ANÁLISE DE UM ALGORITMO DESENVOLVIDO PARA SOLUÇÃO DE DEEP LEARNING UTILIZANDO REDES NEURAIS CONVOLUCIONAIS PARA DIFIERÊNCIAR GATOS DE CACHORROS

ANALYSIS OF A DEVELOPED ALGORITHM FOR DEEP LEARNING SOLUTION USING CONVOLUTIONAL NEURAL NETWORKS TO DIFFERENTIATE CATS

Jonas de Souza TEGA (Aluno)

jonas_tega@hotmail.com

Ciências da Computação, Centro Universitário Padre Anchieta

Carlos Eduardo CÂMARA (orientador)

ccamara@anchieta.br

dinhocamara@gmail.com

Ciências da Computação, Centro Universitário Padre Anchieta

Resumo

Com o passar dos anos, a inteligência artificial passou por diversas evoluções, se segmentado em diversas áreas. Um desse ramos é dedicado a aprendizagem de máquina, que por sua vez vem se destacando com o aprendizado profundo (*Deep Learning*), principalmente na parte de reconhecimento de imagens. Umas das estratégias utilizadas para atingir esse destaque, é utilizar as redes neurais convolucionais para que a máquina concretize seu aprendizado. Sendo assim, este artigo trará os conceitos básicos sobre redes neurais convolucionais, e como a sua fusão com aprendizado profundo se encaixa tão bem para reconhecimento de imagens.

Palavras-Chave

Inteligência Artificial; Aprendizado Profundo; Reconhecimento; Redes Neurais Convolucionais.

Abstract

Over the years, artificial intelligence has undergone several evolutions, if segmented into several areas. One of these branches is dedicated to machine learning, which in turn has been highlighted by Deep Learning, mainly in the image recognition part. One of the strategies used to achieve this highlight is to use convolutional neural networks to make the machine realize its learning. Therefore, this article will introduce the basics of convolutional neural networks, and how their fusion with deep learning fits so well for image recognition.

Keywords

Artificial Intelligence; Deep Learning; Recognition; Convolutional Neural Networks.

INTRODUÇÃO

Desde de muito tempo atrás, o ser humano tenta compreender como funciona a inteligência humana e com o passar dos tempos esse conhecimento foi se formalizando, dando início a uma nova investigação, que visa avançar os conhecimentos adquiridos sobre inteligência, para formar algo que pensa desde a sua essência, criando assim o campo da Inteligência Artificial.

Com o passar dos anos, e o avanço da tecnologia, o aprendizado de máquina passou a ser cada vez mais comum, principalmente após a segunda guerra mundial, e hoje em dia é um dos assuntos mais pesquisados e desenvolvidos no ramo da tecnologia da informação. Assim, inúmeras pesquisas foram desenvolvidas para máquinas efetuar determinadas ações, ao ponto em que as máquinas possuem a capacidade de desenvolver determinado conhecimento sobre uma ação, podendo desenvolver suas próprias técnicas para melhor efetuar a ação.

Atualmente o estudo de inteligência artificial se divide em diversas áreas, focando em diferentes formas de aprendizado de máquina. Dentre eles um que vem se destacando é o *Deep Learning*, que tem como objetivo uma aprendizagem mais profunda que, através de um conjunto de algoritmos de Redes Neurais Artificiais, modelam melhor uma quantidade bem superior de dados, visando o aprendizado de máquinas através da observação dessa massa de dados. Isso permite, por exemplo, aprofundar seus níveis de comparação, a ponto de conseguir diferenciar certas imagens, podendo assim trazer um resultado de busca muito mais preciso e rápido.

Esse tipo de técnica é utilizado por plataformas de busca, como o Google utiliza para realizar busca de imagens, até para plataformas de segurança que utiliza reconhecimento facial de pessoas em meio à multidão, ou para um sistema de segurança.

Através dessa crescente pesquisa em torno desse tema, esse projeto analisa os componentes básicos para que uma Rede Neural Densa (*Deep Learning*) consiga operar, e avaliar os resultados que ela permite no reconhecimento automático de imagens.

A maneira utilizada para fazer essa análise, foi emprestar um código feito por Kinsley, disponível em ***How to use your trained model:*** *Deep Learning basics with Python, TensorFlow and Keras p.6* [16], que diferencia se uma determinada imagem é um cachorro, ou um gato, explicando detalhadamente, e testando mais de um modelo de Redes Neurais Densa, a fim de perceber como elas operam, e como suas estruturas funcionam. Todas as estruturas utilizadas no código, serão descritas, para maior entendimento do mesmo.

CONCEITOS BÁSICOS

INTELIGÊNCIA ARTIFICIAL

"A inteligência artificial (IA) é simplesmente uma maneira de fazer o computador pensar inteligentemente. Isso é conseguido estudando como as pessoas pensam quando estão tentando tomar decisões e resolver problemas, dividindo esses processos de pensamento em etapas básicas e desenhando um programa de computador que solucione problemas

usando essas mesmas etapas. A IA então fornece um método simples e estruturado de se projetar programas complexos de tomada de decisão". (LEVINE, DRANG, EDELSON 1988)

Seu início se deu logo no fim da segunda guerra, em 1950 quando Alan Turing fez uma proposta chamada de teste de Turing, na qual se propõe avaliar se uma máquina que possui uma inteligência satisfatória ou não. Para fazer a qualificação, o teste envolve uma conversa entre duas pessoas em uma máquina, porém cada um está isolado do outro. Uma das pessoas faz perguntas, como em um diálogo normal, e dessas perguntas veem respostas simultâneas, tanto da máquina, quanto da outra pessoa. Caso o questionador não conseguisse diferenciar com certeza qual a resposta estava vindo da máquina, e qual vinha da outra pessoa, a mesma era classificada com uma inteligência satisfatória, já que seu sistema de pensamentos para responder as perguntas não conseguia ser diferenciado do humano. [1,9]

Para a máquina conseguir êxito no teste, ela precisaria possuir processamento de linguagem natural, representação de conhecimento, raciocínio automatizado, aprendizado de máquina. Possuindo essas quatro características, a máquina seria capaz de se comunicar, armazenar informações, comparar e manipular uma conclusão e por fim aprender tudo aquilo que está sendo absorvido nas outras três características. Essa deveria a única forma de uma máquina passar no teste. [1]

Um outro teste proposto, o Teste de Turing Total, se baseava na mesma ideia, porém dessa vez o questionador iria interagir diretamente com a máquina. O que diferenciaria as circunstâncias do outro teste, assim para conseguir executar tal alteração, a máquina precisaria ter uma tela, com possibilidade de o computador receber objetos oferecidos pelo interrogador, essa tese é chamado de teste de Turing total. [1]

A partir deste teste, surgem mais dois conceitos essenciais para a IA sendo eles a visão computacional, para receber objetos, e a robótica, para manipular e movimentar-se, assim passaria a estudar a forma como um computador "observa", "percebe" e manipula objetos, junto com a robótica, trazendo o estudo de que uma máquina de forma autônoma possa manipular objetos precisamente e, na maioria dos casos, fazendo-o de uma forma que supera a capacidade humana. [1]

REDES NEURAIS ARTIFICIAIS

Começou a ser fortemente pesquisada nos anos 90, apesar de já se ter havido algumas publicações na década de 50. As redes neurais possuem o intuito de resolver diversos tipos de problemas, porém obtendo uma alta capacidade de mapear sistemas não lineares, e através dele aprender os comportamentos envolvidos através dos dados obtidos. [4]

"Redes neurais artificiais são modelos computacionais inspirados no sistema nervoso de seres vivos. Possuem a capacidade de aquisição e manutenção do conhecimento (baseado em informações e podem ser definidas como um conjunto de unidades de processamento, caracterizadas por neurônios artificiais, que são interligados por um grande número de interconexões (sinapses artificiais), sendo representadas aqui por vetores/matrizes de pesos sinápticos". (SILVA, 2016)

A primeira publicação foi em 1943, escrita por *McCulloch & Pitts*, analisando um neurônio biológico e reproduzindo um modelo matemático inspirado nesse neurônio biológico. Assim surgiu a primeira ideia de um neurônio artificial. *Em 1949, Hebb propôs o primeiro método de treinamento para uma rede, baseando-se em hipóteses e observações de caráter neurofisiológico*. Outros grandes marcos dessa época se deram no fim da década de 50 e início da década de 60, com o desenvolvimento do primeiro neuro-computador denominado *Mark I – Perceptron*. Outro modelo de redes neurais é a

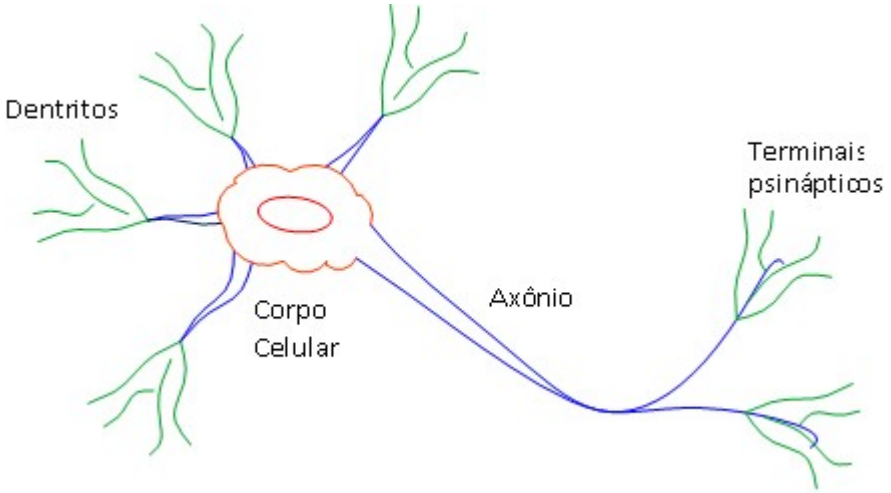
rede *Adaline* (*ADAptative Linear Element*) e, posteriormente, *Madaline*, podendo pôr em prática as ideias de redes neurais artificiais e deixando essas ideias cada vez mais palpáveis. Porém a publicação de *Perceptrons - an introduction to computational geometry* em 1969 por *Minsky & Papert*, "*acabou colocando os estudos da área de neurocomputação em um grande inverno*". *Nesta publicação foi exposta uma limitação das redes neurais artificiais, por serem constituídas de apenas uma única camada, e por não conseguirem aprender um relacionamento entre as entradas e saídas de funções lógicas bem simples como XOR (ou exclusivo), por exemplo. Com essa demonstração mostrava-se a impossibilidade dessas redes realizarem a correta classificação de padrões para classe não linearmente separáveis.* Passaram os anos, computadores evoluíram, algoritmos foram otimizados, e os estudos sobre o sistema nervoso biológico tiveram novas descobertas. *Com esses fatores Rumelhart, Hinton e Williams, publicam Parallel Distributed Processing, no qual eles desenvolveram um algoritmo intitulado backpropagation, que permitia ajustar os pesos em uma rede com mais de uma camada, solucionando os problemas referentes ao XOR, e quebrava assim a barreira imposta em 1969. A partir de então, muitos estudos passaram a se desenvolver na área novamente.* [4]

As principais características de uma rede neural artificial, adaptação por experiência, capacidade de aprendizado, habilidade de generalização, organização de dados, tolerância a falhas, armazenamento distribuído e facilidade de prototipagem. Atualmente pode-se aplicar os conceitos de redes neurais em diversas áreas, porém a que mais vem se destacando acaba sendo a área de reconhecimento de padrões (utilizada para análise de determinadas ações, ou utilizada na área de segurança para reconhecimento facial, ou da saúde para reconhecimento de diagnósticos de cânceres e várias outras doenças automaticamente), e de agrupamento de dados (muito utilizado para agrupar diversos dados espalhados pela internet, e filtrá-los, mostrando somente dados válidos para a análise). [4, 5, 11]

NEURÔNIO BIOLÓGICO

É uma célula nervosa, Figura 1, pode ser definida como o processador biológico, altamente estimulável, age em paralelo através de impulsos eletroquímicos, com a funcionalidade de processar, transmitir, pensar, e memorizar os dados e informações adquiridas através dos nossos receptores (olhos, mãos, boca, etc.). [4,6]

Figura 1. Fisiologia de um neurônio biológico. [6]

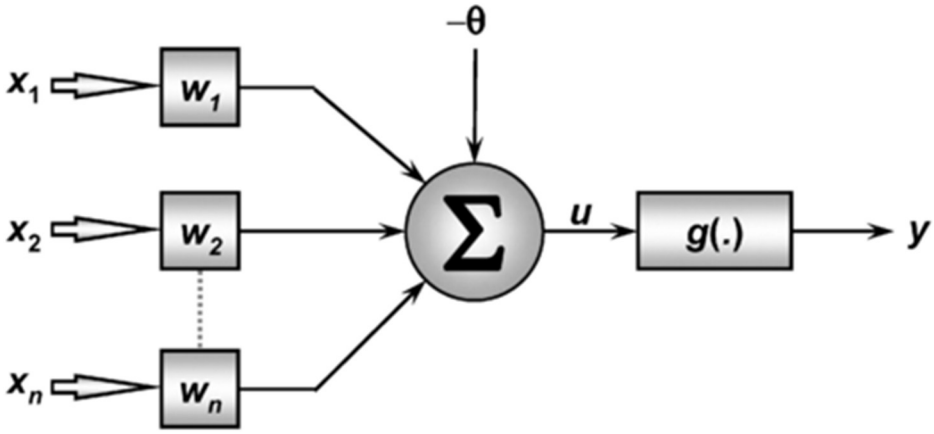


A responsabilidade de processar todas as informações advindas dos dendritos é do corpo celular. Os dendritos, são ramificações localizadas nas pontas dos neurônios. Após processada a informação é transmitida através do axônio por impulsos elétricos, que por final repassa a informação necessária para os próximos neurônios através de suas terminações sinápticas, Figura 1. [4,6]

NEURÔNIO ARTIFICIAL

Inspirado no neurônio biológico, o neurônio artificial, Figura 2, tenta emular a mesma tarefa da sua inspiração, assim, basicamente, ele receberá a informação, irá processá-la, em seguida irá repassá-la processada para outro neurônio. As interligações de vários neurônios formam uma rede denominada rede neural, com paralelismo e alta conectividade. [4]

Figura 2. Fisiologia de um neurônio artificial. [4]



Sua funcionalidade se baseia nos sinais de entrada $\{x_1,\ x_2,\ x_n\}$ sendo multiplicadas pelos pesos sinápticos na entrada do neurônio, $\{w_1,\ w_2, \dots ,\ w_n\}$, onde está especificado se a informação apresentada na entrada e relevante ou não. Em seguida, todos os resultados adquiridos através destes pesos são somados no combinador aditivo linear $\{\Sigma\}$, e, após obter a soma, ela passa pela soma de um limiar de ativação$\{-\theta\}$, sendo o resultado gerado, o valor do potencial de ativação $\{u\}$, caso $u > -\theta$, o neurônio produzirá um potencial excitador, caso contrário, será inibidor; Por fim, u passa por uma função de

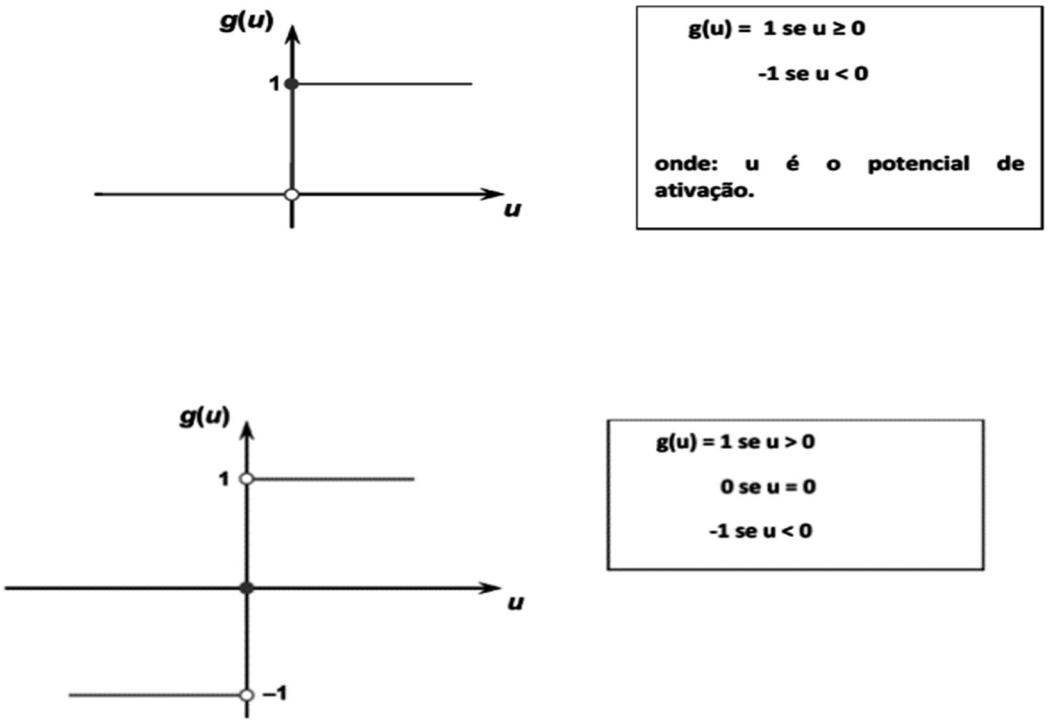
ativação $\{g\}$, limitando o valor adquirido no começo do processo, gerando o sinal de saída $\{y\}$, que poderá ser o resultado final de uma rede, ou o sinal de entrada de um próximo neurônio. [4,7]
As expressões do resultado gerado por um neurônio, propostas por Pitts e McCulloch são:

$$u = \sum_{i=1}^{n} w_i \, . \, x_i - \theta$$

$$y = g(u)$$

"As funções de ativação podem ser divididas em dois grupos, sendo parcialmente diferenciáveis, e totalmente diferenciáveis." [4]
Quando as derivadas de primeira ordem forem inexistentes, é considerada parcialmente diferenciável, já que as diferênças menores que o ponto a serem pesquisados são inexistentes. Esta classe possui três funções principais: função degrau, função degrau bipolar, função rampa simétrica. [4]
Nas funções degrau, e degrau bipolar, Figura 3, são basicamente iguais; ambas analisam o resultado, porém no degrau, o neurônio só é ativado quando possui um valor maior que 0, enquanto a degrau bipolar possui o valor nulo do neurônio, que será quando ele atingir o valor 0. [4, 13]

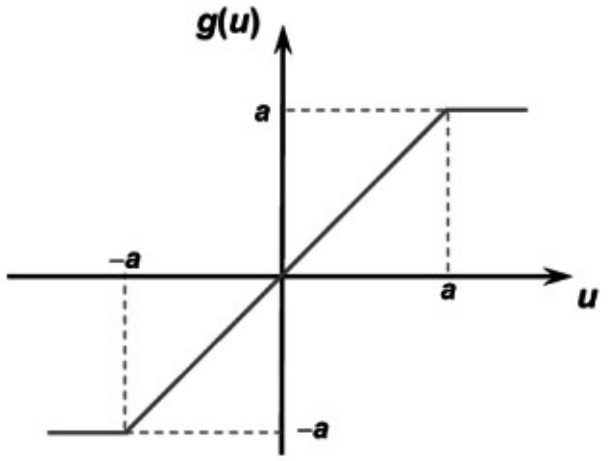
Figura 3. Representação gráfica das funções degrau, e degrau bipolar. [13]



Já a função rampa simétrica, Figura 4, possui um diferencial das outras duas, já que essa ao invés de fazer uma análise entre 0 e 1, ela determina qual o espaço será o de ativação,

assim essa função mostra a evolução da ativação desde seus valores negativos, até seus valores positivos, criando assim a rampa. [4, 13]

Figura 4. Representação gráfica da função rampa simétrica. [13]



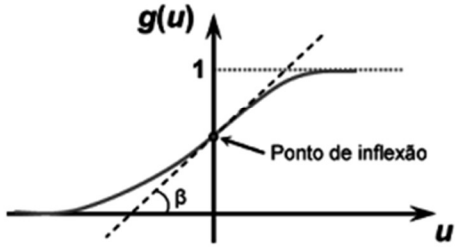
$$g(u) = \begin{cases} a & \text{se } u > a \\ u & \text{se } -a \leq u \leq a \\ -a & \text{se } u < a \end{cases}$$

Para a função ser totalmente diferenciável, as derivadas de primeira ordem são conhecidas e analisadas em todo domínio da função. Possui quatro funções mais conhecidas dessa divisão.

A primeira e a segunda são muito parecidas, sendo elas função logística, e tangente hiperbólica, Figura 5. A diferença básica entre elas é que a função logística, $0 < g(u) < 1$, sendo que $g(u)$ tende a 1, ou seja, os valores 0 e 1 são assíntotas. A função tangente hiperbólica, $-1 \leq g(u) \leq 1$. O mesmo para esta função, ou seja, -1 e 1 a curva é assintótica a -1 e a 1. Elas utilizam uma expressão, e seus valores são analisados entre 0 e 1 na primeira, e na segunda entre -1 e 1. [4, 13]

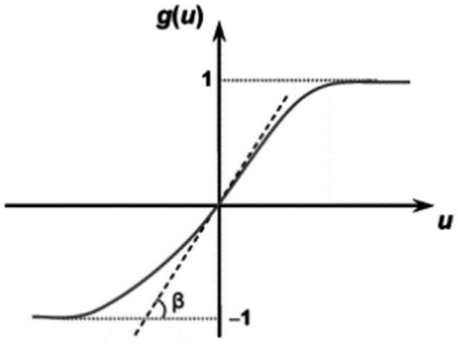
Outra função utilizada nessa divisão, é a gaussiana, Figura 6, que consiste em analisar a saída do neurônio, fazendo com que ele seja igual aos valores que possui o mesmo potencial de ativação, $\{u\}$, e que esteja n mesma distância de seu centro. Para marcar o centro da função, utiliza-se o valor de c (a média), e o desvio padrão, para analisar se ambas possuem o mesmo valor. [4, 13].

Figura 5. Representação gráfica da função logística. [13]



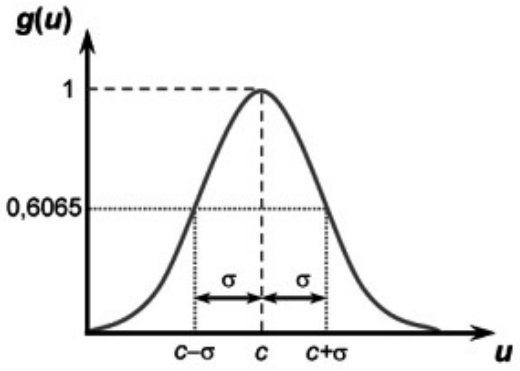
$$g(u) = \frac{1}{1 + e^{-\beta * u}}$$

onde β é uma constante associada ao nível de inclinação da função logística.



$$g(u) = \frac{1 - e^{-\beta * u}}{1 + e^{-\beta * u}}$$

Figura 6. Representação gráfica da função gaussiana. [13]



$$g(u) = e^{\frac{-(u*C)^2}{2*\sigma^2}}$$

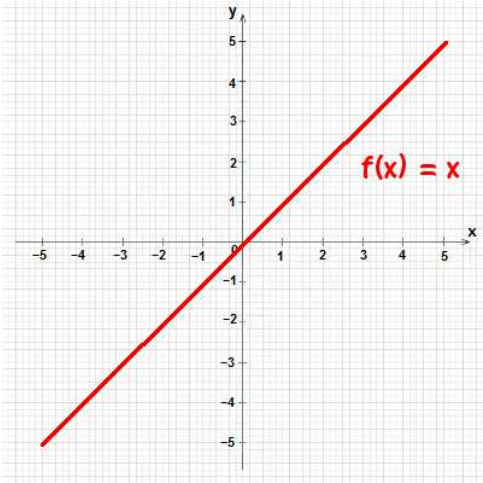
onde:

c: parâmetro que define o centro da função gaussiana.

σ: desvio padrão

Por fim, a última e a função linear, que acaba sendo utilizada para mapear o comportamento entre as variáveis de entrada e saída, já seus valores são iguais ao potencial de ativação. [4, 15]

Figura 7. Representação gráfica da função linear. [15]



DEEP LEARNING (APRENDIZADO PROFUNDO)

Baseado nos princípios do teste de Turing, o aprendizado profundo passa a ser uma evolução do aprendizado de máquina, logo, seu intuito é criar mecanismos que diante de um problema possam resolvê-lo a partir de uma grande análise previamente já feita pela máquina; podendo assim em um caso de reconhecimento de imagem fazer com que a máquina reconheça o que foi pré-determinado, através de um mecanismo treinado por ela, trazendo assim a resposta mais próxima do esperado. [10]
De acordo com Deng e Yu, *"o aprendizado profundo são várias técnicas de treinamento, agindo (analisando) em várias camadas de dados não lineares para analisar padrões e classificações"*. Com a grande ascensão das GPUs, o processamento de várias camadas passou a ser cada vez mais eficiente e rápido, e com essa maior facilidade e, com a queda de custo de hardware, atualmente é umas das áreas mais desenvolvidas dentro das universidades, ainda mais por ser uma técnica aprimorada (que passou a ser explorada) a pouco tempo. Mas apesar de recente, muitas empresas já fazem estudos e desenvolvimentos dentro dessa área. [11]

REDES CONVOLUCIONAIS

A convolução é uma operação matemática que atua sobre duas funções gerando uma terceira, que geralmente é a primeira função modificada $(f(x) * g(x)\, f'(x))$.[11]
Uma convolução unidimensional discreta, sendo, $x \in \mathbb{Z}$, é definida por [5, 11]:

$$f * g[x] = \sum_{m=-\infty}^{\infty} f[m]g[x-m]$$

"A partir da convolução, em meados de 2006, pesquisadores da CIFAR (Canadian Institute for Advanced Research) adicionaram a esse conceito matemático em uma rede neural artificial, a fim de produzir uma rede que não precisasse de informações pré-rotuladas, assim a máquina possuiria um método de aprendizagem não supervisionada".

Essa nova técnica combinada às novas GPUs, gerou uma rede neural muito mais confiável e eficiente. [11]

"Para uma rede neural ser convolucional, ela precisa ter uma convolução em pelo menos uma de suas camadas. Em um cenário de reconhecimento de imagens utilizando uma rede neural artificial a convolução funcionaria da seguinte maneira, uma imagem seria dividida em seus pixels, formando assim uma matriz f (x), para analisar essa imagem, é utilizada um filtro g (x), que terá seus pesos invertidos em g'(x), e esse por sua vez, irá passar por toda a matriz efetuando o cálculo convolucional, que ao final de sua execução, irá ter uma nova matriz f'(x), também conhecida como feature map", Figura 8 e 9. [5, 11]

Figura 8. Função, Filtro, e Filtro balanceado para operação. [5]

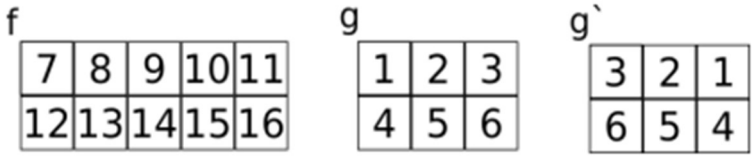
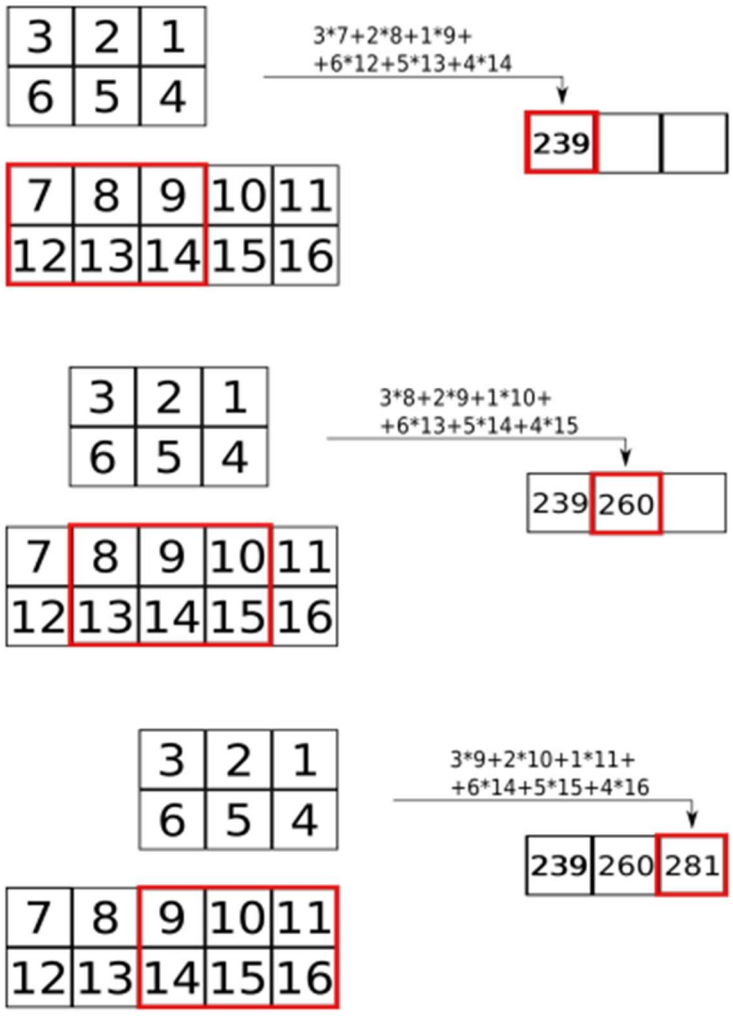


Figura 9. Exemplo funcional de uma convolução em uma matriz. [5]



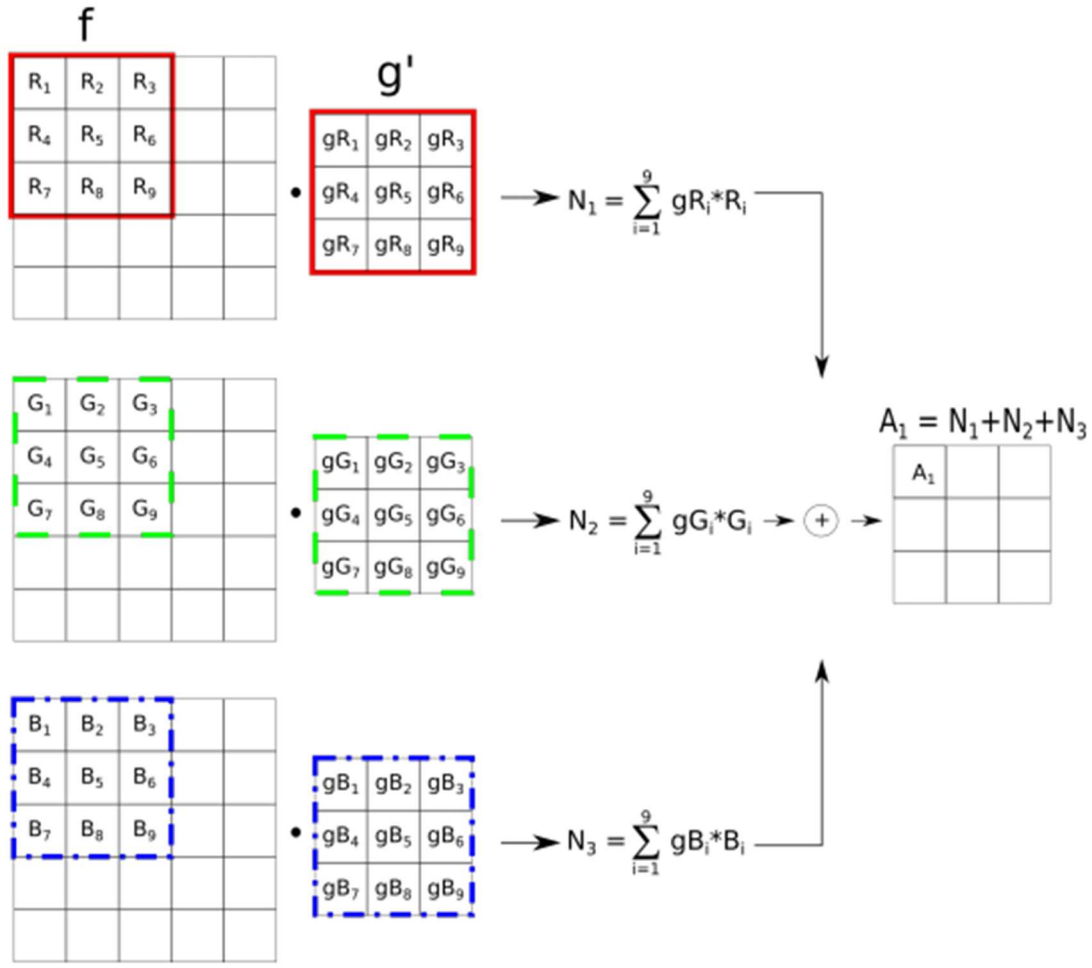
Para o tratamento de imagens ao invés de utilizar a convolução unidimensional, utiliza-se a convolução bidimensional, para fazer análise dos eixos x e y da matriz adquirida na imagem, a equação assume seguinte forma:

$$f * g[x,y] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[n,m]\, g[x-n, \quad y-m]$$

[5;11]

Se a imagem analisada estiver em preto e branco, o processo bidimensional acima é o utilizado, mas caso seja uma imagem colorida, o processo passa a ser analisado por 3 filtros, sendo cada um responsável por analisar (bidimensionalmente) uma cor do sistema RGB (Vermelho, Verde, Azul) de cada vez. Ao final da análise, os resultados dos três *feature maps* são somados, gerando apenas um *feature map* RGB para uma próxima análise, conforme exemplo apresentado na Figura 10. [5]

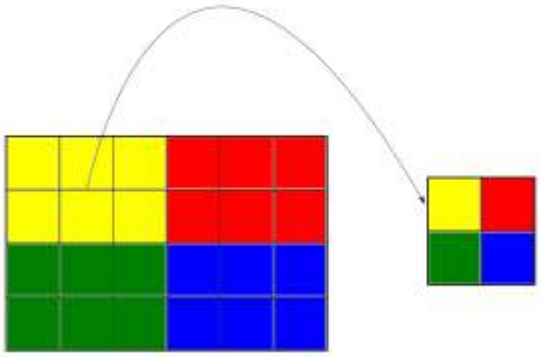
Figura 10. Exemplo funcional de uma convolução em uma matriz de uma imagem colorida, utilizando uma matriz F e G' para cada cor do sistema RGB (Vermelho, Verde, Azul). [5]



POOLING

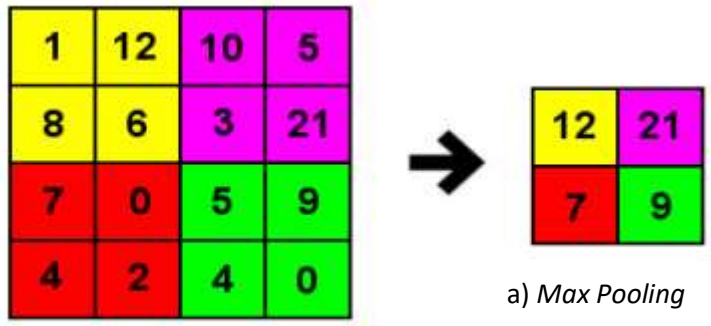
Para haver uma maior agilidade em analisar as imagens, seria preciso que os *feature maps* gerados fossem reduzidos, porém sem perder as informações necessárias, nesse caso, utiliza-se as camadas denominadas *pooling*, Figura 11, reduzindo um *feature map* para um tamanho, que consiga representar a mesma coisa em um ambiente menor. [5;11]

Figura 11. Exemplo básico de como funciona o *Pooling*. [5]

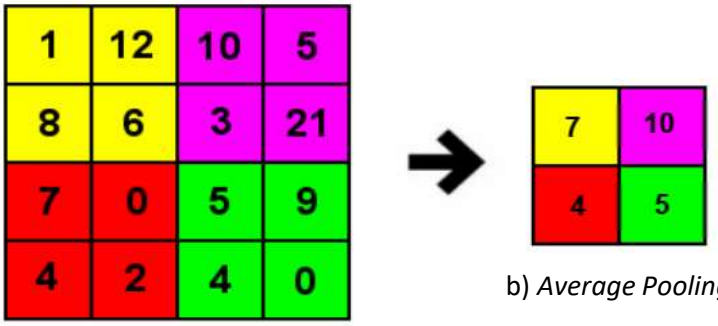


Os dois cálculos mais comuns referentes ao *pooling* são o *Average Pooling*, que calcula a média dos valores da região para a saída; e o *Max Pooling*, que assume o maior valor da região como resultado de saída, como no exemplo da Figura 12. [5]

Figura 12. Exemplo funcional de como funciona o *Max Pooling*, e o *Average Pooling*. [5, 11]



a) *Max Pooling*



b) *Average Pooling*

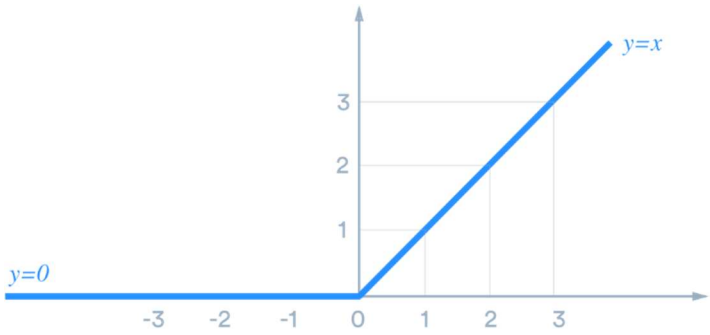
Além de reduzir o tamanho do *feature map* o *pooling* gera a invariância da imagem, que é a ação de representar a parte principal do *feature map*, para a classificação de imagens.

Esse recurso é muito importante, já que ele pode por exemplo, destacar o focinho de um cachorro, já podendo ter um bom argumento para a classificação final. [11]
Em geral, as camadas de *pooling* são responsáveis pelo *downsampling* de um *feature map*, geralmente aparecendo após uma camada convolucional, sendo algo indispensável para uma rede neural, pois cria a invariância e diminui o tamanho do *feature map*, reduzindo o número de análises que ela irá executar. Essas duas camadas em sequência, são inspiradas na neurociência visual [5;11]

ATIVAÇÃO

Para a rede obter êxito na sua parte de reconhecimento, é necessário que a mesma se adeque a modelos não lineares, já que ela não irá avaliar sempre a mesma foto, assim ela precisa se adaptar a cada análise, para poder achar o determinado padrão. Como já citado em neurônios artificiais, há uma multiplicação entre camadas seguida de uma função de ativação (podendo ser ou não linear), para cumprir a ação descrita. Se a rede não passar pela função de ativação, ela simplesmente não cumprirá a sua ação. [5]
Apesar de existirem diversas funções de ativação, a principal a ser utilizada na maioria das redes convolucionais é a *ReLU* (*Rectified Linear Unit*) (Figura 13), devido a sua baixa complexidade matemática, e por possuir uma alta eficiência para análises computacionais, fazendo que muitas vezes economize tempo no treinamento da rede. Ela é responsável por ignorar todas as entradas que possuam um resultado inferior a zero, e retorna o valor da função, caso ele seja positivo, ou seja $x < 0 \ \therefore f(x) = 0, x \geq 0 \therefore f(X) = x$. Atualmente, *ReLU*, é uma das funções mais utilizadas entre as camadas de convolução, principalmente por analisar as evoluções pelas quais máquina passa. [5,12, 18]

Figura 13. Gráfico ilustrativo da função *ReLU* (*Rectified Linear Unit*). [12]

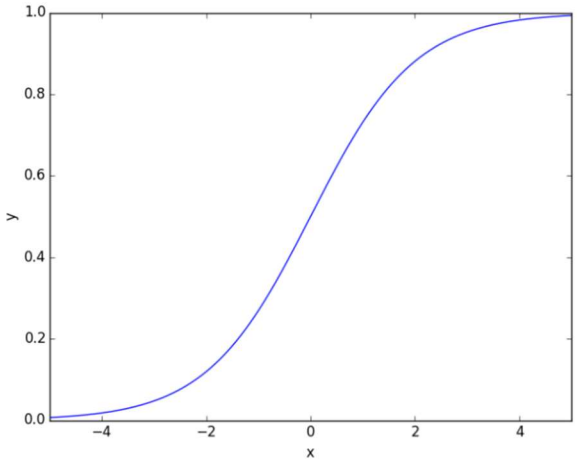


Geralmente ao final das camadas, é aplicada uma ativação diferente, a fim de medir a evolução pela qual a rede passou, medindo desde de a sua perda até aos seus ganhos. Nessa rede, a função final acaba sendo a *sigmoid*, Figura 14, sendo definida como:

$$f(x) = \frac{1}{1+e^{-x}}$$

[5, 14]

Figura 14. Gráfico ilustrativo da função *Sigmoid*. [14]



A sigmoid atuará na fase de reconhecimento, e ela representa uma probabilidade que o valor irá atingir, fazendo assim a previsão da imagem. [14]

REDES PROFUNDAS

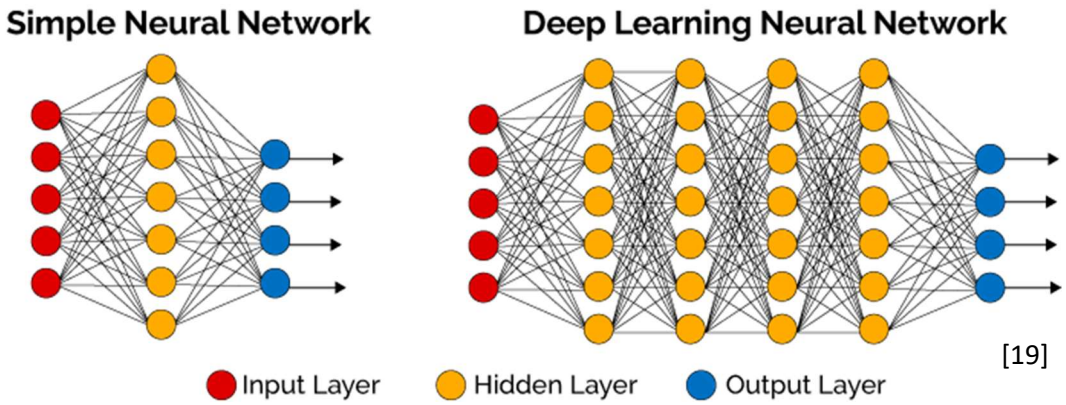
São as redes utilizadas para fazer as várias comparações dos *features maps*, é assim denominada, redes profundas por ter múltiplas camadas ocultas, sendo que nenhuma dessas camadas é uma camada de entrada ou uma camada de saída. Sua funcionalidade é receber os dados de uma determinada camada, e repassar eles para diversas outras, até o final que geralmente termina com uma função que irá classificar o dado. Um exemplo utilizado seria:

$$h_1 = ReLU(xW_1)$$
$$h_2 = ReLU(h_1W_2)$$
$$h_1 = ReLU(h_2W_3)$$
$$f(x) = Sigmoid(h_3)$$

[5]

No exemplo, h_1 representa a entrada da camada profunda, nela possui uma função de ativação ReLU, seguida do valor adquirido, e do peso sináptico da camada, no caso W_1. A rede segue nas camadas h_2 e h_3, sendo que o valor recebido em cada uma para fazer o cálculo, é o resultado da camada anterior. Até chegar a função final $f\ (x)$, que irá classificar o resultado da última camada profunda (h_3), nesse caso, foi utilizada a função *sigmoid*. [5]

Figura 15. Gráfico ilustrativo entre uma rede neural artificial e uma rede neural artificial profunda. [19]



[19]

DESENVOLVIMENTO

O código a ser analisado foi produzido por *Harrison Kinsley* em "***How to use your trained model:*** *Deep Learning basics with Python, TensorFlow and Keras p.6*" [16]. Baseando nessa publicação, foi escrita uma visão e tradução do material disponibilizado, nele apresenta-se uma rede utilizando *Deep Learning* que consegue realizar a classificação entre cachorros e gatos. Para o desenvolvimento da análise, foi utilizado uma plataforma virtual do ambiente Anaconda, com Python 3.6, TensorFlow 1.13.1, e a biblioteca Keras.

O código fonte está totalmente disponível em [16] na língua inglesa.

BASE DE DADOS

Para poder efetuar a análise foi preciso utilizar uma base de dados (*dataset*) disponibilizada no *download center* da *Microsoft* [17], contendo 25000 imagens de gatos e cachorros com conteúdo e tamanho diferenciável. Dessa forma, foi preciso preparar o *dataset* para o poder utilizar na rede. Nesta aplicação 70% do *dataset* foi utilizado para a base teste, e 30% foi utilizado para a validação.

O primeiro passo é a importação de algumas bibliotecas para efetuar as alterações nas imagens. A biblioteca *numpy* para criação de matrizes, a *matplotlib* gerando gráficos 2D para os futuros resultados do treinamento, o *OS* para trazer as informações de sistemas necessárias para manipulação de arquivos, o *cv2* para manipulação de imagens, e por fim o *tqdm* para obter uma barra de evolução, facilitando a análise visual do processo.

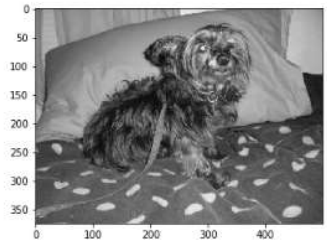
Figura 16. Importações utilizadas para criar o *dataset* (extraído de trechos desenvolvido pelo autor).

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm
```

Figura 17. Imagem original utilizada (extraído do *dataset* disponível em [17])

Figura 18. Conversão de imagem em matriz, e de colorida para preto e branco (extraído de trechos desenvolvido pelo autor).

```
DATADIR = r"E:\tcc\PetImages"

CATEGORIES = ["Dog", "Cat"]

for category in CATEGORIES:  # criando categorias
    path = os.path.join(DATADIR,category)  # criando caminho para imagens
    for img in os.listdir(path):  # passagem das iamgens
        img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)  # convertendo para uma matriz
        plt.imshow(img_array, cmap='gray')  # criando a imagem na matriz
        plt.show()  # mostrando

        break  # mostrando somente uma imagem
    break
```




```
print(img_array)
```

```
[[117 117 119 ... 133 132 132]
 [118 117 119 ... 135 134 134]
 [119 118 120 ... 137 136 136]
 ...
 [ 79  74  73 ...  80  76  73]
 [ 78  72  69 ...  72  73  74]
 [ 74  71  70 ...  75  73  71]]
```

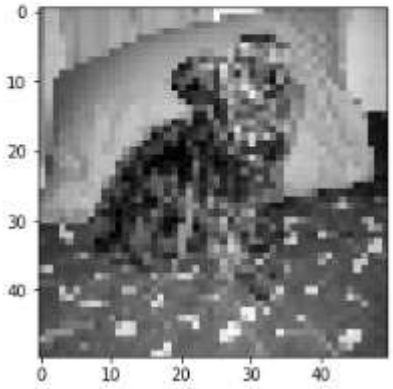
```
print(img_array.shape)
```

```
(375, 500)
```

Em seguida, a mesma imagem tem seu tamanho alterado para o padrão que será utilizado no projeto. Para essa análise, todas as imagens possuem o tamanho 50x50 pixels.

Figura 19. Conversão do tamanho da imagem para 50x50 pixels (extraído de trechos desenvolvido pelo autor).

```
IMG_SIZE = 50

new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
plt.imshow(new_array, cmap='gray')
plt.show()
```



Terminado o exemplo, o mesmo processo é repetido para todas as imagens do *dataset*, essas que estão separadas em duas pastas diferentes, já que cada *dataset* servirá para mostrar a rede o que é um cachorro e o que é um gato.

Figura 20. Conversão de todas as imagens para a criação do *dataset* (extraído de trechos desenvolvido pelo autor).

```
training_data = []

def create_training_data():
    for category in CATEGORIES:  # criando categorias

        path = os.path.join(DATADIR,category)  # criando caminho para imagens
        class_num = CATEGORIES.index(category)  # separa as categorias em 0 (cachorro) e 1 (gato)

        for img in tqdm(os.listdir(path)):
            try:
                img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)  # converte para matriz
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # converte para 50x50
                training_data.append([new_array, class_num])  # adiciona a informação ao trainig_data
            except Exception as e:  # mostra os erros
                pass

print(len(training_data))
```

```
100%|██████████| 12501/12501 [00:59<00:00, 208.80it/s]
100%|██████████| 12501/12501 [00:50<00:00, 245.70it/s]
24946
```

Com o *dataset* em mãos, se embaralha a ordem das imagens, para que não haja uma sequência grande de cachorros e depois uma sequência de gatos, o que acabaria prejudicando a rede neural artificial, Figura 21. Por fim, importa-se a biblioteca *pickle* para salvar o mesmo em uma sequência de bytes, assim, não é necessário executar esses passos novamente para qualquer treino da rede. Será utilizado também o método *dump* que captura dos dados do objeto para gerar os bytes, e assim poder transmitir as informações dentro da rede, como demostrado na Figura 22.

Figura 21. Embaralhando a ordem das imagens (extraído de trechos desenvolvido pelo autor).

```
import random

random.shuffle(training_data)
```

```
X = []
y = []

for features,label in training_data:
    X.append(features)
    y.append(label)

print(X[0].reshape(-1, IMG_SIZE, IMG_SIZE, 1))

X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
```

```
[[[[200]
   [ 62]
   [176]
   ...
   [ 56]
   [ 53]
   [ 57]]

  [[205]
   [198]
   [ 68]
   ...
   [ 62]
   [ 63]
   [ 64]]

  [[204]
   [203]
   [ 77]
   ...
   [ 70]
   [ 93]
   [ 96]]

  ...
```

Figura 22. Salvar o *dataset* (extraído de trechos desenvolvido pelo autor).

```
import pickle

pickle_out = open("X.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()

pickle_out = open("y.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

DESENVOLVIMENTO DE UMA APLICAÇÃO DE REDE NEURAL

Nesta seção mostramos como desenvolver uma aplicação de Aprendizado de Máquina utilizando *TensorFlow*, a biblioteca *Keras* em *Python*. Como na criação do *dataset*, inicialmente se faz os *imports* das bibliotecas utilizadas na rede neural, a maioria das bibliotecas a serem utilizadas foram retiradas do *Keras*, que é uma API de redes neurais, utilizada no *TensorFlow* (biblioteca focada em aprendizado de máquinas desenvolvida pelo *Google*). Os modelos utilizados nesse projeto foram: *Sequential*, para utilizar de seu modelo de rede pré-construído, e adicionar as camadas de densidade (*Dense*) para prever os rótulos, a camada responsável pela ativação do neurônio (*Activation*), a camada *Flatten*, utilizada para transformar a matriz em vetor, a camada de convolução (*Conv2D*) que efetuará a convolução nos dados processados, e pôr fim a camada de *pooling* (*MaxPooling2D*) para reduzir os dados analisados. Além dos *imports* relacionados a rede neural, foi adicionado *pickle* novamente para salvar ao final da execução, *time* para verificação de tempo, e o *TensorBoard* para ver os resultados da rede em forma gráfica. Apresentando a chamada de *imports* na Figura 23.

Figura 23. *Imports* para construção da rede neural profunda (extraído de trechos desenvolvido pelo autor).

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import TensorBoard
import pickle
import time
```

Após os *imports*, são carregados os *datasets* criados com o *pickle* e em seguida o *dataset* é passado por uma normalização de seus dados. Para isso, realiza a operação $\frac{X}{255.0}$, pois o valor máximo que o pixel poderá possuir é 255, logo, é feita essa divisão para se obter o valor normalizado do dado. É declarado depois o número de camadas densas, o tamanho de cada camada (densa e convolucional), e o número de camadas convolucionais. Visualização de como é realizada a importação na Figura 24.

Figura 24. Importação e normalização dos *datasets*, e atribuição de número e tamanho das camadas densas e convolucionais (extraído de trechos desenvolvido pelo autor).

```
pickle_in = open("X.pickle","rb") #importação dos datasets
X = pickle.load(pickle_in)

pickle_in = open("y.pickle","rb")
y = pickle.load(pickle_in)

X = X/255.0 #normalização do dataset

dense_layers = [0, 1, 2] #número de camadas densas
layer_sizes = [32, 64, 128] #tamanho da camda densa
conv_layers = [1, 2, 3] #número de camadas convolucionais
```

Inicia-se a criação da rede neural, cria-se uma estrutura de repetição baseada nos valores atribuídos na Figura 24, e para não confundir os resultados gerados, cada análise é gravada com o nome dos dados da estrutura de repetição. Para iniciar a rede neural, é utilizado o modelo *Sequential* para criar a estrutura, a camada de entrada será uma camada convolucional, com o número de camadas de *layer_size*, um filtro 3x3, e uma matiz de entrada do *dataset*. Terminada, os dados são passados pela camada de ativação (sendo utilizada a função *ReLU*), e pôr fim a camada *pooling*, utilizando o método de *max-pooling* (*MaxPooling2D*) para obter os resultados relevantes da matriz analisada, apresentada na Figura 25.

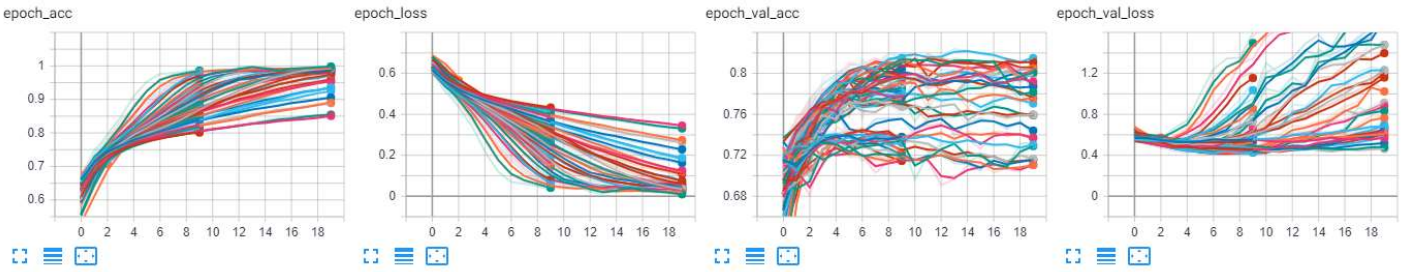
Na sequência, todo o processo é refeito (convolução, ativação e *pooling*), porém com a diminuição de uma camada convolucional conforme Figura 25.

Passando por essas duas etapas, as informações adquiridas na matriz são transformadas em um vetor, e repassadas para análise nas camadas densas, após o processamento, é passada pela ativação, novamente utilizando a função *ReLU*. A análise final é passada por uma camada densa, e a ativação que irá classificar a imagem, para essa ação a função a ser utilizada será a *Sigmoid*. Para analisar o resultado final, é chamado o *TensorBoard* para análise gráfica, apresentada na Figura 26.

Figura 25. Rede neural convolucional densa (extraído de trechos desenvolvido pelo autor).

```
for dense_layer in dense_layers:
    for layer_size in layer_sizes:
        for conv_layer in conv_layers:
            NAME = "{}-conv-{}-nodes-{}-dense-{}".format(conv_layer, layer_size, dense_layer, int(time.time()))
            print(NAME)

            model = Sequential() #estrutura da rede neural

            model.add(Conv2D(layer_size, (3, 3), input_shape=X.shape[1:])) #rede convolucional recebendo a primeira imagem
            model.add(Activation('relu')) #camada de ativação
            model.add(MaxPooling2D(pool_size=(2, 2))) #camada pooling

            for l in range(conv_layer-1):
                model.add(Conv2D(layer_size, (3, 3)))  #rede convolucional recebendo dados da outra rede
                model.add(Activation('relu')) #camada de ativação
                model.add(MaxPooling2D(pool_size=(2, 2))) #camada pooling

            model.add(Flatten()) #tranformação de matriz em vetor

            for _ in range(dense_layer):
                model.add(Dense(layer_size)) #camada densa com o numero atribuido em layer_size
                model.add(Activation('relu')) #camada de ativação

            model.add(Dense(1)) #camada densa com uma camada
            model.add(Activation('sigmoid')) #camada de ativação para classificar

            tensorboard = TensorBoard(log_dir="logs/{}".format(NAME))
```

Figura 26. Resultado gráfico através do TensorBoard (extraído de trechos desenvolvido pelo autor).



Terminada a rede, ela é compilada através de uma função de perda (*binary_crossentropy*), um otimizador (*adam*), e a métrica de acerto (*accuracy*). Após a compilação, é feito um ajuste através do método *fit*, e treina o modelo em um determinado número de épocas (*epochs*). E então os resultados começam a aparecer na tela. Na Figura 27 representa está parte do código, e a Figura 28 exibe uma pequena parte dos resultados gerados.

Figura 27. Compilação e ajustamento da rede, para então executar e mostrar os resultados (extraído de trechos desenvolvido pelo autor).

```
            model.compile(loss='binary_crossentropy',
                          optimizer='adam',
                          metrics=['accuracy'],
                          )

            model.fit(X, y,
                      batch_size=32,
                      epochs=20,
                      validation_split=0.3,
                      callbacks=[tensorboard])

model.save('64x3-CNN.model')
```

Figura 28. Parte dos resultados gerados (1 convolução, 32 nós, 0 densas, 20 épocas) (extraído de trechos desenvolvido pelo autor).

```
1-conv-32-nodes-0-dense-1574614753
Train on 17462 samples, validate on 7484 samples
Epoch 1/20
17462/17462 [==============================] - 8s 457us/sample - loss: 0.6183 - acc: 0.6621 - val_loss: 0.5711 - val_acc: 0.7
104
Epoch 2/20
17462/17462 [==============================] - 8s 446us/sample - loss: 0.5513 - acc: 0.7239 - val_loss: 0.5699 - val_acc: 0.7
022
Epoch 3/20
17462/17462 [==============================] - 8s 469us/sample - loss: 0.5200 - acc: 0.7444 - val_loss: 0.5919 - val_acc: 0.6
808
Epoch 4/20
17462/17462 [==============================] - 8s 448us/sample - loss: 0.4983 - acc: 0.7608 - val_loss: 0.5486 - val_acc: 0.7
247
Epoch 5/20
17462/17462 [==============================] - 8s 446us/sample - loss: 0.4800 - acc: 0.7774 - val_loss: 0.5318 - val_acc: 0.7
401
Epoch 6/20
17462/17462 [==============================] - 8s 449us/sample - loss: 0.4622 - acc: 0.7844 - val_loss: 0.5276 - val_acc: 0.7
```

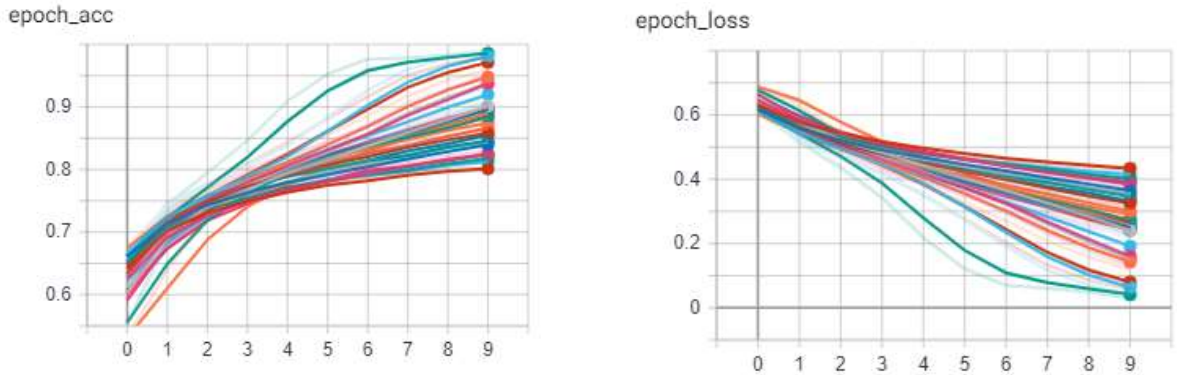
TESTES E RESULTADOS

O teste busca analisar qual o melhor resultado entre várias redes neurais com diversas características diferentes como: número de camadas densas, número de camadas convolucionais, número de nós por camadas e, também, qual o número de épocas para obtenção dos resultados. Como o material utilizado para esse trabalho tem uma função didática, o autor analisa várias redes neurais, com a intenção de mostrar como cada camada pode ter uma determinada influência na rede neural. Para mostrar essas diferenças ele trabalhou com modelos possuindo de zero a duas camadas densas, e de uma a três camadas convolucionais, ambas possuem nós de 32, 64, ou 128 camadas internas, possuindo 10 épocas (0 a 9) por análise. Lembrando que o *dataset* utilizado está dividido em 70% das imagens para aprendizado (17500 imagens) e 30% das imagens para validação (7500 imagens). Dadas as características, as redes analisadas estão descritas na Tabela 1. Os resultados dos testes estão expressos na Figura 29.

Tabela 1. Estruturas das redes criadas por [16] para análise.

Convolução	Nós	Densa
1 conv	32 nodes	0 dense
1 conv	64 nodes	0 dense
1 conv	128 nodes	0 dense
1 conv	32 nodes	1 dense
1 conv	64 nodes	1 dense
1 conv	128 nodes	1 dense
1 conv	32 nodes	2 dense
1 conv	64 nodes	2 dense
1 conv	128 nodes	2 dense
2 conv	32 nodes	0 dense
2 conv	64 nodes	0 dense
2 conv	128 nodes	0 dense
2 conv	32 nodes	1 dense
2 conv	64 nodes	1 dense
2 conv	128 nodes	1 dense
2 conv	32 nodes	2 dense
2 conv	64 nodes	2 dense
2 conv	128 nodes	2 dense
3 conv	32 nodes	0 dense
3 conv	64 nodes	0 dense
3 conv	128 nodes	0 dense
3 conv	32 nodes	1 dense
3 conv	64 nodes	1 dense
3 conv	128 nodes	1 dense
3 conv	32 nodes	2 dense
3 conv	64 nodes	2 dense
3 conv	128 nodes	2 dense

Figura 29. Resultado dos testes efetuados por [16] (extraído de trechos desenvolvido pelo autor).



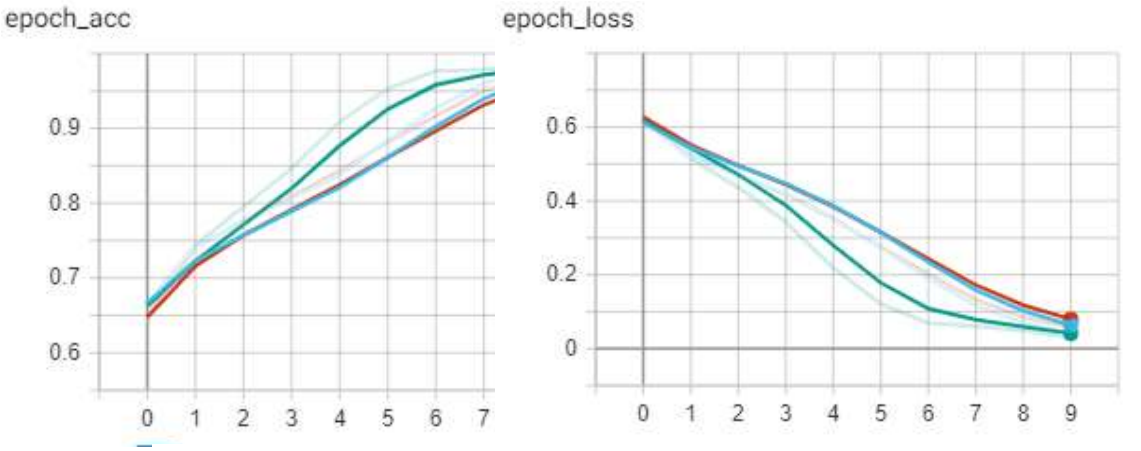
a) Épocas de acurácia

b) Épocas de perda

Analisando os gráficos de teste, é possível perceber que a rede é funcional devido aos seus valores de acurácia (Figura 29 a) estrarem entre 0.8 e um pouco acima de 1.0; e os valores de perda (Figura 29 b) estarem abaixo de 0.5. Esses são bons valores, pois a

acurácia mostra que em teste a máquina chegou a pelo menos 80% de aprendizado sobre as imagens e, o valor de perda, significa que sua perda de informação adquirida é de menos de 50%. O resultado poderia ser negativo caso a acurácia fosse baixa (menor que 70%), ou caso a taxa de perda fosse muito alta (acima de 50%), mesmo tendo uma taxa de acurácia alta, pois não adianta a máquina aprender, e perder seus dados em sequência. Apesar dos bons resultados, existem alguns modelos que se destacam mais que os outros nessa fase de testes. Os três principais (Figura 30) a serem utilizados foram os que possuíam uma convolução, 64 nós (vermelha) e 128 nós (azul e verde), e uma camada densa (azul), ou duas (vermelha, verde). Dando um principal destaque para a rede com uma camada convolucional, 128 nós, e duas camadas densas (verde).

Figura 30. Melhores resultados dos testes efetuados por [16] (extraído de trechos desenvolvido pelo autor).



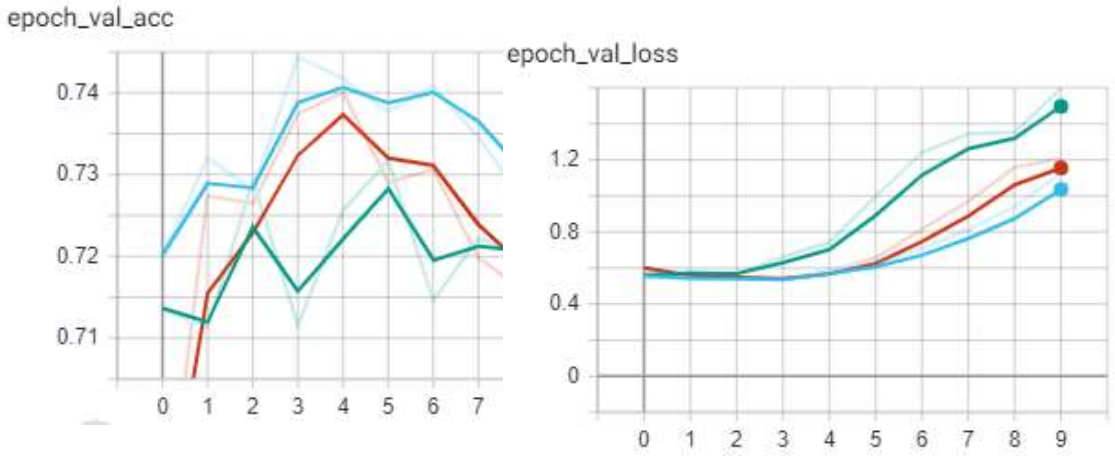
a) Épocas de acurácia b) Épocas de perda

Tabela 2. Resultados dos testes do autor

Modelo	Aprendizado	Perda	Tempo'	Cor
1-conv-128-nodes-1-dense	0.9767	0.0774	~13.54	Azul
1-conv-64-nodes-2-dense	0.9765	0.0670	~4.52	Vermelho
1-conv-128-nodes-2-dense	0.9871	0.0398	~13.44	Verde

Porém ao analisar as suas validações, percebe-se que a inteligência não atingiu a mesma eficiência que nos testes. A rede 1-conv-128-nodes-1-dense, até desempenha um bom papel no aprendizado, porém tem uma taxa de perda muito alta, já que a mesma tem que chegar o mais próximo de 0, como os exemplos contidos na Tabela 2. A Figura 31, e a Tabela 3 mostram o resultado negativo que esses três modelos obtiveram ao validar as imagens.

Figura 31. Resultados das validações efetuadas por [16], referente a tabela1 (extraído de trechos desenvolvido pelo autor).



a) Épocas de acurácia b) Épocas de perda

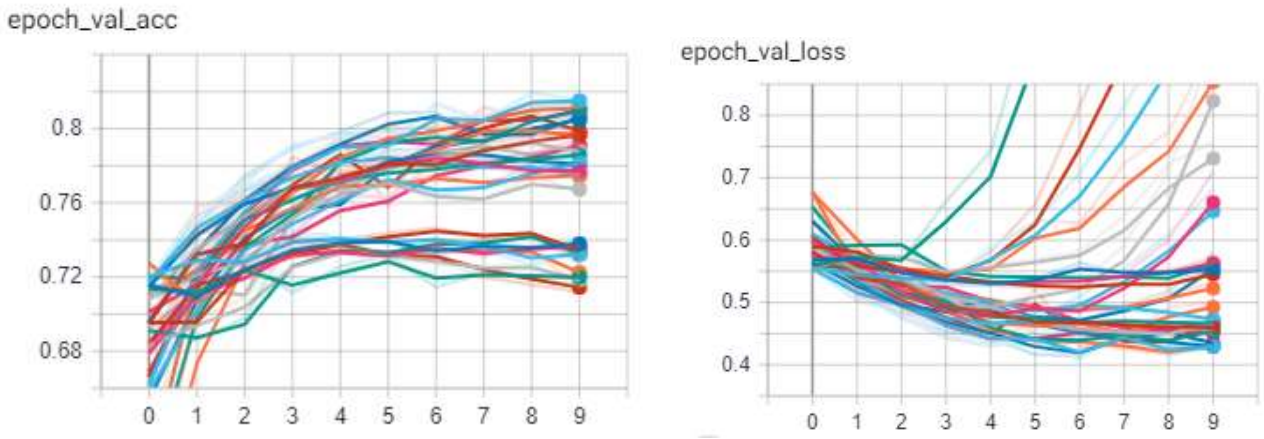
Tabela 3. Resultados das validações do autor

Modelo	Aprendizado	Perda	Tempo'	Cor
1-conv-128-nodes-1-dense	0.7356	0.9490	~13.54	Azul
1-conv-64-nodes-2-dense	0.7159	1.3458	~4.52	Vermelho
1-conv-128-nodes-2-dense	0.7218	1.5668	~13.44	Verde

É normal os resultados da validação serem um pouco piores do que os dos testes, pois não está em um ambiente de teste, sendo assim ela trata os dados de uma forma mais aprofundada do que em uma zona de testes, em que pequenas diferenças são ignoradas para um funcionamento mais rápido. Porém os casos na Tabela 3, apesar de terem valores aceitáveis de acurácia (acima de 70% nesse caso), a variação da taxa de perda aumentou muito chegando a ultrapassar o valor 1.0 (nesse caso mais de 100% das infomações adiquiridas no processamento de imagens foi misturada), nessas validações, as redes não conseguiriam fazer as distinções, já que as mesmas misturaram as informações adiquiridas, analisándo de uma forma geral, nesses casos apresentados, as chances de ela acertar se a imagem é de gato ou cachorro, seria na base de chute. Sendo assim, nem sempre o que é adiquirido nos treinos é sera realizado na validação.

Vamos analisar as validações das redes neurais citadas no começo do capítulo, que foram propostas por [16] na Figura 32.

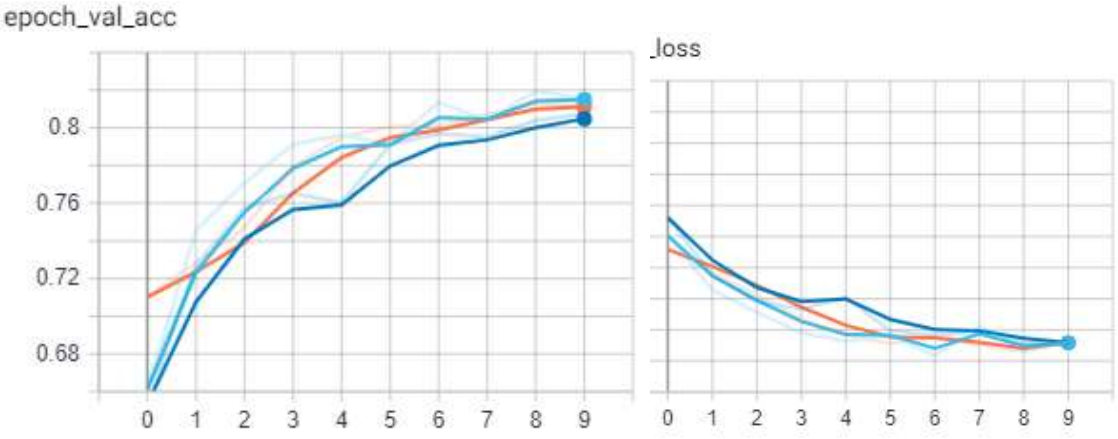
Figura 32. Resultado das validações efetuadas por [16] (extraído de trechos desenvolvido pelo autor).



a) Épocas de acurácia b) Épocas de perda

Os modelos funcionais apresentados nos gráficos (Figura 32) são os que se possuem um valor de acurácia (Figura 32 a) acima de 0.7 (nesse caso todas se encaixam), e estejam com um valor de perda (Figura 32 b) abaixo de 0.5, para ter certeza de que o medelo apresentado possua a capasidade de aprender, e manter as informações adquiridas. Analisando visualmente o gráfico, e eliminando os modelos que visualmente não atendem os pré requistos, chegamos a três soluções (Figura 33), que mostram as melhores caracteristicas para poder executar essa opração.

Figura 33. Resultado das validações com menor taxa de perda efetuadas por [16] (extraído de trechos desenvolvido pelo autor).



a) Épocas de acurácia b) Épocas de perda

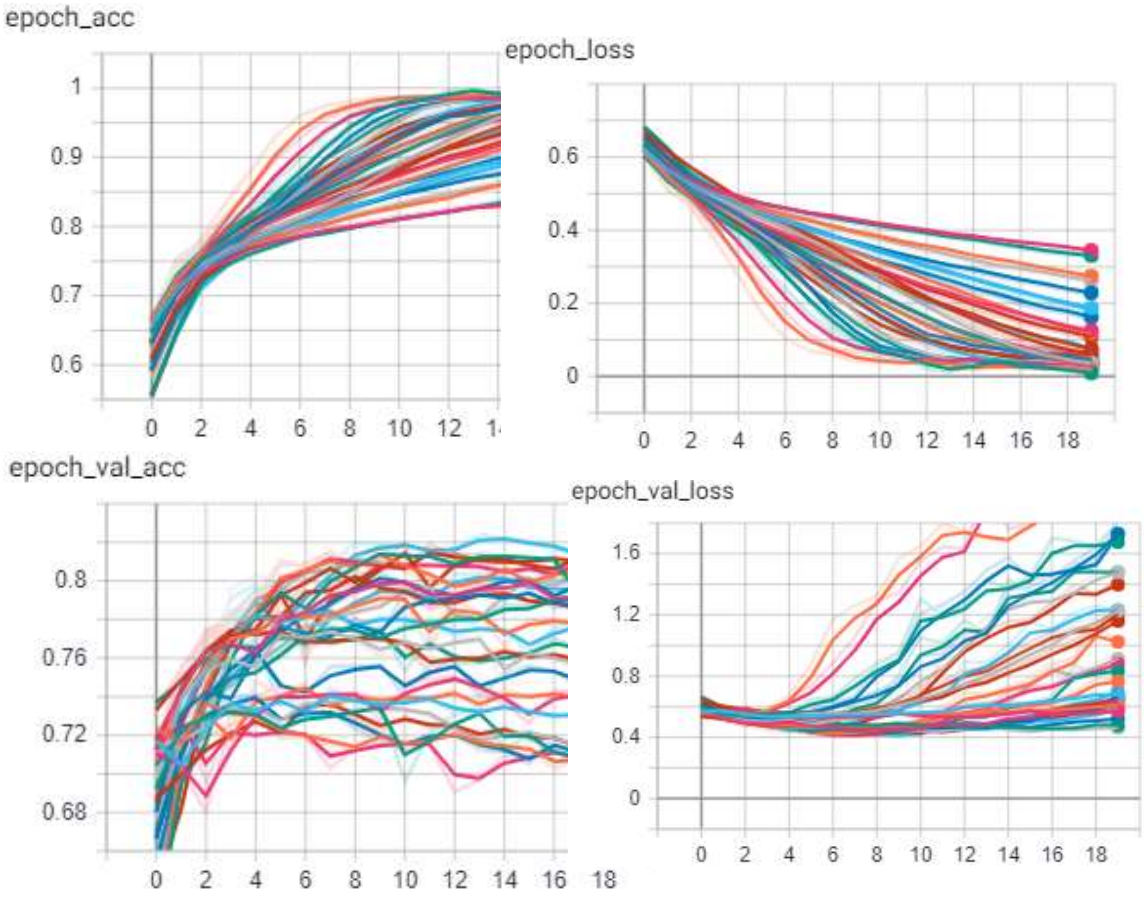
Tabela 4. Resultado das validações com menor taxa de perda efetuadas pelo autor

Modelo	Aprendizado	Perda	Tempo'	Cor
3-conv-128-nodes-0-dense	0.8180	0.4360	~14.65	Azul Claro
3-conv-64-nodes-0-dense	0.8112	0.4467	~6.4	Laranja
3-conv-32-nodes-2-dense	0.7981	0.4450	~3.17	Azul Escuro

Analisando a Tabela 4, percebese que o número de convoluções faz uma grande diferença na validação da rede, já que os modelos que apresentaram as melhores validações possuem três camadas convolucinais, o que gerou a melhoria na taxa de perda, e até mesmo na taxa de aprendizado da máquina. Isso ocorre pelo maior tratamento dos dados ao serem analisados nas camadas convoolucionais adicionadas.

Baseando-se nessas redes, foram adicionadas mais dez epocas (0 a 19), para ver a variação que a rede pode obter, e se as taxas de aprendizado e perda iriam obter uma melhora, já que os dados seriam analisados por um periodo maior.

Figura 34. Resultado dos testes e validações efetuadas com 20 épocas (extraído de trechos desenvolvido pelo autor).

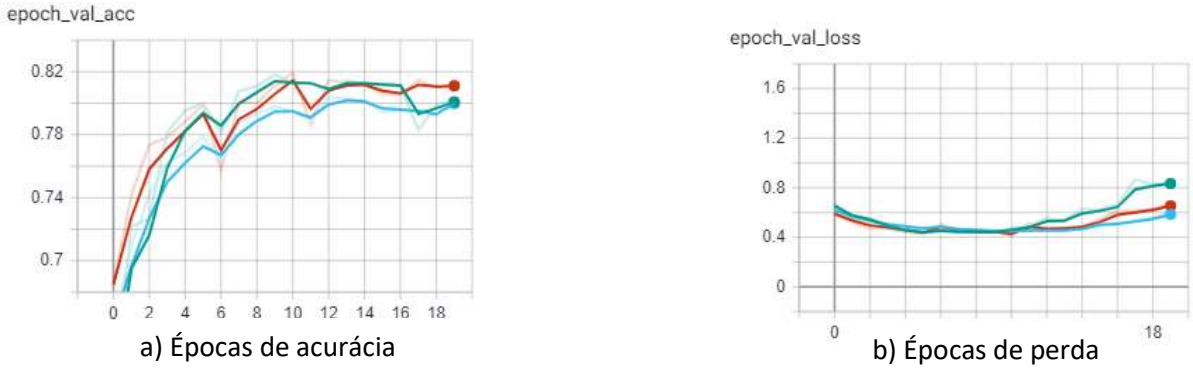


a) Épocas de acurácia b) Épocas de perda

Visualmente, o aumento de épocas (Figura 34) foi negativo para os modelos de redes neurais. Nas validações, o valor de acurácia ficou acima de 0.7 (Figura 34 a), porém os valores de perda (Figura 34 b) se mostram maiores do que a versão com 10 épocas (Figura 32 b). No gráfico de perda (Figura 34 b) é evidente de que as redes neurais ou mantem um padrão desde o início, ou senão, começam a ter variações após um tempo de análise, que fazem com que o valor da perda aumente, o que é indesejavel para uma rede neural. Muito provavelmente, isso ocorre devido a padronização encontrada em sequência, o que pode acabar confundindo a máquina.

Para uma melhor comparação, as melhores estruturas encontradas com dez épocas(Figura 31 e Tabela 4), serão comparadas com as de 20 épocas (Figura 35 e Tabela 5), com relação a ativação delas, já que nos gráficos de teste (Figura 34),os resultados apresentados são superiores aos dados aprensetados nos gráficos de testes com 10 épocas (Figura 29).

Figura 35. Resultado das validações efetuadas com 20 épocas com as estruturas da tabela 4 (extraído de trechos desenvolvido pelo autor).



a) Épocas de acurácia

b) Épocas de perda

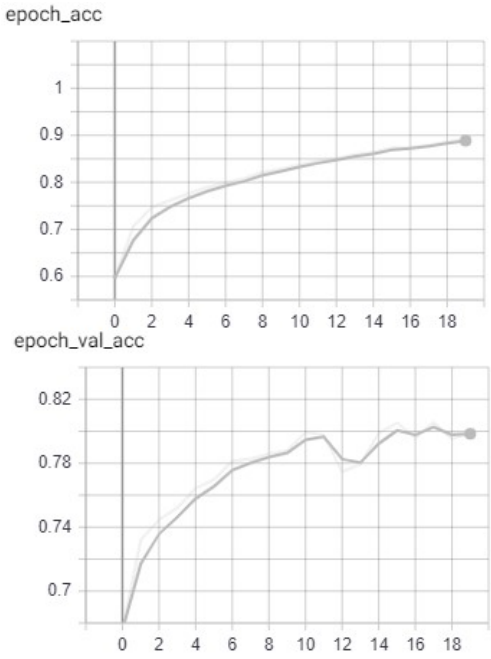
Tabela 5. Resultado das validações com base na tabela 3, possuindo 20 épocas.

Modelo	Aprendizado	Perda	Tempo'	Cor
3-conv-128-nodes-0-dense	0.8030	0.8443	~29.29	Verde
3-conv-64-nodes-0-dense	0.8113	0.6729	~13.00	Vermelho
3-conv-32-nodes-2-dense	0.8038	0.6046	~6.34	Azul

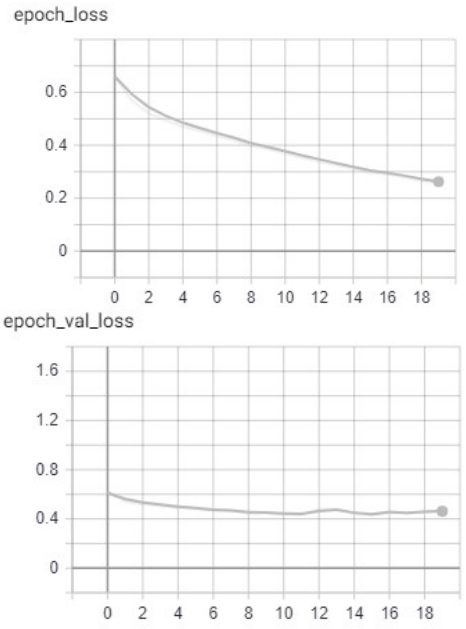
Comparando as tabelas 4 e 5, é nítido que os três casos regrediram em melhorar suas taxas, ao ponto de que o modelo 3-conv-128-nodes-0-dense com 10 épocas possuía o melhor resultado, enquanto o mesmo modelo com 20 épocas tem sua taxa de perda aumentada de 0.4360 para 0.8443.

Executando a mesma análise de gráficos da Figura 32, e eliminado as redes com os maiores valores de perda, chegou-se a um novo modelo de rede neural, sendo esse o que possuía a menor taxa de perda, chegando mais perto do desejável, sendo ele composto por 3-conv-32-nodes-0-dense. Sua representação está na Figura 36.

Figura 36. Resultado dos testes e validações efetuadas com o modelo 3-conv-32-nodes-0-dense com 20 épocas (extraído de trechos desenvolvido pelo autor).



a) Épocas de acurácia



b) Épocas de perda

Tabela 6. Resultado das validações efetuadas com o modelo 3-conv-32-nodes-0-dense com 20 épocas.

Modelo	Aprendizado	Perda	Tempo'	Cor
3-conv-32-nodes-0-dense	0.7988	0.4640	~6.35	Cinza

Se compararmos esse modelo (Figura 36 e Tabela 6), com os modelos da Tabela 5, percebe-se a grande diferença em relação a taxa de perda, que como já citado, é a prinicipal forma de saber se a inteligência obtem progresso ou não, por não perder as informações adiquiridas.

Parando para analisar os números das Tabelas 4, 5 e 6, é notável que em todas as estruturas de melhor taxa possuem 3 camadas de convolução. Na sua grande maioria, não utiliza as camadas densas (salvo 3-conv-32-nodes-2-dense, que utiliza somente 32 nós, o que evita o conflito de informação que deve estar tendo dentro das camadas). Referente aos nós, em dez épocas, sua variação não é tanto perceptiva, ainda mais que a rede que obteve a melhor taxa de perda (0.4360), possuía 128 nós; porém ao aumentar o número de épocas e estendendo o processo, percebe-se que quanto maior o número de nós, maior a chance da rede neural se perder na hora da validação de dados (totalmente notável na mudança de 0.4360 para 0.8443 no modelo 3-conv-128-nodes-0-dense). Tudo fica mais aparente, quando o melhor modelo com 20 épocas é dado como 3-conv-32-nodes-0-dense, mostrando que trabalhando com um menor número de nós, ele consegue manter uma constante na sua taxa de perda.

Baseando-se nesses fatos e nos trabalhos de [5] e [7], um novo teste foi sugerido. Baseando nos dados já presentes, os modelos a serem testados irão possuir três camadas convolucionais, tendo entre 32, 64, e 128 nós, afim de achar alguma diferença entre eles, e possuíram zero ou três camadas densas, afim de analisar como a camada densa interfere no processo. Como já testado, o número de épocas influencia muito no resultado final, e isso provavelmente se deve a repetição de processos, que ao fazer várias vezes o mesmo caminho, acaba se perdendo na hora da validação. Para resolver esse problema, será adicionado duas camadas de *dropout*, conforme descrito em [5] e [7]. A camada de *dropout* será responsável por eliminar nós aleatoriamente, a cada vez que a informação passar por uma camada; assim pretende se criar uma variação entre as épocas, fazendo com que com a sua taxa de perda na validação se mantenha baixa. E para reanalisar a questão de extensão de épocas, foram adicionas mais 20, sendo assim, esse novo teste será realizado por 40 épocas (0 a 39).

Para as realizações de cima, as alterações feitas no código, serão representadas na Figura 37.

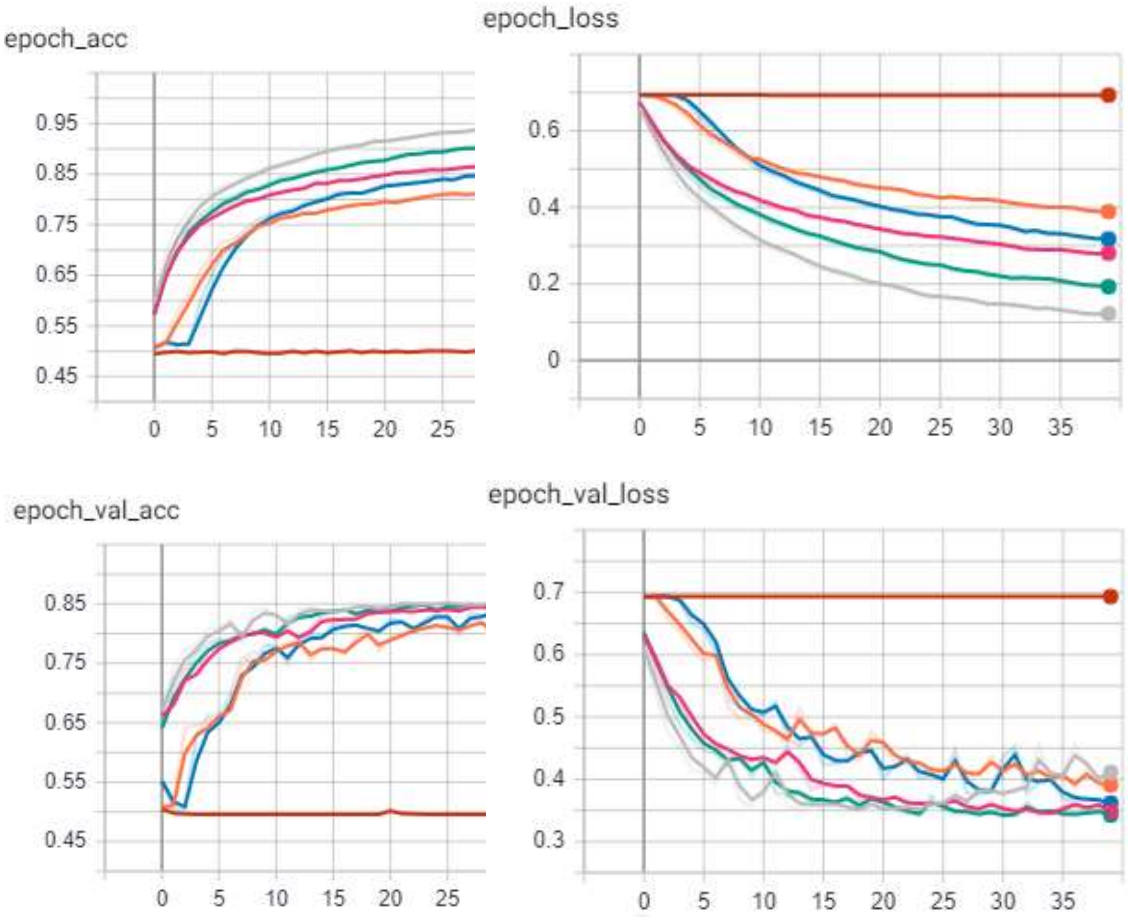
A primeira camada de dropout, é de 0.25, significando que 25% dos nós são eliminados por passagem, e a seguinte é de 0.5, representado 50% dos nós.

Figura 37. Alterações para inclusão da camada *dropout* (extraído de trechos desenvolvido pelo autor).

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import TensorBoard
import pickle
import time

pickle_in = open("X.pickle","rb")
X = pickle.load(pickle_in)

pickle_in = open("y.pickle","rb")
y = pickle.load(pickle_in)

X = X/255.0

dense_layers = [0,3]
layer_sizes = [32, 64, 128]
conv_layers = [3]

for dense_layer in dense_layers:
    for layer_size in layer_sizes:
        for conv_layer in conv_layers:
            NAME = "{}-conv-{}-nodes-{}-dense-{}".format(conv_layer, layer_size, dense_layer, int(time.time()))
            print(NAME)

            model = Sequential()

            model.add(Conv2D(layer_size, (3, 3), input_shape=X.shape[1:]))
            model.add(Activation('relu'))
            model.add(MaxPooling2D(pool_size=(2, 2)))

            for l in range(conv_layer-1):
                model.add(Conv2D(layer_size, (3, 3)))
                model.add(Activation('relu'))
                model.add(MaxPooling2D(pool_size=(2, 2)))
                model.add(Dropout(0.25))

            model.add(Flatten())

            for _ in range(dense_layer):
                model.add(Dense(layer_size))
                model.add(Activation('relu'))
                model.add(Dropout(0.5))

            model.add(Dense(1))
            model.add(Activation('sigmoid'))

            tensorboard = TensorBoard(log_dir="logs/{}".format(NAME))

            model.compile(loss='binary_crossentropy',
                          optimizer='adam',
                          metrics=['accuracy'],
                          )

            model.fit(X, y,
                      batch_size=32,
                      epochs=40,
                      validation_split=0.3,
                      callbacks=[tensorboard])

model.save('64x3-CNN.model')
```

Após as alterações os resultados se mostraram bastante positivos, confirmando a importancia de se gerar variações em uma rede muito loga. Os resultados estão expostos na Figura 38.

Figura 38. Resultado dos testes e validações efetuadas com 40 épocas, e duas camadas *dropout* (extraído de trechos desenvolvido pelo autor).



a) Épocas de acurácia b) Épocas de perda

Analisando o gráfico, percebe-se a melhoria obtida na maior parte dos modelos, tendo somente um que não atingiu as espectativas (3-conv-128-nodes-3-dense), sendo assim, autilização do dropout se mostar essencial para casos que possua várias épocas.

Tabela 7. Resultado das validações efetuadas com 40 épocas, e duas camadas *dropout*.

Modelo	Aprendizado	Perda	Tempo'	Cor
3-conv-128-nodes-0-dense	0.8326	0.4167	~60.00	Cinza
3-conv-64-nodes-0-dense	0.8545	0.3392	~26.70	Verde
3-conv-32-nodes-0-dense	0.8506	0.3430	~12.70	Rosa
3-conv-64-nodes-3-dense	0.8435	0.3590	~30.00	Azul
3-conv-32-nodes-3-dense	0.8252	0.3889	~13.4	Laranja
3-conv-128-nodes-3-dense	0.6932	0.4956	~58.00	Vermelho

Ao analisar os dados da Tabela 7, é nótavel a melhoria obtida principalmente em 3-conv-64-nodes-0-dense, tendo sua taxa de perda em 0.3392, marca que nenhuma das outras redes (tabelas 4, 5 e 6) conseguiram atingir antes. Mostrando que uma boa base para diferenciar duas espécies, seria uma de 3 camadas convolucinais, possuindo 32 ou 64 nós,

utilizando duas camadas de dropout (0.25 e 0.5) e nenhuma camada densa, para evitar conflito de informações.

CONCLUSÕES

No trabalho foi desenvolvido uma rede neural artificial, que tem a capacidade de distinguir cachorros de gatos, sua base foi retirada de [16], ele já estava funcional e cumpria o que havia sido proposto, porém em uma tentativa de analisar seu comportamento com épocas maiores, percebeu-se uma inconstância no código, que fazia com que ele se perdesse; e após algumas análises em [5] e [7], algumas alterações foram feitas no código original. Essas alterações proporcionaram uma melhoria, consertando o problema encontrado no código.

Inicialmente o código continha estruturas que possuíam de zero a duas camadas profundas, de uma a três camadas convolucionais, e nós nos tamanhos de 32, 64 e 128, duas camadas de *pooling*, e três ativações, sendo duas *ReLU* e uma *Sigmoid*, atuando em 10 épocas. Para fazer a análise da rede foi utilizado um *dataset* de 25000 imagens. Na primeira análise, percebeu-se a grande importância de se possuir camadas convolucionais, e que se ao utilizar as redes densas para esses tipos de casos de diferenciação de imagens na rede neural podem acabar se atrapalhando e misturando as informações necessárias para fazer a validação das imagens. Em uma segunda análise, a rede foi colocada a interagir em 20 épocas, para ver seu desempenho, que acabou obtendo mal resultados. Por fim com a base utilizada em [5] e [7], duas camadas de *dropout* foram adicionas a rede, o que fez os dados que antes se perdiam, tomassem caminhos diferentes, assim a rede se obrigava a descobrir novos padrões para diferenciar as espécies, para testar as épocas, essa versão adotou 40 épocas, a fim de ver se o prolongamento da rede seria um problema.

Por fim, através das análises de todas as bases deu para concluir que o melhor modelo para seu utilizar em uma análise de espécies seria uma rede neural com a camada densa final, três camadas convolucionais, nós nos tamanhos de 32 ou 64, duas camadas de *pooling*, e três ativações, sendo duas *ReLU* e uma *Sigmoid*, atuando em 40 épocas. Essas duas redes apresentadas foram as que obtiveram o melhor resultado nas análises, e se mostram muito prestativas para efetuar essa ação.

Em geral, foi possível comprovar a grande capacidade que uma rede neural artificial pode possuir, e como suas camadas podem fazer diferença no resultado final. Essas redes se mostram muito úteis não somente na diferenciação de imagens, mas também em outras áreas, com segurança, saúde, educação, entre outras. Se mostra como sendo uma das grandes tecnologias a serem adotadas em um futuro próximo para a execução de tarefas no nosso dia a dia.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] RUSSELL, Stuart; NORVIG, Peter. **Inteligência Artificial.** 3. ed. Rio de Janeiro: Editora LTC, 2013.

[2] KINLAW, C. Ryan; DUNLAP, Linda L.; D'ANGELO, Jeffrey A. **Relations between faculty use of online academic resources and student class attendance.** 2012. Disponível em: <www.elsevier.com/locate/compedu>. Acesso em: 14 abr. 2019.
[3] AZOFF, Michael. **Machine Learning in Business Use Cases**. Abril 2015. Disponível em:<http://images.nvidia.com/content/pdf/Ovum_Machine_Learning_in_Business_Use_Cases.pdf>. Acesso em: 24/06/2019.
[4] SILVA, Ivan Nunes da; SPATTI**, Dailo Hernane; FLAUZINO,** Rogério Andrade. Redes Neurais Artificiais: Para Engenharia e Ciências Aplicadas. 2. ed. São Paulo: Artliber, 2016
[5] MAZZA, Leonardo Oliveira. **APLICAÇÃO DE REDES NEURAIS CONVOLUCIONAIS DENSAMENTE CONECTADAS NO PROCESSAMENTO DIGITAL DE IMAGENS PARA REMOC¸AO DE RUÍDO GAUSSIANO**. TCC (Graduação) - Curso de Engenharia Controle e Automação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2017
[6] MOREIRA, Catarina. Neurónio. **Revista de Ciência Elementar**, Lisboa, v. 1, n. 1, p.1-3, out. 2013. Trimestral. -. Disponível em: <https://rce.casadasciencias.org/rceapp/art/2013/008/>. Acesso em: 27 out. 2019.
[7] OLIVEIRA, Patrick Francisco. ANÁLISE DE DESEMPENHO DE UM ALGORITMO DESENVOLVIDO PARA SOLUÇÃO DE DEEP LEARNING UTILIZANDO REDES NEURAIS CONVOLUCIONAIS PARA ANÁLISE DE CONTRASTE DE IMAGENS. **Revista Ubiquidade**, Jundiaí, v. 2, n. 1, p.84-105, jan. 2019. Semestral. Disponível em: <http://www.portal.anchieta.br/revistas-e-livros/ubiquidade/pdf/2019/artigo-ubiquidade-vol2-artigo5.pdf>. Acesso em: 4 nov. 2019.
[8] LEVINE, Robert I.; DRANG, Diane E.; EDELSON, Barry. **INTELIGÊNCIA ARTIFICIAL E SISTEMAS ESPECIALISTAS:** APLICAÇÕES E EXEMPLOS PRÁTICOS. São Paulo: Mcgrowhill, 1988.
[9] RICH, Elaine; KNIGH, Kevin. **Inteligência Artificial.** 2. ed. São Paulo: Mcgrowhill, 1995.
[10] AGUIAR, Gabriel Jonas; ZARPELÃO, Bruno Bogaz; BARBON JUNIOR, Sylvio. **Melhoria de Contraste em Imagens Digitais baseado em Inteligência Artificial.** Londrina, 2017.
[11] FERREIRA, Alessandro dos Santos. **Redes Neurais Convolucionais Profundas na Detecção de Plantas Daninhas em Lavoura de Soja.** 2017. 80 f. Dissertação (Mestrado) - Curso de Pós Graduação em Ciências da Computação, Universidade Federal de Mato Grosso do Sul, Campo Grande, 2017.
[12] LIU, Danqing. **A Practical Guide to ReLU:** Start using and understanding ReLU without BS or fancy equations. 2017. Disponível em: <https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7>. Acesso em: 23 nov. 2019.
[13] PALMIERE, Sérgio Eduardo. **Introdução a Redes Neurais Artificiais.** 2016. Disponível em: <https://www.embarcados.com.br/redes-neurais-artificiais-introducao/>. Acesso em: 23 nov. 2019. (site das funções)
[14] BRIXIUS, Nathan. **The Logit and Sigmoid Functions.** 2016. Disponível em: <https://nathanbrixius.wordpress.com/2016/06/04/functions-i-have-known-logit-and-sigmoid/>. Acesso em: 23 nov. 2019.
[15] RIBEIRO, Amanda Gonçalves. **Função Linear.** Disponível em: <https://mundoeducacao.bol.uol.com.br/matematica/funcao-linear.htm>. Acesso em: 24 nov. 2019.
[16] KINSLEY, Harrison. **How to use your trained model:** Deep Learning basics with Python, TensorFlow and Keras p.6. 2018. Disponível em:

<https://pythonprogramming.net/using-trained-model-deep-learning-python-tensorflow-keras/>. Acesso em: 24 nov. 2019.
[17] MICROSOFT. **Dogs vs Cats dataset from Microsoft.** Disponível em: <https://www.microsoft.com/en-us/download/confirmation.aspx?id=54765>. Acesso em: 24 nov. 2019.
[18] WANG, Pu et al. Rectified-Linear-Unit-Based Deep Learning for Biomedical Multi-label Data. **Interdisciplinary Sciences**: Computational Life Sciences, [s.l.], v. 9, n. 3, p.419-422, 11 nov. 2016. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s12539-016-0196-1

[19] SOARES, Vinicius. **"Nobel da Computação" vai para os pais do Deep Learning**. Disponível em: <https://www.institutodeengenharia.org.br/site/2019/04/01/nobel-da-computacao%E2%80%8B-vai-para-os-pais-do-deep-learning/>. Acesso em: 8 dez. 2019.