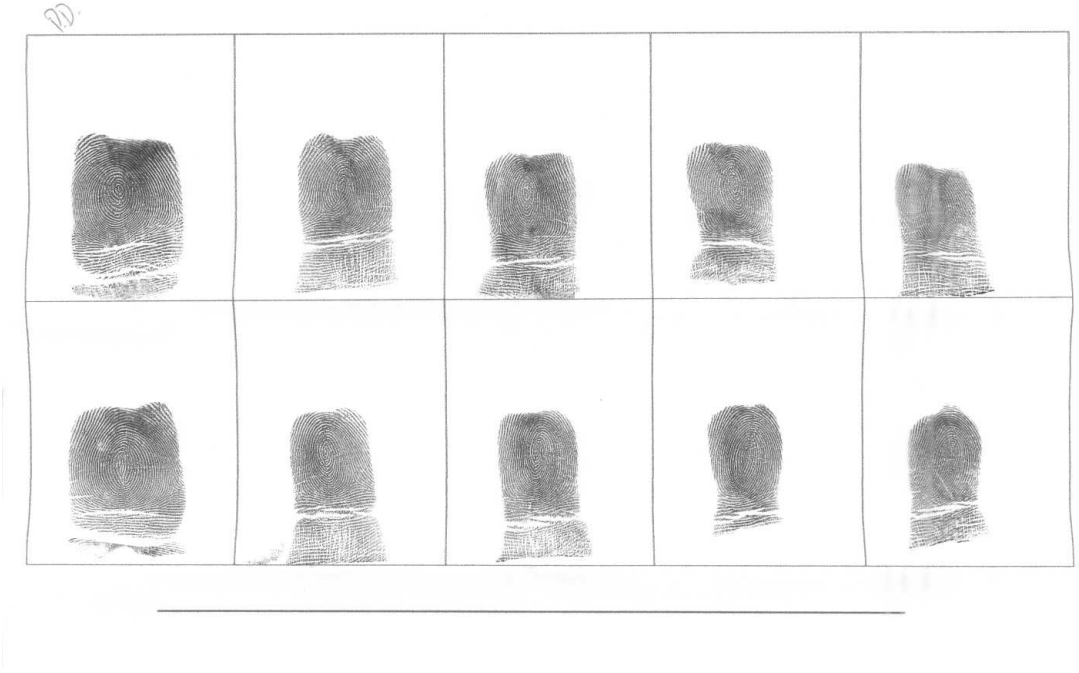
Figura 37 – Componentes detectados a partir das fotos do veículo, lanternas em amarelo, faróis em verde e retrovisores em branco.

Fonte: Próprio autor.

4 Segmentação Automática de Fichas Decadactilares

De modo a atender outra demanda da empresa, está sendo implementado um sistema para segmentação automática de fotos de fichas decadactilares, documentos que registram as digitais dos dez dedos de um indivíduo e permitem sua identificação biométrica, tem-se um exemplo na Figura 38. O sistema será implementado utilizando técnicas clássicas de visão computacional e espera-se que ele permita uma rápida digitalização dos bancos de digitais oriundos de regiões com acesso restrito à tecnologia, não possuindo sensores especializados para a captura direta das digitais ou para a digitalização das fichas já existentes.

Figura 38 – Exemplo de ficha decadactilar.



Fonte: Próprio autor.

O projeto foi iniciado no dia 03/12/2019 e está atualmente em andamento. Até o momento foi adotada uma metodologia e uma versão inicial do algoritmo já foi implementada utilizando OpenCV C++, resta adaptar o código já existente para um aplicativo *mobile*. Até o momento o sistema funciona a partir da detecção de marcações posicionadas nas extremidades da ficha, permitindo que se localizem as posições ocupadas pelas digitais e posteriormente se realize o corte da imagem.

4.1 Banco de Dados

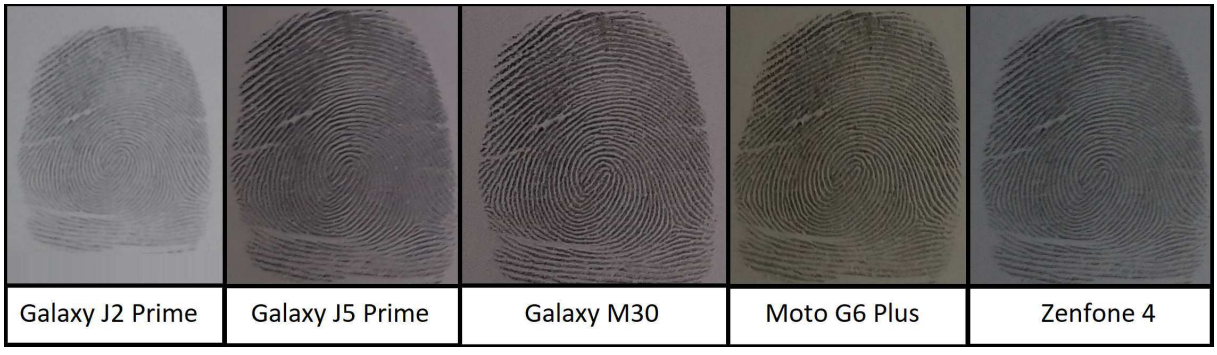
Partiu-se de um conjunto de fichas decadactilares produzido pela empresa anteriormente para dar início a este projeto. Esse banco de dados continha 6 amostras com as digitais de 23 dos funcionários, totalizando 138 fichas disponíveis.

Como o sistema será implementado em formato *mobile* foi necessário identificar modelos de telefone cuja câmera era capaz de proporcionar fotos com qualidade suficiente das digitais. Consequentemente, foi montado um novo banco de dados com fotos das fichas decadactilares e, para tal, foram utilizados cinco modelos de celulares distintos, sendo eles:

- *Moto G6 Plus*: Câmera de 12 megapixels.
- *Galaxy J2 Prime*: Câmera de 8 megapixels.
- *Galaxy J5 Prime*: Câmera de 13 megapixels.
- *Galaxy M30*: Câmera de 13 megapixels.
- *Zenfone 4*: Câmera de 12 megapixels.

Como pode ser visto na Figura 39, com exceção do *Galaxy J2 Prime*, os celulares utilizados foram capazes de tirar fotos com qualidade suficiente para que fosse possível visualizar grande parte dos padrões das digitais. De fato, após realizar testes com digitais cortadas manualmente foi verificado que o sistema utilizado pela Vsoft era capaz de identificar se duas amostras de digitais obtidas a partir de fotografias eram ou não referentes ao mesmo dedo.

Figura 39 – Diferenças nas digitais coletadas a partir de modelos distintos de celulares.



Fonte: Próprio autor.

4.2 Metodologia

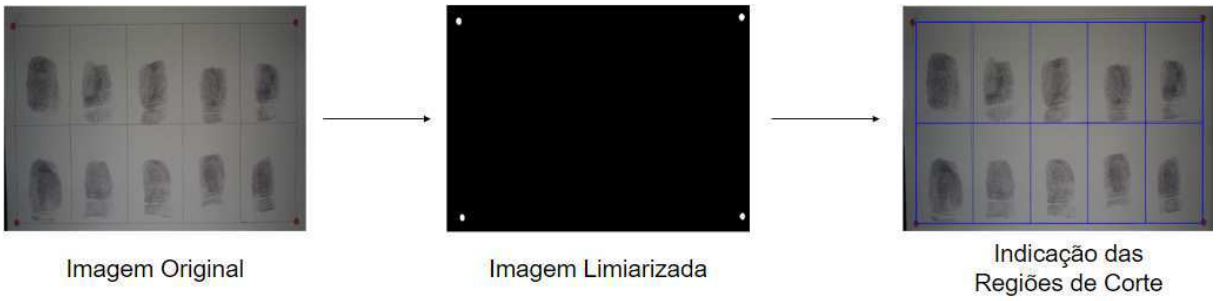
Para que seja possível realizar o pareamento das digitais coletadas é necessário que elas apresentem resolução de 500 DPI, ou seja, uma polegada na imagem deve ser representada por 500 *pixels*. Para garantir essas dimensões, utilizou-se do fato de as fichas utilizadas apresentarem tamanhos conhecidos, 10×20 centímetros ao todo, permitindo que se inferisse o DPI das imagens a partir dos pontos localizados. Uma vez determinado o DPI é possível redimensonar a imagem para que ela apresente a resolução desejada.

Foram utilizados círculos vermelhos posicionados nas extremidades da ficha como marcações. Elas foram detectadas a partir de uma limiarização no espaço de cores HSV e, para tal, foi determinando a região desse espaço que continha as cores das marcações, permitindo que se filtrasse a foto da ficha. Após esse processo, tem-se uma imagem limiazarada que contém apenas os pontos da imagem original cujas cores se localizavam na região especificada do espaço HSV.

Embora esse processo não seja livre de ruído, a limiarização elimina uma porção suficiente da imagem para que um simples detector de manchas, como a classe *SimpleBlobDetector* do OpenCV, possa encontrar a posição central das marcações. O processo está ilustrado na Figura 40, onde pode-se ver, no centro, a imagem binarizada produzida pela limiarização, sendo os *pixels* brancos os pontos que não foram eliminados e que serão analisados pelo detector.

Dispondo das coordenadas centrais das marcações foi possível determinar os limites e dimensões da região que engloba todas as digitais. Como cada digital ocupa um espaço de tamanho padronizado, encontrar as coordenadas dos retângulos que as englobam é uma tarefa simples, ilustrada na Figura 40, à direita. Os pontos detectados são utilizados para determinar o canto superior esquerdo, a altura e a largura da região de interesse. Assim, as regiões de corte podem ser determinados a partir de incrementos de um quinto da largura e um meio da altura partindo das coordenadas do canto superior esquerdo.

Figura 40 – Operações realizadas para a segmentação das digitais.



Fonte: Próprio autor.

4.3 Resultados Parciais

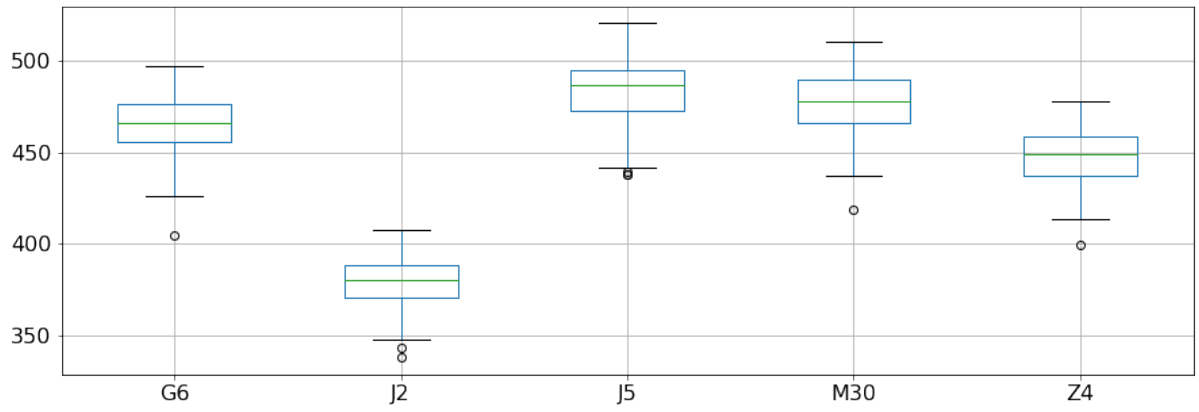
Até o momento de escrita deste relatório o projeto ainda estava em andamento, de modo que os resultados aqui apresentados são parciais e não envolvem a parte do projeto implementada em *Android*. Foram realizadas avaliações com relação ao DPI das imagens produzidas pelos modelos de celular utilizados e com relação à capacidade do algoritmo implementado de extrair corretamente a região de interesse de cada foto.

4.3.1 Viabilidade dos Celulares

Para avaliar a viabilidade de se utilizar as fotos dos celulares mencionados anteriormente, foi calculado o DPI de todas as imagens coletadas e a distribuição dos valores obtidos pode ser vista no diagrama de caixa da Figura 41. No gráfico, a linha verde de cada coluna representa a mediana dos DPIs obtidos para cada um dos celulares, sendo as linhas azuis acima e abaixo desta respectivamente o terceiro e primeiro quartis, de modo que a região delimitada por elas contém 50% das amostras. As linhas pretas são escolhidas de modo a englobar a imensa maioria dos resultados, deixando de fora apenas os valores que fogem aos valores típicos da distribuição, representados por círculos.

Uma análise do diagrama de caixa deixa evidente que as fotos do *Galaxy J2 Prime* apresentam qualidade consideravelmente inferior, visto que seus resultados encontram-se distruibuídos em uma região consideravelmente abaixo de 500 DPI. Os demais celulares apresentaram boa performance, conforme poderia-se esperar a partir da Figura 39, sendo os modelos *Galaxy J5 Prime* e *Galaxy M30* responsáveis pelas melhores imagens. Determinou-se, então, que o sistema atenderia celulares cujas câmeras apresentassem resolução de pelo menos 12 *megapixels*, garantindo assim que as digitais coletadas tivessem qualidade próxima do desejado.

Figura 41 – Distribuição dos DPIs nas fotos de cada celular.



Fonte: Próprio autor.

4.3.2 Coeficiente de Similaridade de Jaccard

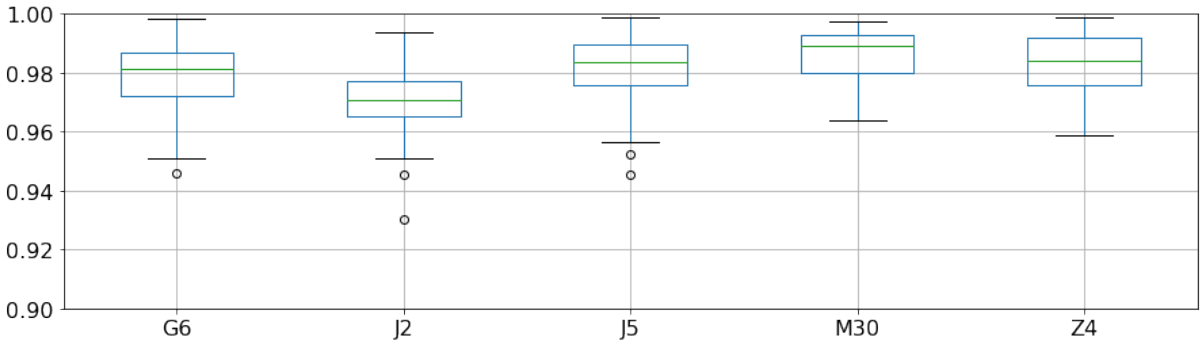
Esse índice foi utilizado para avaliar a performance do algoritmo quanto à identificação da região de interesse, ou seja, da ficha decadactilar. Esse índice, que também é chamado de IoU (*Intersection over Union*) é computado a partir de duas regiões, uma correspondente à porção detectada pelo algoritmo e outra correspondente à referência, de modo que representa a razão entre a interseção e a união das duas regiões. No caso deste projeto, a referência é a própria ficha decadactilar e a porção detectada é o retângulo montado pelo algoritmo para extrair as digitais.

$$IoU = \frac{A_{D \cap R}}{A_{D \cup R}} = \frac{A_{D \cap R}}{A_D - A_{D \cap R} + A_R} \tag{4.1}$$

O IoU é calculado segundo a Equação 4.1, onde A_D e A_R indicam as áreas detectadas e de referência, respectivamente e $\cap$ e $\cup$ indicam respectivamente as operações de interseção e união. Esse índice fornece uma pontuação ideal quando a interseção e a união de ambas as regiões apresentarem mesma área, o que só acontece caso ambas sejam a mesma região. Por outro lado, pontuações baixas, ≈ 0, são obtidos se não houver interseção, se o detector localizar apenas uma pequena porção muito pequena da referência ou se uma região muito maior que a referência for retornada. Assim, quanto mais próximo o IoU estiver da unidade, mais próxima da referência será a detecção do sistema.

Os resultados obtidos estão disponíveis no diagrama de caixa da Figura 42 para todos os modelos de celular testados. Observa-se que todos os IoU calculados, independentemente do celular, foram superiores a 0,94, de modo que pode-se inferir que o segmentador projetado funcionou conforme desejado. Pode-se ver também que os resultados para o celular *Galaxy J2 Prime* se distribuíram por uma faixa de valores levemente mais baixos que os demais aparelhos, o que indica que fotos tiradas em baixa resolução podem prejudicar a performance da aplicação.

Figura 42 – Distribuição dos IoU obtidos por modelo de celular.



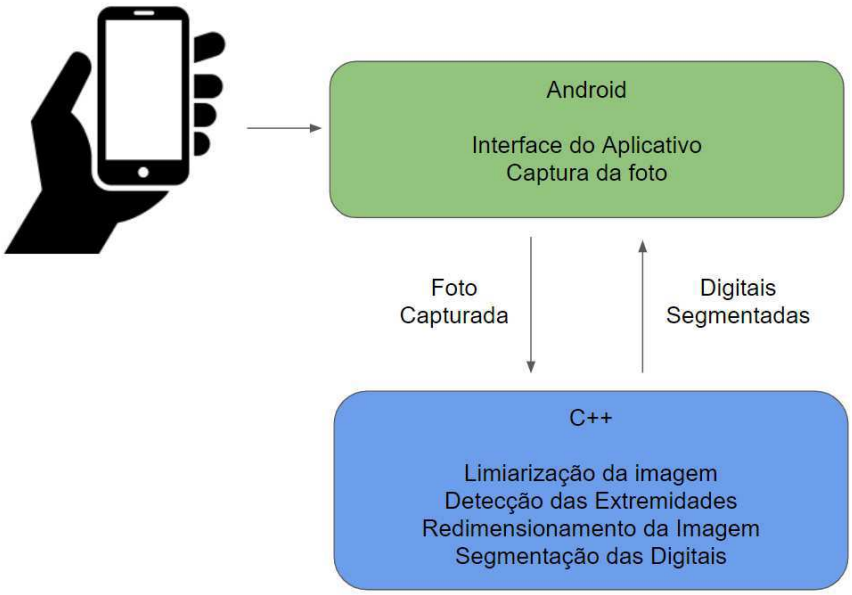
Fonte: Próprio autor.

4.4 Aplicativo *Mobile*

Conforme mencionado, o projeto será finalizado com a concepção de um aplicativo que está atualmente em produção. Essa aplicação terá uma interface simples e a parte executada no sistema operacional *Android* servirá meramente para a captura de fotos das fichas decadactilares, sendo o restante do processamento realizado em C++ como pode ser visto na Figura 43.

O aplicativo já está em desenvolvimento e tem previsão de conclusão para o mês de Fevereiro. Embora seja consideravelmente simples, realizar a comunicação entre a porção do sistema que roda em Android e o restante do sistema em C++ tem se mostrado uma tarefa não trivial porque aplicações em C++ não podem ser executadas diretamente no Android, sendo necessário um uso de um compilador cruzado (*cross-compiler*) para que o código seja executado.

Figura 43 – Fluxograma da aplicação *Mobile*.



Fonte: Próprio autor.

5 Considerações Finais

Neste estágio foram implementados dois projetos na área de Visão Computacional, sendo eles um sistema de inspeção veicular e um segmentador automático de fichas decadactilares. O desenvolvimento desses projetos exigiu conhecimentos adquiridos ao longo de toda a graduação em Engenharia Elétrica, em especial as disciplinas de Álgebra Linear, Cálculo, Processamento Digital de Imagem e Técnicas de Programação. Além disso, foram pertinentes estudos mais profundos sobre Visão Computacional e *Deep Learning*.

Além de fornecer o estímulo à pesquisa, a experiência proporcionou a imersão do estagiário no ambiente da empresa, permitindo que se trabalhasse diretamente com sua equipe de pesquisa. Considera-se essa oportunidade de se aproximar do ambiente empresarial uma experiência indispensável para a formação profissional, pois evidencia os pontos fortes e fracos da formação do estagiário frente ao mercado de trabalho, lhe permitindo reforçar suas deficiências e adquirir novas qualificações.

O contato mais prático com o projeto de arquiteturas para Redes Neurais Convolucionais evidenciou algumas características dessa área. Ficou claro o potencial de expansão e versatilidade que *Deep Learning* possui e como pode ser muito atrativo para pesquisadores e empresas dos mais diversos segmentos. Foi observado como mudanças simples na arquitetura da rede podem ter grandes impactos no processo de treinamento e na capacidade de aprendizado da rede. Também ficou evidente que frequentemente é mais importante trabalhar na construção de um bom *dataset* e no pré-processamento dos dados do que na própria arquitetura da rede para alcançar os resultados desejados.

Também ficou claro que apesar da revolução causada pelas Redes Neurais na ciência de Visão Computacional, as técnicas clássicas ainda tem sua importância e podem ser utilizadas para a concepção de projetos relevantes. Por vezes o uso de inteligência artificial é um exagero ou, mesmo em casos em que são empregadas técnicas de *Deep Learning*, a realização de um pré-processamento das imagens pode simplificar a arquitetura e reduzir o tempo de resposta do sistema de forma considerável, configurando situações em que ter noções sobre processamento de imagens é bastante conveniente.

Uma última observação a ser realizada é que a grande vantagem de algoritmos de *Deep Learning* é permitir que o sistema suporte um conjunto de dados bem mais genérico como entrada, algo raramente alcançado por técnicas clássicas de Visão Computacional. No entanto, nem sempre os recursos disponíveis para o projeto são compatíveis com sua utilização, de modo que limitações como a ausência de um banco de dados grande ou dos recursos computacionais necessários para rodar a aplicação num tempo aceitável podem fazer com que seja preferível o uso de alternativas menos custosas.

Bibliografia

DUMOULIN, V.; VISIN, F. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016. Citado 2 vezes nas páginas 24 e 25.

FRANCOIS, C. *Deep learning with Python*. [S.l.]: Manning Publications Company, 2017. Citado 6 vezes nas páginas 14, 16, 17, 33, 34 e 35.

FUKUSHIMA, K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, Elsevier, v. 1, n. 2, p. 119–130, 1988. Citado na página 23.

GÉRON, A. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. [S.l.]: "O'Reilly Media, Inc.", 2017. Citado na página 14.

HE, K. et al. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778. Citado na página 35.

HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. Citado 3 vezes nas páginas 27, 28 e 35.

KHAN, S. et al. A guide to convolutional neural networks for computer vision. *Synthesis Lectures on Computer Vision*, Morgan & Claypool Publishers, v. 8, n. 1, p. 1–207, 2018. Citado 2 vezes nas páginas 15 e 24.

LECUN, Y. et al. Handwritten digit recognition with a back-propagation network. In: *Advances in neural information processing systems*. [S.l.: s.n.], 1990. p. 396–404. Citado na página 23.

RASCHKA, S.; MIRJALILI, V. *Python machine learning*. [S.l.]: Packt Publishing Ltd, 2017. Citado na página 26.

ROSEBROCK, A. Deep learning for computer vision with python. pyimagesearch. 2017. Citado 3 vezes nas páginas 17, 25 e 26.

RUDER, S. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017. Citado na página 36.

SAMUEL, A. *Some studies in machine learning using the game of checkers. Reprinted in EA Feigenbaum & J. Feldman (Eds.)(1963). Computers and thought*. [S.l.]: McGraw-Hill, 1959. Citado na página 14.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. Citado na página 35.

STOKES, M. et al. A standard default color space for the internet-srgb. *Microsoft and Hewlett-Packard Joint Report*, 1996. Citado na página 12.

TOET, A. Multiscale color image enhancement. *Pattern Recognition Letters*, Elsevier, v. 13, n. 3, p. 167–174, 1992. Citado na página 12.

WU, C.-E.; CHAN, Y.-M.; HEALTHCARE, A. V. On merging mobilenets for efficient multitask inference. 2019. Citado na página 36.

YANG, L. et al. A large-scale car dataset for fine-grained categorization and verification. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2015. p. 3973–3981. Citado na página 31.

YU, F. et al. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018. Citado na página 32.