Pró-Reitoria Graduação Presencial.
Coordenação do Curso de Engenharia Elétrica.

Artigo apresentado como um dos pré-requisitos para a obtenção do grau de bacharel em Engenharia Elétrica, 2020/01.

CLASSIFICAÇÃO DE PLACAS DE TRÂNSITO COM O USO DE REDES NEURAIS CONVOLUCIONAIS

Erasmo de Almeida Neto (erasmo.aln@gmail.com); Felipe Moreira da Silva Santos (felipemoreiraa@hotmail.com); Profº Msc. Felipe Santana Santos (felipe.ssantos@souunit.com.br).

RESUMO

Com o crescimento do número de acidentes de trânsito a cada ano, especificamente aqueles decorrentes de falha humana, surgiram diversas iniciativas para lidar com este problema, uma delas é o uso de veículos autônomos. Diante disso, este artigo tem o objetivo de realizar a classificação de placas de trânsito, uma subtarefa, porém muito importante, que é realizada nos veículos autônomos. A classificação foi feita com o uso de uma rede neural convolucional, em conjunto com técnicas de regularização, como dropout e data augmentation, com o objetivo de melhorar seu desempenho. Ao total, 12 CNNs foram testadas, abrangendo os otimizadores Adam e RMSprop, as funções de ativação ReLU, Leaky ReLU e Swish, além de testar o desempenho do modelo com o uso de decaimento exponencial da taxa de aprendizado. Dentre os 12 modelos propostos, a CNN com o otimizador Adam, ativação Leaky ReLU e com o uso de decaimento exponencial após a décima época, obteve o melhor desempenho, atingindo 99.66% de acurácia, superando o desempenho humano, cuja média foi de 98.84% e o melhor indivíduo obteve 99.22%.

Palavras-chave: *Redes neurais convolucionais. Visão Computacional. Aprendizado de máquina.*

ABSTRACT

With the growth in the number of traffic accidents each year, specifically those resulting from human error, several initiatives have emerged to deal with this problem, one of which is the use of autonomous vehicles. Therefore, this article aims to classify traffic signs, a subtask, however very important, that is performed in autonomous vehicles. The classification was made using a convolutional neural network, in conjunction with regularization techniques, such as dropout and data augmentation, in order to improve their performance. In total, 12 CNNs were tested, covering the optimizers Adam and RMSprop, the activation functions ReLU, Leaky ReLU and Swish, in addition to testing the model's performance with the use of exponential learning rate decay. Among the 12 models proposed, the CNN with Adam optimizer, Leaky ReLU activation and with the use of exponential decay after the tenth epoch, obtained the best performance, reaching 99.66% accuracy, surpassing human performance, whose average was 98.84% and the best individual obtained 99.22%.

Keywords*: Convolutional neural networks. Computer vision. Machine learning.*

1 INTRODUÇÃO

A cada ano morrem 1,35 milhão de pessoas em decorrência de acidentes de trânsito. De acordo com a NHTSA (*National Highway Traffic Safety Administration*) cerca de 94% dos acidentes fatais são causados por falha humana. Dentro dessa categoria, 41% são decorrentes de erros de reconhecimento, como por exemplo desatenção do motorista, distrações internas e externas. Outra grande parcela, equivalente a 33% é referente a erros de decisão do motorista, como: excesso de velocidade, manobras ilegais e suposições falsas a respeito de outros motoristas (World Health Organization, 2018; SINGH, 2015).

De todos os acidentes causados por falha humana, mais de 74% destes poderiam ser evitados apenas por seguir as leis e recomendações de trânsito. A partir deste problema, surgiram muitas propostas de incentivo ao uso e desenvolvimento de veículos autônomos, uma delas se chama *Preparing for the Future of Transportation: Automated Vehicles 3.0* desenvolvida pelo Departamento de Transporte dos Estados Unidos com o objetivo de utilizar estes veículos autônomos para reduzir acidentes nas vias (U.S DOT, 2018).

Com o aumento do poder computacional, da disponibilidade de dados e do avanço na inteligência artificial, é possível desenvolver algoritmos cada vez mais robustos capazes de superar o desempenho de humanos. De acordo com Haenssle e colaboradores (2018) uma CNN (*Convolutional Neural Network*) treinada foi capaz de superar um grupo de 58 dermatologistas no reconhecimento de câncer de pele. Dado o desempenho desses algoritmos, é possível usá-los em veículos para reconhecer objetos e fazer a sua classificação, como placas de trânsito, outros veículos, pedestres e semáforos.

Dentro da área de visão computacional existem dois ramos principais: detecção e classificação. A detecção é o ato de dada uma imagem, retornar a localização de um objeto específico. A classificação é a segunda etapa do processo, que se resume a classificar ou categorizar tal objeto. Não basta ter uma precisão alta na detecção e falhar na classificação ou vice-versa (KIAEE et al., 2018).

Como citado anteriormente, se do total de acidentes causados por falha humana, cerca de 74% podem ser prevenidos por total atenção na via e cumprimentos das leis, um veículo autônomo que possui uma acurácia alta ao classificar placas de trânsito já seria capaz de eliminar alguns riscos como excesso de velocidade e ultrapassagem em trechos não permitidos (U.S DOT, 2018).

Diante disso, o objetivo deste artigo será desenvolver uma CNN (que já se provou ser um método muito eficiente quando se trata de imagens) capaz de classificar diversos tipos de placas de trânsito e comparar com o desempenho humano. Além disso, serão feitas comparações com diversos métodos ao longo do processo, justificando assim todos os hiperparâmetros escolhidos para esta aplicação.

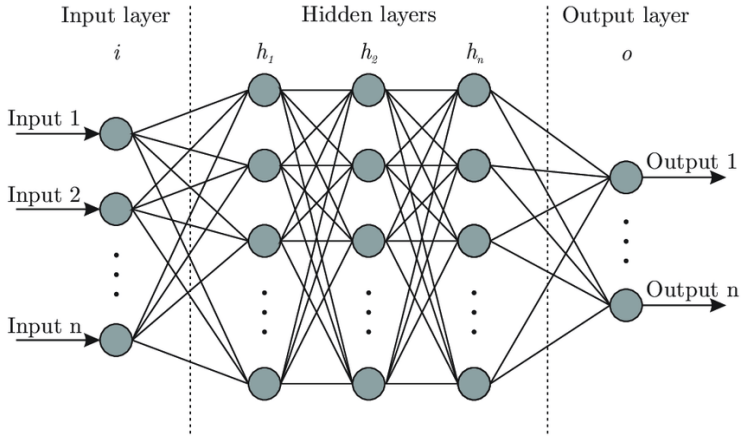
2 REDES NEURAIS ARTIFICIAIS (RNAs)

As redes neurais artificiais, mais especificamente as *Multi Layer Perceptrons* (MLP), se tornaram bastante populares após a publicação de um artigo escrito por Rumelhart, Hinton e Williams (1986), onde um novo procedimento de aprendizagem foi apresentado: o *backpropagation* (também chamado de *backprop*). Por conta da dominância deste algoritmo atualmente, outros métodos não serão abordados. Sendo

assim, o processo de aprendizagem de uma MLP consiste na repetição de 4 etapas: *forward propagation*, cálculo da função de custo (ou perda), *backpropagation* e atualização dos parâmetros.

A partir da Figura 1 é possível identificar 3 tipos diferentes de camadas: entrada (*input*), ocultas (*hidden*) e saída (*output*). Cada círculo cinza representa um *perceptron* (ou neurônio) e cada neurônio está interconectado com as camadas vizinhas, daí surge o nome de camada totalmente conectada, abreviadas por FC (*Fully Connected layers*). Cada conexão entre neurônios possui um parâmetro associado chamado de peso (*weight*), e cada camada possui um viés (*bias*), esses são os parâmetros que serão alterados ao longo do processo de aprendizagem (RAMSUNDAR; ZADEH, 2018).

Figura 1 - Rede neural artificial com múltiplas camadas



Fonte: (SANDHU, 2020)

2.1 *Forward propagation*

O *forward propagation* (também chamado de *forward pass*) é a primeira fase do processo de aprendizagem, que consiste em duas etapas:

1. Calcular a soma ponderada das entradas e adicionar o valor do *bias* à essa soma, conforme equação (1);
2. Aplicar o resultado numa função não-linear chamada de “ativação”, conforme equação (2).

$$Z_j = \sum_{i=1}^{m} x_i \cdot \omega_{ij} + b \qquad (1)$$

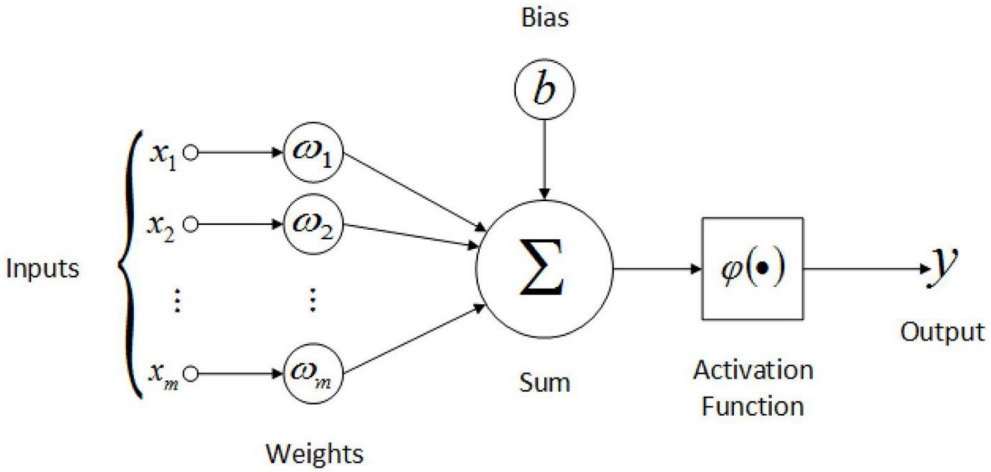
Onde Z_j representa o valor “pré-ativação” do neurônio j. O problema de apenas usar Z na fase do *forward propagation* é que independente de quantas camadas forem adicionadas, o resultado sempre será fruto de uma função linear. Daí surge a necessidade de uma função de ativação, que será responsável por introduzir não-linearidade à rede neural. Desta forma é possível identificar padrões que uma função linear não seria capaz, o que aumenta o poder de generalização da rede (GÉRON, 2019).

$$a_j \approx \hat{y}_j = \varphi_j(Z_j) \qquad (2)$$

Onde φ representa a função de ativação, a é o valor de ativação das camadas intermediárias, enquanto $\hat{y}$ é referente a saída da última camada. É importante ressaltar que esta função deve ser diferenciável, caso contrário não será possível realizar o *backpropagation* (GÉRON, 2019).

A etapa de *forward propagation* para um único neurônio pode ser vista na Figura 2, bastando lembrar que o *output* deste neurônio pode se tornar parte do *input* da próxima camada (caso faça parte de uma camada oculta) (RUMELHART; HINTON; WILLIAMS, 1986).

Figura 2 - Representação de um neurônio artificial



Fonte: (AHIRE, 2018)

2.2 Função de Perda

A saída dos neurônios da última camada (*output layer*) representa a predição da rede neural, ou seja, qual classe tem mais probabilidade de representar o conjunto de dados inserido (as entradas). Para determinar o desempenho da predição, é preciso uma função que meça o quão distante este valor está do resultado real, esta função recebe o nome de “custo” ou “perda”. Em problemas de classificação, a função de perda *cross-entropy* (também chamada de *log loss*) é amplamente utilizada e está definida na equação (3) (NIELSEN, 2015).

$$L = -\sum_{i=1}^{m} y_i \ln \hat{y}_i \tag{3}$$

Onde y_i é um valor binário (0 ou 1) indicando quais classes estão presentes e $\hat{y}_i$ é o valor de ativação referente à classe i. Diante disto, quando $\hat{y}_i$ se aproxima de 1 (valor máximo), a perda tende a 0, o que significa que a predição está muito próxima do valor real. Caso contrário, o valor da perda aumenta, ou seja, a probabilidade de a predição estar certa é muito baixa.

2.3 *Backpropagation*

O *backpropagation* é o responsável por calcular a influência de cada parâmetro no valor da perda, ou seja, seu gradiente. Isso é possível, pois a função de perda é composta, contendo as equações (2) e (1). Desta forma, é possível calcular seu gradiente em relação aos parâmetros através da regra da cadeia do cálculo

diferencial, conforme pode ser visto na equação (4), onde é calculada a taxa de variação da perda referente aos *weights* (GOODFELLOW; BENGIO; COURVILLE, 2016).

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial Z} \frac{\partial Z}{\partial w} \tag{4}$$

A partir da equação (4), é possível obter o gradiente em relação ao *bias* de forma análoga. Todas as equações mostradas levam em conta apenas um neurônio, porém para múltiplas camadas com mais de um neurônio o processo é semelhante. A diferença é que a cada camada adicional, a saída de um neurônio se torna a entrada de outro, estendendo dessa forma a regra da cadeia até as camadas iniciais, o que se torna um problema a depender da função de ativação (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.4 Atualização dos parâmetros

Com o *backpropagation* finalizado, é realizado um procedimento chamado de *gradient descent*, onde cada parâmetro é atualizado de acordo com dois valores: a taxa de aprendizagem e o seu gradiente. Esse procedimento é definido pela equação (5) (RUDER, 2016).

$$w \leftarrow w - \alpha \frac{\partial L}{\partial w} \tag{5}$$

Onde α é a taxa de aprendizagem e determina o tamanho do “passo” que será dado em direção ao ponto mínimo da função L. Esse processo é realizado até que o valor da métrica utilizada seja satisfeita ou após um número determinado de iterações acontecer (chamadas de épocas, do inglês *epochs*).

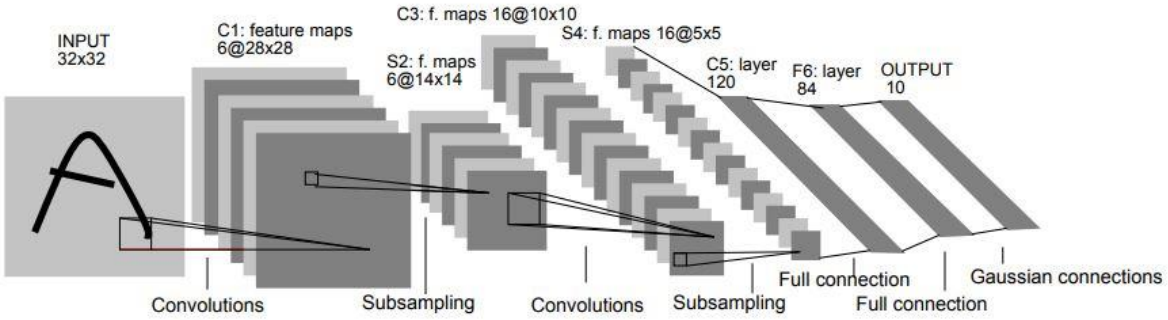
3 REDES NEURAIS CONVOLUCIONAIS

Em 1998, Yann LeCun e colaboradores publicaram um artigo que foi um marco na área de redes neurais, principalmente no campo da classificação de imagens. Neste artigo foi apresentada a rede neural convolucional (do inglês *Convolutional Neural Networks*, comumente abreviada por CNN), que foi projetada especificamente para lidar com dados 2D. De forma geral, as CNNs são capazes de extrair características locais e compartilhá-las por toda a imagem, ao contrário das MLPs, que tratam cada *pixel* diferentemente, o que resulta em dois problemas: aumenta consideravelmente a quantidade de parâmetros que serão treinados, e possui redundância, já que *pixels* próximos possuem alta correlação entre si. Características elementares como bordas horizontais e verticais que são identificadas numa região da imagem, provavelmente também serão úteis em outras regiões, dessa forma é possível reduzir de forma significativa a quantidade de parâmetros a serem treinados, além de melhorar o desempenho da rede neural.

Uma CNN básica geralmente possui uma arquitetura sequencial composta por 3 tipos de camadas: convolução, *pooling* e camadas totalmente conectadas. No processo de aprendizagem são realizadas as mesmas etapas que numa MLP, a diferença são as operações realizadas em cada etapa, que serão determinadas pelo tipo de camada (O’SHEA; NASH, 2015).

Na Figura 3 está representada a *LeNet-5*, uma CNN criada com o objetivo de classificar diferentes dígitos manuscritos de 0 a 9, contidos no conhecido *dataset* MNIST (*Modified National Institute of Standards and Technology*). Nela é possível identificar 3 tipos de operações: convoluções, *subsampling* e *full connection*. A operação de *subsampling* é essencialmente a mesma operação que o *pooling*, embora este último termo seja mais usado atualmente (LECUN *et al.*, 1998).

Figura 3 - Representação da *LeNet-5*

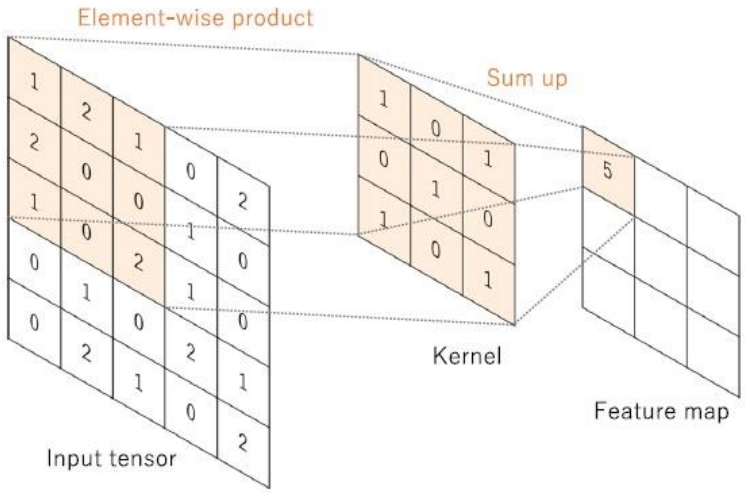


Fonte: (LECUN *et al.*, 1998)

3.1 Camada convolucional

O objetivo de uma camada convolucional é extrair o máximo de características relevantes de uma imagem. Esse processo é feito por uma operação linear chamada de convolução, composta por duas etapas: o produto entre os elementos do tensor de entrada e o *kernel* (também chamado de filtro), e a soma desses produtos, como pode ser visualizado na Figura 4 (YAMASHITA *et al.*, 2018).

Figura 4 - Processo de convolução



Fonte: Adaptado de (YAMASHITA *et al.*, 2018)

A imagem que será classificada pela CNN está representada pelo *input tensor* na Figura 4, o *kernel* também é um tensor, porém, seus elementos não são constantes como o de entrada, são parâmetros que serão treinados no processo de aprendizagem feito pelo *backpropagation*. A operação de convolução pode ser definida de forma geral pela equação (6) de acordo com Ian Goodfellow e colaboradores (2016).

$$(K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \quad (6)$$

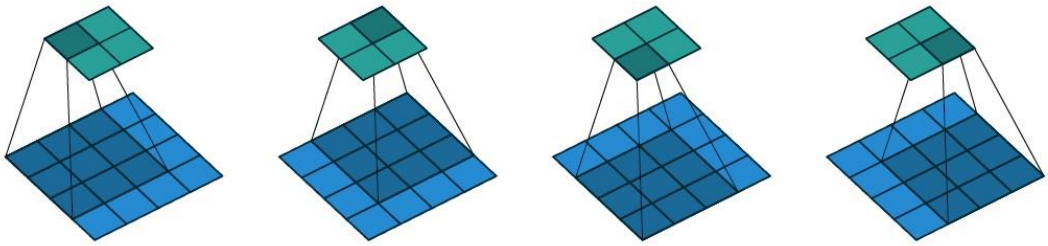
Onde K é o *kernel*, I é a imagem e o asterisco representa a operação de convolução. Os índices i, j, m e n, são referentes às dimensões da imagem (i, j) e do *kernel* (m, n). A Figura 4 pode ser reproduzida pela equação (6), onde $i = 0$, $j = 0$, $m = 3$ e $n = 3$, dessa forma cada elemento da imagem é multiplicado pelo seu correspondente do *kernel* e após o somatório, o resultado representa um elemento do mapa de características (do inglês *feature map*), que uma vez completo, podem representar desde características elementares, como bordas verticais e horizontais, até mais complexas como formatos de rostos (GOODFELLOW; BENGIO; COURVILLE, 2016; YAMASHITA *et al.*, 2018).

Numa camada de convolução, existem 4 hiperparâmetros (que não serão ajustados pelo *backpropagation*) que devem ser definidos: a quantidade de filtros, o tamanho do *kernel*, o valor do *stride* e o valor do *padding*, sendo os dois primeiros autoexplicativos. O *stride* representa o tamanho do "passo" do *kernel* por iteração, ou seja, a distância entre a posição de dois *kernels* consecutivos, sendo assim, quanto maior o valor do *stride*, menor será a resolução do mapa de características. A equação (7) define o tamanho do *feature map* após aplicar o *stride* (ALBAWI; MOHAMMED; ALZAWI, 2017; YAMASHITA *et al.*, 2018).

$$M = 1 + \frac{N - F}{S} \quad (7)$$

Onde N representa o tamanho da imagem ($N \times N$), F o tamanho do filtro ou *kernel* ($F \times F$) e S representa o valor do *stride*. Aplicando a equação (7) à Figura 4, tem-se uma imagem de tamanho 5, um *kernel* de tamanho 3 e um *stride* de 1, portanto, o *feature map* resultante deve ter tamanho 3×3 de acordo com a equação, o que pode ser confirmado observando a Figura 4.

Figura 5 - Representação da convolução sem *padding*



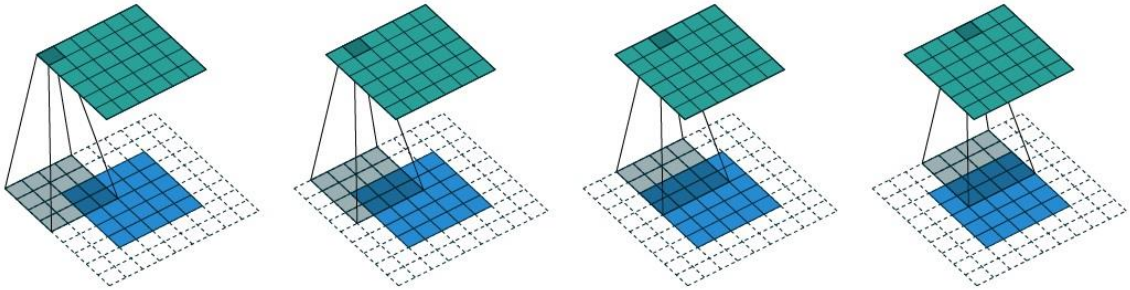
Fonte: (DUMOULIN; VISIN, 2018)

Durante a convolução, é possível notar através da Figura 5, que os cantos da imagem são utilizados na operação somente uma vez, desta forma muita informação nessa região é perdida. Para resolver este problema, adicionam-se bordas à imagem, geralmente com valores iguais a 0. Este processo é chamado de *zero-padding*, onde novos pixels são adicionados à imagem, porém, não terão impacto na convolução, pois são nulos. O tamanho do mapa de características após o *stride* e *padding* é definido pela equação (8) (ALBAWI, MOHAMMED; ALZAWI, 2017; DUMOULIN; VISIN, 2018).

$$M = 1 + \frac{N + 2P - F}{S} \quad (8)$$

Onde P representa o valor do *padding*. A Figura 6 mostra este processo para $N = 5$, $P = 2$, $F = 4$ e $S = 1$. Como resultado, M deve ser igual a 6 para satisfazer a equação (8) e de fato é o que a figura apresenta.

Figura 6 - Representação da convolução com *padding* igual a 2



Fonte: (DUMOULIN; VISIN, 2018)

3.2 Camada *Pooling*

O objetivo principal da camada *pooling* é reduzir a complexidade para as próximas camadas, tornando a CNN invariante a pequenas distorções. As duas formas mais populares de operações *pooling* se chamam *max pooling* e *average pooling*. Ao contrário da camada convolucional, o resultado dos produtos entre a entrada e o *kernel* não serão somados. Se a operação *max pooling* for usada, apenas o maior valor será passado ao *feature map*, analogamente, se a operação *average* for utilizada, será passada a média dos valores. É bastante comum um *kernel* da camada *pooling* possuir *stride* 2 e tamanho 2×2, reduzindo a dimensão da imagem pela metade (ALBAWI; MOHAMMED; ALZAWI, 2017; YAMASHITA *et al.*, 2018).

4 FUNÇÕES DE ATIVAÇÃO

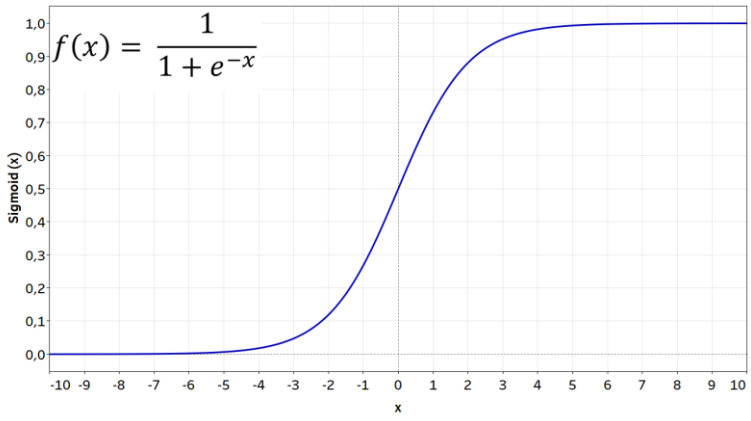
As funções de ativação são muito importantes numa rede neural, pois irão adicionar não-linearidade ao modelo, tornando-o capaz de representar polinômios de alto grau, melhorando assim o seu desempenho durante o treinamento. Com o avanço do poder computacional, as redes neurais se tornaram mais profundas, o que gerou alguns problemas, como o *vanishing gradient*, decorrente da multiplicação em cadeia dos gradientes (que geralmente tem valor menor que 1), fazendo com que a cada passo após o *backpropagation*, o valor resultante diminua. Dessa forma, os parâmetros das primeiras camadas praticamente não são atualizados (NWANKPA *et al.*, 2018).

Por muito tempo, a função sigmóide foi utilizada na implementação das redes neurais, porém logo foi substituída por melhores alternativas. Atualmente, a mais utilizada é a função *ReLU* e suas variações (*Leaky ReLU*, por exemplo), porém, assim como a função sigmóide apresentou problemas, o mesmo ocorre com a *ReLU*, e outra possível alternativa surgiu: a função *Swish* (PEDAMONTI, 2018; RAMACHANDRAN; ZOTH; LE, 2018).

4.1 *Sigmóide*

A função sigmóide pode ser vista na Figura 7, onde o seu valor varia de 0 a 1, sendo diferenciável em todo o domínio. Glorot e Bengio (2010) descobriram através de experimentos, que esta função é bastante suscetível ao *vanishing gradient*, já que o maior valor possível da sua derivada é igual 0,25. Dessa forma, não é recomendado utilizá-la para redes mais profundas, pois os valores atualizados pelo gradiente descendente tendem a 0 nas camadas iniciais, porém, ainda assim, ela é bastante usada em problemas de classificação binários (apenas 2 classes), mas apenas na camada de saída.

Figura 7 - Gráfico da função sigmóide

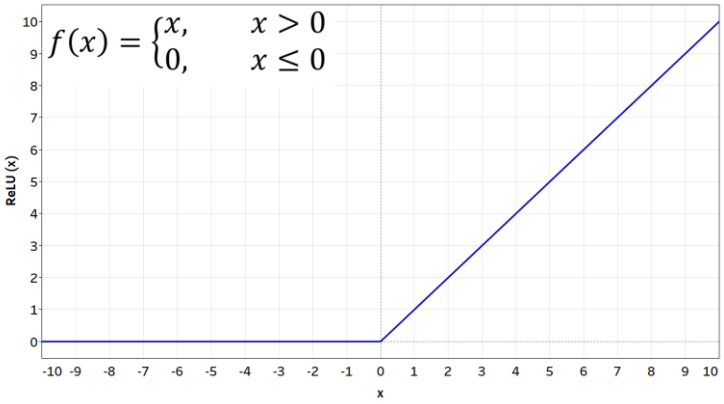


Fonte: Autoria própria (2020)

4.2 *Rectified Linear Unit* (*ReLU)*

Para resolver os problemas de *vanishing* e *exploding gradient* (os gradientes se tornam cada vez maiores), foi proposta a utilização da função *ReLU*, que pode ser visualizada na Figura 8. Para valores menores que 0, os parâmetros não são atualizados e acima de 0 o seu gradiente equivale a 1. Desta forma, é possível evitar tanto o crescimento, quanto o decaimento descontrolado dos gradientes, além de ser uma função com baixo custo computacional (PEDAMONTI, 2018).

Figura 8 - Gráfico da função ReLU

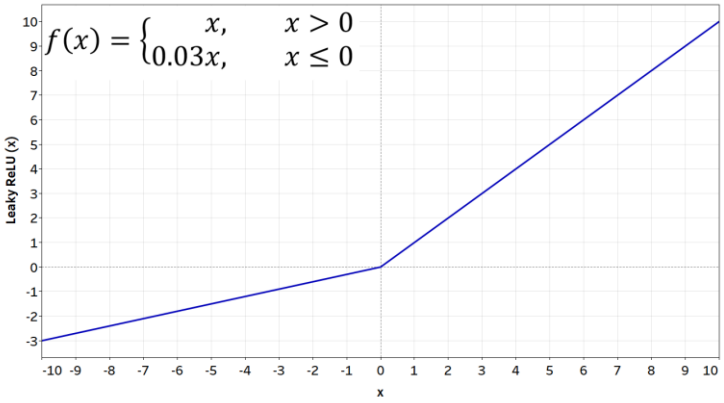


Fonte: Autoria própria (2020)

4.3 *Leaky ReLU*

Um problema decorrente do uso da *ReLU*, é que devido ao gradiente ser 0 para valores negativos, alguns neurônios podem ficar “presos” nessa situação e nunca sofrerem alteração após o *backpropagation*. Para resolver isto, é adicionada uma inclinação no domínio negativo, garantindo que todos os neurônios sejam atualizados. A função *Leaky ReLU* está representada na Figura 9, com o valor da inclinação igual a 0,03 (MAAS; HANNUN; NG, 2013).

Figura 9 - Gráfico da função *Leaky ReLU* para $\alpha = 0{,}03$

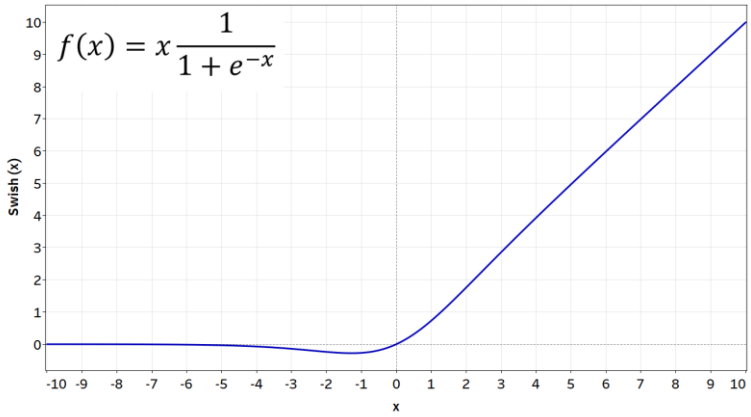


Fonte: Autoria própria (2020)

4.4 *Swish*

Apesar de a *ReLU* e suas variações obterem desempenhos muito bons, Ramachandran, Zoth e Le (2017) apresentaram uma nova função de ativação capaz de superá-las: *Swish*, que pode ser vista na Figura 10. Esta função obteve melhor desempenho em diversos *datasets* em comparação à *ReLU*, como por exemplo, o CIFAR-100. Os gráficos das duas funções são muito semelhantes (como pode ser visto na Figura 8 e Figura 10), porém a *Swish* é diferenciável em todo o seu domínio, o que a torna uma função mais suave, em contraste com o “salto” que a *ReLU* possui na vizinhança do 0 (RAMACHANDRAN; ZOPH; LE, 2018).

Figura 10 - Gráfico da função *Swish*



Fonte: Autoria própria (2020)

5 OTIMIZADORES

O gradiente descendente foi apresentado na equação (5), como uma forma de atualizar os parâmetros na direção do ponto mínimo da função. Esse processo é chamado de otimização e são executados pelo otimizador. O gradiente descendente é um dos métodos de otimização mais antigos e comuns, que consiste em atualizar os parâmetros na direção oposta do gradiente. Apesar de convergir para o mínimo da função, esse processo não é feito de forma rápida devido as oscilações nas mudanças dos gradientes (SUN *et al.*, 2019).

Um método criado com o objetivo de resolver este problema, é o *momentum*. Este otimizador ao invés de tratar cada "passo" individualmente, os "passos" anteriores são considerados, ou seja, os gradientes anteriores contribuem diretamente para o cálculo do próximo gradiente. Outro algoritmo chamado *AdaGrad* faz uma melhoria no *momentum*, que permite alterar a taxa de aprendizado de forma adaptativa, diminuindo-a à medida que se aproxima do ponto mínimo. O *momentum* pode ser definido pela equação (9) (RUDER, 2016).

$$v_t \leftarrow \gamma v_{t-1} + \alpha \frac{\partial L}{\partial w} \qquad (9)$$
$$w \leftarrow w + v_t$$

Onde v_t representa a soma exponencial dos gradientes no instante t e γ se chama o termo do *momentum* e representa a fração que os gradientes anteriores irão receber no cálculo do próximo (varia de 0 a 1). Assim, é possível acelerar o processo de convergência, pois a cada iteração, o "passo" na direção do ponto mínimo irá aumentar (daí o termo *momentum*), pois os gradientes antigos irão se acumular.

5.1 RMSprop

O otimizador RMSprop, definido na equação (10), faz um pequeno incremento ao *momentum* e ao *AdaGrad*, onde ao invés de ser calculado o acúmulo de todos os gradientes passados, é calculada a média exponencial quadrada num intervalo definido, desta forma, a convergência ocorre mais rápido, pois os gradientes passados não irão afetar a taxa de aprendizado da mesma forma que no *AdaGrad*, fazendo com que o RMSprop convirja para mínimo global mais rapidamente (SUN *et al.*, 2019).

$$V_t = \sqrt{\beta V_{t-1} + (1-\beta){g_t}^2} \qquad (10)$$
$$w_{t+1} = w_t - \alpha \frac{g_t}{V_t}$$

Onde V_t representa a média exponencial quadrada dos gradientes no instante t, β é o parâmetro de decaimento exponencial, g_t é o gradiente do parâmetro (nesse caso, w_t).

5.2 *Adaptative Moment Estimation* (Adam)

Adam é um otimizador que foi introduzido por Diederik Pieter Kingma e Jimmy Lei Ba (2015), onde foi proposta uma combinação entre o algoritmo de *momentum* e o RMSprop. Há apenas uma alteração ao introduzir o *momentum* ao Adam, que ao invés de ser calculada a soma dos gradientes passados, é calculada a média

exponencial, como pode ser vista na equação (11). O Adam também calcula a média exponencial quadrada dos gradientes anteriores, da mesma forma que no RMSprop, porém a equação que define o "passo" é diferente, já que serão consideradas duas medidas combinadas (GERÓN, 2019).

$$v_t = \beta_1 v_{t-1} - (1-\beta_1) g_t$$
$$V_t = \sqrt{\beta_2 V_{t-1} - (1-\beta_2) {g_t}^2} \quad (11)$$
$$w_{t+1} = w_t - \alpha \frac{v_t}{V_t} g_t$$

Onde v_t representa a média exponencial dos gradientes anteriores e V_t a média exponencial quadrada dos gradientes anteriores, analogamente ao *momentum* e RMSprop, respectivamente.

6 TÉCNICAS DE REGULARIZAÇÃO

Após o processo de aprendizagem, é comum ocorrer um fenômeno chamado *overfitting*, que é quando a rede neural aprende tão bem o conjunto de treino que não é capaz de generalizar para os conjuntos de teste. Uma das causas disso é decorrente da complexidade do modelo, ou seja, redes neurais com muitas camadas tendem a cair em *overfitting* devido à alta capacidade de se ajustar ao conjunto de dados, pois podem representar funções muito complexas (CARUANA; LAWRENCE; GILES, 2000).

6.1 *Dropout*

Uma das técnicas de regularização desenvolvidas para lidar com esse problema, se chama *dropout*. Essa técnica consiste na exclusão aleatória de uma parcela de neurônios durante o aprendizado, desta forma, é possível reduzir a complexidade do modelo, forçando-o a realizar predições com apenas uma amostra da rede neural. A cada iteração, são selecionados neurônios aleatórios para serem "excluídos", o que acaba afetando o desempenho durante a etapa de treino, porém resulta numa melhor generalização no conjunto de teste (SRIVASTAVA *et al.*, 2014).

6.2 *Data augmentation*

Essa técnica tem o objetivo de ampliar o conjunto de treino, realizando diversas transformações nos dados, criando "cópias" com algumas modificações. Com imagens, por exemplo, é possível gerar diversas imagens rotacionadas, ampliadas ou deslocadas a partir de uma imagem original, como pode ser observado na Figura 11. Com isso, mais imagens são processadas, aumentando o conjunto de treino e reduzindo a chance de *overfitting* (SHORTEN; KHOSHGOFTAAR, 2019).

Figura 11 - Exemplos gerados com o uso de *data augmentation*

Fonte: Autoria própria (2020)

7 METODOLOGIA

Todas as ferramentas e informações necessárias para reproduzir este artigo, serão descritas nessa seção, incluindo as versões das bibliotecas e plataformas usadas, além de todos os hiperparâmetros selecionados, com as respectivas justificativas.

7.1 Base de dados e pré-processamento

A base de dados (do inglês *dataset*) utilizada neste trabalho foi retirada do *site* do Instituto de Neuroinformática (INI), com sede em Bochum, Alemanha. Este *dataset* foi utilizado numa competição chamada GTSRB (*German Traffic Sign Recognition Benchmark*), realizada em 2011 com o intuito de estimular o desenvolvimento de novos algoritmos que pudessem superar a *performance* humana nesta tarefa (classificação de placas de trânsito) (STALLKAMP *et al.*, 2012).

O *dataset* possui um total de 39.209 imagens para treino e 12.630 para teste, contendo 43 classes diferentes, numeradas de 0 a 42. A identificação de cada placa está descrita na Tabela 1.

Tabela 1 - Numeração e descrição das placas

Nº	Descrição	Nº	Descrição
0	20 km/h	22	Estrada irregular
1	30 km/h	23	Estrada escorregadia
2	50 km/h	24	Estrada estreita à direita
3	60 km/h	25	Obras rodoviárias
4	70 km/h	26	Semáforo
5	80 km/h	27	Faixa de pedestre
6	Fim: 80 km/h	28	Crianças
7	100 km/h	29	Ciclistas
8	120 km/h	30	Estrada com gelo ou neve
9	Proibido ultrapassar	31	Animais selvagens
10	Proibida ultrapassagem para caminhões	32	Fim de restrição
11	Cruzamento	33	Apenas direita

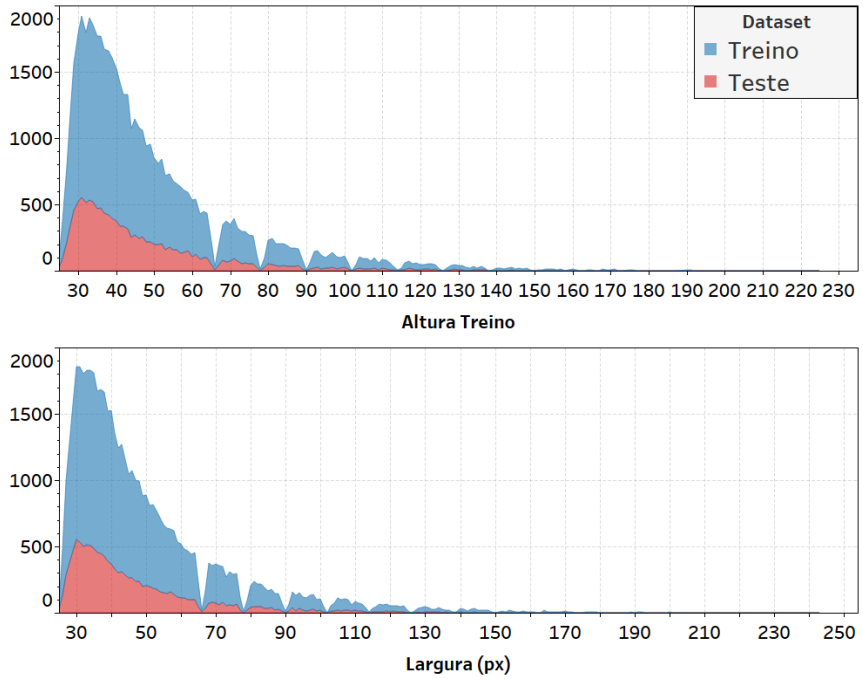
Continua

Nº	Descrição	Nº	Descrição
12	Pista preferencial	34	Apenas esquerda
13	Dê a preferência	35	Siga em frente
14	Pare	36	Apenas direita ou em frente
15	Nenhum veículo permitido	37	Apenas esquerda ou em frente
16	Proibido caminhões	38	Mantenha-se à direita
17	Não entre	39	Mantenha-se à esquerda
18	Perigo	40	Rotatória
19	Curva à esquerda	41	Fim: proibido ultrapassar
20	Curva à direita	42	Fim: proibida ultrapassagem para caminhões
21	Curva dupla, primeira à esquerda		

Fonte: Autoria própria (2020)

A maior parcela das imagens apresenta resolução (medida em pixels) por volta de 30x30, como pode ser visto na Figura 12, portanto todas foram redimensionadas para 32x32, já que a CNN necessita que todas tenham o mesmo tamanho. Além disso, a distribuição de imagens por classe não está balanceada, variando desde 210 imagens para as placas de 20km/h até 2.220 para as placas de 30km/h. Apesar do desbalanceamento, com o uso de *data augmentation* é possível mitigar este problema. Modificações aleatórias foram aplicadas às imagens durante o treinamento, incluindo rotação de até 10 graus, taxa de deslocamento vertical e horizontal de até 10% e *zoom* de até 15%.

Figura 12 - Distribuição da resolução das imagens



Fonte: Autoria própria (2020)

Todas as imagens possuem 3 dimensões, estando no espaço de cores RGB. De acordo com Sermanet e LeCun (2011), o espaço de cores YUV demonstrou maior desempenho em extrair características relevantes utilizando apenas o canal Y, que pode ser obtido a partir da imagem RGB através da equação (12).

$$Y = 0{,}2125R + 0{,}7154G + 0{,}0721B \quad (12)$$

Onde R, G e B representam os valores normalizados da intensidade de cada pixel no seu respectivo canal (*Red, Green, Blue*). A Figura 13 mostra algumas imagens antes e depois de serem convertidas.

Figura 13 - Amostras do *dataset* em RGB (acima) e em Y (abaixo)

Fonte: Autoria própria

A validação cruzada *holdout* foi utilizada, já que os *datasets* de treino e teste já vêm separados, portanto, não houve necessidade de dividir o conjunto de treino em treino/validação ou utilizar o método *k-folds*.

6.2 *Tensorflow* e *Keras*

Tensorflow é uma plataforma desenvolvida pela Google, especializada em algoritmos de aprendizado de máquina, já o *Keras* é uma biblioteca de alto nível implementada em cima do *Tensorflow*, criada com o objetivo de desenvolver modelos e arquiteturas de redes neurais de uma maneira simples e intuitiva. A versão do *Tensorflow* 2.1 foi utilizada, onde o *Keras* foi incorporado à plataforma como um módulo interno (ABADI *et al.*, 2016).

Por se tratar de uma plataforma de alto nível, nenhuma etapa da aprendizagem precisou ser implementada do zero em *Python*, ao invés disso foram usadas funções já desenvolvidas pelo *Tensorflow*, pois já estão otimizadas e foram escritas em *C++*, o que aumenta consideravelmente a velocidade de processamento em comparação a *Python*.

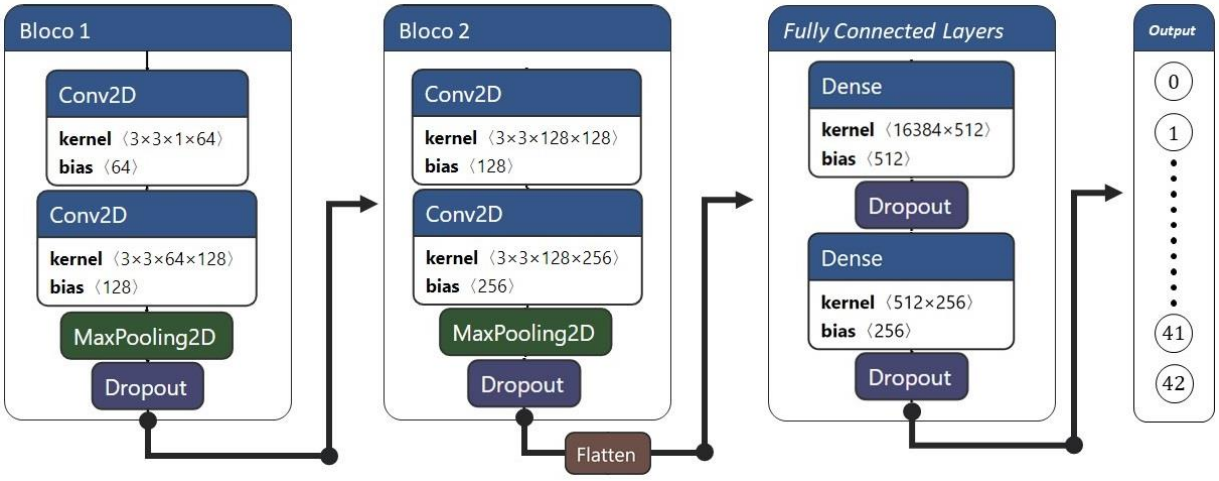
6.3 Arquitetura da CNN

Geralmente as camadas de convolução são seguidas de uma camada *pooling* e isso compõe um bloco. Porém, várias redes neurais mais robustas e conhecidas, como a VGG, *Inception* e ResNet possuem várias camadas de convolução antes da *pooling*. Desta forma é possível extrair mais características antes da informação ser perdida. Por se tratar de imagens relativamente pequenas, duas camadas de convolução são suficientes, seguidas de uma *pooling* e um *dropout* para evitar *overfitting* (SIMONYAN; ZISSERMAN, 2015; SZEGEDY *et al.*, 2015; HE *et al.*, 2016).

A arquitetura geral da rede neural pode ser vista na Figura 14, dividida em 3 blocos e uma camada de saída, sendo os dois primeiros blocos responsáveis pela etapa convolucional e o último bloco contendo as camadas totalmente conectadas. Todas as camadas convolucionais possuem um *stride* de 1 e *padding* do tipo "*same*", evitando a redução de dimensão após a convolução. As camadas *pooling* possuem

kernels com tamanho 2x2. Os valores de *dropout* para os blocos de convolução possuem valor 0.25 e para o bloco FC possuem valor 0.5.

Figura 14 - Arquitetura da CNN utilizada



Fonte: Autoria própria

Como pode ser observado na Figura 14, no primeiro bloco são utilizados 64 e 128 filtros nas duas primeiras camadas de convolução, ambos com tamanho 3x3. No segundo bloco, as duas camadas de convolução possuem 128 e 256 filtros, respectivamente, e novamente filtros de tamanho 3x3.

Após as camadas convolucionais foi feita a operação de *flatten*, com o objetivo de preparar os resultados para as camadas FC. São utilizadas duas camadas totalmente conectadas ao final da rede, possuindo 512 e 256 neurônios, respectivamente. A última camada, como se espera, possui 43 neurônios.

6.4 Treinamento

Durante o processo de treinamento, é possível monitorar as métricas a cada *epoch* com o uso de um método do *Keras* chamado *callbacks*, portanto, foram utilizadas técnicas como *early stopping* e *learning rate decay*. Além disso, o modelo salvo não representa o resultante da última época, mas sim o que obteve melhor desempenho durante o treinamento.

A acurácia no conjunto de teste é avaliada a cada época; não havendo alteração maior que 0,0001 após 10 épocas, o treinamento é interrompido. Foram treinados ao total 12 modelos, sendo eles, a combinação entre os otimizadores Adam e RMSprop, as funções de ativação *ReLU*, *Leaky ReLU* e *Swish*, e a presença ou ausência do decaimento exponencial da taxa de aprendizado. Nos casos sem decaimento exponencial definido, o próprio otimizador se adapta no decorrer do treinamento. Com o decaimento definido, a taxa de aprendizado começa em 0,001 e decai exponencialmente após a décima época.

7 RESULTADOS E DISCUSSÕES

Os resultados obtidos após o treinamento dos 12 modelos, podem ser vistos na Tabela 2, com os melhores resultados por função de ativação destacados em negrito. Para as 3 diferentes funções de ativação, os modelos que utilizaram o otimizador Adam obtiveram os melhores resultados. E dentre as funções de ativação, a *Leaky*

ReLU obteve o melhor desempenho em 3 das 4 configurações possíveis.

Por questão de visualização, o parâmetro de decaimento da taxa de aprendizado foi abreviado para LRS (*Learning Rate Schedule*).

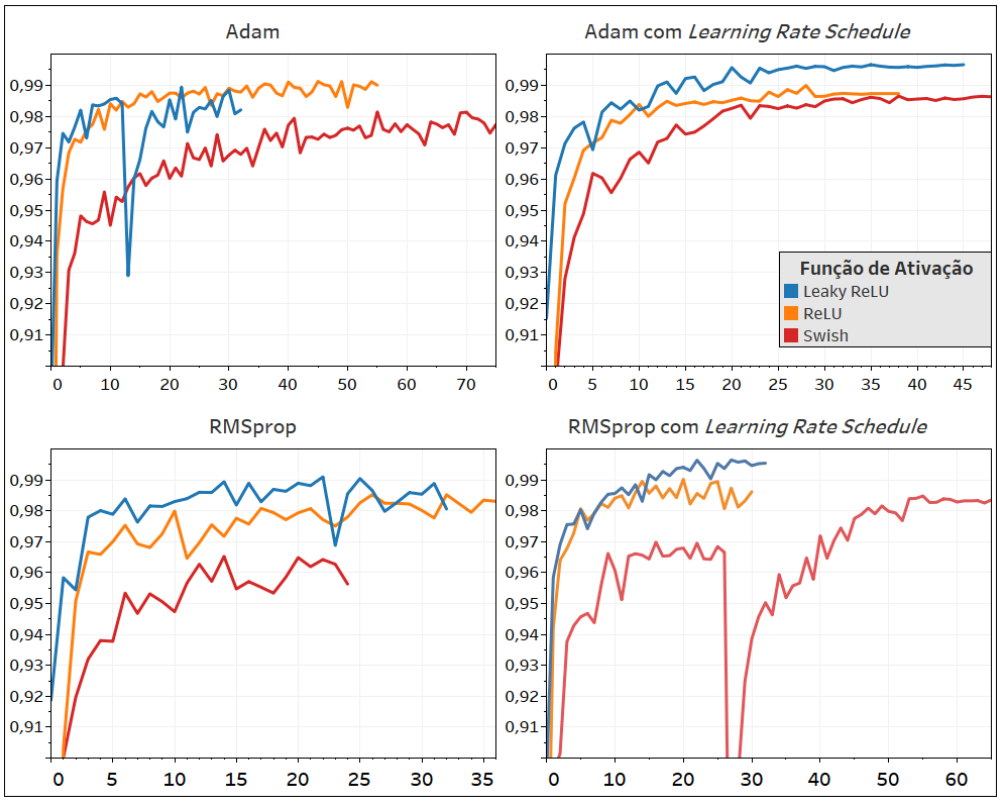
Tabela 2 - Acurácia dos modelos, em %

	Adam	Adam LRS	RMSprop	RMSprop LRS
ReLU	**99,12**	98,99	98,52	99,02
Leaky ReLU	98,94	**99,66**	99,10	99,64
Swish	98,15	**98,65**	96,52	98,48

Fonte: Autoria própria

As curvas de aprendizado de época *versus* acurácia de cada modelo podem ser vistas na Figura 15. Dentre os 12 modelos, o que obteve o melhor desempenho utilizou o Adam como otimizador, *Leaky ReLU* como ativação e foi utilizado o decaimento exponencial, atingindo 99,66% de acurácia na *epoch* 35.

Figura 15 - Desempenho entre Adam e RMSprop (*Epoch* x Acurácia)



Fonte: Autoria própria

De acordo com Stallkamp e colaboradores (2012), responsáveis por hospedar a competição, foram abordadas 32 pessoas para rotular uma amostra do *dataset* contendo 400 imagens. O desempenho humano nesta competição foi dividido em dois resultados: a média de todos os participantes e o melhor desempenho dentre eles. Ambos podem ser vistos na Tabela 3, juntamente com o modelo que obteve maior acurácia na Tabela 2.

Tabela 3 - Comparação com o desempenho humano

Método	Acurácia (%)
Desempenho humano (média)	98,84
Desempenho humano (melhor indivíduo)	99,22
CNN proposta	**99,66**

Fonte: Autoria própria

Desta forma, é possível identificar que a acurácia da CNN superou de forma significativa o desempenho humano, incluindo o melhor indivíduo. Das 12.630 imagens de teste, este modelo errou a classificação em apenas 45. Dentre as 45 imagens, foi observado que muitas delas possuíam características extremas, como iluminação muito forte ou muito fraca, placas tortas e obstruídas por objetos.

8 CONCLUSÕES

Diante dos resultados obtidos, com uma arquitetura de rede neural relativamente simples e pouco poder computacional, foi possível mostrar que embora os seres humanos possuam uma alta habilidade no reconhecimento de imagens, algoritmos mais avançados como as redes neurais convolucionais, são capazes de superá-los nessas atividades, trazendo mais segurança e eficiência nesta tarefa.

Embora o ato de controlar um veículo autônomo não se resuma à classificação de placas de trânsito, esta etapa é fundamental diante de todo o processo, pois sem a correta classificação de objetos como pedestres, outros veículos e placas de trânsito, a segurança do passageiro não pode ser garantida, tornando o veículo autônomo um projeto inviável. Apesar das imagens terem resoluções relativamente pequenas (32x32 pixels), todos algoritmos podem ser ampliados para trabalhar com imagens de qualidade superior, não se restringindo a este caso específico.

Apesar de apenas o melhor modelo ter sido selecionado na comparação com o desempenho humano, mais da metade de todos os modelos utilizados superaram a média dos indivíduos, deixando evidente o potencial das redes neurais em substituir humanos na tarefa de reconhecimento de imagens, dada sua confiabilidade e desempenho.

REFERÊNCIAS

ABADI, Martín et al. *TensorFlow:* Large-scale machine learning on heterogeneous distributed systems. [S.l.]: ArXiv:1603.04467, 2016. Disponível em: <https://arxiv.org/abs/1603.04467>. Acesso em: 8 de jun. 2020.

AHIRE, Jayesh. *The Artificial Neural Networks Handbook:* Part 4. [S.l.], nov. 2018. Disponível em: <https://medium.com/@jayeshbahire/the-artificial-neural-networks-handbook-part-4-d2087d1f583e> Acesso em 8 de jun. 2020.

ALBAWI, Saad; MOHAMMED, Tareq; ALZAWI, Saad. *Understanding of a Convolutional Neural Network*. In: The International Conference on Engineering and Technology, 2017, Antalya, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.

CARUANA, Rich; LAWRENCE, Steve; GILES, C. Lee. *Overfitting in neural nets:* Backpropagation, conjugate gradient, and early stopping. In: Advances in Neural Information Processing Systems, v.13, p402, MIT Press, 2000, Denver.

DUMOULIN, Vincent; VISIN, Francesco. *A guide to convolution arithmetic for deep learning*. [S.l]: Arxiv: 1603.07285, 2018. Disponível em: <https://arxiv.org/abs/1603.07285>. Acesso em: 8 jun. 2020.

GÉRON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* 2. ed. Farnham: O'Reilly UK Ltd., 2019.

GLOBAL STATUS REPORT ON ROAD SAFETY 2018. Geneva: World Health Organization; 2018. Licence: CC BY-NC-SA 3.0 IGO.

GLOROT, Xavier; BENGIO, Yoshua. *Understanding the difficulty of training deep feedforward neural networks.* In: International Conference on Artificial Intelligence and Statistics, 13., 2010, Montreal.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. [S.l.]: MIT Press, 2016; Disponível em: <http://www.deeplearningbook.org>. Acesso em: 8 jun. 2020.

HAENSSLE, H.A et al. Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists. *Annals of Oncology*, [s.l.], v. 29, n. 8, p.1836 – 1842, ago. 2018. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0923753419341055>. Acesso em: 8 jun. 2020.

HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. *Deep residual learning for image recognition*. In: Computer Vision and Pattern Recognition, 2016, Las Vegas.

KIAEE, Nadia; HASHEMIZADEH, Elham; ZARRINPANJEH, Nima. A survey on Object detection and Classification methods. In: The International conference on Intelligent Decision Science, 3., 2018, Tehran.

KINGMA, Diederik P.; BA, Jimmy. *Adam: A method for stochastic optimization*. In: International Conference for Learning Representations, 3., 2015, San Diego.

LECUN, Y.; BOTTOU, L.; HAFFNER, P. Gradient-based learning applied to document recognition*. Institute of Electrical and Electronics Engineers,* New York, v.86, n.11, p. 2278-2324, dez. 1998. Disponível em: <http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf>. Acesso em: 8 jun. 2020.

MAAS, Andrew; HANNUN, Awni; NG, Andrew. *Rectifier nonlinearities improve neural network acoustic models*. In: International Conference on Machine Learning, v.30, 2013, Stanford.

NIELSEN, Michael A. *Neural Networks and Deep Learning.* [S.l.]: Determination Press, 2015.

NWANKPA, C.; IJOMAH, W.; GACHAGAN, A.; MARSHALL, S. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning.* [S.l]: Arxiv: 1811.03378, 2018. Disponível em: <https://arxiv.org/abs/1811.03378>. Acesso em: 8 jun. 2020.

O'SHEA, K.; NASH, R. *An introduction to convolutional neural networks*. Lancashire: Arxiv:1511.08458, 2015. Disponível em: <https://arxiv.org/abs/1511.08458>. Acesso em: 8 jun. 2020.

PEDAMONTI, D. *Comparison of non-linear activation functions for deep neural networks on MNIST classification task.* Edinburgo: Arxiv: 1804.02763, 2018. Disponível em: <https://arxiv.org/abs/1804.02763>. Acesso em: 8 jun. 2020.

RAMACHANDRAN, P.; ZOPH, B.; LE, Q. V. *Searching for activation functions*. [S.l]: Arxiv: 1710.05941, 2018. Disponível em: <https://arxiv.org/abs/1710.05941>. Acesso em: 8 jun. 2020.

RAMSUNDAR, Bharath; ZADEH, Reza Bosagh. *TensorFlow for Deep Learning.* O'Reilly Media, [s.l], 2018.

RUDER, Sebastian. *An overview of gradient descent optimization algorithms.* [S.l]: Arxiv:1609.04747, 2016. Disponível em: <https://arxiv.org/abs/1609.04747>. Acesso em: 8 jun. 2020.

RUMELHART, D.; HINTON, G.; WILLIAMS, R. Learning internal representations by backpropagating errors. *Nature*, [s.l.], v.323, p. 533–536, jul. 1986.

SANDHU, Sunil. *How Neural Networks process input data*. AI In Plain English, abr 2020. Disponível em: <https://medium.com/ai-in-plain-english/my-notes-on-neural-networks-adf3e49657f8>. Acesso em: 8 de jun. 2020.

SERMANET, Pierre; LECUN, Yann. *Traffic sign recognition with multi-scale convolutional networks.* In: International Joint Conference on Neural Networks, 2011, San Jose. Disponível em: <https://ieeexplore.ieee.org/document/6033589?arnumber=6033589>. Acesso em: 8 de jun. 2020.

SHORTEN, Connor; KHOSHGOFTAAR, Taghi. A survey on image data augmentation for deep learning. *Journal of Big Data,* Boca Raton, v.6, n. 60, 2019. Disponível em: <https://doi.org/10.1186/s40537-019-0197-0>. Acesso em: 8 de jun. 2020.

SIMONYAN, Karen; ZISSERMAN, Andrew. *Very deep convolutional networks for large-scale image recognition*. In: Computer Vision and Pattern Recognition, 2015, Boston.

SINGH, S. *Critical reasons for crashes investigated in the National Motor Vehicle Crash Causation Survey*. (Traffic Safety Facts Crash•Stats. Report No. DOT HS 812 115). Washington, DC: National Highway Traffic Safety Administration, 2015.

SRIVASTAVA, Nitish et al. *Dropout:* A simple way to prevent neural networks from overfitting. Journal of Machine, Learn. Res., v.15, n.1, p.1929–1958, January 2014.

STALLKAMP, Johannes et al. *Man vs. computer:* Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, [S.l.], v.32, p323–332, ago. 2012. Disponível em: <https://doi.org/10.1016/j.neunet.2012.02.016>. Acesso em: 8 de jun. 2020.

SUN, Shiliang; CAO, Zehui; ZHU, Han; ZHAO, Jing. *A Survey of Optimization Methods from a Machine Learning Perspective*. [S.l]: Arxiv: 1906.06821, 2019. Disponível em: <https://arxiv.org/abs/1906.06821>. Acesso em: 8 jun. 2020.

SZEGEDY, Christian et al. *Going deeper with convolutions*. In: Computer Vision and Pattern Recognition, 2015, Boston.

U.S. DEPARTMENT OF TRANSPORTATION. *Preparing for the Future of Transportation:* Automated Vehicles 3.0. Washington – DC, 2018.

YAMASHITA, R.; NISHIO, M., DO, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. *Insights into imaging,* [S.l], v.9, n.4, p. 611-629, jun. 2018. Disponível em: < https://doi.org/10.1007/s13244-018-0639-9 >. Acesso em: 8 jun. 2020.

Este documento possui 6438 palavras.