DESENVOLVIMENTO E COMPARAÇÃO DE REDES NEURAIS CONVOLUCIONAIS PARA CLASSIFICAÇÃO DE OBJETOS

Klebiano Kennedy da Silva Lima[1], Matheus da Silva Menezes[2]

Resumo: A área do aprendizado de máquina possui diversas aplicações nos mais variados ramos da engenharia, medicina, estatística e computação. Uma das aplicações mais conhecidas é a sua utilização na classificação de imagens e escrita, onde a Rede Neural Convolucional (*Convolutional Neural Network - CNN*) vêm apresentando bons resultados para este tipo de problema. Neste trabalho foram apresentadas algumas análises teóricas dos componentes mais importantes da CNN e seus parâmetros principais, assim como os métodos mais utilizados para otimizar o treino das redes neurais. Além disso, será realizado o desenvolvimento, otimização e treino de algumas CNNs para a detecção de três diferentes tipos de objetos (carro, placa de pare de trânsito e vaso de planta) com precisão maior do que 95%. Para isso foram treinados múltiplos modelos otimizando seus parâmetros e camadas principais, comparando a rede neural treinada com alguns dos melhores modelos pré-treinados na área de classificação de imagens como *Mobilenet*, *VGGNet*-16, *GoogleNet*, *DenseNet* e *VGGNet*-19.

Palavras-chave: Redes neurais; aprendizado de máquina; classificação de objetos; transferência de aprendizagem; inteligência artificial

1. INTRODUÇÃO

As ferramentas de aprendizado de máquina obtiveram um grande valor em diversas áreas como a engenharia, computação e até mesmo na medicina. Esse impulso em usabilidade ocorreu em conjunto com o aumento da capacidade de processamento dos computadores e da quantidade de dados para análise. Uma área em que as técnicas de aprendizado de máquina obtiveram um elevado progresso foi na resolução de problemas de classificação, como no diagnóstico médico, controle de qualidade em processos industriais, reconhecimento de escrita e fala, entre outros. Na classificação, é importante que os dados de entrada sejam padronizados e rotulados, método conhecido como aprendizado supervisionado [1].

Uma das principais técnicas de aprendizado de máquina são as Redes Neurais Artificiais (RNAs), que apresentam a vantagem de aproximar qualquer função geral, proporciona ajuste automático aos dados, não possui a necessidade de modelos fundamentais na base e apresentam um comportamento não-linear, o que é ideal para a utilização em modelos no mundo real. Para melhor compreender o conteúdo do presente artigo, o leitor deve estar familiarizado com o funcionamento básico de redes neurais. Para consulta sugerimos [1][2][3].

Um tipo especifico de RNA que apresentou resultados superiores comparado a qualquer outra técnica de aprendizado de máquina foram as Redes Neurais Convolucionais (*ConvNets* ou CNNs, *Convolutional neural networks*), essas redes apresentam a vantagem de ter os neurônios, que são funções matemáticas que recebem um vetor e o transformam em um valor escalar, conectados a apenas um pequeno conjunto de neurônios da camada anterior. Esse método de escada faz com que esse sistema seja capaz de aprender características como bordas para a primeira camada e elementos inteiros do objeto na última camada [3].

O campo das *ConvNets* pode ser dado como início na década de 1960, quando os neurofisiologistas Hubel e Wiesel, realizando experimentos no córtex visual de gatos e macacos, descobriram que essa região apresenta neurônios que disparam com estímulos visuais como a mudança em arestas retilíneas [4]. Inspirados pelo trabalho de Hubel e Wiesel, foi desenvolvida, na década de 1980, a rede neural artificial conhecida como *Neucognitron* [5], que apresentou as primeiras habilidades de reconhecimento de padrões mesmo com mudanças de posicionamento e com distorções. O *Neucognitron* foi utilizado no reconhecimento de letras manuscritas.

Na década de 1990 foi desenvolvida a *LeNet* [6], que apresentou uma taxa de rejeição de apenas 9% no reconhecimento de letras manuscritas em um banco de dados de CEP (Código de Endereçamento Postal) dos correios dos Estados Unidos, sendo assim uma das primeiras redes neurais utilizadas em uma aplicação comercial. Essa rede obteve baixa necessidade de pré-processamento devido a realização da padronização de todas as imagens de entrada para um formato de 16 por 16 *pixels*.

A falta de capacidade computacional e de conjuntos de dados para treino acarretou no desenvolvimento lento na área das *ConvNets* até a década de 2010, quando o modelo de rede neural conhecido como *AlexNet* [7] apresentou os melhores resultados na competição *Imagenet LSVRC 2012* [8], onde diferentes tipos de ferramentas computacionais competem na classificação de imagens. A arquitetura *AlexNet* conseguiu superar os modelos

estado da arte da época com ampla margem, obtendo uma taxa de erro de 16,4% na categoria com mais de 1,2 milhões de imagens e 1000 classes de objetos diferentes. Após o sucesso da *AlexNet*, o campo das redes neurais convolucionais obtiveram um aumento expressivo de descobertas e inovações para o reconhecimento de imagens e outras diversas aplicações, diminuindo a cada ano a taxa de erro no *Imagenet* com redes cada vez maiores e mais complexas. Os erros na classificação de imagens a cada ano dos modelos participantes na *Imagenet* podem ser vistos na Figura 1.

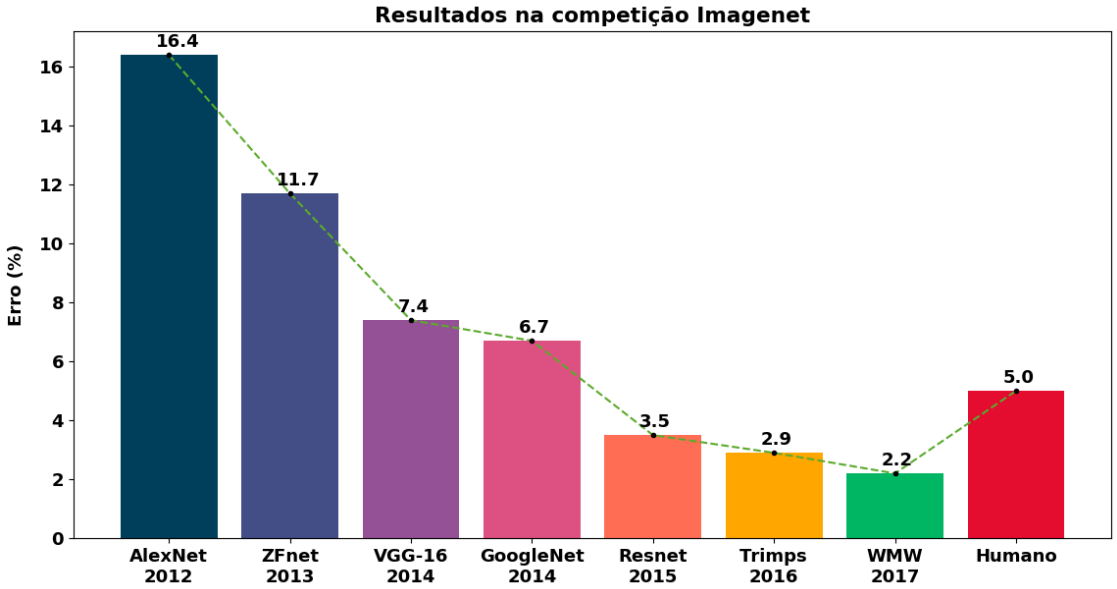

Figura 1. Erro dos melhores modelos participantes na *Imagenet* a cada ano. (Autoria própria)

Com isso, foram traçados alguns objetivos para este trabalho: descrever o funcionamento de uma rede neural convolucional, desenvolver modelos e treina-los em um conjunto de imagens com até três categorias de objetos diferentes, como carro, placa de pare de trânsito e vaso de planta. Após isso, os melhores modelos serão selecionados e comparados para a escolha do melhor modelo geral. O objetivo desse modelo é apresentar uma taxa de acerto maior do que a meta selecionada de 95% no conjunto de testes utilizado. O melhor modelo selecionado passará também por uma comparação com os modelos estado da arte atuais por meio de transferência de aprendizagem.

2. FUNDAMENTAÇÃO TEÓRICA

Para o melhor entendimento da estrutura de uma rede neural convolucional, neste trabalho será utilizado como base a arquitetura *AlexNet*, devido a poucas mudanças nas estruturas após a sua criação comparada as redes estado da arte atuais, as mudanças mais drásticas foram em relação ao aumento da capacidade computacional envolvendo matrizes por meio de GPU (*Graphics Processing Unit*, ou Unidade de Processamento Gráfico) e, consequentemente, o aumento do número de camadas neurais e do tamanho dos conjuntos de dados.

A arquitetura do *AlexNet* possui oito camadas totais, sendo dessas cinco camadas convolucionais (*Convolutional Layers*), que é o local onde acontece o maior número de operações, e três camadas *fully-connected* (FC, ou em tradução livre, totalmente conectadas), onde cada entrada está conectada a cada saída. A última camada FC é conhecida como camada de saída (*Output*), que apresenta uma distribuição entre as classes de objetos definidos no início da rede, no caso da *AlexNet*, a camada de saída possui uma distribuição de 1000 diferentes rótulos de objetos [7].

As imagens de entrada em uma *ConvNet* precisam ser padronizadas em relação as suas dimensões, pois as operações realizadas nas camadas convolucionais são dependentes das matrizes formadas pelos *pixels* de entrada. As dimensões das imagens de entrada da *AlexNet* são de 224 por 224 *pixels* e 3 canais (*Red-Green-Blue*, ou RGB), cada uma dessas imagens é passada para a primeira camada convolucional, depois para a segunda e assim consecutivamente, até a camada final, onde é apresentada uma distribuição para definir a qual das 1000 classes essa imagem pertence. Na Figura 2 é mostrada a arquitetura da *AlexNet* e todas as suas conexões, essa figura demonstra duas redes em paralelo pois no artigo original o modelo foi treinado utilizando duas GPUs em paralelo.

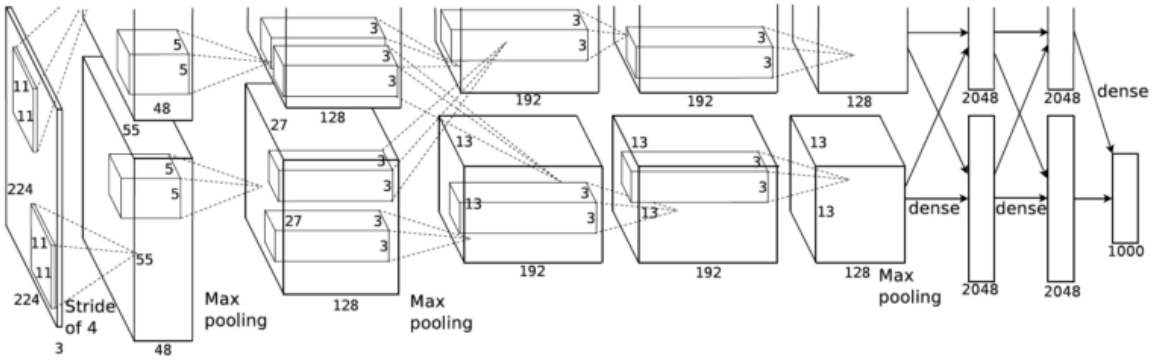

Figura 2. Ilustração da arquitetura utilizada na *AlexNet*. [7]

2.1. Camada Convolucional

A camada convolucional apresenta a parte da estrutura que exibe o maior número de operações em uma *ConvNet*. Sendo sempre a primeira camada após a entrada da rede, a camada convolucional constitui de um número de filtros (também conhecidos como neurônios ou *kernels*) que realiza uma operação de convolução nas matrizes de entrada. No caso do *AlexNet* a janela de filtro (também conhecido como campo receptivo, traduzido de *receptive field*) da primeira camada apresenta uma dimensão de 11 por 11 *pixels* com um passo (*stride*) de 4 *pixels* para percorrer a matriz de entrada.

O filtro apresenta uma matriz de números (conhecidos como pesos) que desliza pela imagem de entrada, multiplicando os valores com *pixels* originais e somando esses em um valor único. A junção desses valores quando o campo receptivo percorre toda a imagem formará a primeira matriz de saída (conhecida como mapa de ativação). A arquitetura *AlexNet* utiliza 96 filtros na primeira camada convolucional, que apresentam pesos para cada canal de cor (RGB) e esses são inicializados aleatoriamente. Com isso pode ser calculado a quantidade de parâmetros para a primeira camada convolucional da *AlexNet* com a seguinte equação:

$$P_{1C} = (Cr^2 \cdot Ca) \cdot f \quad (1)$$

no qual *Cr* é a dimensão do campo receptivo, *Ca* é o número de canais da imagem de entrada e *f* é a quantidade de filtros. Aplicando os valores da *AlexNet* na equação (1) obtém-se a quantidade de parâmetros da primeira camada convolucional em $(11 \cdot 11 \cdot 3) \cdot 96 = 34.848$ parâmetros. A quantidade de parâmetros em uma rede neural é de grande importância para a velocidade final de processamento computacional, quanto maior o número de parâmetros, maior será a memória necessária para computar essa rede. Para circundar esse problema, é comum se utilizar em *ConvNets* a separação das imagens de entrada em lotes, processando esses lotes separadamente e salvando apenas os parâmetros mais importantes como os pesos computados. A operação básica de um neurônio em uma rede neural pode ser descrita pela seguinte equação [9]:

$$\mathcal{F}(xi, Wi, b) = (Wi \cdot xi) + b \quad (2)$$

onde w_i representa o peso contido no filtro, x_i representa o respectivo *pixel* da imagem de entrada e o *b* é o viés (*bias*), que é um parâmetro atualizado assim como os pesos. Com todos esses valores obtidos, é então feita a soma desses parâmetros e passado o resultado obtido em uma função de ativação não-linear (discutido mais profundamente na seção 2.1.2) para representar o valor de saída. Na Figura 3, é possível obter uma melhor visualização de como é feita essa operação de convolução em uma *ConvNet*, onde o filtro (de dimensões 3 por 3) desliza na imagem de entrada realizando a operação da Equação (2) e seguindo, depois de passar pela função de ativação, para formar o pixel de saída (pixel convolvido).

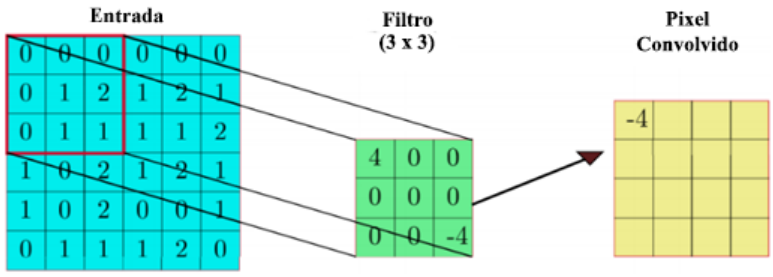

Figura 3. Operação de convolução em uma *ConvNet*. (Adaptado) [10]

Como pode ser visto na Figura 3, o processo de convolução pode acarretar em uma matriz de saída com dimensões menores do que a matriz de entrada, isso gera uma grande diminuição das dimensões espaciais da matriz, o que depois de diversas convoluções, pode gerar uma perda de informações das características da imagem de entrada, para circundar esse problema pode ser utilizado o método de *Zero Padding* (Em tradução livre, Preenchimento de zero), onde uma quantidade de zeros, são adicionadas nas bordas da matriz de entrada e com isso pode ser repassada as dimensões espaciais para a matriz de saída independente do passo utilizado no filtro.

2.1.1 Inicialização de parâmetros

Parâmetros numéricos como os pesos e os vieses em uma *ConvNet* são, geralmente, inicializados com valores aleatórios para que depois sejam modificados com o método de otimização do gradiente descendente. Os valores para o vetor do viés são, geralmente, inicializados em zero, já os pesos necessitam de métodos de inicialização de valores aleatórios mais complexos como a distribuição normal, distribuição uniforme, distribuição normal truncada, gerador de matriz ortogonal aleatória e diversos outros.

Apesar da dificuldade em desenvolver um método único de inicialização que funcione melhor do que outros em aplicações distintas, geralmente na área de redes neurais, um dos métodos mais utilizados é conhecido como inicialização *Xavier*, ou *Glorot Uniform*, nomeado em homenagem ao seu desenvolvedor *Xavier Glorot* [11]. Esse inicializador faz com que os pesos sejam iniciados em valores que se mantem em um alcance razoável em todas as camadas da rede neural. A inicialização *Xavier* pode ser definida na seguinte equação:

$$Var(W) = \frac{2}{(N_{ent} + N_{sai})} \tag{3}$$

no qual Var(W) é a variância da distribuição, que pode ser normal ou uniforme para o neurônio a ser analisado, N_{ent} é o número de neurônios na entrada e N_{sai} é o número de neurônios na saída. Esse método se mostrou eficaz sendo utilizado em conjunto com o otimizador de gradiente descendente estocástico e com as funções de ativação não-lineares [11].

2.1.2 Funções de ativação

Como demonstrado pela Equação (2), a principal operação em uma *ConvNet* é linear, isso faz com que a rede neural não apresente bons resultados para problemas não-lineares como a extração de características em imagens. Para resolver esse problema, após a camada convolucional em uma *ConvNet*, é aplicada uma camada de ativação, onde uma função não-linear como a função logística e tangente hiperbólica (tanh) são aplicadas para compor a saída. Em redes neurais modernas, a função preferível é conhecida como unidade linear retificada (ou *rectified linear unit*, *ReLU*) por apresentar tempo de treinamento baixo comparado com outras funções e diminuir a probabilidade de exibir o problema de desaparecimento do gradiente, onde o gradiente pode diminuir a um valor baixo que torne impossível a atualização dos pesos da rede neural, provocando dificuldades no treino e até mesmo parada total do treino da *ConvNet* [12]. As equações para a função logística (eq. 4), tangente hiperbólica (eq. 5) e ReLU (eq. 6), respectivamente, são definidas abaixo, assim como as curvas de cada função podem ser vistas na Figura 4:

$$f_1(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

$$f_2(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{5}$$

$$f_3(x) = max(0, x) \tag{6}$$

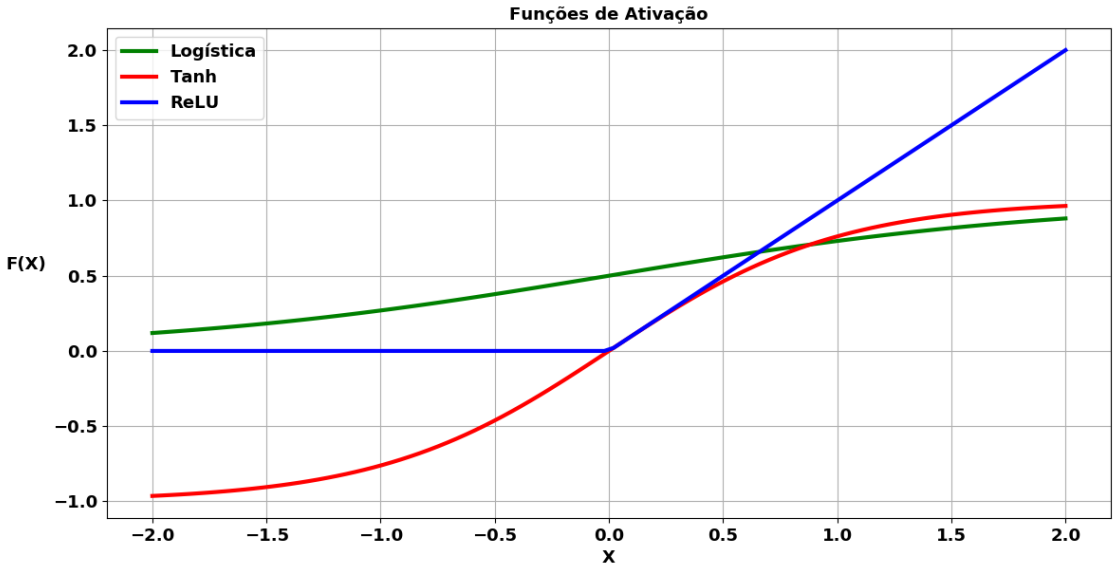

Figura 4. Curvas das funções de ativação logística, Tanh e ReLU. (Autoria Própria)

Após a aplicação da função de ativação, é geralmente aplicada uma camada de *Pooling* antes de partir para a próxima camada convolucional. O *Pooling* tem o objetivo de diminuir as dimensões espaciais do mapa de ativação e com isso diminuir a quantidade de parâmetros e capacidade computacional necessária. Essa camada se apresenta de forma similar à camada convolucional, onde um filtro percorre o mapa de ativação e realiza uma operação para formar os dados de saída.

Diversas operações podem ser empregadas no *Pooling*, sendo a operação conhecida como *Max Pooling* uma das mais utilizadas, pois apresenta uma grande capacidade de extração de características como bordas e linhas mesmo com a diminuição da resolução. No método de *Max Pooling*, é formada a matriz de saída pelos valores máximos do mapa de ativação depois da aplicação de um filtro de dimensões *n x m*, sendo o controle de resolução da saída definido pelo passo do filtro. Outro método bastante utilizado é o *Average Pooling*, onde ao invés de selecionar o valor máximo dentro do filtro como no *Max Pooling*, é realizada uma média entre esses valores. Na Figura 5, é possível visualizar a operação de *Max Pooling* com um filtro de dimensões 2 por 2.

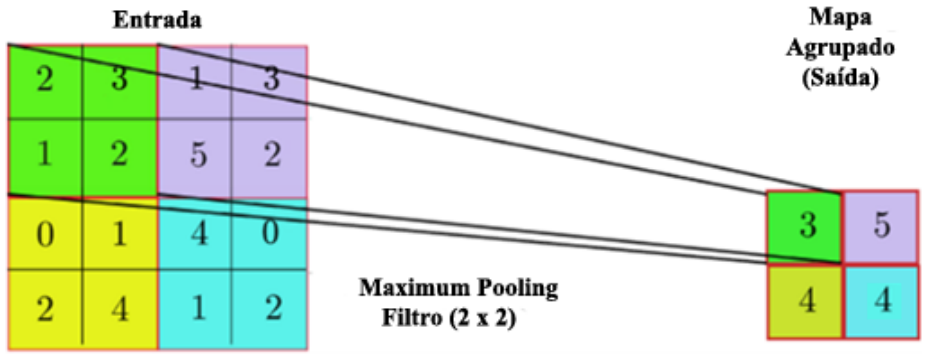

Figura 5. Operação de *Max Pooling* com um filtro de dimensões 2 por 2. (Adaptado) [10]

2.2. Treino e resultados da *ConvNet*

2.2.1 Retropropagação (*Backpropagation*)

A passagem das imagens de entrada, camada convolucional e camadas de *Pooling* seguem, de início, um caminho de avanço para frente (*Feed Forward*) até a camada *fully-connected* final, essa camada, também conhecida como camada de saída, é responsável por classificar a imagem de entrada em uma das classes inicialmente fornecidas, como gato, cachorro, carro etc. A comparação entre a imagem de entrada e o resultado classificado de saída é feito por meio de uma função de perda ou custo, onde a função conhecida como Entropia Cruzada (*cross-entropy loss*) é uma das mais utilizadas.

Na função de custo *cross-entropy*, os valores obtidos pela equação (2) são tratados como probabilidades não normalizadas e aplicadas na seguinte equação:

$$C_i = -log(\frac{e^{fyi}}{e^{fj}}) \tag{7}$$

No qual C_i é o custo no i-ésimo elemento da matrix de entrada, y_i é o valor que especifica a categoria correta a ser classificada definida no início da rede neural, e f_j é a função classificadora conhecida como *Softmax* [13], que pode ser definida pela equação a seguir:

$$\mathcal{F}(z) = \frac{e^z}{\sum_k e^{z_k}} \tag{8}$$

No qual *Z* é um vetor de valores reais obtidos pela Equação (2). A função *Softmax* compacta esses valores em um vetor de valores entre 0 e 1 como uma classe de probabilidades. Após a obtenção dos custos de cada parâmetro, é então calculado a média entre todos esses valores e obtido um número único final para a perda da *ConvNet*. Com esse número é possível saber se a rede neural apresenta progresso no aprendizado.

O ajuste do vetor peso ocorre com a aplicação de algoritmo para computar o gradiente, sendo o valor do custo utilizado para confirmar se a rede está aprendendo ou não. O vetor gradiente leva uma função de grandes dimensões espaciais e realiza a operação para a aproximação do ponto de mínimo local, onde a função de custo apresentará um valor mais baixo. Para a validação dos novos pesos e conferir a sua generalização para imagens, é então mensurado a precisão utilizando um conjunto único de imagens de teste, onde é comparado a habilidade da rede na classificação correta das imagens.

Como os neurônios apresentam uma função suave da entrada (*input*) com os pesos, é possível computar o gradiente utilizando a técnica conhecida como retropropagação [14], onde é realizada uma aplicação prática da regra da cadeia para derivadas. A retropropagação, como o próprio nome insinua, apresenta um sentido contrário ao fluxo normal da rede neural, essa se inicia na saída e segue até a entrada da rede, calculando o gradiente em cada conexão.

3. METODOLOGIA

De modo a realizar o desenvolvimento dos modelos e treina-los, primeiramente é necessário a obtenção das imagens de teste e seleção do número de categorias. Para este trabalho foi selecionado um conjunto de três categorias de objetos diferentes, com intuito de verificar a generalização do modelo treinado, os objetos escolhidos necessitam apresentar uma grande distinção entre si, que foram: carro, placa de pare de trânsito e vaso de planta.

3.1. Preparação dos dados e parâmetros de entrada

A obtenção das imagens foi feita por meio de *datasets* conhecidos e fotos próprias, o *Dataset Cars* da universidade de *Stanford* foi utilizada para a aquisição das imagens de carros, esse *dataset* possui 16.185 imagens de 196 diferentes tipos de carros e de marcas [15], as imagens dos vasos e placas foram obtidas por meios próprios. No total foram utilizadas aproximadamente 29 mil imagens, sendo essas separadas em 60% para o conjunto de treino e 40% para o conjunto de testes, nesses conjuntos as imagens são separadas em pastas dependendo de sua classe. Na figura 6, são demonstradas algumas imagens dos conjuntos de treino, testes e validações.

Figura 6. Exemplos de imagens contidas nos conjuntos de treino, testes e validações. (Autoria própria)

Como forma de facilitar a programação das redes e funções, foi utilizada a API (*Application Programming Interface*) Keras para a implementação do código, aumento de dados, treino e previsões das *ConvNets*. Esta API é baseada na linguagem de programação *Python* e fornece as funções e métodos mais utilizados em redes neurais em poucas linhas de código, onde não é necessário a implementação completa de partes como a retropropagação, cálculo do gradiente e atualização das matrizes de grandes dimensões [16]. A programação do código principal foi realizada utilizando a plataforma *Google Colaboratory*, um serviço na nuvem com ambiente *Jupyter Notebook*, que tem como objetivo a pesquisa e educação em aprendizado de máquina [17]. A plataforma *Colaboratory* oferece uma GPU e memória RAM em nuvem por um tempo limitado de uso e sem custos.

Dois dos principais problemas no treino de redes neurais são o sobreajuste (*overfitting*) e subajuste

(*underfitting*). No primeiro, o modelo apresenta baixa performance no conjunto de testes e alta no conjunto de treino, isso demonstra que a rede apresenta baixa generalização para as classes e imagens. Já no subajuste, o modelo apresenta baixa performance no conjunto de testes e treino, demonstrando que o modelo treinado não apresenta os parâmetros suficientes para alcançar a generalização.

Diversas técnicas foram desenvolvidas para amenizar os problemas de *overfitting* e *underfitting*, onde um dos métodos mais utilizados é a de aumento dos dados de entrada, de forma artificial por meio de transformações [7]. Neste trabalho, como o conjunto para a classe carros apresentou um número de imagens maior do que as outras classes (vaso e placas de pare), foram realizadas transformações nas imagens de entrada para gerar novas amostras nessas classes, o que proveu um conjunto de dados de igual tamanho para cada uma das categorias. Algumas das transformações utilizadas foram rotação no eixo central da imagem, aumento de largura e altura, aumento e diminuição de zoom e mudanças nos canais *RGB*.

Outra técnica utilizada para diminuir a ocorrência de *overfitting* é conhecida como *Dropout* [18], onde uma porção dos neurônios são configurados para zero ou "desligados", esse método diminui a dependência de um neurônio em outro, isso faz com que características diferentes tenham mais probabilidade de serem detectadas e repassadas pela rede.

A entrada da rede neural necessita que as imagens apresentem uma padronização de dimensões e cores. Para isso, as imagens nos conjuntos de treino, testes e validação foram redimensionadas para um tamanho de 100x100 *pixels*, como forma de normalizar cada imagem também foram realizados reescalonamento nas cores RGB, onde os valores digitais entre 0 e 255 foram transformados em valores flutuantes entre 0 e 1, esses redimensionamentos foram realizados antes da aplicação na *ConvNet* para diminuir a quantidade de computação necessária para o treino.

Com intuito de diminuir a quantidade de memória utilizada para o treino da *ConvNet*, o método de lote (*batch*) foi implementado. Neste método as imagens são divididas e treinadas em pequenos lotes, onde são salvos apenas os parâmetros principais como os pesos e vieses e, então, é calculada a função de perda e precisão ao final, concluindo uma época (*epoch*). O lote de 100 imagens foi selecionado para o treino neste trabalho pois apresentou resultados satisfatórios com a memória oferecida pela plataforma *Colaboratory*.

3.2. Definição do modelo

A criação do modelo de rede neural convolucional segue uma forma linear de amontoamento de camadas. Para construir esse modelo na API Keras é utilizada a classe *Sequential*, onde as camadas da rede podem ser adicionadas facilmente utilizando o método *add*. Como o modelo a ser treinado tem aplicações em imagens, a convolução utilizada é a de duas dimensões (*Conv2D* no Keras). Neste método são repassadas a quantidade de filtros desejadas, dimensões do filtro, função de ativação (seção 2.1.2), passo dos filtros, inicialização de pesos e vieses (seção 2.2.1) e outros argumentos mais específicos. Para a primeira Conv2D, também é necessário repassar as dimensões da imagem de entrada.

No Keras, a aplicação da camada de *Pooling é* feita utilizando a função *MaxPooling2D*, que recebe argumentos de tamanho do filtro para a diminuição de escala (seção 2.1.2), passo do filtro e outros parâmetros mais específicos como o preenchimento por zero. A aplicação do *Dropout* no Keras também é feita por meio de função, que aplica o *dropout* na entrada e recebe um argumento do tipo ponto flutuante entre 0 e 1. Esse argumento representa a fração dos neurônios de entrada que serão desligados, a seleção desses neurônios é feita de forma aleatória.

Como a camada de saída (*fully-connected*) está conectada a todos os neurônios da camada anterior, é necessário transforma-la em um vetor com apenas uma dimensão. Para a realização dessa transformada é aplicada a função *Flatten*, na qual o vetor de entrada é redimensionado e então repassado para a próxima camada, que geralmente é uma camada *fully-connected*. A camada de saída no Keras é obtida com a função *Dense*, que recebe argumentos de um número inteiro positivo que vai formar a dimensão do espaço de saída onde é implementada a equação 2. O *Dense* também recebe a função de ativação de saída, geralmente é utilizada a função *Softmax* (seção 2.2.1) para redes com mais de duas classes.

Após a criação da rede neural convolucional no Keras, é então aplicado o método *Compile*. Este método configura o modelo para o treino e recebe os argumentos dos tipos de funções de perda que serão empregados. O método otimizador utilizado para computar o gradiente recebe o parâmetro da taxa de aprendizado, que define o passo com que a função de perda será atualizada. Os otimizadores utilizados para esse trabalho foram o RMSprop [19] e Adam [20] devido à alta performance e baixa computação necessária. O método *compile* também recebe o argumento da métrica utilizada para a avaliação do modelo durante o treinamento e teste.

Após a compilação do modelo, foi então realizado o treino utilizando a função *fit_generator* da API Keras, que recebe como argumentos as imagens e categorias das pastas de treino e validação, número máximo de épocas e os passos necessários para completar uma época. Nesse trabalho foi utilizado o número de imagem de treino dividido pelo tamanho do lote escolhido. Como esses valores não modificaram durante os treinos, foi utilizado o valor de 167 passos por época. A função *fit_generator* também recebe os parâmetros de *callback* internos ao Keras. Um desses parâmetros foi o *ModelCheckpoint*, este *callback* foi utilizado como forma de *backup*, onde os modelos foram salvos a cada época caso a precisão de validação apresentasse um incremento, desse modo, caso houvesse algum erro durante a execução do algoritmo, o modelo não seria perdido. Todos os modelos treinados foram

salvos em arquivos do tipo *Hierarchical Data Format* (HDF5) pois apresenta tamanho de armazenamento baixo e fácil leitura com o Keras. O HDF5 foi projetado para armazenar e organizar extensas quantidades de dados, principalmente matrizes multidimensionais do tipo homogêneo.

Com todas as redes treinadas, foi realizada a comparação entre cada uma delas, conferindo os dados de perdas e precisão e, então, selecionando a rede que apresenta os maiores valores e melhor encaixe de curvas, demonstrando que o modelo treinado não apresentou *overfitting* ou *underfitting*. Após a seleção do melhor modelo, foi feita uma comparação dessa rede com alguns modelos pré-treinados do estado da arte contidos na API Keras. Esses modelos passarão por um método de transferência de aprendizagem utilizando parâmetros semelhantes e mesmo conjuntos de dados ao da rede selecionada.

4. RESULTADOS E DISCUSSÕES

No total foram treinadas 63 redes neurais diferentes para a realização desse trabalho, onde em cada uma foi efetivada a mudança de parâmetros como número de filtros, número de camadas convolucionais e diferentes valores para a taxa de aprendizado, fração dos neurônios de entrada que serão desligadas nas camadas dropout e para a dimensão do espaço de saída das camadas *fully-connected*. Como as redes apresentaram semelhanças em relação ao número de camadas, o tempo de treino de cada rede apresentou um valor médio de 150 minutos. A seleção das melhores redes foi realizada levando em consideração as perdas e precisões de validação e treino e também uma análise de curvas para conferir se ocorreu *overfitting* ou *underfitting*. Os resultados de precisão e perdas finais obtidos para as três melhores redes treinadas podem ser vistos na Tabela 1 e as curvas podem ser visualizadas na Figura 7.

Tabela 1. Valores finais das perdas e precisão dos melhores modelos treinados. (Autoria própria)

Modelo	Perdas		Precisão	
	Treino (%)	Teste (%)	Treino (%)	Teste (%)
a	9,45	10,25	96,68	96,63
b	8,47	8,97	97,19	97,02
c	5,59	8,29	98,16	97,41

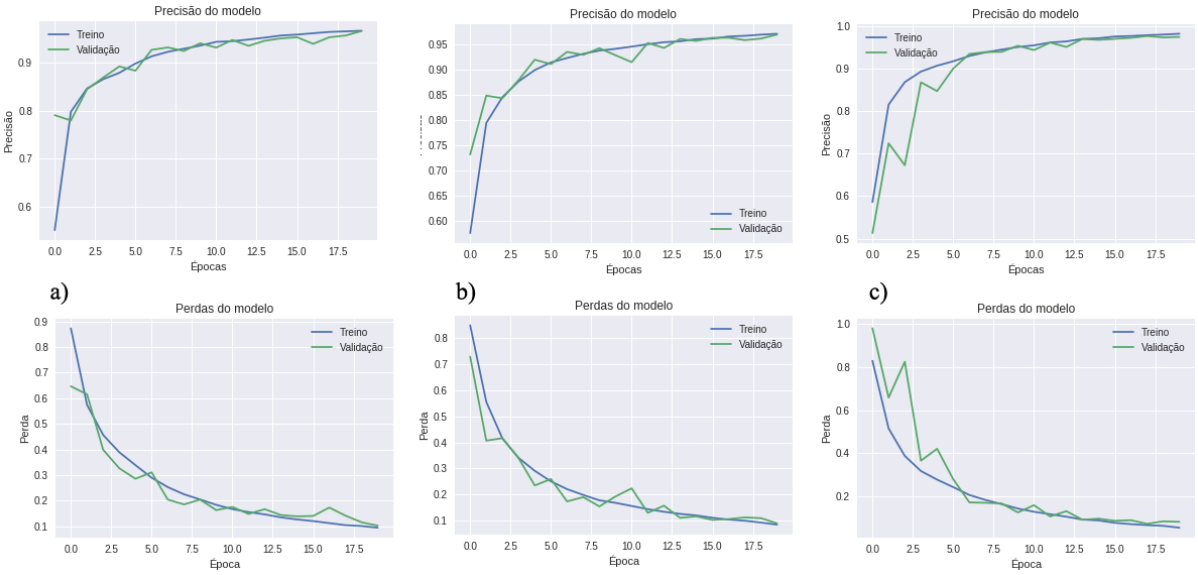

Figura 7. Resultados das três melhores *ConvNets* treinadas (modelos a,b e c). (Autoria própria)

Na Figura 7, é possível conferir as curvas de perdas e precisão de cada modelo. Apesar do modelo c apresentar um valor maior para a precisão, as suas curvas de precisão e perda apresentam início irregular em comparação aos outros modelos, o que pode indicar um problema de sobreajuste. Por este motivo a curva selecionada para a comparação com os modelos estado da arte no *Imagenet* foi a do modelo b, que apresenta um encaixe mais suave nos seus devidos gráficos de precisão e perdas.

Com todos os dados necessários obtidos, foi alcançado o objetivo na criação de uma rede neural convolucional com precisão em testes maior que 95%. A estrutura da *ConvNet* do modelo b pode ser visualizada na Tabela 2. O tamanho dos filtros utilizados nas camadas convolucionais e de *Max Pooling* foram de 2 por 2 *pixels* e o passo do filtro de 1 por 1 *pixel*, a função ativação utilizada em todas as camadas convolucionais e *fully connected* foi a

ReLu. O método de *Zero Padding* foi utilizado em todas as camadas convolucionais. O total de parâmetros treináveis da rede foi de 4.913.219.

Tabela 2. Estrutura da rede neural convolucional do modelo b. (Autoria própria)

Camada	Número de filtros	Dimensões da Saída
Entrada	Nenhum	100x100
Convolucional 1	32	100x100
Convolucional 2	32	100x100
Max Pooling 1	32	50x50
Convolucional 3	64	50x50
Convolucional 4	64	50x50
Max Pooling 2	64	25x25
Dropout 1	Neurônios desligados = 30%	
Convolucional 5	128	25x25
Convolucional 6	128	25x25
Max Pooling 3	128	12x12
Fully Connected 1	Comprimento do vetor = 256	
Dropout 2	Neurônios desligados = 50%	
Fully Connected 2	Comprimento do vetor = 256	
Dropout 3	Neurônios desligados = 50%	
Fully Connected 3 (Saída)	Comprimento do vetor = 3	

A rede selecionada apresentou uma alta precisão de acertos nos conjuntos de imagens de treino e testes, e demonstrou um encaixe de curva satisfatório, porém ainda assim a rede pode ter apresentado alguma forma de sobreajuste ou subajuste. Para comprovar se esses problemas ocorreram pode ser testado a rede neural utilizando a função *predict* da API Keras, que facilita a aplicação da rede em um conjunto de imagens desejadas e tem como saída os resultados da classificação em uma distribuição de probabilidades.

A rede neural do modelo b e suas curvas foram analisadas como forma de comparação com as redes pré-treinadas estado da arte presentes no módulo *Applications* da API Keras. As redes com pesos e vieses pré-treinadas utilizadas para a comparação foram a VGG-16, VGG-19, *mobilenet*, *GoogleNet* e *DenseNet*. Essas redes foram treinadas utilizando o método de transferência de aprendizagem por ajuste fino, onde os pesos e vieses continuam os mesmos do modelo original, sendo atualizado no treino apenas as últimas camadas *fully-connected*. Isso faz com que o tempo de treino de redes mais complexas como a *VGGNet*, *GoogleNet* e *DenseNet* seja baixo, apresentando uma média de 153 minutos para cada modelo.

Para realizar a comparação com o modelo b foi mantida a mesma quantidade de camadas finais e os mesmos parâmetros dos comprimentos dos vetores e de neurônios desligados, as camadas finais selecionadas para as redes pré-treinadas foram as *fully-connected* 1, *dropout* 2, *fully-connected* 2, *dropout* 3 e *fully-connected* 3 da tabela 1. A Figura 9 demonstra as curvas de precisão dos modelos pré-treinados da API Keras nos conjuntos de imagens de testes e de treino.

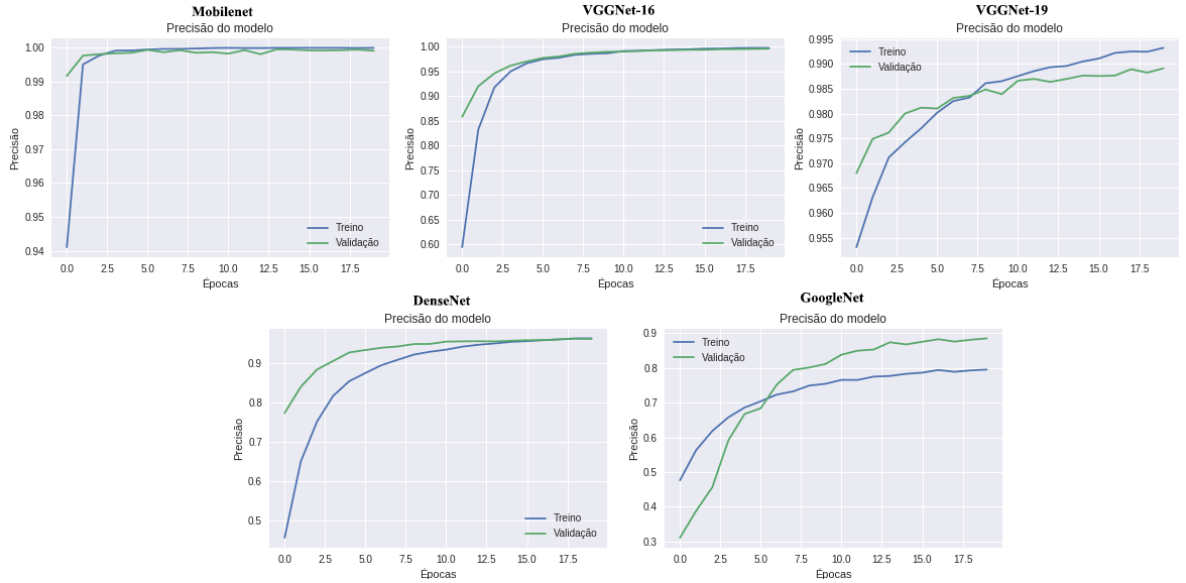

Figura 9. Precisão dos modelos pré-treinados nos conjuntos de testes e treino. (Autoria própria)

Na Figura 9 é possível notar que três redes apresentaram sobreajuste, sendo elas a *Mobilenet*, *VGGNet*-16 e *VGGNet*-19, que pode ter ocorrido devido aos parâmetros utilizados nas camadas finais de treino, o que pode ser evitado modificando os parâmetros das camadas *fully-connected* e adicionando maior número de *dropout*. Durante o treino também foi notado que a taxa de aprendizado oferece grande variação para o resultado final do modelo. Para o treino das cinco redes pré-treinadas foi utilizada uma taxa de aprendizado de $1 \cdot 10^{-5}$. O modelo GoogleNet apresentou um problema de subajuste, que pode ser solucionado acrescentando o número de camadas, filtros e neurônios na rede ou aumentando o número de imagens de treino e testes.

O modelo pré-treinado que apresentou o melhor resultado foi a *DenseNet*, onde as curvas de validação e treino exibiram um crescimento suave, não atingindo o valor máximo de 100%, como o exemplo do sobreajuste da *MobileNet* e *VGGNet*-16. A arquitetura *DenseNet* apresenta alta complexidade e pode ser desenvolvida para escalar uma grande quantidade de camadas com baixa probabilidade de exibir os problemas de uma *ConvNet* comum como o sobreajuste e degradação [21]. O modelo *DenseNet* alcançou uma precisão de treino e testes, respectivamente, de 96,36% e 96,24%, sendo as perdas obtidas para o treino e testes, respectivamente, de 10,43% e 14,62%. Uma comparação entre os principais parâmetros das redes neurais treinadas pode ser vista na tabela 3.

Tabela 3. Comparação entre os principais parâmetros das redes treinadas. (Autoria própria)

ConvNet	Perdas	Precisão	Quantidade de Parâmetros		Tempo de treino
	Treino - Teste (%)	Treino - Teste (%)	Treináveis	Não-treináveis	(minutos)
Modelo b	8,47 - 8,97	97,19 - 97,02	4.913.219	0	151
DenseNet	10,43 – 14,62	96,36 – 96,24	4.720.643	7.037.504	146

Na Tabela 3 é possível visualizar que o modelo b apresenta melhores parâmetros finais para as perdas e precisão, mas oferece um maior número de parâmetros treináveis como os pesos e vieses, e consequentemente, maior tempo de treino comparado ao modelo *DenseNet*. O modelo b não apresenta parâmetro não-treinável pois toda a rede foi computada da entrada das imagens até a camada final de saída, já o modelo *DenseNet*, por ser treinado utilizando a técnica de transferência de aprendizagem, possui uma grande quantidade de objetos não-treináveis, que são todos os parâmetros que foram "congelados" da rede pré-treinada no conjunto de dados *Imagenet*. A *DenseNet* apresenta maior profundidade de camadas e menor tempo de treino comparado ao modelo b, indicando a vantagem da utilização do método de transferência de aprendizagem no treino de *ConvNets* em conjuntos de dados próprios.

Apesar dos parâmetros da tabela 3 indicarem uma vantagem do modelo b nas perdas e precisão, as curvas geradas também precisam ser analisadas para a verificação dos problemas de *overfitting* e *underfitting*. Comparando as Figuras 9 e 8 é possível verificar que o encaixe de curvas da *DenseNet* é mais suave do que o modelo b, sempre apresentando o parâmetro de validação sendo acompanhado pelo treino. O modelo b demonstra uma curva de validação que ultrapassa e retrocede o treino em diversos momentos, o que pode indicar *overfitting* ao final da rede.

Para realizar a validação de uma rede neural convolucional, geralmente é utilizado um conjunto de imagens não contidas nos conjuntos de testes e treino e conferido as probabilidades obtidas pela rede na sua saída. A figura

10 demonstra as probabilidades acertadas obtidas pelas redes neurais para imagens não contidas nos conjuntos de treino e de validação.

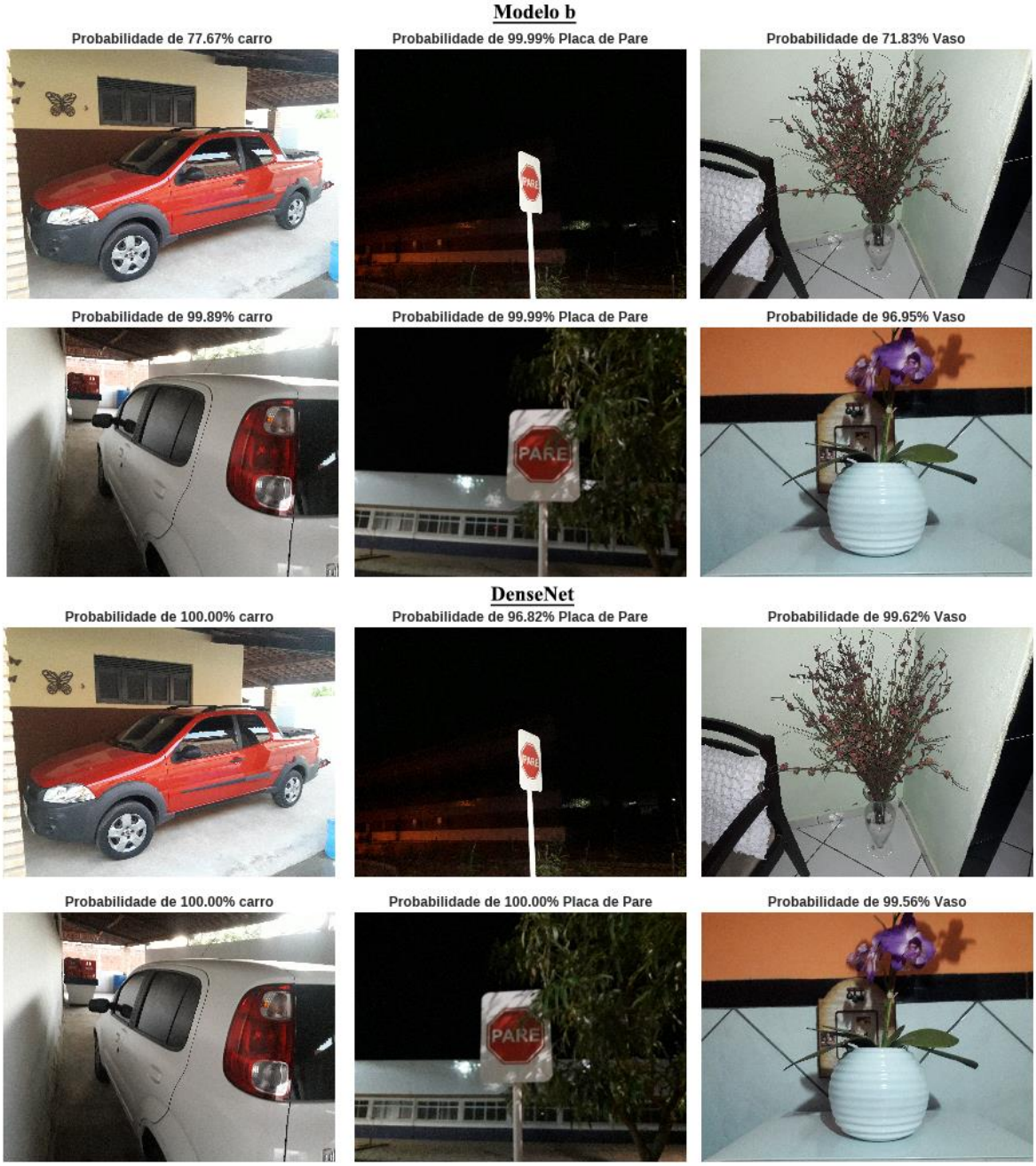

Figura 10. Resultados de saída do modelo b e *DenseNet* em imagens. (Autoria própria)

Os resultados apresentados na Figura 10 demonstram que as duas redes apresentaram alta taxa de acerto, mas o modelo *DenseNet* exibiu maior probabilidade para as distintas categorias de objetos corretos, indicando melhor aprendizado de características em geral.

5. CONCLUSÕES

A área de aprendizado de máquina se mostrou bastante eficaz nas últimas décadas em diversas aplicações do cotidiano, sendo as redes neurais uma das suas ferramentas mais importantes. Neste trabalho foram analisados os parâmetros principais das *ConvNets* e como é realizado seu treino e otimização. No total foram treinados mais de 60 tipos diferentes de *ConvNets* para a detecção nas categorias carro, vaso de planta e placa de pare, foram, então, comparados os seus resultados para extrair o melhor modelo possível com a computação disponível. Nesses modelos foram variados parâmetros como número de camadas convolucionais, aumento ou diminuição de camadas *dropout* e *fully-connected*, taxa de aprendizado e diversos outros. Com isso foi alcançado o objetivo da aquisição

de uma *ConvNet* com precisão maior do que 95%, onde o melhor modelo encontrado (modelo b) atingiu a precisão de testes de 97,02%.

Após o treino completo dos modelos, foi realizada uma comparação com algumas das redes pré-treinadas estado da arte na competição *Imagenet*, onde o modelo do tipo *DenseNet* apresentou os melhores resultados entre as cinco outras redes treinadas, alcançando uma precisão de testes de 96,24%. Com os objetivos concluídos e conhecimentos sobre redes neurais obtidos, neste trabalho foi demonstrado um vasto número de possibilidades para a utilização de *ConvNets* em aplicações diversas, como na detecção de materiais na indústria, aplicações em veículos autônomos, detecção de cânceres em medicina e outros, que podem propiciar diversos estudos nesses respectivos temas.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] KOTSIANTIS, Sotiris B.; ZAHARAKIS, Ioannis D.; PINTELAS, Panayiotis E. Machine learning: a review of classification and combining techniques. Artificial Intelligence Review, v. 26, n. 3, p. 159-190, 2006.

[2] ZHANG, Guoqiang Peter. Neural networks for classification: a survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), v. 30, n. 4, p. 451-462, 2000.

[3] ALOYSIUS, Neena; GEETHA, M. A review on deep convolutional neural networks. In: Communication and Signal Processing (ICCSP), 2017 International Conference on. IEEE, 2017. p. 0588-0592.

[4] HUBEL, David H.; WIESEL, Torsten N. Receptive fields and functional architecture of monkey striate cortex. The Journal of physiology, v. 195, n. 1, p. 215-243, 1968.

[5] FUKUSHIMA, Kunihiko; MIYAKE, Sei. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In: Competition and cooperation in neural nets. Springer, Berlin, Heidelberg, 1982. p. 267-285.

[6] LECUN, Yann et al. Handwritten digit recognition with a back-propagation network. In: Advances in neural information processing systems. 1990. p. 396-404.

[7] KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. 2012. p. 1097-1105.

[8] BERG, Alex; DENG, Jia; FEI-FEI, L. Large scale visual recognition challenge 2010. 2010.

[9] BOUVRIE, Jake. Notes on convolutional neural networks. 2006.

[10] SEKAR, Vinothkumar et al. Inverse Design of Airfoil Using a Deep Convolutional Neural Network. AIAA Journal, p. 1-11, 2019.

[11] GLOROT, Xavier; BENGIO, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. 2010. p. 249-256.

[12] GLOROT, Xavier; BORDES, Antoine; BENGIO, Yoshua. Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics. 2011. p. 315-323.

[13] DENKER, John S.; LECUN, Yann. Transforming neural-net output levels to probability distributions. In: Advances in neural information processing systems. 1991. p. 853-859.

[14] RUMELHART, David E.; HINTON, Geoffrey E.; WILLIAMS, Ronald J. Learning representations by back-propagating errors. nature, v. 323, n. 6088, p. 533, 1986.

[15] KRAUSE, Jonathan et al. 3d object representations for fine-grained categorization. In: Proceedings of the IEEE International Conference on Computer Vision Workshops. 2013. p. 554-561.

[16] CHOLLET, François et al. Keras: The python deep learning library. Astrophysics Source Code Library, 2018.

[17] Google Colaboratory. Disponível em: < https://colab.research.google.com//> Acesso em: 05 de fevereiro de 2019

[18] SRIVASTAVA, Nitish et al. Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, v. 15, n. 1, p. 1929-1958, 2014.

[19] TIELEMAN, Tijmen; HINTON, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, v. 4, n. 2, p. 26-31, 2012.

[20] KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[21] HUANG, Gao et al. Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017. p. 4700-4708.