ANÁLISE DE DESEMPENHO DE UM ALGORITMO DESENVOLVIDO PARA SOLUÇÃO DE DEEP LEARNING UTILIZANDO REDES NEURAIS CONVOLUCIONAIS PARA ANÁLISE DE CONTRASTE DE IMAGENS

PERFORMANCE ANALYSIS OF AN ALGORITHM DEVELOPED FOR DEEP LEARNING SOLUTION USING CONVOLUCIONAIS NEURAL NETWORKS FOR IMAGE CONTRAST ANALYSIS

Patrick Francisco OLIVEIRA

patrickoliveira58@gmail.com
Sistemas de informação, Centro Universitário Padre Anchieta

Carlos Eduardo CÂMARA

ccamara@anchieta.br
dinhocamara@gmail.com
Sistemas de Informação, Ciência da Computação, Centro Universitário Padre Anchieta

Resumo

O aprendizado profundo (*deep learning*) pertence a um tipo de aprendizado de máquina que utiliza as redes neurais artificiais para a resolução de problemas complexos como o reconhecimento de imagens. A rede neural convolucional (*RNC*) é uma arquitetura baseada em redes neurais artificiais, e vem sendo muito aplicada para o aprendizado profundo, trazendo bons resultados no processamento e análise de imagens. Nesse artigo será apresentado conceitos preliminares para o entendimento das *RNCs*, analisando as técnicas para a melhora do contraste das imagens, mostrando o processo de aprendizagem e seus resultados.

Palavras-Chave

Aprendizado profundo. Rede neural convolucional. Contraste.

Abstract

Deep learning belongs to a type of machine learning that uses artificial neural networks for solving complex problems such as the recognition of images. Convolutional neural network (*RNC*) is an architecture based on artificial neural networks, and has been much applied for deep learning, bringing good results in the processing and analysis of images. This article will be presented preliminary concepts for the understanding of the *RNCs*, analyzing techniques for the improvement of the contrast of images, showing the learning process and its results.

Keywords

Deep learning. Convolutional neural network. Contrast.

INTRODUÇÃO

A capacidade dos computadores em realizarem tarefas vem crescendo cada vez mais com o passar dos anos e através disso estão surgindo novas possibilidades. Os estudos voltados ao aprendizado de maquinas é algo recente comparado a outros campos, tendo início após a segunda guerra mundial, com o objetivo de entender como nós seres humanos pensamos e, indo mais além, construindo algoritmos utilizando mecanismos que permitem emular a forma de aprender biologicamente, a Inteligência Artificial (IA) que pode executar as nossas atividades de forma ágil e eficiente.

Diversas áreas estão envolvidas para se desenvolver a IA, disciplinas de filosofia que auxiliam a entender de onde vem o conhecimento e como ele conduz a ação, a psicologia que mostra como nós seres humanos pensamos e agimos, a matemática que é uma das bases do desenvolvimento e que determina o que pode ser computado. Quais regras e formulas determinam os modelos que serão utilizados para fazer os cálculos e as probabilidades. A biologia que mostra como o cérebro funciona, como é feito o processamento das informações, entre outras que contribuem para a construção da inteligência artificial. (VON ZUBEN, 2015)

A Inteligência Artificial, hoje, abrange diversos segmentos onde podem ser aplicadas, sendo eles, geralmente, de grande escala, como os que são usados na área da saúde, automação, ou para uma simples tarefa como um jogo de xadrez.

Com a grande massa de fotos capturadas por câmeras de smartphones, de variadas qualidades, é possível notar que algumas imagens não possuem iluminação necessária, o contraste, para que a visualização do conteúdo presente na foto possa ser identificada da melhor forma.

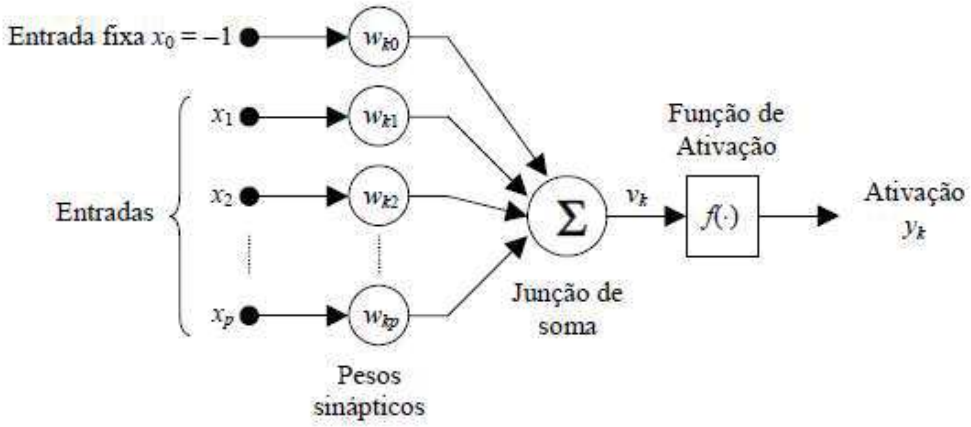
Para se ter uma ideia, o *Google Fotos*, aplicativo de galeria de fotos da google, que possui mais de 500 mil usuários, tem a visualização de 5 bilhões de fotos por dia. Com essa grande quantidade de imagens se torna possível a utilização de IA para poder avaliar essas fotos e realizar algumas ações em cima delas.

Em volta dessa questão, nesse projeto será feito a utilização de deep learning para a implementação de redes neurais convolucionais com o objetivo de melhorar o contraste das fotos, corrigindo a iluminação daquela imagem que foi capturada em condições de pouca iluminação.

REDES NEURAIS

O estudo de redes neurais artificiais começou a partir do interesse de como o cérebro humano realiza diversas atividades complexas, com alto grau de agilidade, tendo como base que, as funções neurais biológicas são armazenadas nos neurônios e nas conexões entre eles. O processo de aprendizagem se dá com a geração de novas conexões entre esses neurônios e a alteração nas conexões existentes. A partir desse conceito foi construído o conceito, um modelo matemático denominado ***redes neurais artificias*** que são um conjunto de neurônios simples agrupados. (HAYKIN, 2009)

Figura 1. Modelo de neurônio artificial.



A figura 1, mostra o neurônio artificial que é a unidade básica de uma rede neural, definido por três elementos básicos:

- **Conjunto de sinapses:** ou links de conexões, caracterizadas por um peso (w_{ki}) ou a sua intensidade, que são definidas como um peso da entrada naquele ramo. Um sinal de entrada x_j conectado ao neurônio k, é multiplicado pelo peso da sinapse w_{kj}. A entrada fixa, x_0, é um parâmetro externo do neurônio em k, que determina uma tendência.
- **Somatório:** ou junção. Realiza a adição do resultado da multiplicação dos sinais de entrada x_i pelas sinapses, w_{kj}, do neurônio. Essa operação constitui um combinador linear.
- **Função de ativação:** φ(.) (f(.)define, ou limita, a amplitude do sinal de saída a um valor finito. Também é definida como uma função limitadora da saída v_k do neurônio. (VARGAS et. al., 2016)

A rede neural é construída através de camadas, que é o agrupamento dos neurônios. Essas camadas são agrupadas, definindo assim a profundidade da rede, de modo que, dependendo de como é feita a organização das camadas, a rede se comportará de forma diferente, podendo apresentar diversos tipos de arquiteturas.

Redes neurais convolucionais

As redes neurais convolucionais são redes artificiais que possuem em uma de suas camadas a operação de convolução. A dificuldade para se treinar uma rede neural de forma eficiente era visível e por isso, outras arquiteturas de redes neurais foram modeladas. Uma dessas redes que apresentou uma maior facilidade para ser treinada comparado a redes completamente conectadas foi a ***Rede Neural Convolucional*** **(RNC).** (FERREIRA. 2017)

As redes neurais convolucionais são muito utilizadas quando existe uma grande quantidade de amostras rotuladas. Algumas vantagens da *RNC*:

- Possuem a capacidade de extrair características relevantes através de aprendizado de transformações
- Depende de um número menor de parâmetros de ajustes do que redes totalmente conectadas com o mesmo número de camadas ocultas.

As *RNCs* podem ser utilizadas na classificação de imagens através do conhecimento sintetizado via *RNC* ou para reconhecer objetos, pessoas entre outros. (FERREIRA. 2017)

A seguir será destacado alguns fundamentos básicos e os componentes de uma rede neural convolucional.

Lenet

Proposta por *Yann LeCun*, foi um dos primeiros projetos de redes neurais convolucionais, utilizado inicialmente para reconhecimento de caracteres manuscritos, tendo surgido propostas para melhoramento dessa rede.

As *RNCs* são formadas por sequências de camadas e cada uma destas possui uma função específica na programação do sinal de entrada. A arquitetura *LeNet* possui três camadas principais: a de *convolução*, de *pooling* e *totalmente conectadas*, sendo que a camada (MAZZA, 2016):

- **Convolucional:** é responsável por extrair atributos dos dados de entrada.
- **Pooling*:*** são responsáveis por reduzir a dimensionalidade dos dados resultantes da camada convolucional.
- **Totalmente conectadas:** são responsáveis pela programação do sinal por meio da multiplicação ponto a ponto e o uso de uma função de ativação (x_0 na figura 1).

O resultado da análise da rede neural convolucional será a probabilidade da imagem pertencer a uma determinada classe para qual ela foi treinada. Abaixo será explicado cada uma das camadas.

Camadas Convolucionais

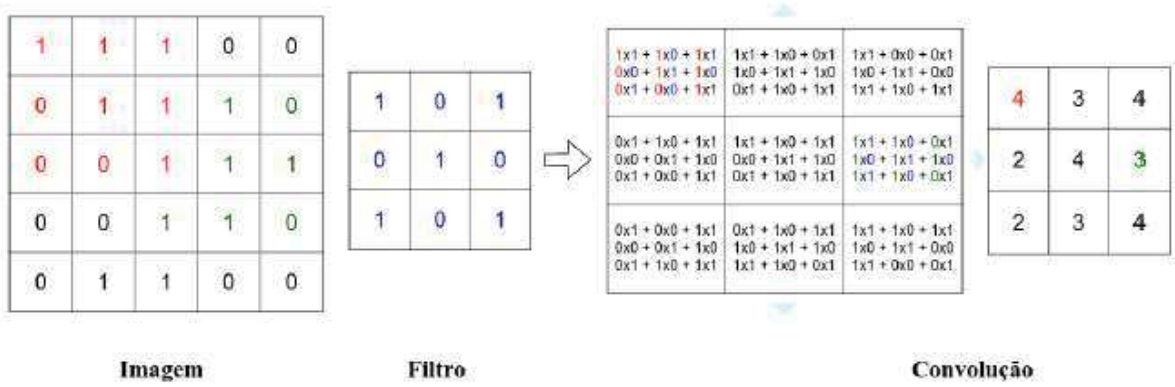
A convolução é uma operação matemática entre duas funções f e g, gerando uma terceira função, que seria uma modificação de f. A expressão que define a convolução, tendo as funções f e g, com a variável continua x e tendo o símbolo * como operador de convolução, é dada por (FERREIRA. 2017):

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\tau) \cdot g(x - \tau) d\tau$$

No processamento de imagem, apresentado na figura 3, onde a imagem é definida como uma função bidimensional, a convolução funciona muito bem para detectar as bordas, suavização de imagem, entre outras aplicações. Nas imagens a convolução funciona da seguinte forma: o somatório da multiplicação de cada elemento da imagem,

com seus vizinhos locais, pelos elementos da matriz que representam o filtro de convolução.

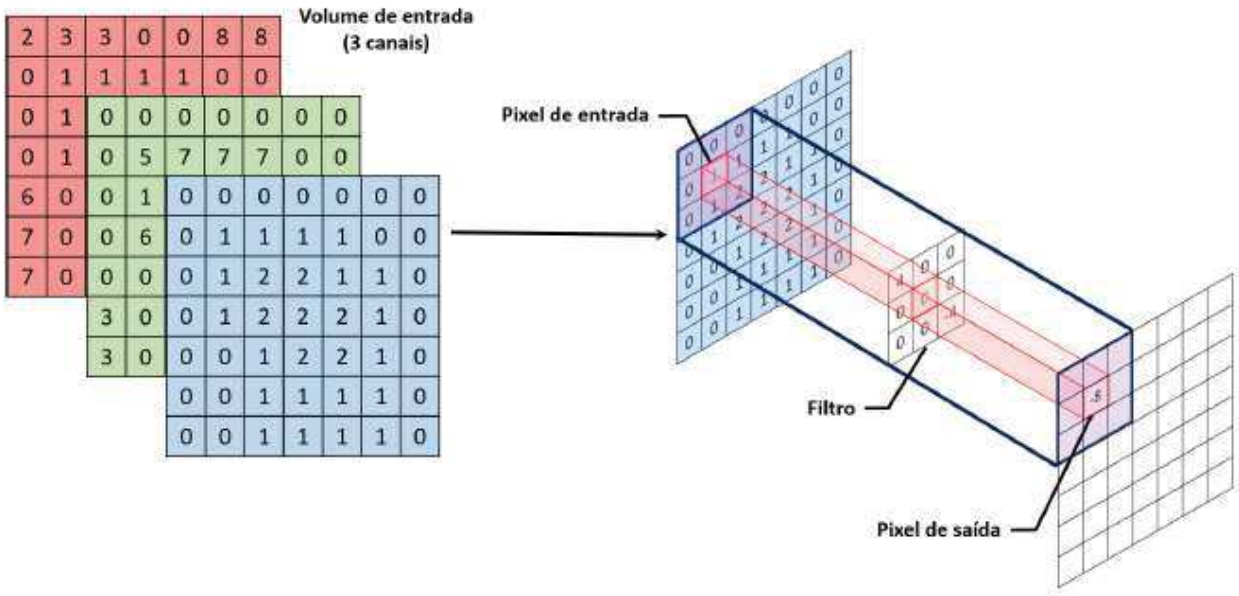
Figura 2: Exemplo de convolução em uma imagem. (FERREIRA. 2017)



Essa camada possui um conjunto de filtros que recebem dados de entrada, chamados de volume. Cada filtro tem dimensão reduzida, mas pode se estender. Se a imagem for colorida ela possui 3 canais e o filtro da primeira camada convolucional terá o tamanho de 5 pixels de altura, 5 de largura, 3 de profundidade e durante o processo de treinamento a dimensão do filtro vai se ajustar de acordo com as características mais importantes dos dados de entrada.

Cada filtro dá origem a uma estrutura conectada que percorre toda a extensão dos dados de entrada. A convolução é feita pela somatória do produto ponto a ponto entre os valores de um filtro e cada posição do volume de entrada. Os resultados passam por uma função de ativação chamada de *Rectified Linear Units*.

Figura 3: Demonstração da convolução entre um filtro 3x3 e o volume de entrada. (ARAÚJO et. al., 2017)

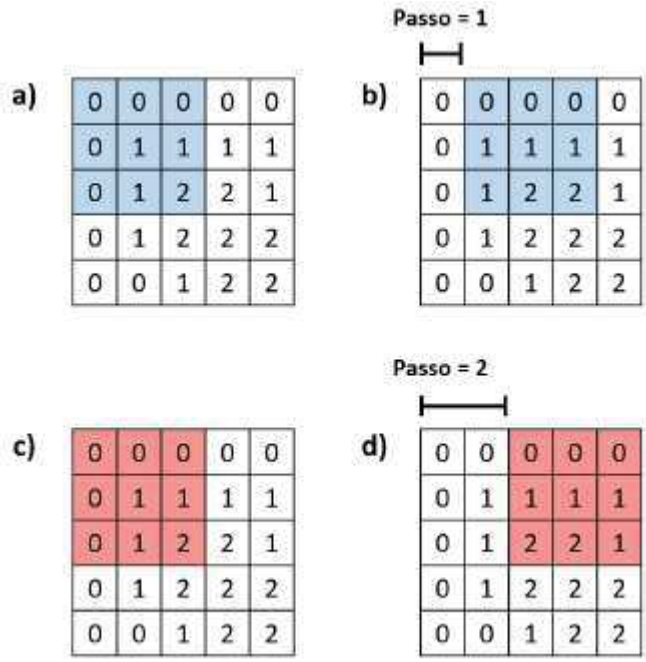


Tendo três parâmetros que controlam o tamanho dos resultados da camada convolucional (ARAÚJO et. al., 2017):

- **Profundidade**: depende do número de filtros utilizados.
- **Passo**: tamanho do salto na operação de convolução.
- **Zero-Padding**: preenche a borda do volume de entrada.

Cada um desses filtros será responsável por extrair as características diferentes dos dados de entrada. Quanto o maior número de filtros, maior será o número de características extraídas dos dados de entrada, porém a complexidade computacional, relativa ao tempo e ao uso de memória será maior.

Figura 4: Ilustração de como o passo influencia o deslocamento de um filtro 3x3 em duas etapas sucessivas da convolução.

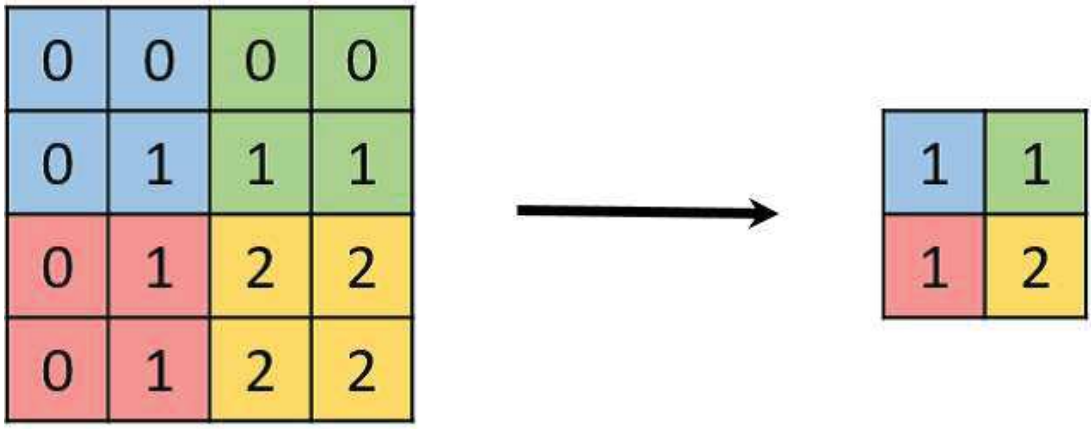


Camada Pooling

Após a camada convolucional existe uma camada de *Pooling*. Esta camada tem como objetivo, reduzir a dimensão espacial do volume de entrada, reduzindo o custo computacional da rede.

A camada *Pooling* (figura 5), recebe os valores de uma parte da região do mapa de atributos, que foram gerados pela camada convolucional, e os substitui por alguma métrica dessa região. A operação *max pooling*, onde é feita a substituição dos valores pelos valores máximos é muito utilizada. Essa operação realiza a eliminação de valores baixos, reduzindo assim a dimensão da representação dos dados, acelerando a computação necessária para as próximas camadas. (ARAÚJO et. al., 2017).

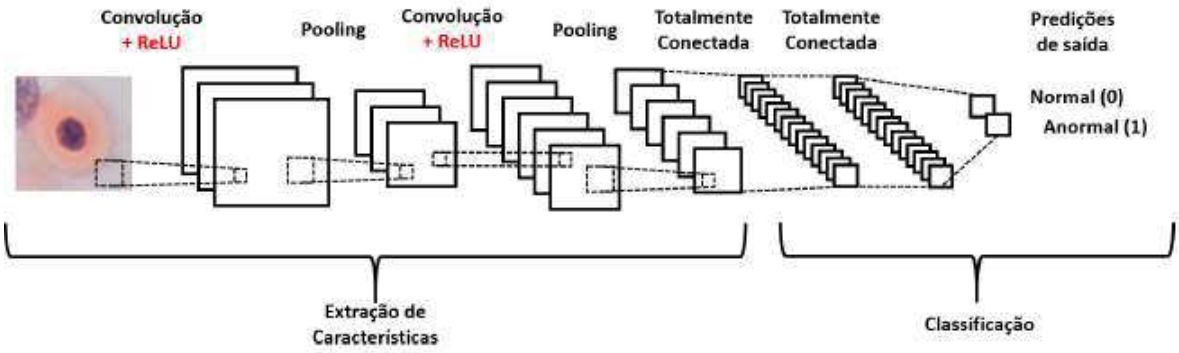
Figura 5: Aplicação de *max pooling* em uma imagem 4x4 utilizando um filtro 2x2.



Camada Totalmente Conectada

Com os resultados das camadas convolucionais e de *Pooling* que irão trazer as características da imagem de entrada, a camada totalmente conectada realiza a classificação da imagem utilizando uma classe pré-determinada para avaliar essas características, como demonstrado na figura 6. (ARAÚJO et. al., 2017)

Figura 6: Demonstração da extração das características de uma imagem pela RNC e a classificação baseadas nessas informações.



A camada é formada por unidades de processamento e essas unidades estão conectadas com a camada anterior e a posterior, por isso o nome “totalmente conectadas”.

É utilizada uma função de ativação chamada de *softmax*. Esta função, recebe um vetor como input e realiza o processamento desse valor analisando a probabilidade de ele pertencer a uma das classes na qual a rede foi treinada.

Outra função muito utilizada é a dropout que reduz o tempo de treinamento, sendo que, ela remove, aleatoriamente a cada iteração de treinamento uma porcentagem das unidades de processamento de uma determinada camada e as recoloca na próxima

camada, evitando overfitting que seria o ajuste excessivo aos dados de treinamento, reduzindo a taxa de acerto. (ARAÚJO et. al., 2017)

PROCESSAMENTO DE IMAGEM

O processamento de imagem consiste em realizar operações matemáticas nos dados, procurando melhorar os aspectos espectrais e espaciais da melhor forma possível, atendendo uma determinada necessidade de uma aplicação. Podem ser separados em dois tipos de técnicas de processamento, sendo elas: de realce e de classificação. Essas técnicas são utilizadas após o pré-processamento, tendo como objetivo corrigir os erros que foram encontrados nos dados processados. (MENESES & ALMEIDA, 2012)

Existem diversas causas para os erros encontrados nas imagens digitais, podendo ser um problema de sensor, que com o passar do tempo acaba perdendo sua precisão ou até o equipamento que compõem o sensor. Outras causas que não envolve a ferramenta utilizada para captura, mas o ambiente que será capturado é a *atmosfera.* Ela pode gerar diversas modificações na imagem, ocorrendo distorções na escala, imprecisão na posição espacial dos pixels, detrimento no contraste ou armazenar incorretamente os valores dos pixels. Ao extrair os dados de uma determinada imagem, na maioria das vezes, é preciso realizar correções para compensar os erros que ela traz, obtendo dados mais precisos. Algumas dessas correções são realizadas em uma das etapas do processamento chamada pré-processamento. Visando corrigir certas imperfeições da imagem, sendo que esses erros são identificados a partir da causa da distorção. Essa etapa é muito importante para o decorrer do processamento, pois, auxilia as próximas fases do processamento, onde são aplicadas as técnicas de transformação, fazendo assim com que as modificações não realçassem as falhas da imagem. (MENESES & ALMEIDA, 2012) (MARTINS, 2017)

Os algoritmos tratam a imagem de forma matemática, com dados diretamente relacionados ao processo físico, porém a representação desse processamento não é totalmente fiel a do mundo real, tendo que ser analisada e interpretada pelo usuário para um melhor entendimento dos dados. Não possuindo um fluxo fixo de processamento para que o usuário siga, dando assim, diversas possiblidades na hora da extração dos dados, atendendo os requisitos necessários naquela situação e que não necessariamente irá atender outras áreas.

Contraste

O contraste é uma medida qualitativa e que está relacionada com a distribuição dos tons de cinza de uma imagem. Esses tons de cinza, às vezes, podem ser difíceis de serem percebidos, pois em determinadas imagens a variação de pixels cinzas possuem uma pequena diferença, sendo complicado perceber os detalhes da textura e do espectro. (MARTINS, 2017)

Por isso existem algumas maneiras básicas para se realçar o contraste da imagem, analisando o ponto de vista espectral e espacial da imagem. Utilizando o espectro da imagem como base é possível realçar o contraste elevando o valor dos pixels em relação ao seu vizinho, desconsiderando o valor que esse pixel vizinho possui.

Baseando-se no espaço, os realces são feitos levando em consideração o valor do pixel vizinho, dependendo assim da distribuição espacial dos pixels que estão próximos ao pixel (ou a uma região de pixels) no qual será feito o reajuste do valor do contraste. Essas técnicas utilizadas para melhorar o realce do contraste das imagens são consideradas transformações de espaço, pois, realizam alterações nos espaços espectrais e espaciais da imagem. Uma das principais técnicas para transformar o espaço da imagem, é o realce de contraste por expansão histográfica (LAKSHMI et. al., 2016).

DESENVOLVIMENTO

O desenvolvimento da aplicação que está sendo realizada neste trabalho foi baseado no trabalho de Mazza (MAZZA, 2016): "Aplicação de redes neurais convolucionais densamente conectadas no processamento digital de imagens para remoção de ruído gaussiano", onde emprestamos este programa para fazer o estudo do comportamento de uma simulação usando *Deep Learning* para o tratamento de imagens, alternado o contraste. Descrevemos a seguir como é montado um programa em python usando as bibliotecas Keras e TensorFlow.

Tensor Flow

O *tensorflow* é uma biblioteca de código aberto utilizada para computação numérica e aprendizado de máquina, disponibilizado pela Google em novembro de 2015. Atualmente uma das bibliotecas mais utilizadas para o aprendizado profundo, ela reúne uma variedade de modelos e algoritmos de *machine learning* (aprendizado de máquina) e *deep learning* (aprendizado profundo). (HOPE et. al., 2017)

Com esta biblioteca é possível treinar e executar redes neurais profundas em diversas aplicações como a classificação de dígitos manuscritos (*Le Cun*), reconhecimento de imagens, processamento de linguagem natural. Possuindo portabilidade, permitindo que os cálculos e gráficos gerados possam ser executados em uma grande variedade de ambientes e plataformas de hardware, tendo a possibilidade, por exemplo de, com o código idêntico, utilizando a mesma rede neural, realizar o treinamento na nuvem distribuindo em um cluster de várias maquinas ou em um único notebook ().

O núcleo do *tensorflow* é construído em C++, tendo duas interfaces principais de alto nível, com indicadores e interfaces que expressam e executam os gráficos. Sendo desenvolvido o *front-end* em grande parte na linguagem *python*, utilizado pela maioria dos desenvolvedores e cientista de dados. Outro ponto importante é a flexibilidade, que permite expressar os modelos com facilidade, substituindo os blocos da rede por outros, analisando os resultados e projetando novos blocos.

Keras

O *keras* é uma API de redes neurais de alto nível, desenvolvida utilizando a linguagem *python*, podendo ser executado junto ao *tensorflow*. Desenvolvido para se ter uma rápida realização dos testes, possibilitando a criação de protótipos de forma fácil e rápida, tendo suporte a redes neurais convolucionais e recorrentes, podendo ser executado utilizando a CPU e GPU em conjunto ou somente a GPU do computador.

Oferece APIs consistentes e simples, que facilitam no processo de desenvolvimento, dando um *feedback* claro sobre os erros recorrentes. Possuindo modularidade, tal que as camadas neuronais, otimizadores, funções de ativação, entre outros, são módulos independentes que podem ser combinados para a criação de novos módulos ou de novos modelos. A facilidade para adicionar novos módulos, e a grande quantidade de exemplos dos módulos existentes, auxiliam no desenvolvimento. Utilizando-se da linguagem *python* para descrever os modelos, tornando-os mais fáceis para depurar, além de auxiliar na extensibilidade. ()

Criação da base de dados

Para a criação da base de dados (*dataset*) é necessário realizar algumas importações de determinadas bibliotecas (fig 7). A biblioteca *numpy*, que permite a criação de arranjos, vetores e matrizes com *n* dimensões. A biblioteca *matplotlib* utilizada para a geração de gráficos 2D a partir de arranjos, que irão exibir os resultados do treinamento da rede neural. O modulo *os* do próprio python, que traz as informações sobre o sistema operacional e serão utilizados para trabalhar com o sistema de arquivos, localizando os diretórios onde estão localizadas as imagens da base de dados. O modulo *cv2*, da biblioteca OpenCV, que possibilita a manipulação de imagens e vídeos. A biblioteca *tqdm*, que possibilita a criação de uma barra de progressão, utilizada para a melhor visualização e verificação do processo de criação da base de dados.

Figura 7: Importações das bibliotecas que serão utilizadas na criação do *dataset.*

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

Após as importações é definido o diretório onde se encontra as imagens para a construção da base de dados, além da categoria que os dados estão divididos (fig 8). Em seguida é criada uma função que possui uma estrutura de repetição que irá realizar uma varredura no diretório em que será feita a criação do caminho das categorias e a captura do índice dessas categorias. A outra estrutura de repetição contém uma variável que será

responsável por converter o vetor com os dados da imagem, e outra variável para redimensionar a imagem para padronizar o tamanho. Depois de realizar a tratativa dos dados, eles são inseridos dentro do vetor e a base de dados é criada.

Figura 8: Função de construção da base de dados.

```
DIRETORIODATA = "C:/Users/Patrick/Desktop/TCC/Base de Dados/basededados"
CATEGORIA = ["test", "train"]


dados_treinamento = []


def criacao_base_de_dados():
    for category in CATEGORIA:
        path = os.path.join(DIRETORIODATA,category)
        class_num = CATEGORIA.index(category)


        for img in tqdm(os.listdir(path)):
            try:
                img_array = cv2.imread(os.path.join(path,img)
,cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                dados_treinamento.append([new_array, class_num])
            except Exception as e:
                pass


criacao_base_de_dados()


print(len(dados_treinamento))
```

Depois da criação da base de dados é necessário embaralhar os dados, para evitar que predição dos dados seja previsível. Em seguida é feita a criação do modelo neural, utilizando o modulo *pickle*, que permite realizar a serialização do objeto, transformando-os em uma sequencias de bytes. O método *dump* é responsável por capturar os dados do objeto para a geração de uma sequência de bytes que representem esses dados, possibilitando a transmissão pela rede.

Figura 9: Embaralhamento dos dados e criação do modelo.

```
import random


random.shuffle(dados_treinamento)
for sample in dados_treinamento[:10]:
      print(sample[1])


X=[]
y=[]


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


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


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)
```

Criação do modelo de rede neural convolucional

Primeiramente é necessário realizar algumas importações de bibliotecas que serão utilizadas na criação da rede neural convolucional. A biblioteca do Keras possui funções que auxiliam na construção da rede neural convolucional, além de disponibilizar alguns *datasets* para servir como base para o treinamento. O modelo *Sequential* é um modelo pré-construído do keras (fig 10), onde serão adicionadas as camadas da rede neural convolucional que são: a Camada de convolução (*Conv2D*), e a Camada de Pooling (*MaxPolling2D*). Também é feita a importação da camada *Dense* usada para predição dos rótulos, a camada de *Dropout* que realiza a redução do *overfitting* (ajuste excessivo aos

dados de treinamento, reduzindo a taxa de acerto), e a camada *Flatten* que tem a função de expandir o vetor tridimensional em um vetor unidimensional.

Figura 10: Importações das bibliotecas para a construção da rede neural.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
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
```

A variável *batch size* indica o número total de exemplos de treinamento presentes em um único lote, a *num_classes* detém o número de classes que será utilizado do *dataset* (base de dados) e a variável *epochs* refere-se a quantas vezes os dados de entrada serão usados para atualizar os pesos durante o treinamento do modelo. Quando se tem uma época (*epoch*), o modelo vai utilizar o valor da entrada somente uma única vez. Com o aumento desse valor, os pesos serão atualizados ao término de cada época, gerando melhores resultados. As variáveis *img rows* e *imgcols* correspondem a largura e altura da imagem respectivamente (fig 11).

Figura 11: Declaração das variáveis para utilização no modelo.

```
batch_size = 128
num_classes = 10
epochs = 6
img_rows, img_cols = 28, 28
```

Para o treinamento foi utilizado dois *datasets* de tamanhos diferentes, sendo que o primeiro é composto por 180 imagens no total, sendo 150 imagens com contraste distorcido a partir de 30 imagens originais que serviram como base para as respectivas distorções. Esse *dataset* foi criado utilizando a base de dados de imagens CSIQ, famosa para a realização de testes de algoritmos de avaliação da qualidade de imagem e outros aspectos relacionados a qualidade de imagem. (LARSON & CHANDLER, 2010)

O segundo *dataset* é composto por 923 imagens, utilizando 800 imagens de aviões e 123 de carros, utilizando a base de imagens Caltech 101, que é composta por 101 categorias de objetos, sendo que cada categoria tem por volta de 40 a 800 imagens.

Cada um dos *datasets* foram preparados para terem imagens com tamanhos e cores iguais. O tamanho foi fixado em 150 de largura e 150 de altura e as imagens receberam um efeito de *grayscale*, transformando-as em uma escala de cinza. Após as importações, para facilitar o entendimento e uma melhor visualização dos resultados do treinamento foi utilizado a ferramenta do tensorflow chamada TensorBoard. Para utilizar essa ferramenta é necessário importar uma função da biblioteca keras chamada de *callbacks*, um conjunto de funções que são aplicados durante os estágios do treinamento para obter uma visão dos estados internos e estatísticos do modelo.

Figura 12: Carregando a base de dados.

```
NOME = "RNC-Model-ContrastDataset-{}".format(int(time.time()))

tensorboard = TensorBoard(log_dir='logs/{}'.format(NOME))


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
```

É construído um modelo sequencial (*Sequential*), (fig 16) e, adicionado a ele, as camadas de rede neural convolucional, sendo elas a de convolução (*Conv2D*) e de pooling (*MaxPooling*). Também é adicionado no modelo a camada de *dropout* que irá desativar alguns neurônios da rede aleatoriamente, forçando a encontrar novos caminhos, reduzindo assim o sobre ajuste (*overfitting*), e no final é adicionado a camada *dense* usada para prever a classe apresentado na Figura 13.

Figura 13: Construção da rede neural convolucional.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(28,28,1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

Por fim o modelo é compilado, utilizando uma função de perda, um otimizador e a métrica de acerto. Em seguida é feito um ajuste no conjunto de dados usando a função *fit*, treinando o modelo por um número determinado épocas. Ao final do treinamento é feita a avaliação de perda e de acerto do modelo e, então, os valores são exibidos.

Figura 14: Ajuste dos dados e exibição dos resultados.

```
model.compile(loss=keras.losses.categorical_crossentropy,optimizer=k
eras.optimizers.Adadelta(),metrics=['accuracy'])
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
  validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

TESTES E RESULTADOS

Os testes foram realizados com uma variação na estrutura do modelo, alterando a quantidade de camadas de convolução e profundidade e o número de nós.

Para o treinamento da *RNC* foram utilizados os dois *datasets* criados, utilizando o período de 30 épocas e posteriormente de 50 épocas para analisar a curva de aprendizado da rede nessa progressão de treino. Sendo utilizados para o primeiro *dataset* 125 exemplos para treinamento e 55 para validar e o segundo *dataset* 646 exemplos para testes e 277 para validar. Com o primeiro *dataset* foram obtidos melhores resultados utilizando o modelo com 1 camada de convolução, 64 nós e 1 camada de profundidade. Os modelos com duas camadas de convolução tiveram resultados apresentados na Tabela 1, concluindo que neste caso, o aumento de camadas convolucionais não necessariamente trará melhores resultados na aprendizagem. Com o aumento do número de épocas, os resultados melhoraram, tendo uma acurácia de 1.0 em todos os modelos e uma consequente diminuição na perda.

Figura 15: Gráfico do modelo de treinamento com melhor resultado, que demonstra a progressão do treinamento, tendo no eixo x o número de épocas e no eixo y os valores que vão de 0.00 a 1.00.

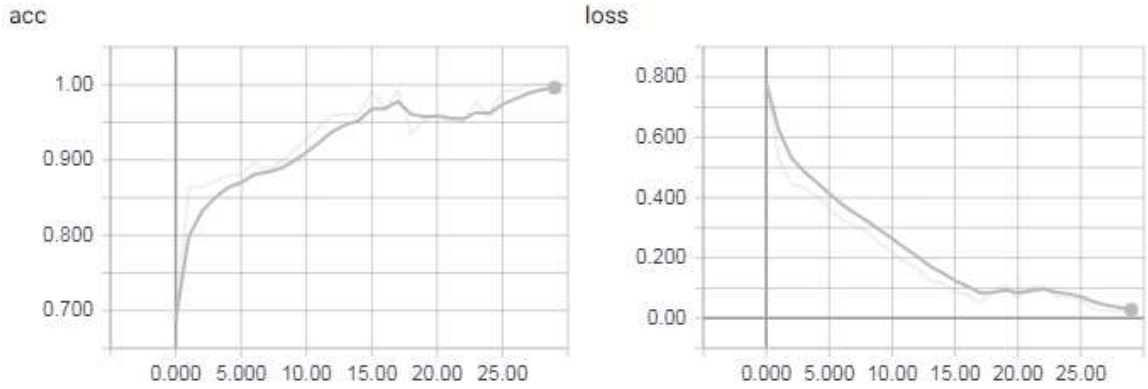


Tabela 1: Resultados de acurácia e perda no aprendizado utilizando o primeiro *dataset*, com o período de 30 épocas.

Modelo de treinamento	***Acurácia***	***Perda***	***Tempo***
Conv=1 | Nós=32 | Prof=0	0.96	0.09	6m 46s
Conv=1 | Nós=64 | Prof=0	1.0	0.03	8m 51s
Conv=1 | Nós=32 | Prof=1	1.0	0.04	9m 32s
Conv=1 | Nós=64 | Prof=1	1.0	0.01	12m 29s
Conv=2 | Nós=32 | Prof=0	0.93	0.16	6m 20s

Conv=2 | Nós=64 | Prof=0	0.95	0.11	8m 15s
Conv=2 | Nós=32 | Prof=1	0.88	0.25	9m 58s
Conv=2 | Nós=64 | Prof=1	0.97	0.06	12m 43s

Figura 16: Um dos gráficos do modelo de treinamento com melhor resultado, que demonstra a progressão do treinamento, tendo no eixo x o número de épocas e no eixo y os valores que vão de 0.00 a 1.00.

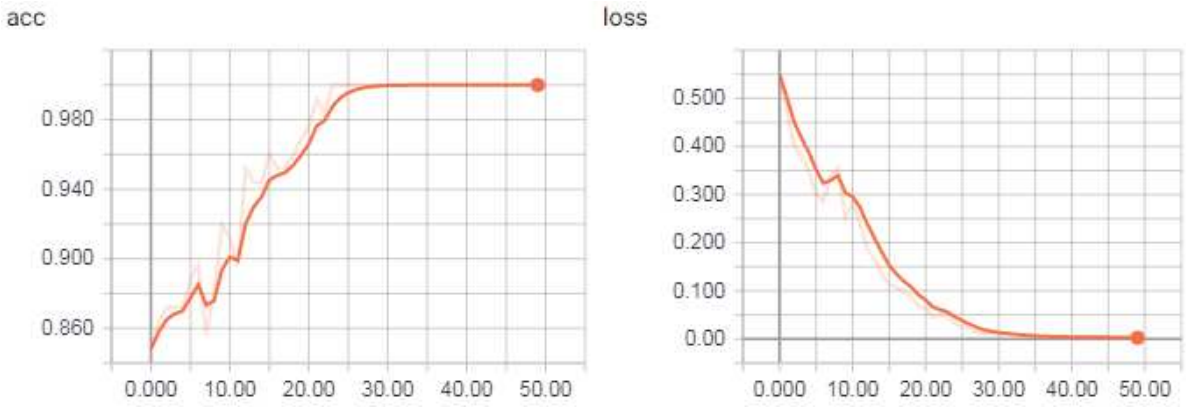


Tabela 2: Resultados de acurácia e perda no aprendizado utilizando o primeiro *dataset*, com o período de 50 épocas.

Modelo de treinamento	***Acurácia***	***Perda***	***Tempo***
Conv=1 | Nós=32 | Prof=0	1.0	0.03	9m 34s
Conv=1 | Nós=64 | Prof=0	1.0	0.01	13m 7s
Conv=1 | Nós=32 | Prof=1	1.0	0.00	13m 48s
Conv=1 | Nós=64 | Prof=1	1.0	0.00	18m 52s
Conv=2 | Nós=32 | Prof=0	1.0	0.04	11m16s
Conv=2 | Nós=64 | Prof=0	1.0	0.01	14m26s
Conv=2 | Nós=32 | Prof=1	1.0	0.01	14m 58s
Conv=2 | Nós=64 | Prof=1	1.0	0.00	18m 38s

Com o segundo *dataset,* apresentado nas Tabelas 3 e 4, os resultados tiveram uma média tanto da acurácia quanto da perda, se mantendo entre 1.0 e 0.0 respectivamente, tendo uma curva de aprendizado ao passar das épocas crescendo e estabilizando, da mesma forma a perda que diminuiu nas primeiras épocas e estabilizou. Isso se repetiu com o aumento no período de épocas.

Figura 17: Um dos gráficos do modelo de treinamento com melhor resultado, que demonstra a progressão do treinamento, tendo no eixo x o número de épocas e no eixo y os valores que vão de 0.00 a 1.00.

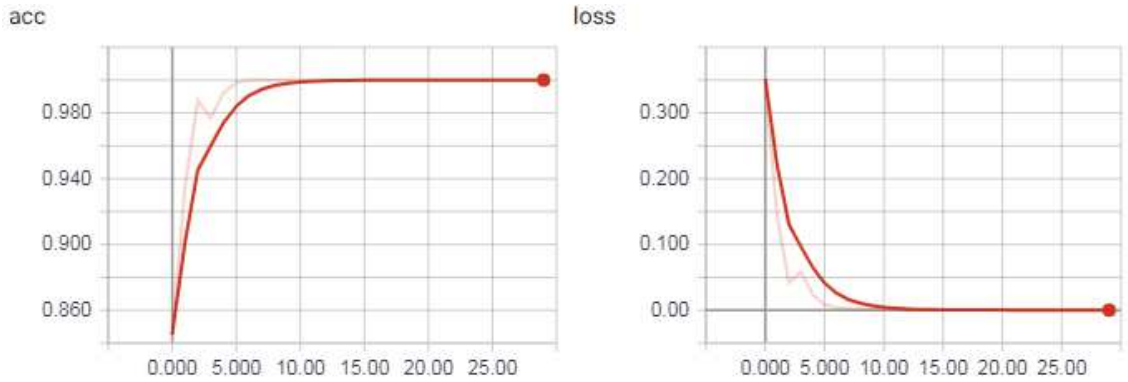


Tabela 3: Resultados de acurácia e perda no aprendizado utilizando o segundo dataset, com o período de 30 épocas.

Modelo de treinamento	***Acurácia***	***Perda***	***Tempo***
Conv=1 | Nós=32 | Prof=0	1.0	0.0	29m38s
Conv=1 | Nós=64 | Prof=0	1.0	0.0	40m 54s
Conv=1 | Nós=32 | Prof=1	1.0	0.0	40m 23s
Conv=1 | Nós=64 | Prof=1	1.0	0.0	59m 8s
Conv=2 | Nós=32 | Prof=0	1.0	0.0	30m 33s
Conv=2 | Nós=64 | Prof=0	1.0	0.0	42m 46s
Conv=2 | Nós=32 | Prof=1	1.0	0.0	1h 4m 27s
Conv=2 | Nós=64 | Prof=1	1.0	0.0	1h 3m 39s

Figura 18: Um dos gráficos do modelo de treinamento com melhor resultado, que demonstra a progressão do treinamento, tendo no eixo x o número de épocas e no eixo y os valores que vão de 0.00 a 1.00.

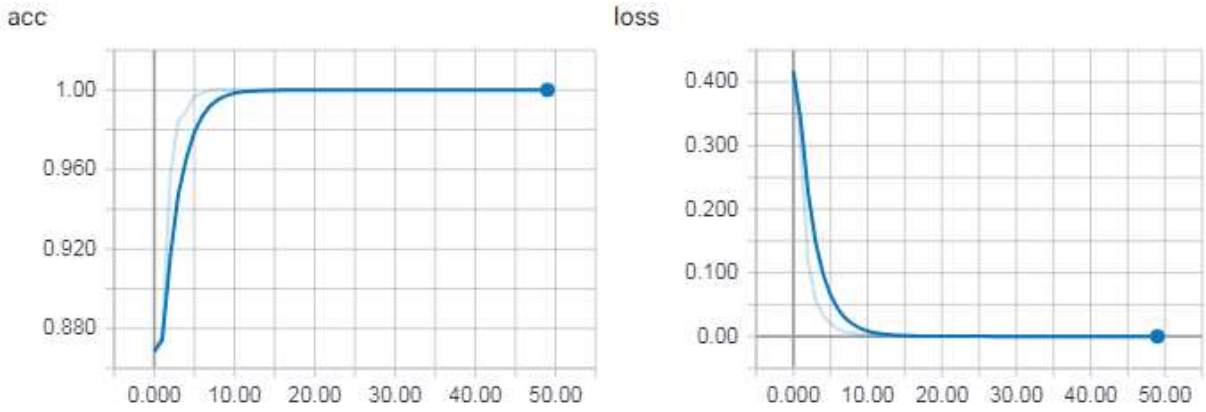


Tabela 4: Resultados de acurácia e perda no aprendizado utilizando o segundo *dataset*, com o período de 50 épocas.

Modelo de treinamento	***Acurácia***	***Perda***	***Tempo***
Conv=1 | Nós=32 | Prof=0	1.0	0.0	54m 19s
Conv=1 | Nós=64 | Prof=0	1.0	0.0	1h 13m 24s
Conv=1 | Nós=32 | Prof=1	1.0	0.0	1h 12m 24s
Conv=1 | Nós=64 | Prof=1	1.0	0.0	1h 27m 45s
Conv=2 | Nós=32 | Prof=0	1.0	0.0	53m 24s
Conv=2 | Nós=64 | Prof=0	1.0	0.0	1h 05m 59s
Conv=2 | Nós=32 | Prof=1	1.0	0.0	1h 31m 03s
Conv=2 | Nós=64 | Prof=1	1.0	0.0	1h 43m 10s

CONCLUSÕES

Neste trabalho foi apresentado o aprendizado profundo utilizando redes neurais convolucionais para o processamento de imagens com variação de contraste. Foram realizados diversos testes com modelos de rede neural convolucional baseado no modelo de rede neural do trabalho do Ferreira (FERREIRA. 2017): "Redes Neurais Convolucionais Profundas na Detecção de Plantas Daninhas em Lavoura de Soja", com uma quantidade de camadas variadas, utilizando dois tipos de *datasets*. O primeiro foi construído com uma base de imagens menor, sendo composto por imagens de contraste distorcido. O segundo *dataset* utilizou uma base de dados maior, com imagens possuindo uma variação no contraste.

Esses modelos foram comparados para identificar o modelo com melhor resultado de aprendizado. Os modelos treinados com o *dataset* 1, treinados em um período com o menor número de épocas, teve resultados melhores com redes que tinham uma quantidade de camadas convolucionais menor, demonstrando que em alguns casos o aumento no número de camadas convolucionais não necessariamente trará uma melhora nos resultados. Com o aumento no período das épocas os resultados foram equivalentes em todos os modelos testados, indicando que quanto maior o período, melhor será o aprendizado.

O treinamento feito utilizando o *dataset* 2 teve resultados que foram semelhantes nos dois períodos de épocas, com modelos com quantidade de camadas diferentes, mostrando que, o aumento da base de dados auxilia numa melhor progressão de aprendizagem da rede neural. Pode se perceber com os resultados de treinamento do *dataset* 2 aconteceu *overfitting*, uma especialização por parte da rede neural, pela baixa quantidade de exemplos de treinamento. Conclui-se que a utilização das redes neurais convolucionais possuem um bom desempenho para análise e processamento de imagens, podendo sugerir como trabalho futuro a extração de característica das imagens, utilizando descritores de imagens, para selecionar os melhores parâmetros gerando assim de forma eficiente e rápida, imagens com uma melhor qualidade.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, Flávio H.D; CARNEIRO, Allan C; SILVA, Romuere R.V; MEDEIROS, Fátima N.S; USHIZIMA, Daniela M; Redes Neurais Convolucionais com Tensorflow: Teoria e Prática. III Escola Regional de Informática do Piauí, 2017.

BEZERRA, Eduardo. Introdução à Aprendizagem Profunda, 2016.

FERREIRA, Alessandro dos Santos. Redes Neurais Convolucionais Profundas na Detecção de Plantas Daninhas em Lavoura de Soja. Campo Grande: UFMS, 2017.

HAYKIN, S. Neural Networks and Learning Machines, Prentice Hall, 2009.

HOPE, TOM; RESHEFF, Y; LIEDER, I; Learning TensorFlow: A guide to building deep learning systems", 2017.

LARSON, E. C.; CHANDLER, D. M. Most apparent distortion: full-reference

Image quality assessment and the role of strategy. *Journal of Electronic Imaging*, International Society for Optics and Photonics, v. 19, n. 1, p. 011006Ű011006, 2010.

LAKSHMI, Hyma T.V; MADHU, T; KAVYA, Sri; DEVI, Geetha. ImageResolution and Contrast Enhancement Using Wavelet Transforms and Contrast Limited Adaptive Histogram Equalization. International Journal of Computer Science and Information Security, 2016.

MAZZA, Leonardo Oliveira. Aplicação de redes neurais convolucionais densamente conectadas no processamento digital de imagens para remoção de ruído gaussiano. Rio de Janeiro, 2016.

MARTINS, Samuel Botter. Introdução ao Processamento Digital de Imagens: Definições Básicas, Espaço de Cores e Histogramas. Universidade Estadual de Campinas.

MENESES, Paulo Roberto; ALMEIDA, Tati. Introdução ao processamento de imagens de sensoriamento remoto. Brasília, 2012.

VARGAS, A.; PAES, A; VASCONCELOS, C; Um Estudo sobre Redes Neurais Convolucionais e sua Aplicação em Detecção de Pedestres. Disponível em <*http://sibgrapi.sid.inpe.br/col/sid.inpe.br/sibgrapi/2016/09.12.15.44/doc/um-estudosobre.pdf*> 2016.

VON ZUBEN, F.J. Síntese automática de redes neurais artificiais com conexões arbitrárias, Universidade Estadual de Campinas, 2015.