PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS

ESCOLA POLITÉCNICA

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CLASSIFICAÇÃO DE PLACAS DE TRÂNSITO UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

GUILHERME LOPES DO NASCIMENTO

GOIÂNIA
2021

GUILHERME LOPES DO NASCIMENTO

CLASSIFICAÇÃO DE PLACAS DE TRÂNSITO UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

Trabalho de Conclusão de Curso apresentada à Escola Politécnica, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Orientador(a): Prof. Me. Max Gontijo de Oliveira

GOIÂNIA
2021

GUILHERME LOPES DO NASCIMENTO

CLASSIFICAÇÃO DE PLACAS DE TRÂNSITO UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

Este Trabalho de Conclusão de Curso julgado adequado para obtenção o título de Bacharel em Ciência da Computação, e aprovado em sua forma final pela Escola Politécnica, da Pontifícia Universidade Católica de Goiás, em ____/____/_____.

_____
Profª. Ma. Ludmilla Reis Pinheiro dos Santos
Coordenador(a) de Trabalho de Conclusão de Curso

Banca examinadora:

_____
Orientador: Prof. Me. Max Gontijo de Oliveira

_____
Prof. Me. Fernando Gonçalves Abadia

_____
Profª. Ma. Lucília Ribeiro

GOIÂNIA
2021

Dedico este trabalho a minha família, meus amigos e colegas que sempre me apoiaram nos momentos que mais precisei.

AGRADECIMENTOS

Agradeço a Deus pelo dom da vida, por sempre me proteger, por me abençoar todos os dias com saúde e força para enfrentar os desafios e dificuldades, principalmente nestes últimos cinco anos. Pelas oportunidades que recebi e também pelos objetivos alcançados até o presente momento.

Agradeço à minha família, meus pais Marlene e Lindomar, minha irmã Geovanna, por todo apoio prestado e também pela compreensão neste período. Também faço um agradecimento especial às minhas avós Carmozina e Iraci (in memoriam) por todo carinho e por todos seus valiosos conselhos.

À Pontifícia Universidade Católica de Goiás, juntamente aos professores que fizeram parte desta jornada, por toda experiência e conhecimentos passados. E também a todos os demais funcionários da instituição.

Agradeço ao meu orientador, professor Max Gontijo, por todo apoio, motivação, paciência e dedicação ao longo do desenvolvimento deste trabalho, estando sempre disponível a compartilhar todo o seu vasto conhecimento.

Por fim agradeço a todos meus amigos e colegas, principalmente os que tive a oportunidade de conhecer durante o curso, por todo apoio, colaboração e também pelos momentos de descontração. Também agradeço a todos que de alguma forma contribuíram para que eu pudesse chegar até aqui.

RESUMO

Este trabalho tem como propósito o desenvolvimento de uma Rede Neural Convolucional capaz de realizar a classificação de um conjunto de placas de trânsito brasileiras. Tendo em vista a grande evolução na área de inteligência artificial e também no setor de veículos autônomos, cada vez mais se faz necessário a implementação de novas soluções com o objetivo de resolver os diversos problemas que ainda impedem a circulação deste tipo de veículo. E um destes problemas é justamente a classificação das placas que existem ao longo do trajeto percorrido por estes veículos, para que os mesmos cumpram com as normas/orientações da via. Sabendo que erros nesta classificação podem causar acidentes, este trabalho também busca explorar diferentes testes com variações de entradas e parâmetros que são informados para a rede com a finalidade de apresentar quais delas apresentam uma taxa maior de acertos. Com base no resultado destes testes, também são sugeridos alguns pontos a se desenvolver em trabalhos futuros.

Palavras-chave: Rede Neural Convolucional. Placas de Trânsito. Inteligência Artificial. Classificação de Imagens.

ABSTRACT

This work aims to develop a Convolutional Neural Network capable of classifying a set of Brazilian traffic signs. In view of the great evolution in the field of artificial intelligence and also in the autonomous vehicle sector, it is increasingly necessary to implement new solutions in order to solve the various problems that still impede the circulation of this type of vehicle. And one of these problems is precisely the classification of the signs that exist along the route taken by these vehicles, so that they comply with the rules/guidelines of the road. Knowing that errors in this classification can cause accidents, this work also seeks to explore different tests with variations of inputs and parameters that are reported to the network in order to show which of them have a higher rate of correct answers. Based on the results of these tests, some points to be developed in future work are also suggested.

Keywords: Convolutional Neural Network. Traffic Signs. Artificial Intelligence. Image Classification

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	*Application Programming Interface*
CNN	*Convolutional Neural Network*
IA	Inteligência Artificial
MLP	*Multilayer Perceptron*
N/A	*Not Applicable*
ReLU	*Rectified Linear Unit*
RGB	*Red, Green and Blue*

SUMÁRIO

1. INTRODUÇÃO

Antigamente a inteligência artificial (IA) era mais conhecida pelos filmes e séries, porém nos últimos anos houve um crescimento expressivo na área da computação voltada para esta área e como consequência, esse crescimento trouxe soluções para muitos problemas que já existiam, mas que não tinham recursos para serem solucionados e também possibilitou diversas mudanças que tem como objetivo simplificar as atividades executadas no dia a dia. Desta forma, fazendo com que a IA fosse mais acessível para a população em geral, e esta foi a motivação para o tema deste trabalho.

O principal foco da inteligência artificial é resolver problemas complexos que não possuem uma solução exata, exemplo de problemas nesse escopo são a identificação de criminosos através de câmeras, a predição de valores do mercado financeiro ou até mesmo o diagnóstico médico de uma doença, dentre outros. Estes problemas são bem mais complexos dos que possuem uma solução exata, como o controle de folhas de pagamento, os cálculos para realizar o projeto de uma determinada construção, um controle de estoque, etc. (SICHMAN, 2021).

Grande parte das mudanças que estão sendo possíveis graças à Inteligência Artificial são voltadas para a automatização de atividades que são executadas de forma repetidas e que muitas das vezes consomem um tempo relevante. Como exemplo, o ato de fazer as compras em um supermercado, onde após pegar os itens desejados, muitas das vezes o cliente necessita de aguardar em uma grande fila para somente então poder efetuar o pagamento de suas compras, o que acaba sendo um problema.

Pensando nisso a empresa Amazon desenvolveu uma solução que busca justamente resolver este problema dos supermercados. Trata-se de uma loja física de conveniências montada em um prédio da empresa, na cidade de Seattle, nos Estados Unidos. A proposta da loja consiste em permitir que os clientes obtenham os produtos desejados e possam pagar por eles sem precisar passar por um caixa. Com isso, para ter acesso a loja o cliente precisará possuir um cadastro no aplicativo da loja e só então terá acesso aos produtos. Na loja existem câmeras e sensores que detectam os produtos retirados pelos clientes e também qual o cliente que retirou o produto, e esta é a parte que entra a inteligência artificial para identificar e classificar tanto o produto retirado quanto o cliente que está retirando. Após feita a detecção, o produto

é registrado na conta do cliente e ao se retirar da loja é realizada a cobrança dos produtos em seu cartão, solucionando assim o problema das filas e do registro pelo caixa. E desta forma, com o tempo outras lojas e supermercados também irão adotar essa estratégia em seus estabelecimentos.

Por fim, outro setor que está passando por grandes mudanças é o setor de veículos autônomos, no qual as grandes empresas estão em uma corrida em busca do feito de ser a primeira montadora a fabricar um carro totalmente autônomo, entre elas estão empresas como Tesla, Google, Amazon, Toyota, entre outras. Muitos testes já estão sendo feitos e existe uma expectativa para que até 2030 já existam carros deste tipo em circulação, sendo que provavelmente as empresas de transporte de passageiros como a Uber, sejam as primeiras a terem acesso a eles.

Esta evolução no ramo de automóveis, também só é possível pelo avanço dos algoritmos de inteligência artificial, pois faz parte dos problemas que não possuem uma solução exata como dito anteriormente. Principalmente neste cenário de um veículo poder circular sem que ninguém o controle, é um problema que envolve muitas variáveis, ou seja, muitas situações em que irão necessitar de uma resposta rápida do veículo, como no caso de uma pessoa correr para a frente do veículo em movimento.

Também existem os problemas mais comuns que devem ser solucionados para que o automóvel ande em segurança, nos quais são manter o veículo na faixa correta, garantir que as sinalizações sejam obedecidas por meio da identificação e classificação de placas, semáforos, quebra-molas, etc. Outro ponto importante é a capacidade do veículo desviar de obstáculos sem colocar em perigo os outros veículos ou pedestres ao seu redor.

Tendo em vista os problemas citados, foi analisado de forma mais profunda o problema em que o veículo identifica uma placa de sinalização na via, precisando classificá-la para que consiga seguir as instruções pré-definidas para a placa identificada. Desta forma, o objetivo geral deste trabalho é implementar uma rede neural convolucional capaz de classificar um conjunto de placas de trânsito brasileiras.

Já em relação aos objetivos específicos, foram definidos os seguintes:

- Desenvolver um *dataset* com um conjunto de placas de trânsito brasileiras.
- Desenvolver uma implementação que seja capaz de aprender a classificação de placas a partir do *dataset*.
- Realizar testes combinando diferentes perfis de parametrização com diferentes subconjuntos do *dataset* com algumas modificações, de forma a

encontrar os limites da solução.

1.1 Metodologia Adotada

Referindo-se a natureza deste trabalho, ele é caracterizado como uma pesquisa aplicada, já que busca apresentar os detalhes sobre os procedimentos utilizados para chegar ao objetivo final (WAZLAWICK, 2014).

Em relação aos conceitos apresentados no trabalho, é uma pesquisa bibliográfica, dado que são baseados em pesquisas realizadas em materiais disponibilizados pela comunidade acadêmica.

E por se tratar de um trabalho que realiza testes alterando os parâmetros que são recomendados nas pesquisas realizadas, também se encaixa na pesquisa experimental.

1.2 Organização dos Capítulos

Referente a estrutura, este trabalho está organizado da seguinte forma: no Capítulo 1 foi apresentado uma introdução ao que será abordado no trabalho, juntamente a motivação, o problema e os objetivos.

O Capítulo 2 apresenta o referencial teórico, no qual são explicados os conceitos que foram utilizados para desenvolver o trabalho. Assim como qual a função dos mesmos na implementação da solução.

O Capítulo 3 descreve os materiais e métodos usados, nele contém os detalhes sobre como foi estruturado o *dataset* e também sobre as tecnologias escolhidas para a implementação.

O Capítulo 4 contém todos os passos feitos durante a etapa de desenvolvimento da solução, dentre eles: a definição dos subconjuntos do *dataset*, o pré-processamento realizado, os detalhes da construção, treinamento e validação do modelo.

O Capítulo 5 apresenta o resultado dos testes realizados utilizando os diferentes perfis de parametrização e os subconjuntos do *dataset*.

O Capítulo 6 possui a conclusão do trabalho, comentando também sobre os trabalhos futuros.

2. REFERENCIAL TEÓRICO

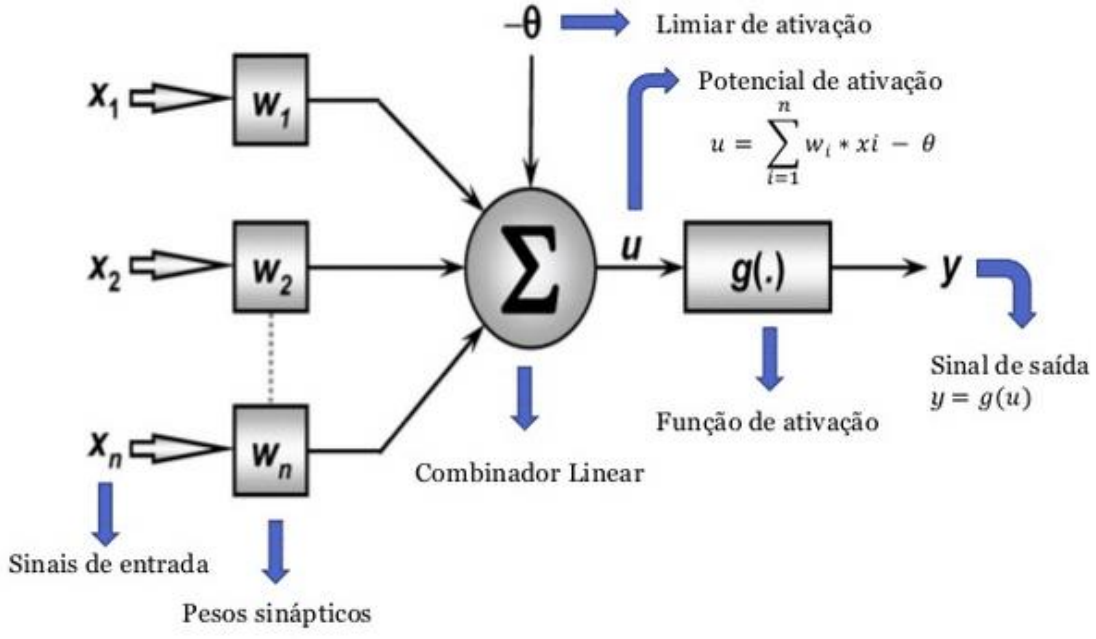
Este capítulo busca esclarecer o motivo pelo qual a Rede Neural Convolucional foi escolhida para ser utilizada neste trabalho, assim como os processos que são feitos para realizar a classificação. Também é explicado o motivo de não utilizar outros tipos de rede, como a *Perceptron* e a *Multilayer Perceptron*.

2.1 *Perceptron*

A rede *perceptron* é considerada uma das redes mais simples e desta maneira recomendada para quem está iniciando na área das redes neurais artificiais, pois permite uma melhor compreensão, ela foi idealizada em 1958 pelo cientista Frank Rosenblatt (ELIZONDO, 2006).

O *perceptron* é constituído por um modelo matemático baseado em um neurônio biológico, e com isso o modelo também pode ser referenciado como um neurônio. Ele possui uma única camada na qual pode possuir *n* entradas, para cada entrada é definido um peso que representa a importância desta entrada. As entradas, juntamente com seus respectivos pesos, passam por um combinador linear e em seguida por uma função de ativação degrau que deve retornar uma única saída com o valor de 0 ou 1, conforme é apresentado na Figura 1 a seguir.

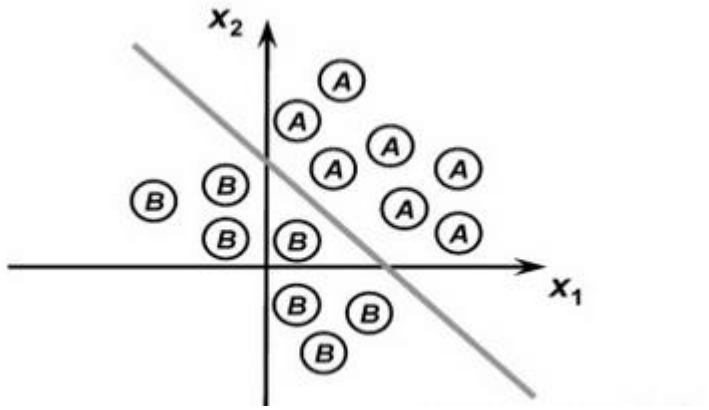
Figura 1 - Ilustração de uma rede *perceptron*



Fonte: Embarcados (2016)

O objetivo do *perceptron* é, tendo um conjunto de dados quaisquer, conseguir separá-los em duas classes distintas, desde que estas classes sejam linearmente separáveis e por este motivo ele pode ser visto como um classificador linear binário (ELIZONDO, 2006). A Figura 2 exemplifica esta classificação.

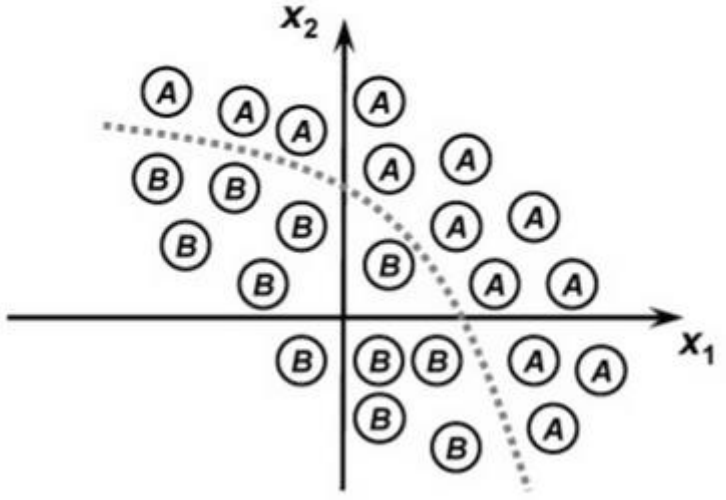
Figura 2 - Conjunto de dados linearmente separáveis



Fonte: Embarcados (2016)

Consequentemente, pelo motivo da rede *perceptron* só conseguir classificar os dados que sejam linearmente separáveis, faz com que ela seja muito limitada, pois a grande parte dos problemas não se encaixam nesta condição, e para estes problemas é indicado o uso da rede *Multilayer Perceptron*, que será abordada na seção 2.2. A Figura 3 representa um conjunto de dados que não é linearmente separável e, portanto, não pode ser classificado pelo *perceptron*.

Figura 3 - Conjunto de dados não linearmente separáveis

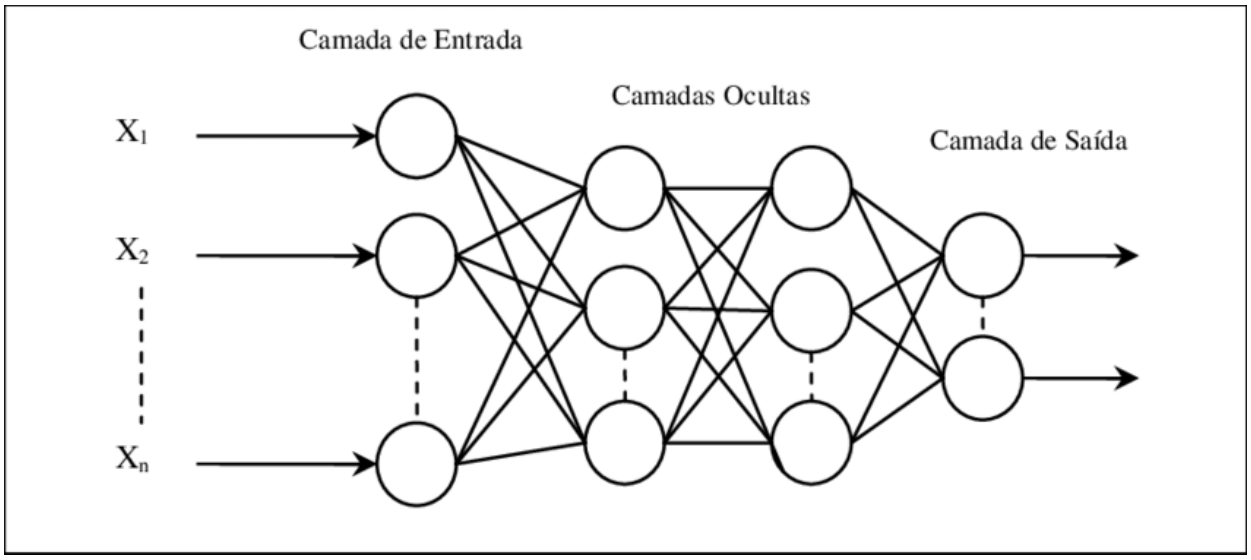


Fonte: Embarcados (2016)

2.2 *Multilayer Perceptron*

A MLP, sigla para *Multilayer Perceptron*, se trata de uma rede neural artificial totalmente conectada que consiste em vários neurônios dispostos em diversas camadas. As MLP's são compostas por três ou mais camadas que são estruturadas da seguinte maneira: uma camada é destinada à entrada de dados, uma ou mais camadas ocultas e uma camada destinada para a saída dos dados. Com exceção dos nós da camada de entrada, cada nó representa um neurônio que faz o uso de uma função de ativação, sendo que as mais utilizadas são a função sigmóide (que retorna um valor de 0 a 1) e a função tangente hiperbólica (que retorna um valor de -1 a 1) (VELO; LÓPEZ; MASEDA, 2014). A Figura 4 apresenta a estrutura de uma MLP com duas camadas ocultas.

Figura 4 - Exemplos de uma MLP com duas camadas ocultas



Fonte: SOBREIRO (2008)

Devido a MLP possuir uma grande flexibilidade de funções, ela é utilizada por diversos sistemas de reconhecimentos de padrões e também em sistemas de processamento de sinais. Em sua arquitetura as informações são transmitidas em apenas um sentido e por este motivo é chamada de *feedforward*. E o treinamento é realizado através de um algoritmo chamado *backpropagation*, em que o foco é buscar pelo mínimo local ou global da função.

O *backpropagation*, traduzido como retropropagação, é um algoritmo que é bastante utilizado no treinamento de redes neurais feedforward como a MLP, ele busca ajustar os pesos de cada neurônio da rede, para isso ele faz o cálculo do

gradiente descendente do erro em relação aos pesos. O objetivo da retropropagação é utilizar o erro da rede, que é o resultado da subtração do valor esperado pelo valor real que a rede retorna como saída, para ajustar os pesos de todos os neurônios da rede. Esse procedimento de ajuste começa da última camada e vai até a camada de entrada e por isto o nome *backpropagation*.

Com essas melhorias a *Multilayer Perceptron* consegue fazer uma classificação com dados que não precisam ser linearmente separáveis e esta é a grande vantagem da MLP quando comparada ao *Perceptron*.

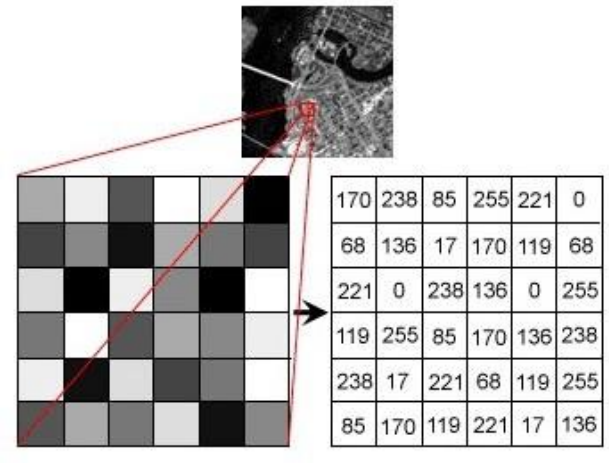
2.3 Rede Neural Convolucional

A Rede Neural Convolucional (em inglês Convolutional Neural Network, abreviado como CNN) é basicamente um algoritmo que implementa o conceito de aprendizado profundo, que busca modelar abstrações de alto nível utilizando um grafo com várias camadas, e com isso é muito utilizada para trabalhar com classificação de imagens. Resumidamente, o principal objetivo de uma CNN, é dado um *dataset* com as imagens que devem ser classificadas, extrair as características de cada imagem e após obter tais características submetê-las ao classificador (O'SHEA; NASH, 2015).

2.3.1 *Imagens*

De modo a facilitar o entendimento da operação realizada pela CNN nas imagens recebidas como entrada, é interessante ter o conhecimento de como uma imagem é tratada computacionalmente. Basicamente as imagens são divididas em dois tipos: as imagens em tons de cinza e as imagens coloridas. Na imagem em preto e branco, é utilizada uma matriz 2D para representá-la, onde cada posição da matriz corresponde a um pixel da imagem e o valor de cada pixel pode variar de 0 (que representa o pixel totalmente preto) a 255 (que representa o pixel totalmente branco) como pode ser observado na Figura 5.

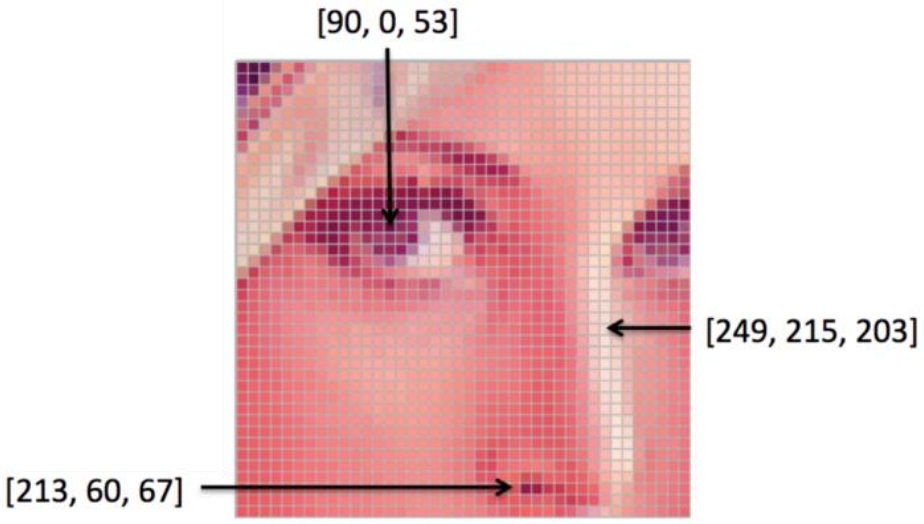
Figura 5 - Representação computacional de uma imagem em tons de cinza



Fonte: Medium (2019)

Referente às imagens coloridas, a sua representação é um pouco mais complexa pois é utilizada uma matriz 3D para comportar os dados das combinações das cores primárias vermelho, verde e azul também conhecidos como RGB (*Red, Green and Blue*). Com esta combinação é possível saber qual a cor de cada pixel e consequentemente obter a imagem completa como mostra a Figura 6.

Figura 6 - Representação computacional de uma imagem colorida



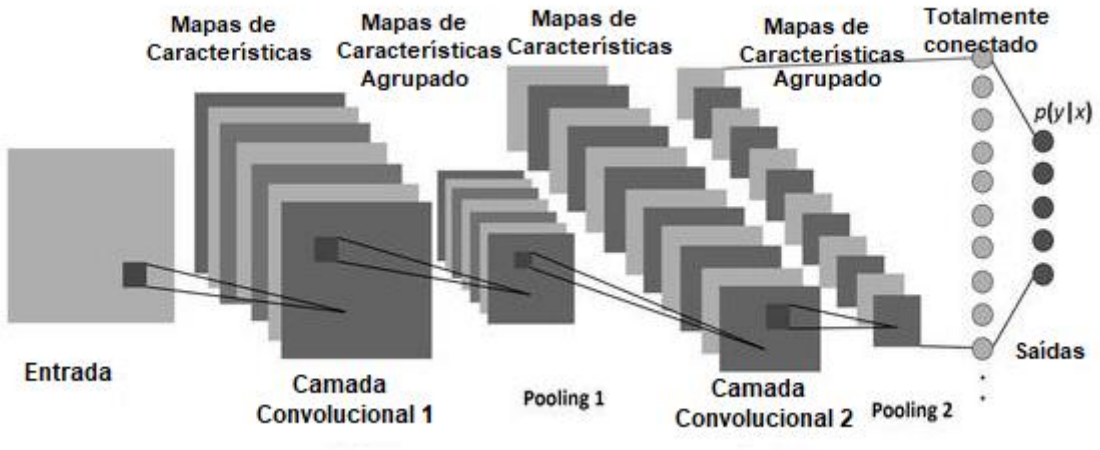
Fonte: Medium (2019)

2.3.2 *Arquitetura*

A arquitetura da CNN é semelhante ao funcionamento do cérebro humano para processar imagens, sendo toda a parte de conectividade dos neurônios e também o processo das informações inspiradas no córtex visual dos seres humanos. Também é buscado reproduzir a forma em que os neurônios respondem aos estímulos captados por seus canais de entrada.

Um dos pontos em que a CNN se destaca é na sua capacidade de continuar obedecendo a relação de vizinhança que pode ser observada nos pixels da imagem. Isso ocorre em paralelo à aplicação dos filtros durante todo o processamento da rede. A Figura 7 apresenta uma representação de como funciona o processo da rede.

Figura 7 - Ilustração do processo de uma Rede Neural Convolucional



Fonte: Adaptado de ALBELWI, MAHMOOD (2017)

2.3.3 *Convoluções*

Como principal característica, a rede neural convolucional é composta por múltiplas camadas, na qual cada uma delas possui uma função distinta e muito importante para garantir o resultado final. Essas camadas são chamadas de convoluções e são aplicadas nos dados de entrada. Elas são compostas por um grande número de neurônios, onde cada um deles é destinado a realizar um determinado filtro em uma parte específica da imagem. Com isto existe uma conexão de cada neurônio com o conjunto de pixels que vieram da camada anterior e também são atribuídos pesos para cada conexão e esses pesos juntamente com a combinação de entradas do neurônio geram uma saída para a próxima camada (ALBAWI; MOHAMMED; AL-ZAWI, 2017).

Estes pesos que são atribuídos para cada conexão dos neurônios formam uma matriz que quando multiplicados com o conjunto de pixels recebidos da imagem aplica um determinado tipo de filtro na imagem, esta matriz é também chamada de kernel ou até mesmo de filtro convolucional. Podem ser aplicadas várias matrizes, em que cada uma gera um filtro diferente para a imagem. Alguns fazem o embaçamento da imagem, outros destacam as bordas, assim possibilitando que a imagem seja mais fácil de ser processada pela rede. Na Figura 8 é apresentado alguns exemplos de kernels, utilizando a placa de proibido estacionar.

Figura 8 - Exemplos de efeitos gerados por diferentes kernels

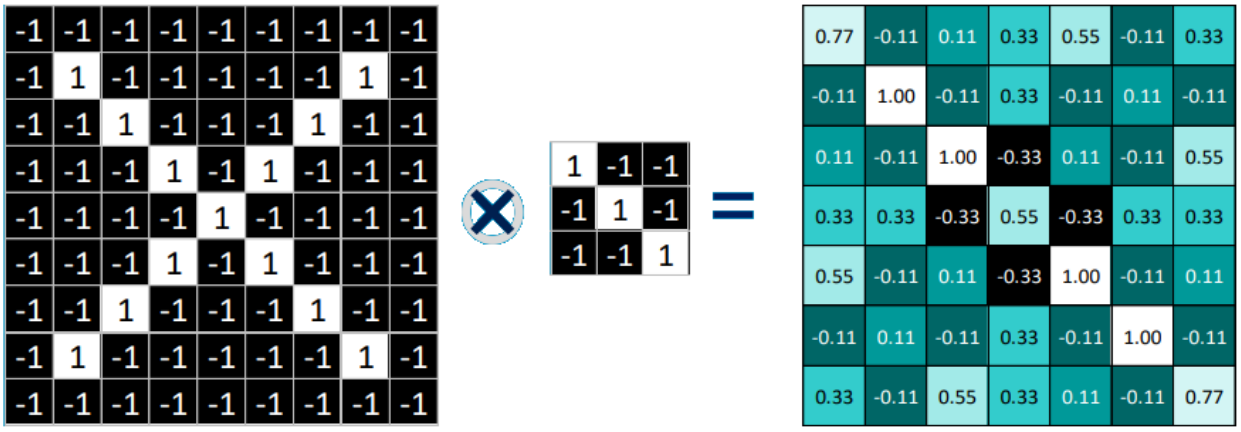
Operação	Kernel	Resultado
Identidade	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Detecção de borda	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Nitidez	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Fonte: Próprio Autor

A principal função do kernel é permitir que possam ser identificadas as características contidas na imagem e não identificar a imagem inteira. Com isso, no caso de uma rede que tenha como objetivo classificar humanos, por exemplo, os kernels não irão extrair diretamente o humano da imagem, mas sim suas características como o cabelo, os olhos, a boca, o nariz, dentre outras. Outro ponto importante é saber que essa extração de características é representada dentro kernel por uma troca de informação, ou seja, se a troca de informação da imagem observada é similar a troca de informação representada no kernel significa a detecção de uma característica (*feature*) expressa pelo kernel utilizado.

Na Figura 9 é possível observar essa extração de características utilizando o kernel. Neste caso, o objetivo é detectar se a imagem contém um *X*. Mas como descrito anteriormente, um kernel muitas das vezes não será responsável por extrair toda a imagem de uma vez, mas sim uma determinada característica. Neste exemplo o kernel utilizado irá buscar extrair uma diagonal da imagem, que é uma característica do *X*. Após convoluir o kernel por toda imagem, a matriz gerada apresenta os maiores valores justamente no local que possui essa diagonal expressa pelo kernel, e os menores valores estão localizados na diagonal oposta.

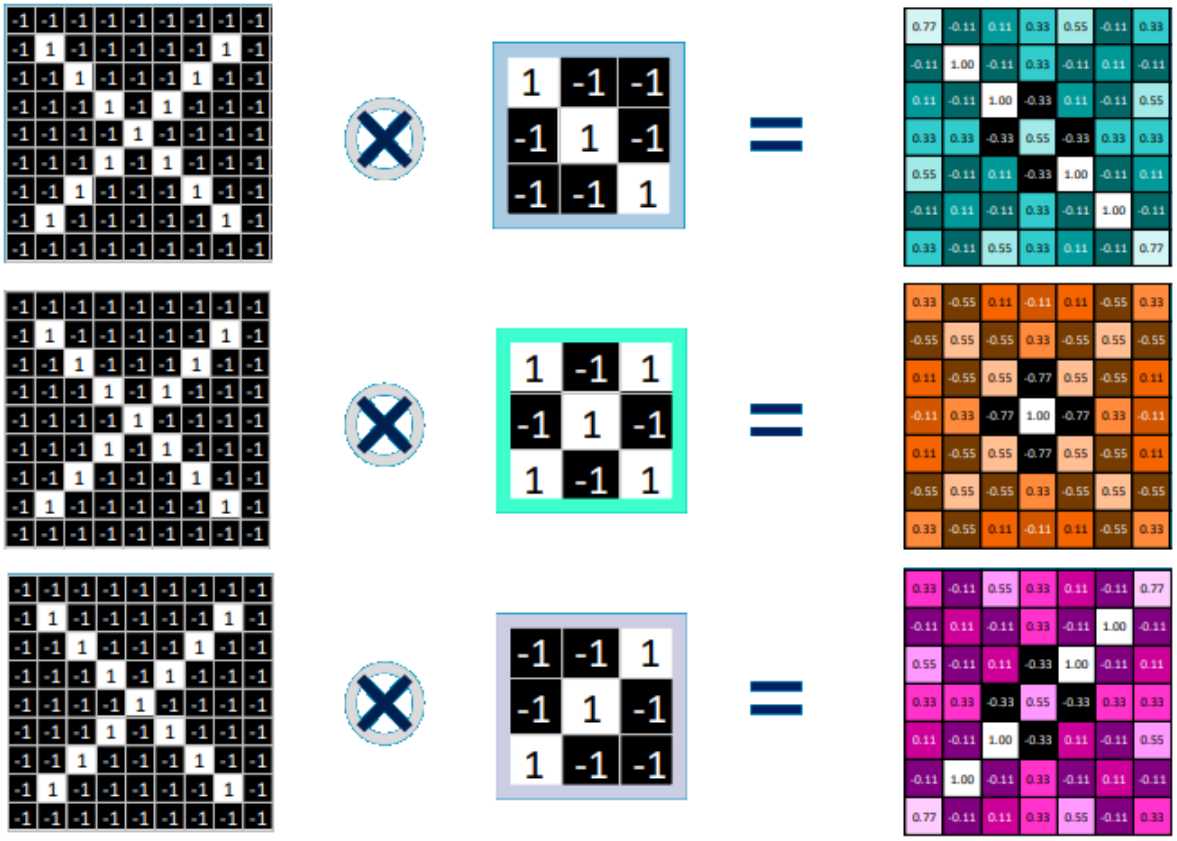
Figura 9 - Exemplo da aplicação de um kernel



Fonte: SOARES (2018)

Identificar que existe uma única diagonal na imagem é um indicativo de que ela possa ter um '*X*' mas não é o suficiente, e por isto neste exemplo também deve haver no mínimo mais dois kernels em que um seria para extrair uma diagonal oposta e o kernel seria para extrair a intersecção entre as duas diagonais. Desta maneira cada um dos kernels irá gerar uma saída conforme o filtro convolucional utilizado, ou seja, a partir de uma imagem podem ser geradas várias saídas de acordo com a quantidade de filtros convolucionais aplicados. No exemplo da classificação do '*X*' como foram usados três filtros, serão geradas três saídas conforme apresentado na Figura 10.

Figura 10 - Saídas dos filtros convolucionais



Fonte: SOARES (2018)

Diferentemente da estrutura utilizada em um *perceptron*, na qual um neurônio é ligado em todos os neurônios da camada anterior, na estrutura de uma CNN somente uma parte do conjunto de entradas é ligada em cada neurônio. Desta forma, essa mudança de estrutura permite que a CNN faça uma verificação dos campos receptivos locais. Passando a agrupar os neurônios que são da mesma camada em mapas.

Este agrupamento em mapas só é realizado em neurônios de saída que sejam responsáveis por uma mesma área da imagem e com a condição de que esta área tenha sido processada com o mesmo filtro. Nesta etapa também é realizada uma divisão dos pesos para que um determinado grupo de neurônios que esteja agrupado em um mapa consiga aplicar o mesmo filtro em partes distintas da imagem. Essa divisão faz com que a quantidade de parâmetros que devem ser processados seja menor e consequentemente o tempo de treinamento também terá reduzido.

2.3.4 *Padding*

A etapa de *padding* é essencial para corrigir um problema gerado pela etapa de convolução, em que dependendo da dimensão da imagem e do kernel a saída gerada poderá ter uma dimensão menor do que a imagem de entrada. Tendo como base uma imagem em escala de cinza com dimensão (*n x n*) e um kernel com dimensão (*k x k*) a saída gerada após a aplicação do filtro convolucional é representada pela Equação 1.

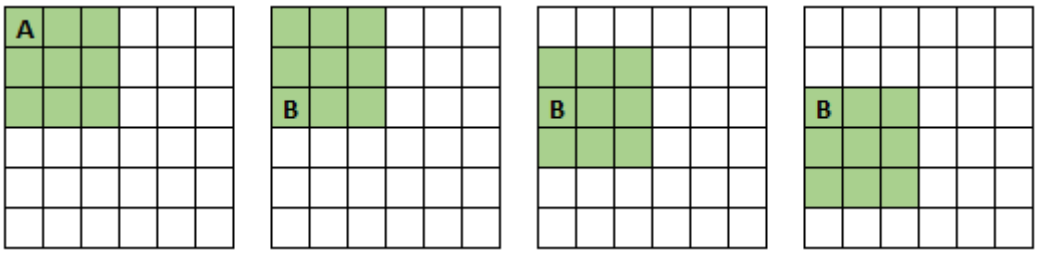
$$(n - k + 1) \times (n - k + 1) \quad (1)$$

Considerando como exemplo uma imagem que possua um tamanho 8 x 8 e um kernel com o tamanho 3 x 3, a imagem retornada após a convolução terá a dimensão 6 x 6. Portanto, toda vez em que a imagem é submetida a etapa de convolução ela irá ficar menor e isto de certa forma limita a quantidade de vezes que o processo de convolução pode ser aplicado na imagem e também faz com que não seja possível estabelecer uma rede mais profunda.

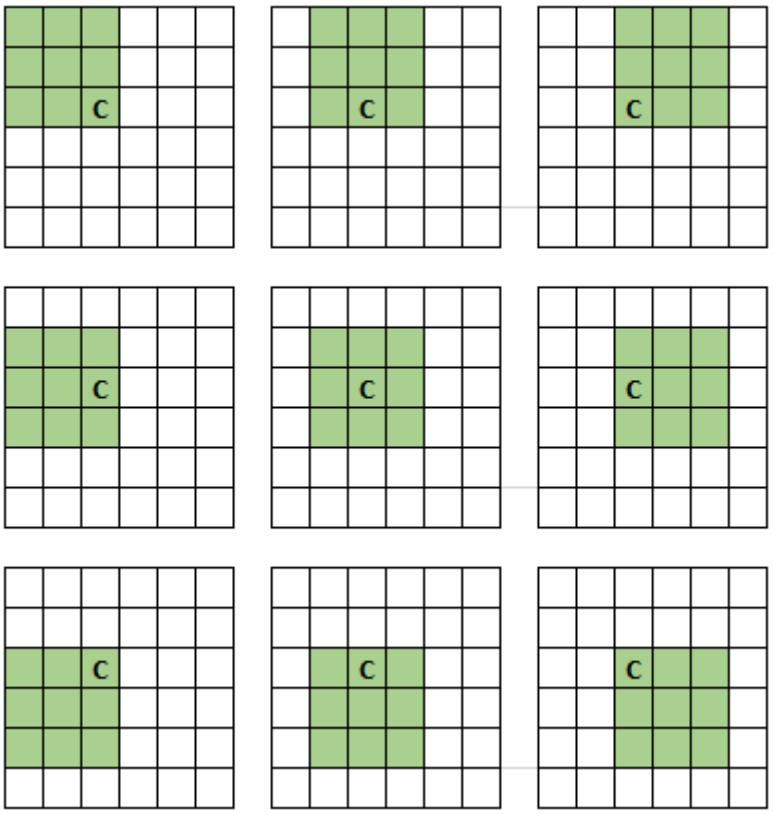
Outro problema que acontece ao aplicar filtro na imagem é de que os pixels localizados no canto e nas bordas da imagem são pouco utilizados, ao contrário dos pixels localizados na região central da imagem que são bem mais utilizados conforme exemplo ilustrado na Figura 11.

Figura 11 – Utilização dos pixels na camada de convolução

A) Pixel no canto B) Pixel na borda



C) Pixel no meio



Fonte: Adaptado de GeeksforGeeks (2021)

Como é possível visualizar no exemplo o pixel A só é utilizado em uma única operação, o pixel B é utilizado três vezes e o pixel C é utilizado nove vezes. A imagem mostra na prática que a utilização dos pixels localizados no centro é bem maior do que a utilização dos pixels que estão na borda e nos cantos. Com isso, no caso em que as bordas da imagem possuam informações relevantes para o processo de treinamento, elas não serão tão utilizadas quanto às informações no centro, podendo assim prejudicar o treinamento.

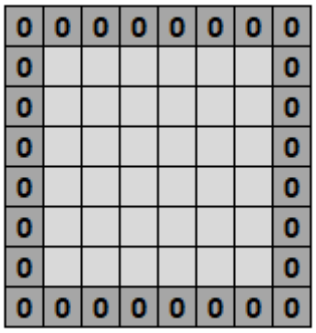
É este o problema que a etapa de *padding* (ou traduzido do inglês, preenchimento) busca solucionar. O processo é basicamente adicionar uma ou mais camadas de pixels ao redor da imagem de entrada, e cada um dos pixels adicionados irá conter o valor zero, conforme mostrado na Figura 12.

O objetivo desse procedimento é fazer com que a imagem não diminua de tamanho após ser aplicado o filtro convolucional. Considerando que p é o número de camadas adicionadas pela etapa de *padding*, o cálculo que retorna o tamanho da imagem após aplicado o filtro é representado pela Equação 2, e no exemplo de uma

imagem (*8 x 8*) com um filtro (*3 x 3*), o tamanho da imagem após a etapa de convolução permanecerá o mesmo.

$$(n + 2p - k + 1) \times (n + 2p - k + 1) \quad (2)$$

Figura 12 - Exemplo da adição da camada de *padding*



Fonte: Adaptado de GeeksforGeeks (2021)

2.3.5 *Função de Ativação*

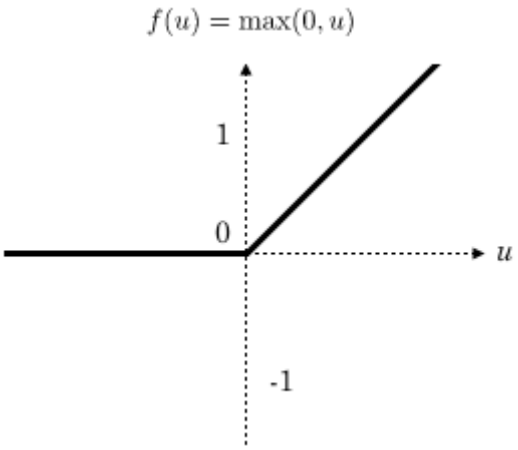
Outra etapa que é normalmente executada após a parte da convolução é a aplicação de uma função de ativação. Ela é utilizada para realizar um pequeno ajuste na saída, decidindo quais neurônios serão ativados. Portanto, nestes casos a função de ativação permite que sejam feitas pequenas modificações nos pesos e também no bias, que permite que estes ajustes sejam mais flexíveis. Esse processo é considerado um dos mais importantes para o aprendizado da rede.

Basicamente, o que a função de ativação faz é uma transformação não linear com base no que é recebido como entrada usando um certo tipo de função, e a saída produzida por essa função é passada como entrada para a camada de neurônios seguintes. Sem esta função de ativação só seria feito uma transformação linear com os pesos e o bias, o que seria mais simples, porém muito limitada.

Existem vários tipos de função de ativação, sendo que cada uma delas pode ser mais recomendada para um problema específico, com o objetivo de permitir que seja possível resolver mais problemas complexos. Os tipos mais conhecidos são: Etapa Binária, Linear, *Sigmóide*, *Tanh*, *ReLU*, *Leaky ReLU* e a *Softmax*.

Uma função que é muito utilizada principalmente na classificação de imagens é a função *ReLU* (Unidade Linear Retificada), que é definida pela função representada na Figura 13.

Figura 13 - Diagrama da função de ativação *ReLU*



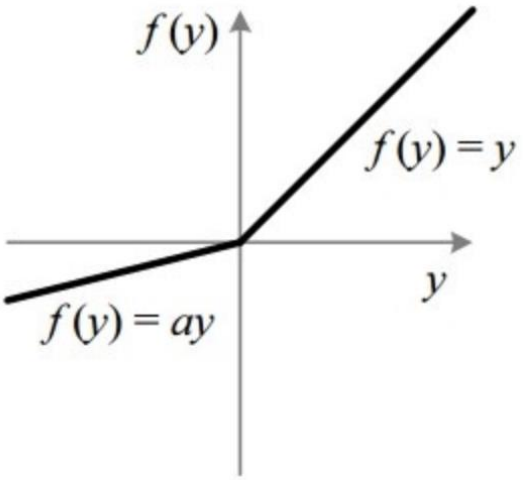
Fonte: PAULY *et al*. (2017)

Uma das grandes vantagens de se utilizar a função *ReLU* na qual se destaca diante as demais funções é que nela os neurônios não são ativados ao mesmo tempo. O procedimento que ela faz é a conversão de todas as entradas negativas, alterando-as para zero e com isso o neurônio não é ativado. Como este processo faz com que apenas alguns neurônios sejam ativados por vez, consequentemente este tipo de função de ativação seja muito mais eficiente em comparação com os demais tipos.

Outra vantagem da função *ReLU* é que sua velocidade de conversão chega a ser seis vezes mais rápida comparando com a função *Tanh* e além disso ela não satura na região positiva. Já em relação às desvantagens, a principal está ligada ao fato de que ocorre uma saturação na região negativa, desta forma o gradiente presente nesta região é zero. Por razão deste problema em que o gradiente é igual a zero é utilizado a função *Leaky ReLU* (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

A função *Leaky ReLU* também é um tipo de função de ativação, que como o próprio nome indica, é baseada na função *ReLU*. A maior diferença entre as duas é que a *Leaky ReLU* possui uma leve inclinação para os valores negativos, ao invés de uma reta horizontal, conforme pode ser observado na Figura 14. Nessa função, o coeficiente de inclinação não é obtido através do treinamento, tendo que ser informado antes que o treinamento seja iniciado.

Figura 14 - Diagrama da função de ativação *Leaky ReLU*



Fonte: AMMAR *et al*. (2019)

2.3.6 *Pooling*

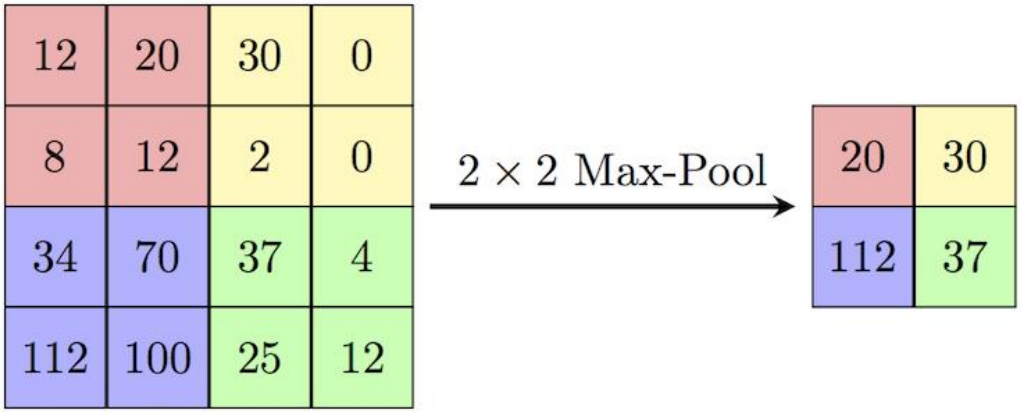
Outro conceito importante na arquitetura da CNN é a camada de *pooling*, também definida como camada de agrupamento local ou global, que podem ser encontradas entre as camadas de convolução. Sua principal função é reduzir o número de parâmetros e também otimizar o processamento da imagem.

Para isso é realizado um processo que combina conjuntos de neurônios que foram recebidos de uma camada anterior, transformando-os em um único neurônio na camada seguinte. Existem dois tipos de operações presentes nesta camada que são o *pooling* máximo, médio e mínimo ou também conhecidos pelos termos *max-pooling*, *average-pooling* e *min-pooling respectivamente*, sendo o *pooling* máximo o mais utilizado para operações com imagens.

No momento da execução desta etapa também deve ser definido o tamanho da janela deslizante que irá percorrer a matriz da imagem aplicando as operações, a este tamanho é dado o nome de *stride*. No *max-pooling*, para cada iteração desta janela deslizante é selecionado o maior valor e após percorrer por toda a matriz, os valores selecionados são utilizados para formar uma nova matriz que é o resultado da operação.

Tendo como exemplo uma matriz de neurônios que possua as dimensões 4 x 4, e que seja definido um *stride* 2 x 2, após executada a camada de *max-pooling* será gerada uma matriz com a dimensão 2 x 2 e seu conteúdo será os maiores valores de cada iteração da janela, conforme mostra a Figura 15.

Figura 15 - Exemplo da operação de *max-pooling*



Fonte: Computer Science Wiki (2018)

Esse procedimento geralmente é realizado após as camadas de convolução e se trata de uma forma eficaz de identificar uma certa característica em qualquer lugar da imagem. E após ter essa característica identificada, a posição exata dela não é necessária para continuar o treinamento, necessitando apenas de uma localização aproximada. Além disto este processo ajuda a reduzir o número de parâmetros que serão processados nas próximas camadas, sendo mais um fator de melhoria da performance da rede.

2.4 Trabalhos Relacionados

Esta seção visa apresentar alguns trabalhos relacionados, são eles:

YANG, Yi *et al.* (2015) apresentou uma implementação para detecção e classificação de placas de trânsito em tempo real utilizando a CNN, buscando fazer este procedimento de forma rápida. Para isso, foi proposto um módulo de detecção que (no período da implementação) era 20 vezes mais rápido do que o melhor módulo de detecção existente (no mesmo período).

ZHANG, Jianming *et al.* (2019) propôs uma implementação de duas novas redes baseadas na Rede Neural Convolucional, para a classificação de placas de trânsito. Para melhorar a precisão do reconhecimento das placas, foi implementado um novo módulo utilizando recursos com conectividade densa. Desta forma a rede implementada obteve uma taxa de precisão de 93,16%.

3. MATERIAIS E MÉTODOS

Este capítulo apresenta os materiais que foram utilizados na implementação do modelo de classificação, bem como os métodos aplicados para definir como os mesmos são usados.

3.1 *Dataset*

O *dataset* é o nome utilizado para representar um conjunto de dados que são utilizados para efetuar algum treinamento ou análise. No caso deste trabalho, será o responsável por guardar todas as imagens de placas de trânsito que serão utilizadas no treinamento e na validação da rede.

Dependendo da classe desejada, é possível encontrar estes *datasets* já prontos e disponíveis para download em alguns sites como o Kaggle (Kaggle, 2021) que fornece uma grande quantidade de *datasets* de diferentes classes, como por exemplo: animais, carros, pessoas, frutas, dentre outros. Caso seja optado por um *dataset* já pronto, basta integrar ao algoritmo da rede e iniciar os testes.

Outra forma de obter o *dataset* seria criando manualmente, neste caso poderia ser utilizando um carro para percorrer pelas ruas e tirar fotos das placas nas mais diversas condições: com boa iluminação, baixa iluminação, tempo chuvoso, etc. Tudo isto para tentar reproduzir ao máximo os cenários reais que são possíveis de ocorrer durante uma viagem. Porém esta forma de coleta exigiria um custo muito alto, além de ser bem cansativa.

No caso deste trabalho não foi encontrado o *dataset* desejado nos sites voltados para este tema, porém foi encontrado um *dataset* nomeado BRTSD (UFRGS, 2017) disponibilizado pela Universidade Federal do Rio Grande do Sul que contém 2111 imagens de vias de trânsito extraídas com o auxílio de um recurso encontrado no Google Maps e no Google Earth chamado Google Street View, que fornece uma visão panorâmica de várias ruas por todo o mundo. Com isto, pelo próprio navegador de internet é possível navegar pelas ruas e capturar as imagens das placas através de ferramentas de *prints*. Estas imagens foram coletadas em vários tamanhos e com diferentes qualidades, possibilitando um treinamento mais eficiente para o algoritmo.

Entretanto, para o trabalho proposto foi preciso recortar essas imagens de modo que no final a imagem seria composta apenas pela região da placa. Este procedimento foi realizado manualmente com auxílio da ferramenta *Lightshot* que

permite fazer capturas de tela e recortá-las. Como a maioria das imagens do *dataset* não possuíam placas e algumas também eram repetidas, o *dataset* gerado com o recorte das placas ficou com 1100 imagens no total.

Como um subproduto deste trabalho também foi disponibilizado o *dataset* gerado através da plataforma *GitHub* (NASCIMENTO, 2021), a fim de contribuir com futuros trabalhos que sejam desenvolvidos nesta área e necessitem deste material.

Para este trabalho foram escolhidas dez classes de placas para compor o *dataset*, sendo que estas placas fazem parte da categoria de regulamentação. As placas de regulamentação possuem o objetivo de indicar obrigações, limitações e proibições ao condutor do veículo. Ao todo o *dataset* possui 1100 imagens, sendo 110 imagens para cada classe de placas e foi dividido da seguinte maneira:

- 90% para a etapa de treinamento: sendo que desta parte 80% são realmente para o treinamento e 20% para o teste.
- 10% para a validação da rede (utilizadas para analisar o comportamento do modelo com imagens quem não fazem parte do escopo de treinamento).

A seguir a Tabela 1 apresenta as classes das placas presente no *dataset* 1, sendo o *dataset* principal do projeto com as dez classes. Sendo que para cada classe foram 79 imagens para treino, 20 imagens para teste e 11 imagens para validação, totalizando em 110 imagens para cada classe e 1100 imagens no total.

Tabela 1 – Classes presentes no *dataset* 1 (principal)

Classes
Estacionamento regulamentado
Parada obrigatória
Proibido estacionar

Continua

Cont. Tabela 1

Proibido retornar à esquerda
Dê a preferência
Velocidade máxima permitida (60 km/h)
Velocidade máxima permitida (80 km/h)
Circulação exclusiva de bicicletas
Sentido de circulação da via ou pista
Siga em frente

Fonte: Próprio Autor

Na parte das imagens destinadas ao treinamento foi realizada a divisão das mesmas em pastas para cada classe, numeradas de 0 a 9. No algoritmo esta numeração irá indicar a qual classe pertence as imagens contidas nas pastas.

No momento da coleta de imagens no Google Street View, houve o cuidado de pegar as imagens em diferentes ângulos, distâncias e qualidades para tentar fornecer o máximo de cenários possíveis para o algoritmo. No entanto, a ferramenta não conta com imagens escuras ou com tempo chuvoso o que certamente levará o algoritmo a ter dificuldade de reconhecer as placas nestas situações. A Figura 16 apresenta um exemplo de cada classe de placas utilizadas para construir o *dataset*.

Figura 16 - Exemplos de placas de trânsito utilizadas no *dataset*

Fonte: Próprio Autor

3.2 Tecnologias

Para implementar a solução, foram utilizadas algumas ferramentas que são recomendadas para trabalhar com o aprendizado de máquina: a linguagem *Python* e as bibliotecas *TensorFlow* e *Keras*.

3.2.1 *Python*

O *Python* é uma linguagem de programação de alto nível criada por Guido van Rossum em 1991. Ela é uma linguagem interpretada, dinâmica, multiplataforma e possui orientação a objetos, o que permite desenvolver um *software* organizado e modular, facilitando a manutenção do código. Também é conhecida por possuir uma sintaxe de fácil entendimento, sendo a escolhida por muitas pessoas que estão ingressando no mundo da programação.

É utilizada em diversos segmentos como aplicações web, jogos, dentre outros. Mas as áreas que mais aderiram o *Python* foram o Data Science e o Machine Learning, pelo motivo de ter vários frameworks que auxiliam no tratamento e processamento dos dados e também por ter uma comunidade bastante ativa neste segmento.

3.2.2 *TensorFlow*

O *TensorFlow* é uma plataforma de código aberto com inúmeras ferramentas com foco no aprendizado de máquina. Ele foi criado por uma equipe de pesquisa voltada para inteligência artificial chamada Google Brain, buscando facilitar o processo para obter os dados necessários para o treinamento e também a forma como criar os modelos de treinamento, além de um refinamento dos resultados futuros.

De forma resumida, o *TensorFlow* faz um agrupamento de seus vários modelos e algoritmos voltados para o treinamento de redes neurais e os disponibiliza através de uma API que pode ser utilizada em linguagens de programação como o *Python*, no qual foi abordado no tópico anterior, mas também em outras linguagens como C++, JavaScript, dentre outras.

Com estes modelos e algoritmos fornecidos pela API (*Application Programming Interface)* do *TensorFlow* é possível realizar o treinamento de redes neurais profundas para várias aplicações, como a classificação de dígitos manuscritos, reconhecimento de imagens, processamento de linguagem natural,

dentre outros.

Um de seus grandes benefícios é a abstração do código, fazendo com que o desenvolvedor não precise se preocupar com os detalhes da implementação do treinamento e com isso possa focar na aplicação final.

3.2.3 *Keras*

O *Keras* é uma biblioteca de *software* com código aberto, que tem como objetivo auxiliar na implementação de redes neurais artificiais. Além disso, foi escrito em *Python* e é definido como uma API de alto nível, oferecendo uma interface modular, extensível e de fácil entendimento, possibilitando assim experimentos rápidos sem a necessidade de entender a fundo os detalhes da implementação.

Dentre as várias funções fornecidas pelo *Keras* para criação e manipulação de redes neurais artificiais estão as funções de ativação, funções de perda, otimizadores, dentre outras. Outro fator interessante, é que além das redes neurais convencionais, ele também oferece suporte para as redes neurais convolucionais, que é a rede utilizada neste trabalho.

Em 2017 o *TensorFlow*, em sua versão 2.0, passou a utilizar o *Keras* em sua plataforma, o que consequentemente tornou o *Keras* ainda mais conhecido e com uma comunidade cada vez maior.

4. DESENVOLVIMENTO

Neste capítulo serão abordados os procedimentos realizados para a implementação da classificação de algumas das placas de trânsito brasileiras. Explicando desde a parte da organização do banco de dados até a parte do código utilizado para realizar os treinamentos e obter os resultados.

4.1 Definição dos subconjuntos do *dataset*

Pensando nos testes a serem realizados após a implementação da Rede Neural Convolucional e também visando explorar quais seriam os impactos resultantes de alterações no *dataset*, foram definidos alguns subconjuntos do *dataset*.

Cada um destes subconjuntos são divisões do *dataset* principal nas quais são feitas algumas alterações que podem gerar alguma diferença no resultado final. Exemplos destas alterações são: um *dataset* com um número menor de imagens, outro com um número menor de classes, outro mesclando uma quantidade maior de imagens para algumas classes e uma quantidade menor para as outras classes, sendo que a classe representa o tipo da placa.

Em relação a divisão da quantidade de imagens para o treinamento destes *datasets*, foram mantidas as mesmas porcentagens, sendo 80% para treino e 20% para teste. Já referente a parte para validação foi fixado a quantidade de 11 imagens para cada classe. As informações dos subconjuntos do *dataset* são apresentados abaixo com suas devidas alterações.

4.1.1 *Datasets com variação na quantidade de imagens*

Para a primeira parte da divisão do *dataset*, foi considerado a variação no número de imagens em cada *dataset*, de modo a ser possível observar na etapa de resultados se a quantidade de imagens usadas no treinamento influencia na acurácia do modelo. Desta forma, foram obtidos os *datasets* 2, 3 e 4 nos quais as classes presentes nos mesmos são as mesmas presentes no *dataset* 1, de acordo com a Tabela 1. Com isto a única diferença foi na quantidade de imagens, conforme apresentado na Tabela 2.

Tabela 2 – Informações dos *datasets* 2, 3 e 4

Datasets	**Quantidades por classe**			**Total**
	Treino	**Teste**	**Total**	
Dataset 2	56	14	70	**700**
Dataset 3	36	9	45	**450**
Dataset 4	16	4	20	**200**

Fonte: Próprio Autor

4.1.2 *Datasets com variação na quantidade de classes*

A segunda divisão leva em conta a variação da quantidade de classes presentes no *dataset*. Tendo em vista que o *dataset* principal possui um total de dez classes, nesta etapa foram gerados três *datasets* organizados da seguinte forma:

- *Dataset* 5 - contém 8 classes.
- *Dataset* 6 - contém 5 classes.
- *Dataset* 7 - contém 3 classes.

Tendo em vista a variação apenas na quantidade de classes, foi definida a mesma quantidade de imagens para os três *datasets*, sendo que para cada classe foram 48 imagens para treino, 12 imagens para teste, totalizando em 60 imagens para cada classe e a quantidade total dependendo da quantidade de classes de cada dataset.

Segue abaixo as Tabelas 3, 4 e 5 informando as classes das placas presentes em cada *dataset*.

Tabela 3 - Classes presentes no *dataset* 5

Classes
Estacionamento regulamentado
Proibido estacionar
Proibido retornar à esquerda
Dê a preferência

Continua

Cont. Tabela 3

Velocidade máxima permitida (60 km/h)
Velocidade máxima permitida (80 km/h)
Sentido de circulação da via ou pista
Siga em frente

Fonte: Próprio Autor

Tabela 4 - Classes presentes no *dataset* 6

Classes
Estacionamento regulamentado
Proibido estacionar
Proibido retornar à esquerda
Dê a preferência
Velocidade máxima permitida (60 km/h)

Fonte: Próprio Autor

Tabela 5 - Classes presentes no *dataset* 7

Classes
Estacionamento regulamentado
Proibido estacionar
Proibido retornar à esquerda

Fonte: Próprio Autor

4.1.3 *Datasets com variação na quantidade de imagens para cada classe*

Nesta última divisão, foi definida uma quantidade diferente de imagens para cada classe, também com a finalidade de avaliar ao final quais impactos esta diferença de quantidade pode gerar. Nesta parte serão utilizadas cinco classes, a Tabela 6 mostra os dados de cada uma.

Tabela 6 - Informações do *dataset* 8

Classes	Treino	Teste	Total
Estacionamento regulamentado	48	12	**60**
Parada obrigatória	32	8	**40**
Proibido retornar à esquerda	16	4	**20**
Velocidade máxima permitida (60 km/h)	8	2	**10**
Sentido de circulação da via ou pista	4	1	**5**
Total	**108**	**27**	**135**

Fonte: Próprio Autor

4.2 Pré-processamento do *dataset*

Finalizado a divisão dos *datasets* que serão utilizados para os vários testes de treinamento da rede, foi então iniciada a etapa de codificação. Inicialmente o *dataset* desejado é colocado na pasta do projeto e o passo inicial é fazer com que o programa passe por todas as imagens redefinindo o tamanho das mesmas para 30x30 e após isto inseri-las em uma lista. Junto a isto, para cada imagem também é inserida em uma lista o número da pasta em que a imagem se encontra. Este número também chamado de *label* representará a classe da imagem. Por exemplo: as imagens na pasta 0 são da classe estacionamento regulamentado, na pasta 1 se encontram as imagens da classe parada obrigatória, seguindo esta mesma lógica para as demais pastas.

Essa redefinição do tamanho é feita para garantir que o processamento não seja tão alto e consequentemente não ultrapasse o poder de processamento do computador. Neste mesmo objetivo a lista na qual as imagens são inseridas é convertida para o formato *numpy.array*, que é um formato recomendado para trabalhar com vetores e matrizes.

Já na parte da divisão das imagens utilizadas para treino, teste e validação, existe uma função chamada *train_test_split* disponibilizada pela biblioteca *sklearn* que realiza a divisão das imagens de acordo com os parâmetros que devem ser passados

para ela, indicando a porcentagem de imagens para treino e também a porcentagem para teste.

4.3 Construção do modelo

Após feita a parte de pré-processamento das imagens presentes no *dataset* é então iniciada a construção do modelo que será utilizado para o treinamento da rede. Nesta parte são utilizadas uma série de funções representando cada etapa do processo convolucional conforme descrito na seção 2.3, sendo que para cada função são passados os seus respectivos parâmetros. Os passos realizados neste processo são explicados a seguir:

1. Inicialmente é instanciado um modelo sequencial através da função *Sequential*, ele funciona como uma pilha onde são adicionadas as camadas de convolução. É ele também que disponibiliza os métodos que são chamados para realizar o treinamento.
2. Em seguida são adicionadas duas camadas utilizando a função *Conv2D* para cada uma. Cada camada irá criar um kernel de convolução que será processado junto à entrada para gerar um tensor de saídas. Os parâmetros para a criação desta camada são:
    a. *filters* - representa o número de filtros de saída que são gerados ao fim da convolução, em outras palavras será a dimensionalidade do espaço de saída.
    b. *kernel_size* - define os valores de altura e largura do filtro convolucional.
    c. *activation* - especifica qual a função de ativação será utilizada.
    d. *input_shape* - representa a dimensão dos dados de entrada. Só é necessário informar este parâmetro na primeira camada adicionada.
3. Logo após é adicionada uma camada de *pooling* utilizando a função *MaxPool2D*, que irá reduzir a resolução da entrada no das dimensões geradas pela camada anterior. Com isso é definido uma janela deslizante que percorre a imagem obtendo o valor máximo para cada iteração. Para esta função é passado o seguinte parâmetro:
    a. *pool_size* - define os valores de altura e largura da janela deslizante.
4. É adicionada também uma camada *dropout*, utilizando a função *Dropout*. Ela irá definir de forma aleatória um valor, passado por parâmetro, que representa

a probabilidade de remover temporariamente parte da entrada. Este processo é realizado para tentar evitar que aconteça o *overfitting*, que se refere a situação onde o modelo possui um excelente desempenho utilizando os dados de treinamento, porém quando ele é informado outros dados seu desempenho piora consideravelmente. Para a função é passado o parâmetro:

   a. *rate* - Um decimal entre 0 e 1 indicando a probabilidade da remoção temporária de parte da entrada.

5. Feito isto, é novamente adicionado novamente as mesmas camadas anteriores. Ou seja, é adicionada mais duas camadas de convolução, uma camada de *pooling* e uma camada de *dropout*, alterando apenas o valor dos parâmetros que serão descritos posteriormente.
6. Em seguida é adicionada uma camada de *flatten* usando a função *Flatten*, essa camada faz o nivelamento da entrada, convertendo de matriz para vetor.
7. É então adicionada uma camada densa utilizando a função *Dense*. Esta camada representa uma rede neural tradicional densamente conectada. Os parâmetros passados para ela são:
   a. *units* - um número inteiro positivo, indicando a dimensionalidade do espaço de saída.
   b. activation - especifica a função de ativação que será usada.
8. Finalizando a construção do modelo é adicionado mais uma camada de dropout e também uma camada densa, alterando apenas o valor dos parâmetros.

4.3.1 *Parametrização*

Conforme informado durante a construção do modelo, em grande parte das funções utilizadas para adicionar as camadas, foram passados alguns parâmetros para indicar quais características estas camadas precisam ter. Desta forma, esta seção possui o objetivo de esclarecer quais os parâmetros utilizados no modelo construído para este trabalho.

Assim como foi feito com o *dataset*, pensando nos testes que serão feitos após o modelo estar pronto. Foram definidos três perfis de parâmetros para observar quais os efeitos de realizar pequenas alterações nos valores dos mesmos. A Tabela 7 apresenta o valor definido para cada parâmetro nos diferentes perfis.

Tabela 7 - Perfis de parametrização

Função	**Parâmetro**	**Perfil 1**	**Perfil 2**	**Perfil 3**
Conv2D	*filters*	13	10	5
	kernel_size	5x5	3x3	3x3
	activation	relu	relu	relu
MaxPool2D	*pool_size*	2x2	3x3	3x3
Dropout	*rate*	0.25	0.13	0.30
Conv2D	*filters*	26	7	N/A
	kernel_size	3x3	2x2	N/A
	activation	relu	relu	N/A
MaxPool2D	*pool_size*	2x2	2x2	N/A
Dropout	*rate*	0.25	0.13	N/A
Dense	*units*	80	70	70
	activation	relu	relu	relu
Dropout	*rate*	0.5	0.3	0.5
Dense	*units*	10	10	10
	activation	softmax	softmax	softmax

Fonte: Próprio Autor

4.4 Compilação e treinamento do modelo

Após construir o modelo com os parâmetros definidos, foi realizada a etapa de compilação do mesmo utilizando a função *compile*, ela é responsável por aplicar as últimas configurações no modelo de forma que ele fique pronto para ser submetido a etapa de treinamento. Para ela, são passados os seguintes parâmetros:

1. *loss* - deve ser informado o nome da função de perda que será aplicada ao modelo.
2. *optimizer* - define o nome do algoritmo de otimização que será utilizado.

3. *metrics* - especifica as métricas que serão avaliadas pelo modelo durante o treinamento e também o teste.

A Tabela 8 mostra quais os valores passados para estes parâmetros no modelo treinado para este trabalho.

Tabela 8 - Valores dos parâmetros passados para a função *compile*

Parâmetro	**Valor**
loss	categorical_crossentropy
optimizer	adam
metrics	accuracy

Fonte: Próprio Autor

Feita a compilação do modelo foi então iniciado a etapa de treinamento por meio da função *fit* que realiza o treino do modelo por um número fixo de épocas, sendo este número um dos parâmetros da função. Além do número de épocas, também é passado como parâmetro o vetor de imagens de treino, o vetor de *labels* de treino, o número de amostras utilizadas para cada época e os vetores de imagens e *labels* separados para teste. Para o modelo treinado neste trabalho, o número de épocas informado foi 15 e o número de amostras foram 32, sendo estes valores definidos com base nos materiais referenciados no trabalho.

4.4.1 *Plotagem do gráfico*

Após a etapa de treinamento do modelo, com auxílio da biblioteca *matplotlib* foi feita a plotagem do gráfico apresentando a porcentagem de acurácia para cada época do treinamento. Esta acurácia é baseada nos testes feitos com parte das imagens separadas para este fim, conforme mencionado anteriormente.

4.5 Validação do modelo

A última etapa é realizar a validação do modelo, para isso são utilizadas as imagens definidas para a validação, que conforme informado anteriormente são 11 imagens para cada classe. E após finalizado o treinamento do modelo, o programa abre as imagens de validação que não fazem parte do escopo das imagens de

treinamento e chama a função *predict_classes* e retorna a acurácia do modelo para as imagens informadas.

É geralmente nesta parte que é possível identificar o *overfitting*, na situação em que o modelo fica especialista para as imagens utilizadas durante o treinamento, mas quando utilizadas imagens fora deste escopo, ele apresenta uma baixa acurácia.

5. RESULTADOS

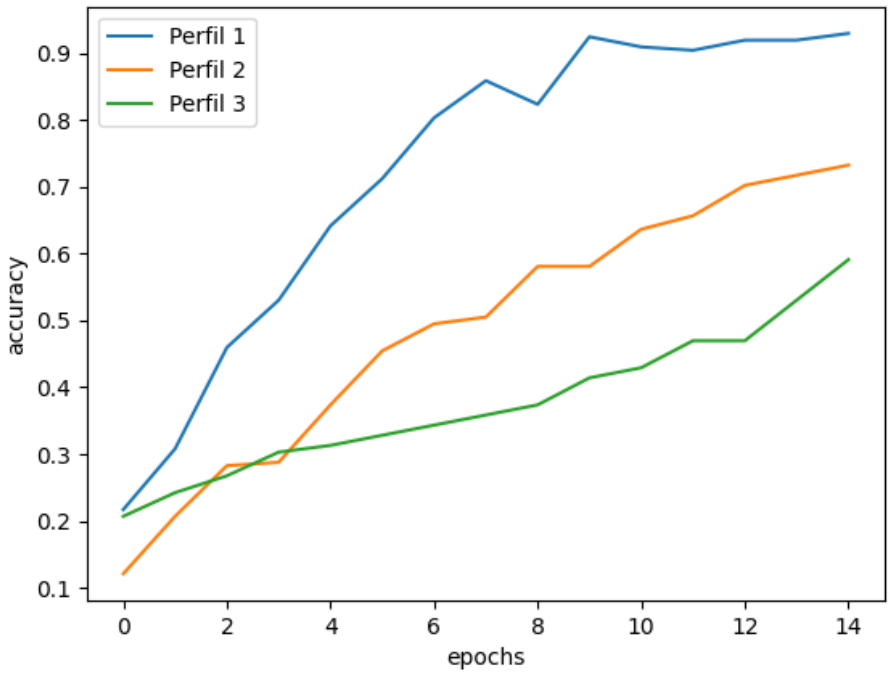
Passado pela parte de desenvolvimento da rede, foi iniciada a etapa de testes com os *datasets* e os perfis descritos anteriormente. Conforme definido na separação dos subconjuntos dos *datasets*, os testes são agrupados por cada tipo de variação para que sejam analisadas as alterações provocadas por elas.

Cada um dos gráficos a seguir apresenta a evolução da acurácia dos testes realizados no decorrer das épocas do treinamento para cada um dos perfis definidos na seção anterior.

5.1 Testes com variação na quantidade de imagens

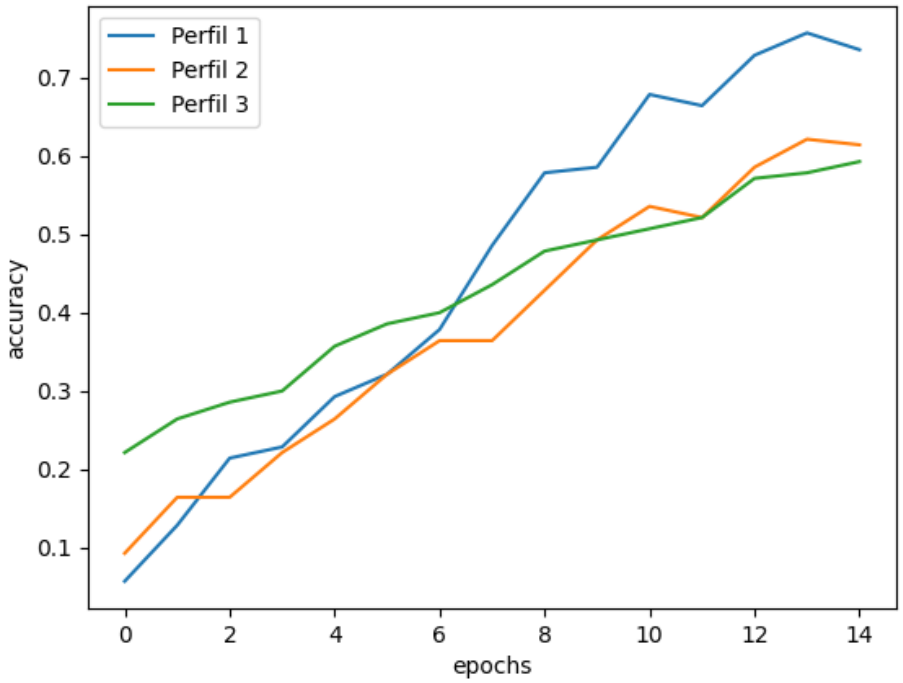
Neste primeiro conjunto de testes é abordado uma situação que gera algumas dúvidas no momento em que é decidido treinar uma rede, no qual diz respeito a quantidade de imagens necessárias para se construir um modelo com uma alta taxa de acertos. E desta maneira este teste busca mostrar quais os impactos nos resultados de modelos treinados com diferentes quantidades de imagens e com diferentes perfis de parâmetros. Os resultados obtidos são apresentados abaixo.

Figura 17 - Evolução da acurácia apresentada pelo teste utilizando o *dataset* 1



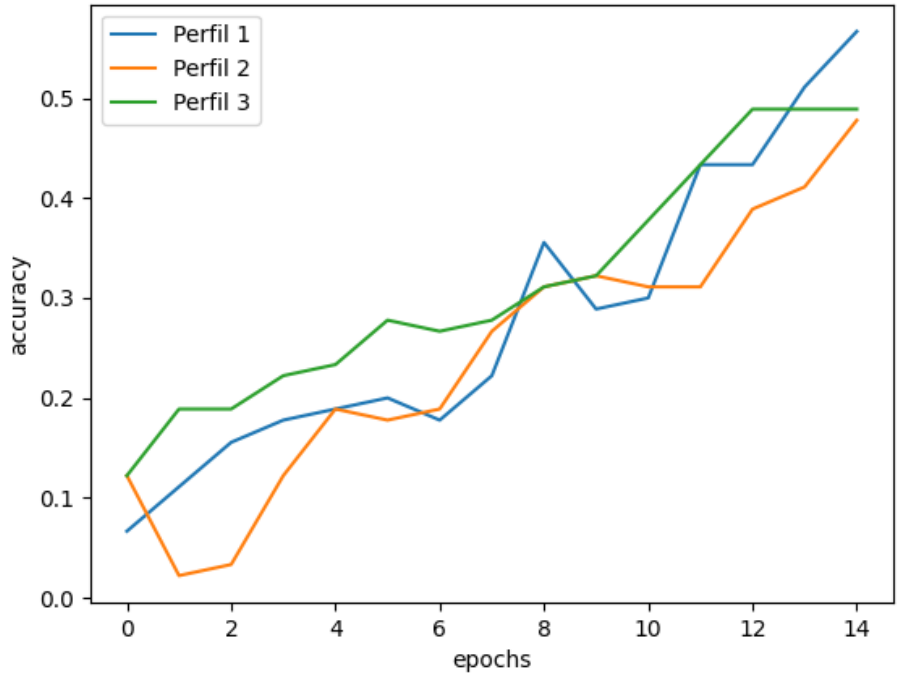
Fonte: Próprio Autor

Figura 18 - Evolução da acurácia apresentada pelo teste utilizando o *dataset* 2



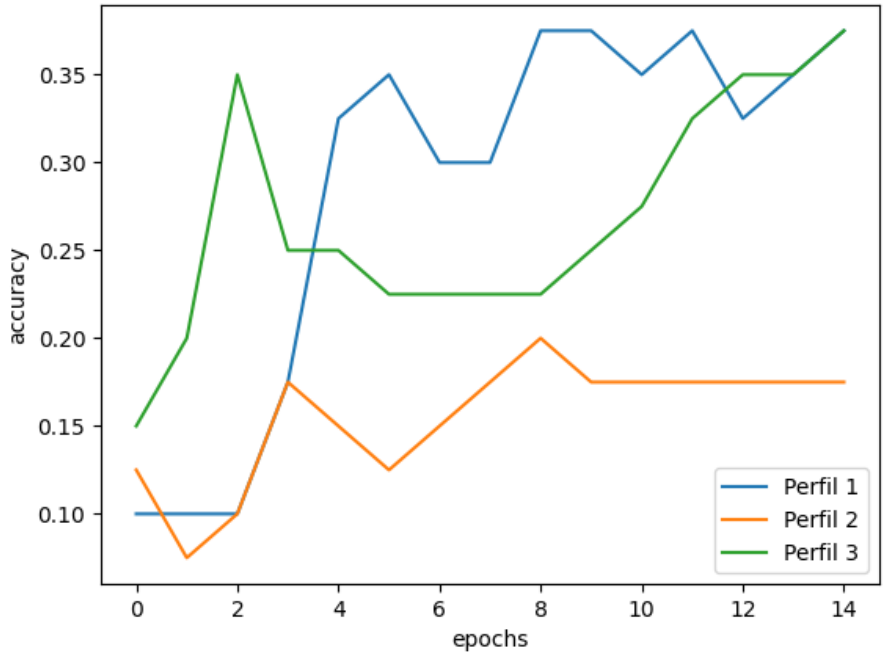
Fonte: Próprio Autor

Figura 19 - Evolução da acurácia apresentada pelo teste utilizando o *dataset* 3



Fonte: Próprio Autor

Figura 20 - Evolução da acurácia apresentada pelo teste utilizando o *dataset* 4



Fonte: Próprio Autor

Tabela 9 - Acurácia dos testes de validação do conjunto de testes 1

Teste	**Acurácia (%)**		
	Perfil 1	**Perfil 2**	**Perfil 3**
Dataset 1	97,27	80,00	65,45
Dataset 2	66,36	54,54	37,27
Dataset 3	57,27	36,36	42,72
Dataset 4	49,09	20,90	40,00

Fonte: Próprio Autor

Analisando a Figura 17, no qual o teste utiliza o *dataset* com a maior quantidade de imagens, é possível observar que ele apresenta uma boa porcentagem de acurácia, principalmente quando observado o perfil 1. Também é possível notar que conforme vão sendo processadas as épocas, a diferença de porcentagem entre os perfis aumenta consideravelmente. Isto indica que, para este teste os parâmetros passados para cada perfil têm um forte impacto no desempenho do modelo.

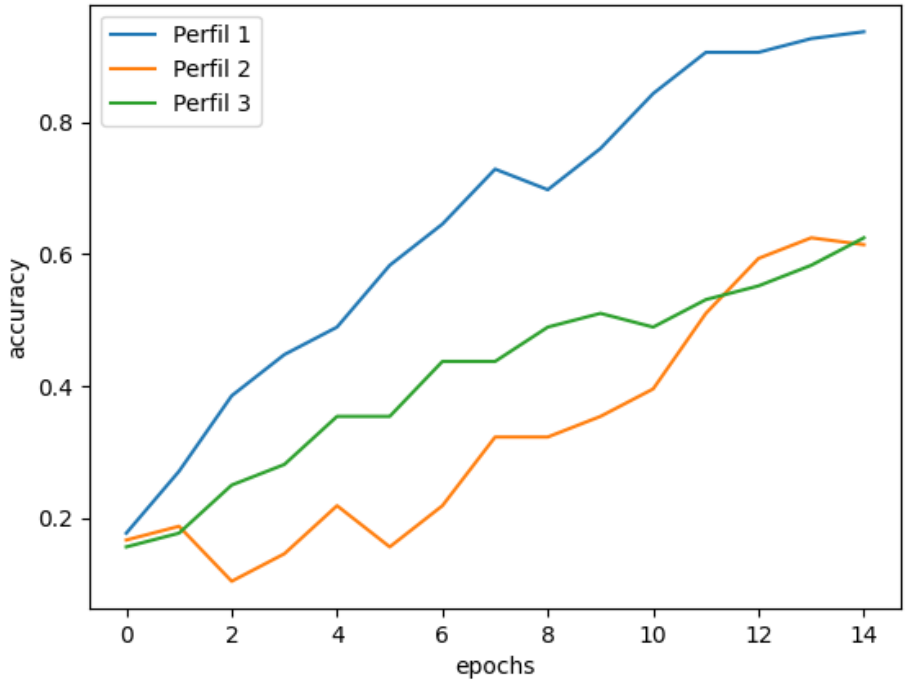
Já quando são observadas as Figuras 18 e 19, pode ser visto que a evolução da acurácia dos perfis se aproxima mais, sendo que a interceptação entre elas ocorre com mais frequência. Por fim, na Figura 20 ocorre uma grande instabilidade em todos os perfis no decorrer das épocas, indicando uma dificuldade em se trabalhar com poucas imagens. Tendo em vista que este teste utiliza o *dataset* com a menor quantidade de imagens deste conjunto de testes.

Analisando de forma geral os quatro gráficos, é indicado que à medida que o número de imagens do *dataset* diminui, a acurácia também diminui e os perfis também apresentam comportamentos diferentes. É possível visualizar também que as acurácias de teste são semelhantes as acurácias de validação exibidas na Tabela 9.

5.2 Testes com variação na quantidade de classes

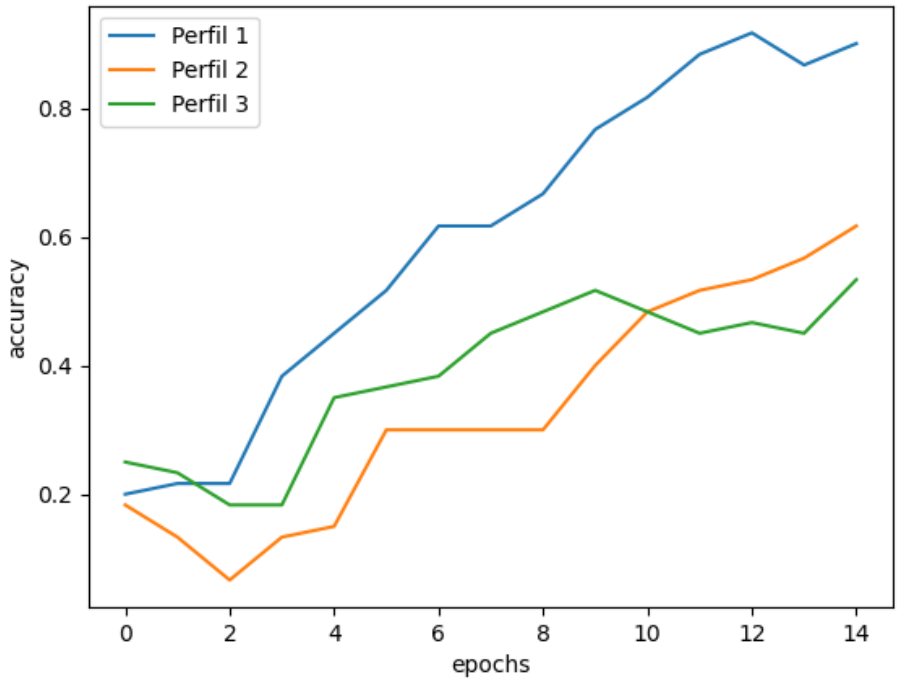
A seguir os resultados obtidos com os testes considerando a variação na quantidade de placas.

Figura 21 - Evolução da acurácia apresentada pelo teste utilizando o *dataset* 5



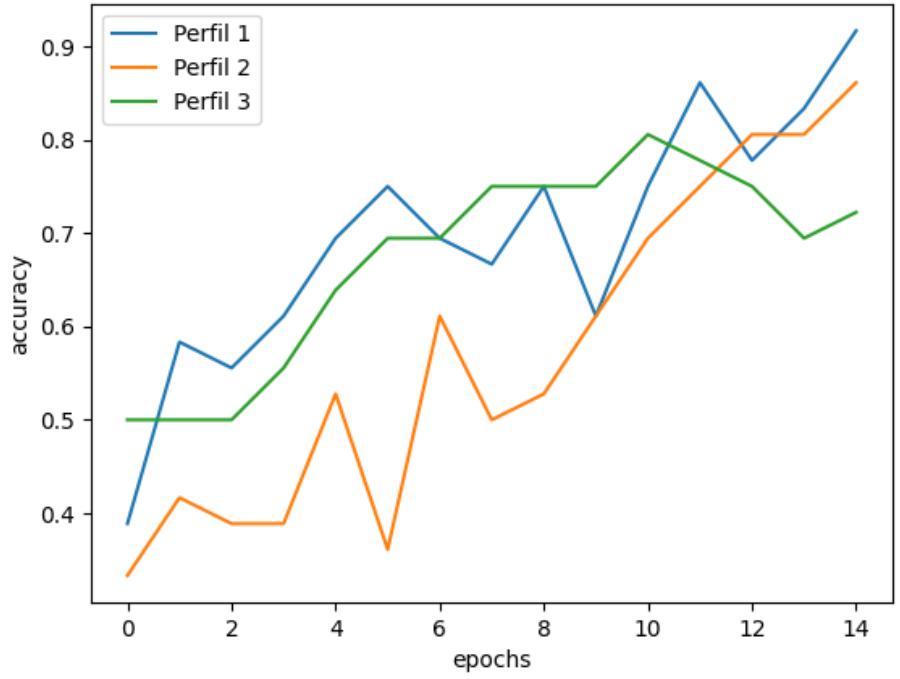
Fonte: Próprio Autor

Figura 22 - Evolução da acurácia apresentada pelo teste utilizando o *dataset* 6



Fonte: Próprio Autor

Figura 23 - Evolução da acurácia apresentada pelo teste utilizando o *dataset* 7



Fonte: Próprio Autor

Tabela 10 - Acurácia dos testes de validação do conjunto de testes 2

Teste	**Acurácia (%)**		
	Perfil 1	**Perfil 2**	**Perfil 3**
Dataset 5	93,18	65,90	55,68
Dataset 6	81,81	67,27	54,54
Dataset 7	87,88	63,63	72,72

Fonte: Próprio Autor

Após analisar os testes com base na variação de quantidade de imagens, este conjunto de testes busca observar os impactos gerados pela variação da quantidade de classes que o modelo deverá classificar. Tendo em vista que com uma maior quantidade de classes, existe uma maior possibilidade de existir classes com características semelhantes que podem confundir o modelo e com isto abaixar sua acurácia. Um exemplo no cenário de placas de trânsito são as classes “Siga em frente” e “Sentido de circulação da via ou pista”, que possuem basicamente a mesma figura, porém em direções diferentes.

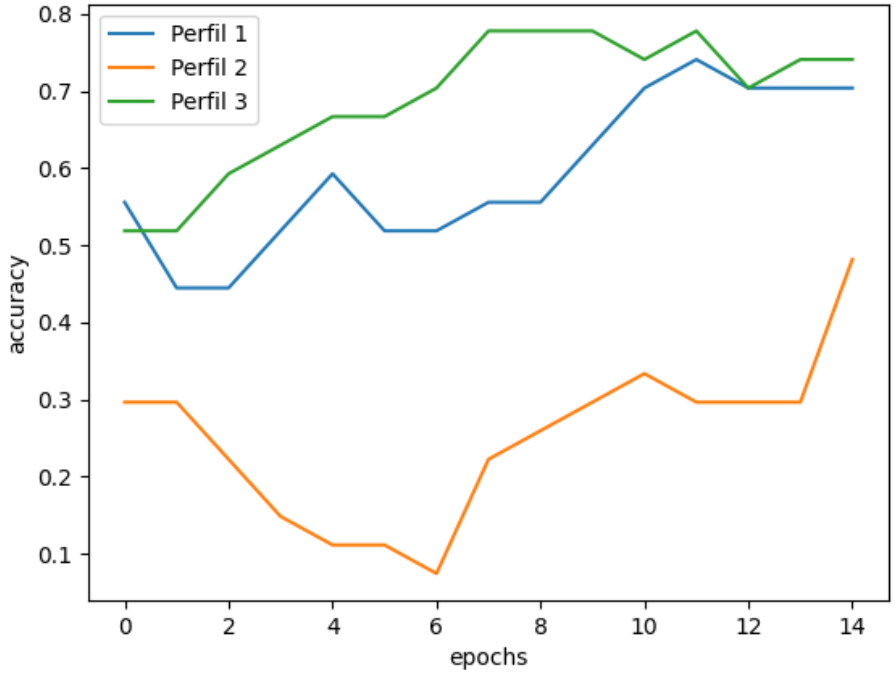
Observando as Figuras 21 e 22 cujo os *datasets* utilizados possuem 8 e 5 classes respectivamente, pode ser visto que possuem uma certa similaridade. No qual o perfil 1 consegue obter um desempenho melhor, chegando próximo aos 90% de acurácia e os perfis 2 e 3 com um desempenho inferior, se aproximando apenas nas últimas épocas. Um ponto de análise importante, é que mesmo o perfil 1 obtendo uma acurácia de teste semelhante no teste representado pela Figura 21 e no teste representado pela Figura 22, a diferença da acurácia de validação entre os dois testes se aproxima de 10%. As acurácias de validação são apresentadas na Tabela 10.

Já na Figura 23, que utiliza um *dataset* com 3 classes, é notável uma melhora na acurácia, principalmente quando observados os perfis 2 e 3. Embora ocorra uma certa instabilidade nos testes com o decorrer das épocas, é perceptível uma melhora no modelo quando trabalhado com uma pequena quantidade de classes.

5.3 Teste com variação na quantidade de imagens para cada classe

Levando em conta os resultados obtidos pelos testes com variação na quantidade de imagens, também foi proposto um teste com variação na quantidade de imagens para cada classe. Ou seja, neste teste a quantidade de imagens é diferente para cada classe do *dataset*. Desta forma, buscando analisar como o treinamento irá se comportar diante destas modificações. Sendo então obtidos os seguintes resultados.

Figura 24 - Evolução da acurácia apresentada pelo teste utilizando o *dataset* 8



Fonte: Próprio Autor

Tabela 11 - Acurácia dos testes de validação do conjunto de testes 3

Teste	**Acurácia (%)**		
	Perfil 1	**Perfil 2**	**Perfil 3**
Dataset 8	40,00	21,82	32,72

Fonte: Próprio Autor

Verificando a Figura 24, que apresenta o resultado deste teste, é possível

observar que a acurácia de teste não é tão alta comparada aos testes anteriores. Uma novidade neste teste é que o perfil 3 apresenta uma acurácia de teste melhor do que o perfil 1 em quase todas as épocas. Por outro lado, o perfil 2 não apresenta um bom resultado, com uma diferença significativa para os demais perfis.

Neste teste também é possível notar uma grande diferença entre a acurácia de teste e a acurácia de validação que é apresentada na Tabela 11. Conforme explicado em seções anteriores, esta situação caracteriza um problema de *overfitting*.

6. CONSIDERAÇÕES FINAIS

Conforme proposto inicialmente, o presente trabalho apresentou uma implementação utilizando Rede Neural Convolucional para realizar a classificação de dez classes de placas de trânsito brasileiras. Para realizar esta implementação, também foram utilizadas as ferramentas propostas, sendo elas a linguagem de programação *Python* que permitiu o uso de bibliotecas para auxiliar no processo de codificação, sendo algumas delas o *TensorFlow* e o *Keras* que foram fundamentais para um desenvolvimento prático e descomplicado.

O trabalho também conseguiu aprofundar nas etapas do processo convolucional, descrevendo todos os passos que a rede realiza internamente para gerar o modelo ao final do treinamento. Desta forma foi possível entender a importância de cada etapa dentro da rede e também os principais fatores que diferenciam a rede convolucional dos demais tipos de rede.

Para a parte de execução dos testes, foi possível obter um *dataset* com várias imagens de vias de trânsito brasileiras. Sendo feito o recorte destas imagens a fim de se obter um *dataset* somente com o enquadramento das placas, além de possuir apenas as dez classes de placas definidas para este trabalho. Com este *dataset* foram gerados mais sete *datasets* com variações no número de imagens, número de classes e no número de imagens por classe, totalizando oito *datasets*. Da mesma maneira foram criados três perfis de parametrizações, com variações nos valores dos parâmetros passados para as camadas utilizadas para construção do modelo. Portanto os testes foram realizados combinando cada perfil de parametrização com os *datasets* gerados.

Os resultados obtidos apresentaram aspectos interessantes para cada conjunto de testes executados. Primeiramente indicando que a quantidade de imagens gera um impacto no modelo, sendo que os modelos treinados com mais imagens tiveram um desempenho melhor. Nos testes com a variação de quantidade de classes não houve um impacto tão grande na acurácia de teste, porém apresentou uma diferença significativa em sua comparação com a acurácia de validação, sendo um problema de *overfitting*. O mesmo problema acontece no teste variando a quantidade de imagens para cada classe, além de também ter uma piora em seus resultados de forma geral. Por fim, em relação aos perfis de parametrização, o perfil 1 teve o melhor desempenho em grande parte dos testes, já os perfis 2 e 3 tiveram o

desempenho condicionado ao *dataset* utilizado.

Outro objetivo proposto que também foi cumprido foi a disponibilização do *dataset* utilizado neste trabalho com as imagens devidamente recortadas e separadas por classe, com a finalidade de cooperar com os futuros trabalhos nesta área.

Para trabalhos futuros, um ponto importante que pode ser feito é aumentar o *dataset*, atribuindo mais imagens ao mesmo com o objetivo de aumentar a acurácia do modelo. Junto a isto, também adicionar as demais classes das placas de regulamentação e também podendo incluir outras categorias de placas como: placas de advertência, placas de sinalização de obras, placas auxiliares dentre outras. Por fim, ainda referente ao *dataset*, poderiam ser adicionadas placas com diferentes condições de tempo/luminosidade, como em momentos de chuvas, neblina e também no período noturno.

REFERÊNCIAS

ALBAWI, Saad; MOHAMMED, Tareq Abed; AL-ZAWI, Saad. **Understanding of a convolutional neural network**. IEE Xplore, 2017. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8308186>. Acesso em: 28 set. de 2021.

ALBELWI, Saleh; MAHMOOD, Ausif. **A Framework for Designing the Architectures of Deep Convolutional Neural Networks**. Entropy, 24 mai. de 2017. Disponível em: <https://www.mdpi.com/1099-4300/19/6/242/htm >. Acesso em: 4 out. de 2021.

AMMAR, S. *et al*. **Using deep learning algorithms to detect violent activities.** Semantic Scholar, 2019. Disponível em: <https://www.semanticscholar.org/paper/Using-deep-learning-algorithms-to-detect-violent-Ammar-Anjum/24bb9e9785c714a8f2470366395f33a2cc339398#citing-papers>. Acesso em: 4 out. de 2021.

Computer Science Wiki. **Rede Perceptron de uma única camada**. 26 fev. de 2018. Disponível em: <https://computersciencewiki.org/index.php/File:MaxpoolSample2.png>. Acesso em: 22 nov. de 2021.

ELIZONDO, D. **The Linear Separability Problem: Some Testing Methods**. IEE Xplore, 6 mar. de 2006. Disponível em: <https://ieeexplore.ieee.org/abstract/document/1603620/authors#authors>. Acesso em: 15 out. de 2021.

Embarcados. **Rede Perceptron de uma única camada**. 2018. Disponível em: <https://www.embarcados.com.br/rede-perceptron-de-uma-unica-camada>. Acesso em: 15 nov. de 2021.

GeeksforGeeks. **CNN Introduction to Padding**. 22 out. de 2021. Disponível em: <https://www.geeksforgeeks.org/cnn-introduction-to-padding>. Acesso em: 20 nov. de 2021.

Kaggle. **Kaggle: Your Machine Learning and Data Science Community**. 2021. Disponível em: <https://www.kaggle.com/>. Acesso em: 17 dez. de 2021.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. **ImageNet Classification with Deep Convolutional Neural Networks**. University of Toronto Computer Science, 2012. Disponível em: <http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf>. Acesso em: 17 dez. de 2021.

Medium. **Uma introdução as redes neurais convolucionais utilizando o Keras**. 2019. Disponível em: <https://medium.com/data-hackers/uma-introdu%C3%A7%C3%A3o-as-redes-neurais-convolucionais-utilizando-o-keras-41ee8dcc033e>. Acesso em: 29 out. de 2021.

NASCIMENTO, Guilherme Lopes. **Dataset de um conjunto de placas brasileiras**. GitHub, 7 dez. de 2021. Disponível em: <https://github.com/guilhermelps/dataset-placas-brasileiras>. Acesso em: 07 dez. de 2021.

O'SHEA, Keiron; NASH, Ryan. **An Introduction to Convolutional Neural Networks**. Cornell University, 2 dez. 2015. Disponível em: <https://arxiv.org/abs/1511.08458 >. Acesso em: 29 set. de 2021.

PAULY, Leo *et al*. **Deeper Networks for Pavement Crack Detection**. ResearchGate, jul. de 2017. Disponível em: <https://www.researchgate.net/figure/ReLU-activation-function_fig3_319235847>. Acesso em: 11 out. de 2021.

SICHMAN, Jaime Simão. **Inteligência Artificial e sociedade: avanços e riscos**. SciELO Brasil, 19 abr. de 2021. Disponível em: <https://www.scielo.br/j/ea/a/c4sqqrthGMS3ngdBhGWtKhh>. Acesso em: 16 nov. de 2021.

SOARES, Anderson. **Redes Neurais Profundas – Deep Learning**. UFG, 2018. Disponível em: <https://ww2.inf.ufg.br/~anderson/deeplearning/20181/Aula%20-%20Redes%20Neurais%20Convolucionais%20Parte%20I.pdf>. Acesso em: 7 out. de 2021.

SOBREIRO, Vinicius Amorim. **UMA ESTIMAÇÃO DO VALOR DA COMMODITY DE AÇÚCAR UTILIZANDO REDES NEURAIS ARTIFICIAIS**. ResearchGate, jan. de 2020. Disponível em: <https://www.researchgate.net/figure/Figura-2-Perceptron-Multicamadas-Fonte-Adaptado-de-LNCC-2008_fig1_228432919>. Acesso em: 16 nov. de 2021.

UFRGS. **BRTSD**. 7 fev. de 2017. Disponível em: <http://www.lapsi.eletro.ufrgs.br/Download/BRTSD>. Acesso em: 18 mai. de 2021.

VELO, Ramón; LÓPEZ, Paz; MASEDA, Francisco. **Wind speed estimation using multilayer perceptron**. ScienceDirect, mai. de 2014. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0196890414001277>. Acesso em: 7 out. de 2021.

WAZLAWICK, Raul Sidnei. **Metodologia de Pesquisa para Ciência da Computação**. 2. ed. São Paulo: Elsevier, 2014.

YANG, Yi *et al.* **Towards Real-Time Traffic Sign Detection and Classification**. IEE Xplore, 12 out. de 2015. Disponível em: <https://ieeexplore.ieee.org/document/7296660>. Acesso em: 9 ago. de 2021.

ZHANG, Jianming *et al.* **Lightweight deep network for traffic sign classification**. ResearchGate, 18 jul. de 2019. Disponível em: < https://www.researchgate.net/publication/334819523_Lightweight_deep_network_for_traffic_sign_classification>. Acesso em: 9 ago. de 2021.

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
GABINETE DO REITOR

Av. Universitária, 1069 ● Setor Universitário
Caixa Postal 86 ● CEP 74605-010
Goiânia ● Goiás ● Brasil
Fone: (62) 3946.1000
www.pucgoias.edu.br ● reitoria@pucgoias.edu.br

RESOLUÇÃO n° 038/2020 – CEPE

ANEXO I

APÊNDICE ao TCC

Termo de autorização de publicação de produção acadêmica

O(A) estudante GUILHERME LOPES DO NASCIMENTO do Curso de Ciência da Computação, matrícula 2017.1.0028.0085-8, telefone: 62 99166-9408 e-mail guilherme250hb@gmail.com, na qualidade de titular dos direitos autorais, em consonância com a Lei nº 9.610/98 (Lei dos Direitos do autor), autoriza a Pontifícia Universidade Católica de Goiás (PUC Goiás) a disponibilizar o Trabalho de Conclusão de Curso intitulado Classificação de Placas de Trânsito Utilizando Redes Neurais Convolucionais, gratuitamente, sem ressarcimento dos direitos autorais, por 5 (cinco) anos, conforme permissões do documento, em meio eletrônico, na rede mundial de computadores, no formato especificado (Texto (PDF); Imagem (GIF ou JPEG); Som (WAVE, MPEG, AIFF, SND); Vídeo (MPEG, MWV, AVI, QT); outros, específicos da área; para fins de leitura e/ou impressão pela internet, a título de divulgação da produção científica gerada nos cursos de graduação da PUC Goiás.

Goiânia, 16 de dezembro de 2021.

Assinatura do(s) autor(es): Guilherme L. do Nascimento

Nome completo do autor: GUILHERME LOPES DO NASCIMENTO

Assinatura do professor-orientador: Max Gontijo

Nome completo do professor-orientador: Me. Max Gontijo de Oliveira