UNIVERSIDADE FEDERAL DO TOCANTINS

CAMPUS UNIVERSITÁRIO DE PALMAS

CURSO DE CIÊNCIA DA COMPUTAÇÃO

PEDRO HENRIQUE ALVES DE MOURA

PROTÓTIPO DE UMA REDE CONVOLUCIONAL PARA PROBLEMAS DE CLASSIFICAÇÃO

PALMAS (TO)

2022

PEDRO HENRIQUE ALVES DE MOURA

PROTÓTIPO DE UMA REDE CONVOLUCIONAL PARA PROBLEMAS DE CLASSIFICAÇÃO

Trabalho de Conclusão de Curso II apresentado à Universidade Federal do Tocantins para obtenção do título de Bacharel em Ciência da Computação, sob a orientação do(a) Prof.(a) Dr. Marcelo Lisboa.

Orientador: Dr. Marcelo Lisboa

PALMAS (TO)
2022

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da Universidade Federal do Tocantins

A474p Alves de Moura, Pedro Henrique.
Protótipo de Rede Neural Convolucional para Problemas de Classificação. / Pedro Henrique Alves de Moura. – Palmas, TO, 2022.
42 f.

Monografia Graduação - Universidade Federal do Tocantins – Câmpus Universitário de Palmas - Curso de Ciências da Computação, 2022.
Orientador: Marcelo Lisboa Rocha

1. Aprendizado Profundo. 2. Redes Neurais Convolucionais. 3. Problemas de Classificação. 4. Ciência da Computação. I. Título

CDD 004

Elaborado pelo sistema de geração automatica de ficha catalográfica da UFT com os dados fornecidos pelo(a) autor(a).

PEDRO HENRIQUE ALVES DE MOURA

PROTÓTIPO DE UMA REDE CONVOLUCIONAL PARA PROBLEMAS DE CLASSIFICAÇÃO

Trabalho de Conclusão de Curso II apresentado à UFT – Universidade Federal do Tocantins – Campus Universitário de Palmas, Curso de Ciência da Computação foi avaliado para a obtenção do título de Bacharel e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Data de aprovação: 7 / 2 / 2022

Banca Examinadora:

Prof. Dr. Marcelo Lisboa Rocha

Profa. Dra. Anna Paula de Sousa Parente Rodrigues

Prof. Dr. Alexandre Tadeu Rossini da Silva

Para todos que me apoiaram.

AGRADECIMENTOS

Os meus mais sinceros agradecimentos pelo suporte e ajuda que a minha família, amigos e o Professor Marcelo Lisboa puderam me oferecer, sou extremamente grato por tudo que foram responsáveis para o desenvolvimento deste projeto.

RESUMO

A proposta do trabalho é empregar Redes Neurais Convolucionais capazes de identificar e resolver problemas de classificação. Para isso, um protótipo de tal rede é desenvolvido com o intuito de comparar os resultados obtidos com outros estudos presentes na área, tendo em vista que algumas das dificuldades enfrentadas na área de Aprendizado de Máquina se referem a catalogar, analisar e por fim classificar grandes quantidades de dados com uma alta taxa de acurácia.

Palavra-chave: Aprendizado Profundo. Redes Neurais Convolucionais. Problemas de Classificação. Ciência da Computação.

ABSTRACT

The purpose of the work is to employ Convolutional Neural Networks capable of identifying and solving classification problems. For this, a prototype of such a network is developed in order to compare the results obtained with other studies present in the area, considering that some of the difficulties faced in the area of Machine Learning refer to cataloging, analyzing and finally classifying large amounts of data with a high accuracy rate.

Keywords: Deep learning. Convolutional Neural Network. Classification Problems. Computer Science.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Diante da evolução da computação em todos os seus contextos e aplicações, desde uma simples calculadora a computadores capazes de elaborar grandes equações e trabalhar com uma imensa quantidade de informação, há espaço para que diversas áreas possam constantemente evoluir e exibir resultados que impactem de alguma forma, problemas criados pelo ser humano.

A interdisciplinaridade presente na Computação, ao simular traços e adaptar conhecimentos de outras áreas para a tecnologia, estimula uma visão mais lógica do mundo, assim como o ser humano e alguns animais são capazes de diferenciar e entender objetos, classificando-os, o aprendizado de máquina pode contribuir para uma aplicação mais eficiente caso encontre problemas em grandes quantidades de dados, o que geralmente demanda tempo e esforço do homem.

Portanto, Problemas de classificação são questões relacionadas a como fazer com que a máquina ”aprenda”a reconhecer padrões e classificar informações dentro de um limite de respostas previamente definidas, assim, de acordo com os dados de entrada, capaz de resolver as mais variadas situações, entrando assim no ramo de aprendizado profundo.

O aprendizado profundo com redes neurais (do inglês *Deep Learning* ou DL) é um assunto bastante abordado para resolver problemas relacionados a classificação, de forma que possui vários tipos diferentes de redes neurais inspiradas em diversas formas de processamento biológicos, a Rede Neural convolucional será a abordada neste estudo para elaborar a hipótese sobre sua capacidade de resolver estes problemas.

As redes neurais convolucionais (do inglês *Convolutional Neural Networks* ou CNN) foram criadas a partir do entendimento do processamento de imagens visuais no cérebro para classificação de imagens com um comportamento semelhante a neurônios de um sistema nervoso central e demonstram uma alta taxa de desempenho, visto que diferente de redes feed-forwards que aplicam a vetorização de imagens, as redes convolucionais fornecem uma capacidade de capturar dados relevantes de imagens através da aplicação de filtros de acordo com a necessidade.(SCHMIDHUBER, 2015)

Mesmo uma Rede neural Convolucional sendo normalmente usada para classificar imagens a ideia de que haveria resultados positivos ao utiliza-la para grupos de dados variados surgiu após o estudo do (ÖRKCü; BAL, 2011), em que ao realizar vários testes datasets variados, estabelecia um comparativo entre diversas técnicas e algoritmos.

A proposta deste trabalho consiste em diferenciar de vários métodos para resolver problemas de identificação e classificação ao utilizar redes neurais convolucionais, espera-se comparar algumas técnicas já utilizadas no meio da computação e com isso buscar por um algoritmo capaz de ser tão ou mais eficiente que os atuais existentes, visto que a capacidade do aprendizado profundo convolucional em aceitar diversos parâmetros em suas camadas de execução costuma ter uma alta taxa de aceitação.

Estabelecendo, portanto, a hipótese de que uma Rede Neural Convolucional é capaz de apresentar desempenho positivo quando comparada a outras técnicas dentro da área de aprendizado profundo e as vistas no trabalho proposto por (ÖRKCü; BAL, 2011).

2 OBJETIVOS

A seguir serão apresentados o Objetivo Geral e Objetivos Específicos desse trabalho.

2.1 Objetivo Geral

- Avaliar a capacidade e desempenho da acurácia do protótipo de uma rede neural convolucional para problemas de classificação em relação ao trabalho de ÖRKCÜ e BAL (2011).

2.2 Objetivos Específicos

- Propor um algoritmo de Rede neural Convolucional.
- Aplicar este algoritmo a grupos de dados selecionados.
- Avaliar o desempenho da acurácia obtida comparando com os resultados do trabalho proposto por ÖRKCÜ e BAL (2011).

3 REFERENCIAL TEÓRICO

Ao analisar o projeto é necessário compreender todo o conceito e áreas de pesquisa necessários para o desenvolvimento e solução do problema anunciado, uma vez que há o material exposto sobre redes neurais convolucionais e problemas de classificação, possuem informações novas com bastante frequência.

3.1 Deep learning e redes neurais

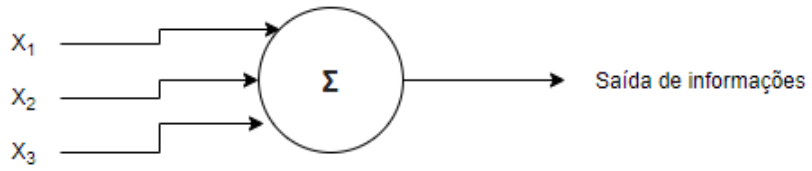
O aprendizado profundo é um ramo da área de aprendizado de máquina, que, com estruturas, filtros e métodos de aplicação, permite modelos computacionais com várias camadas de processamento a aprenderem a representar informações com diversos níveis de abstração, muitas vezes utilizando estruturas baseadas em sistemas biológicos como o sistema nervoso e seus neurônios.(LECUN; BENGIO; HINTON, 2015)

O Deep learning, tem como uma de suas principais características trabalhar com níveis grandes de dados não estruturados e por meio do treinamento de seus diversos tipos de redes neurais conseguir classificar seguindo padrões específicos. Normalmente a técnica de aprendizado supervisionado é a mais utilizada para treinar as redes neurais visto que, ao entregar a resposta esperada no processo de aprendizado, qualquer variação do dado de saída poderá ser ajustado com os parâmetros para que as próximas iterações do sistema possam se aproximar da resposta.(GOODFELLOW; BENGIO; COURVILLE, 2016)

Uma rede neural pode ser classificada como um aproximador de funções, visto que ao possuir os valores de entradas, suas camadas possuem pesos e vieses onde o objetivo principal é se assemelhar a alguma das respostas desejadas na camada de saída.(ZEILER; FERGUS, 2013a)

A estrutura de uma rede neural simples pode ser percebida a seguir, ao detalhar o funcionamento do perceptron, que foi o primeiro modelo de rede neural, desenvolvido nas décadas de 1950 e 1960 pelo cientista Franken Rosenblatt(SCHMIDHUBER, 2015) :

Figura 1 – Modelo Simplificado de um perceptron e seu funcionamento

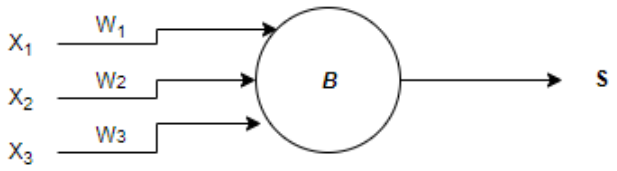


O Percepetron, como pode ser visto na Figura 1 segue um modelo matemático básico como visto na equação 1.Se a operação de soma ponderada resultar num valor menor que o limiar

a saída se tornará 0, se a operação obtiver um valor igual ou maior, saída será 1. Resultando num sistema condicional de acordo com os valores de entrada(ZEILER; FERGUS, 2013b)

$$f(x) = \begin{cases} 0 -> if & : \sum_j w_j x_j < Limiar \\ 1 -> if & : \sum_j w_j x_j \geq Limiar \end{cases} \quad (1)$$

Figura 2 – Modelo de um Perceptron com peso (W) e viés (B)



Toda rede neural apresenta uma função de custo, o que afeta o comportamento de suas camadas ocultas, ao gerar uma saída correspondente a fórmula utilizada.Além do viés, que muda o conceito de limiar, o que causa alterações no comportamento do perceptron.O peso caracterizado na Figura 2 influencia diretamente o resultado da Rede Neural, ao isolar parêmtros específicos com o objetivo de facilitar o sistema de se aproximar do objetivo esperado.(GOODFELLOW; BENGIO; COURVILLE, 2016)

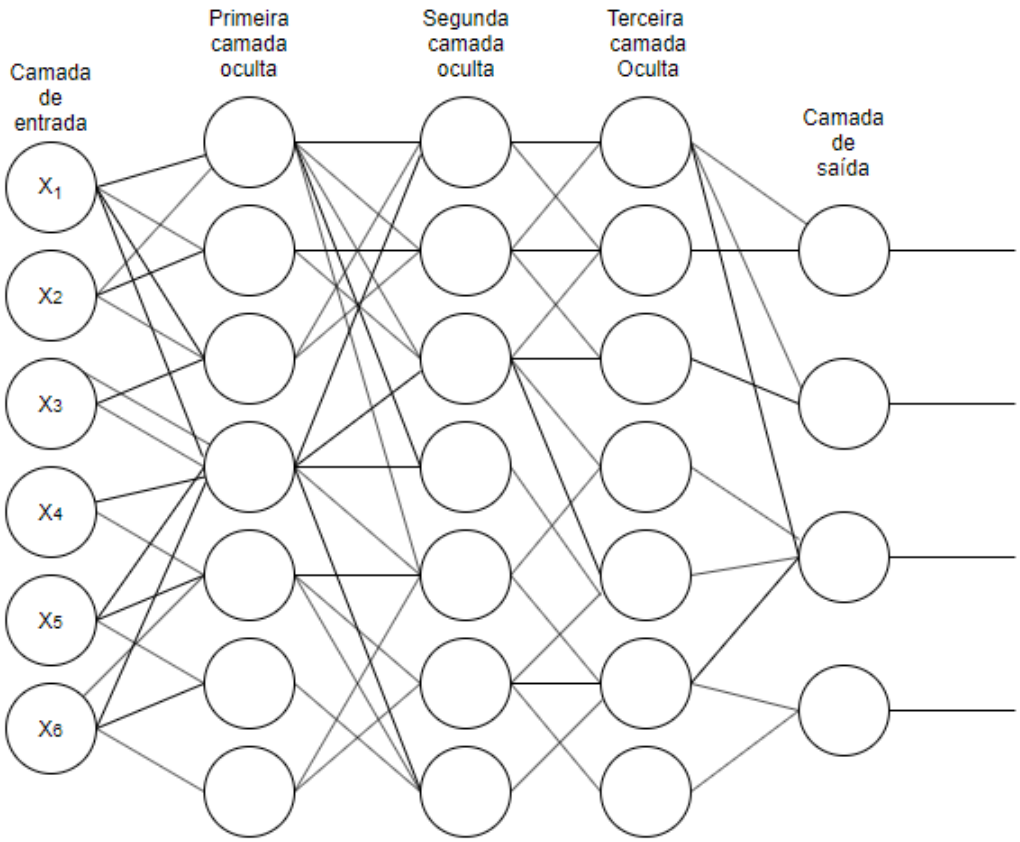
3.2 Rede Neural Convolucional

As redes neurais convolucionais são diferentes de outras redes neurais, pois nem todas as suas camadas estão conectadas, visto que essa rede neural almeja levar em conta sua taxa de pré-processamento e o treinamento necessário para que consiga chegar numa solução mais próxima do ideal (GOODFELLOW; BENGIO; COURVILLE, 2016). Conforme pode ser observado na Figura 3 o modelo simples de uma Rede Neural Convolucional.

A imagem desejada é transformada em uma matriz de pixels,ocorre o surgimento de campos receptivos locais, ou seja, pequenas matrizes são formadas com os pixels originais da imagem e se tornam as entradas dos neurônios da primeira camada oculta, gerando viés e pesos que possuem como característica, o fato de serem compartilhados, resultando numa taxa menor de processamento de informações ao longo do processo, possuindo parâmetros estabelecidos, seguindo a fórmula matemática de custo de função de acordo com a necessidade do problema.Para condensar a respostas das camadas escondidas de neurônios, se utiliza as camadas de pooling ou de agrupamento, com o intuito de reduzir o número de parâmetros nas camadas posteriores.(VARGAS; PAES; VASCONCELOS, 2016)

A arquitetura de uma rede neural convolucional (CNN), apresenta diversos hiper-parâmetros necessários para sua execução, assim como os neurônios são importantes para cada camada poder processar as informações, há a paciência que define a quantidade máxima de vezes em que a

Figura 3 – Estrutura simples de uma rede neural convolucional



CNN irá processar para validar o dataset, praticando o que é chamado de early stopping quando não há um progresso após determinado número de iterações o código irá parar a execução da rede neural.

O Batch size é importante para uma rede neural pois define o número de amostras que serão analisadas durante a etapa de processamento, pois ao dividir a entrada original de dados em lotes menores que a quantidade original, permite que o treinamento necessite de menos memória da máquina e aumentando a velocidade de execução, saltando para soluções boas em tempos menores, ao invés de usar todo o conjunto de dados e correr riscos desnecessários como overfitting.

As camadas densas e de dropout em uma arquitetura CNN previnem o sobreajuste (overfitting) da rede, a primeira por fazer com que uma camada se conecte com todas as outras posteriores e segunda cria uma máscara sob a rede em questão, que nulifica temporariamente a contribuição de alguns neurônios para as próximas camadas e deixa todas as outras intactas, resultando em um processo que se comporta de forma similiar treinamento de diferentes redes neurais, estatisticamente, aumentando assim a possibilidade de caminhos e a forma como a rede neural irá aprender pesos e bias para cada treinamento, portanto, cria uma média dos valores encontrados, eficaz para aumentar a acurácia e impedir o overfitting.

O método Early stopping necessário em etapas mais avançadas de desenvolvimento para

evitar o overfitting e o underfitting, uma vez que poucas épocas podem não ser o sufuciente para chegar ao nível desejado de desempenho e o overfitting é quando épocas em excesso prejudicam o treinamento, assim definindo um teto no qual, após um número específico de testes, se não houver melhora, a rede neural irá encerrar sua execução.

A Técnica de validação cruzada é utilizada como forma de separar subconjuntos dos grupos de dados, como forma de evitar sobreajuste e garantir um treinamento eficaz da rede, o método utilizado neste projeto será o K-fold no qual separa os subconjuntos em K pastas em que uma delas será a base de comparação entre as demais, assim estabelecendo uma média (SALZBERG, 1997).

A Técnica de grid search e hiperparametrização é responsável por avaliar a acurácia da rede neural ao definir um conjunto de valores para cada hiperparâmetro julgados necessários para a execução e arquitetura do sistema, de tal forma que seja possível identificar a melhor combinação destes valores.

A Taxa de Aprendizado (learning rate) é um hiperparâmetro útil para a rede neural ao determinar o tamanho de cada etapa de iteração do treinamento e o ritmo no qual os pesos são atualizados, assim, define a velocidade de aprendizagem da rede neural.

As funções de ativação são modelos matemáticos aplicados na rede neural para definir quando um neurônio deve ou não ser ativado em alguma camada, determinando se a informação é importante ou pode ser desconsiderada durante o treinamento, evitando custo maior de processamento, o que torna assim a rede mais esparsa e fácil de ser computada, há diversos tipos de funções de ativação e cada uma delas tem suas características e aplicações visto que é uma área de pesquisa ativa e em constante evolução.

A Função Relu, ou a função de ativação de unidade linear rectificada define o processamento de diversos tipos de redes neurais, Como pode ser visto na fórmula presente na equação 2, a função Relu determina se o neurônio é ou não relevante de acordo com a necessidade de cada rede.

$$Relu(x) = \begin{cases} 0, quando : x < 0 \\ 1, quando : x \geq 0 \end{cases} \tag{2}$$

A função de ativação Sigmoid binária, possui como característica principal em sua normalização entre 0 e 1, diferente da sua contraparte bipolar, em que vai de -1 a 1; estabelece os limites para as saídas dos resultados suavizando e reduzindo o tempo de processamento ao facilitar o corte de um grande número de dados dentro de uma rede neural, seus pontos positivos também influenciam nos pontos negativos em que quanto mais próximo de 0 o gradiente se encontra, significa que a rede não está aprendendo realmente.

$$Sigmoid(x) = \begin{cases} \frac{1}{(1+e^x)} : x < 1 \end{cases} \tag{3}$$

A função de ativação Softmax é uma versão extendida da sigmoid, pois engloba a logística sigmoidal em um campo exponencial, generalizando o número de classes possíveis dentro da fórmula, sendo bastante efetivo para determinar probabilidades e comumente utilizado na última camada de ativação para normalizar a saída de dados dentro de um axioma determinado.

$$Softmax(x) = \left\{ \frac{\exp(x_i)}{\sum_j \exp(x_j)}; x < 1 \right. \tag{4}$$

A função Selu por sua vez origina da sigla Scaled Exponential Linear Unit, é mais recente que suas contrapartes também exponenciais como Leaky Elu, Elu e Relu, porém apresenta a auto normalização da rede com o uso de novas varíaveis como o lambda e o alfa, que escalonam o grau de complexidade realizando a tarefa de reduzir o número de dados e identificar os valores corretos simultaneamente, afim de dimuir o tempo de processamento, maximizando eficiência.

$$Selu = \lambda = \begin{cases} \alpha(e^x - 1), quando : x < 0 \\ x, quando : x \geq 0 \end{cases} \tag{5}$$

A Exponential linear Unit apresenta pontos positivos em sua utilização, por reduz o gradiente e impede que os valores de saída sejam zero, assim pode exibir comportamentos complicados, como tempo de processamento maior, em razão de conseguir alcançar números negativos.

$$Elu(x) = \begin{cases} \alpha(e^x - 1), quando : x \leq 0 \\ x, quando : x > 0 \end{cases} \tag{6}$$

Como pode ser percebido, uma rede neural apresenta diferentes formas de comportamento, de acordo com o tipo de função de ativação e outras parâmetros para seu funcionamento, a estrutura básica porém pode ser estabelecida como camada de entrada, camadas ocultas e camadas de saída.

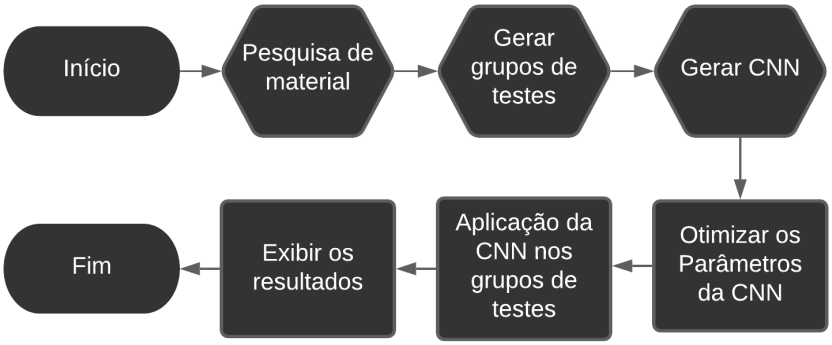
3.3 Ferramentas

As ferramentas utilizadas ao longo do trabalho foram a Anaconda, plataforma de datascience que possui diversos programas úteis na área de análise e ciência de dados (ANACONDA, 2017), assim, foram utilizados a IDE Spyder e Jupyter Notebook que estão presentes dentro da plataforma, desenvolvendo os códigos de forma direta e em formato markdown por meio de células, por fim o Google Colaboratory se fez necessário em razão de utilizar programação em nuvem e as máquinas virtuais do Google (SILVA, 2020), diminuindo assim o trabalho da máquina local para treinar a rede neural.

4 METODOLOGIA

Na figura 4, é exibido o fluxograma do trabalho, explicando graficamente como todo seu desenvolvimento foi executado. O projeto tem em sua abordagem inicial a necessidade de criar o protótipo de uma rede neural convolucional para analisar grupos de dados genéricos e assim estabelecer um caminho em resolver problemas de classificação o que resulta em todo um domínio do estudo de técnicas e ferramentas necessárias, além de compreender uma quantidade massivas de dados e modelos estatísticos, a pesquisa portanto é quantitativa, pois gera a revisão de dados importantes em sua análise.

Figura 4 – Fluxograma do Trabalho



Há viés de estudo aplicado quando relacionado a sua natureza, visto que a especificação do projeto futuro será relacionado a apenas uma área em questão, e seu objetivo principal possui função explicativa como forma de entender os resultados adquiridos,relativo aos procedimentos necessários, o estudo de caso será de extrema importância para avaliar a possibilidade do emprego de uma rede neural convolucional ser efetiva em relação a outras redes.

O projeto fará uso de ferramentas como o Anaconda (ANACONDA, 2017), um programa feito com o propósito de auxiliar a área de Data Science, Deep learning e testes de redes neurais seu emprego tem como propósito comparar alguns algoritmos de identificação de imagens com redes neurais, assim comparando dados, a complexidade dos algoritmos e taxa de processamento.

Após a criação do Protótipo de Rede Neural Convolucional, a escolha dos hiperparâmetros a serem utilizados determinariam o nível de proficiência, iniciando assim a etapa de testes e coleta dos resultados. Posteriormente a coleta dos resultados, seria realizado a análise e comparação entre os vários datasets para uma rede neural multilayer perceptron e o protótipo da CNN, a fim de validar a proposta de que uma rede neural convolucional poderá apresentar uma performance positiva em sistemas não específicos, por isso a utilização de datasets de diferentes áreas, tamanhos e parâmetros.

O fato da pesquisa continuar sendo quantitativa - experimental, dentro de um espaço amostral com uma taxa de reprodutibilidade difícil, demonstra que a computação tem muito a evoluir em sua parametrização de resultados, tendo em vista que replicar um processo e entender seus nuances, exige entendê-lo completamente. Portanto a dificuldade em chegar a respostas satisfatórias, exigem testes de validação cruzada entre todos os dados obtidos, além de progedir a área de estudo com grupos de dados que apresentam saídas binárias e avaliar suas perfomances com diferentes filtros de ativação.

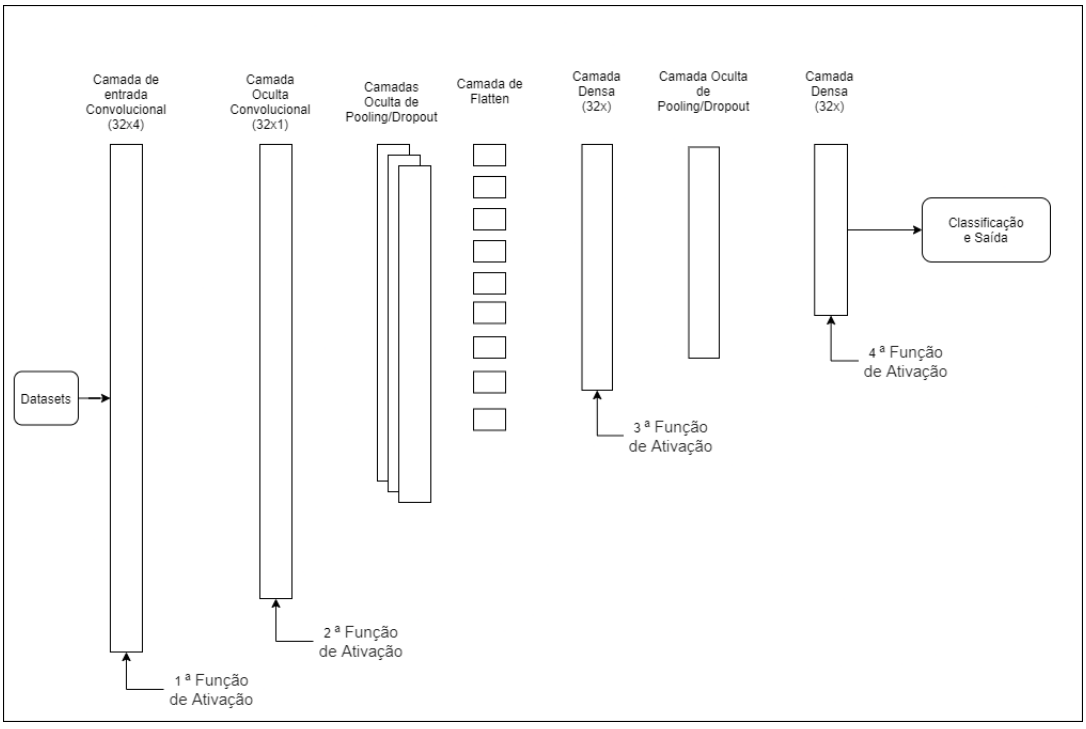
Por conseguinte, os filtros de ativação mais comuns seriam analisados e testados na rede neural convolucional: Sigmoid, Relu, Elu, Selu, Softmax,Tangencial e os resultados mais otimistas seriam separados e postos em debate para análise.

Após os datasets escolhidos para o passo final de aquisição de informações, que se destacaram por terem apresentado dados ruins quando interpretados e processados na primeira etapa de desenvolvimento da rede neural convolucional, rede esta, que utilizava o filtro de ativação Relu para qualquer tipo de resultado, não obstante a estrutura almejava entender as funções de ativação em uma rede neural, comparar e avaliar os resultados obtidos, além formular uma hipótese para as diferenças nos tipos de processamento da rede neural com diversos filtros de ativação diferentes em suas camadas.

A continuidade do projeto utiliza-se de técnicas de hiperparametrização dos dados da rede neural, padronizando os valores de Batch size, Learning Rate, Épocas e Early stopping, com o intuito de analisar a perfomance obtida com os mais diversas configurações da CNN.

A estrutura da rede neural utilizada tem por ventura a representação gráfica como pode ser observada na imagem 5, assim estabelece o arquétipo do funcionamento do projeto.

Figura 5 – Representação gráfica da CNN



Ao revisar artigos e pesquisas da área de classificação de dados em que se utiliza redes neurais, foi estabelecido dez grupos de dados genéricos para serem utilizados provenientes do UCI machine learning repository (DUA; GRAFF, 2017) e que, quando usados para classificação, costumam ser alguns dos padrões de dados de testes (ÖRKCü; BAL, 2011).

Para avaliação de todas as versão do protótipo desenvolvidos de CNNs, foram utilizados os conjuntos de dados (*datasets*) apresentados com suas principais características na tabela 1. Esses *datasets* estão disponíveis no site da UCI (<https://archive.ics.uci.edu>)

Tabela 1 – Tabela com os grupos de dados pertencente ao UCI ML

Dataset	Número de grupos	Quantidade de unidades	Número de variáveis
Breast Cancer	2	683	10
Bupa Live	2	345	6
Boston Housing	2	1012	13
Pima Diabetes	2	768	8
Iris	3	150	4
Vehicle	4	846	18
Dermatology	6	358	34
Glass	6	214	9
E.coli	8	336	7
Yeast	10	1484	8

O trabalho proposto por (ÖRKCü; BAL, 2011) foi usado como base inicial, utilizando os mesmos datasets de sua abordagem para realizar a comparação dos resultados entre ambos os projetos e tratamento de dados, excluindo seções de valores que não estão presentes.

O Breast Cancer possui 683 unidades, com 10 números de variáveis, sua saída binária determina se o nódulo corresponde é benigno ou maligno.

A Bupa Live é um dataset que possui 345 unidades, 6 variáveis e é responsável por identificar a presença ou não de problemas no fígado, logo a saída também é binária.

Boston housing possui 1012 unidades com 13 variáveis e sua saída binária identifica se a casa em questão terá um valor alto ou baixo.

Pima diabetes possui 768 unidades e 8 variáveis, sua saída é binária pois identifica se o paciente está com os exames de sangue normais ou não.

O Iris dataset possui 150 unidades e 4 variáveis, classifica seus dados em três tipos distintos de plantas iris.

o grupo de dados Vehicle possui 846 unidades e 18 variáveis, sua saída representa 4 tipos diferentes de automóveis para serem classificados.

Dermatology classifica suas 358 unidades de 34 variáveis em 6 tipos de doenças de pele.

Glass é um grupo de dados utilizado na investigação forense por classificar suas 214 instâncias e 9 variáveis em 6 tipos de vidro como saída.

E.coli possui 336 unidades e 7 variáveis com chance de classifiacr 8 características pertinentes a fisiologia da bactéria e seu comportamento.

Yeast possui 1484 instâncias com 8 colunas e 10 grupos de saída, que classifica o grupo de dados em tipos de infecções e doenças possíveis.

Com os datasets escolhidos, ocorreu o processo de definição de métricas para a comparação, dentre elas a *K-fold*. *K-fold* consiste em treinar a rede neural com o *dataset* específico *K* vezes (SALZBERG, 1997). Neste estudo, foi definido o valor de *K* sendo igual a 10. Assim, o grupo de dados é dividido em 10 partes, em que 9 são treinadas e 1 é utilizada como teste, após isso, é possível estabelecer a média entre as partes e assim obter um resultado mais confiável.

Como pré-processamento dos dados, os *datasets* que possuíssem alguns campos de informações perdidas, não preenchidos, ou preenchidos incorretamente (*missing values ou NaN*) eram corrigidos, de tal forma que as respectivas linhas foram excluídas, para que não houvessem problemas maiores durante a próxima etapa de classificação.

Deve-se atentar que uma rede neural tem como proposta de avaliação de perfomance a sua precisão relacionada ao tempo e quantidade de dados inseridos, assim, para evitar resultados incorretos ou inconclusivos, além de problemas na medição de tempo, fatores externos também devem ser medidos (SOKOLOVA; LAPALME, 2009), como o equipamento a ser utilizado no processamento e outras variáveis de ambiente.

A CNN em questão, desenvolvida neste projeto inicial, tem os seguintes hiper-parâmetros:

- número máximo de épocas (*epochs*) igual a 500,
- paciência (*patience*) com valor 100,
- a taxa de aprendizado (*learn_rate*) igual a 0.01,
- a taxa de *dropout* (*dropout_rate*) igual a 0.2,
- o tamanho do batch (*batch_size*) igual a 50,
- função de custo (*loss function*),
- entropia cruzada (*categorical_crossentropy*).

Assim, fica estabelecido um padrão para os testes, para os testes com a validação cruzada K-fold, o padrão para K foi definido como K = [10].

Os fatores externos previamente citados são a estrutura e capacidade do hardware em questão. Neste trabalho, para as quantidades de dados presentes e com os recursos disponíveis, foi utilizada uma máquina com a seguinte configuração: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz, 8 Gb de RAM, GPU Nvidia 920MX com 2 Gb de RAM e sistema operacional Windows 10/64 bits. Ressalta-se a importância da utilização de GPU para acelerar o processamento de técnicas de *Deep Learning*.

O amadurecimento do projeto e sua evolução resultou no estudo da hiperparametrização associado a validação cruzada dentro dos grupos de dados escolhidos, com o objetivo de ampliar a taxa de sucesso e performance da rede neural convolucional, desta forma estabelecendo

conjuntos de variações de parâmetros a serem testados e analisados. Logo os Hiperparâmetros passaram a ser:

- *epochs* = [100, 200, 500]
- *batch_size* = [10, 50, 100]
- *dropout_rate* = [0.1, 0.2]
- *learn_rate* = [0.001, 0.01]
- *loss* = [’categorical_crossentropy’, ’binary_crossentropy’]

E após a definição dos hiperparâmetros iniciais, a realização de outra etapa de testes porém com o método de early stopping, adicionando a paciência, sendo ela igual a 100.

A princípio foi utilizado a ferramenta Anaconda (ANACONDA, 2017) com a sua IDE própria de desenvolvimento Spyder, em conjunto com o Jupyter notebook, porém com o grande volume de processamento necessário, apenas o hardware especificado não foi capaz de suprir a demanda, logo se fez a utilização do Google Colaboratory.

5 RESULTADOS OBTIDOS

Nesse capítulo serão apresentados os resultados obtidos dos vários modelos de *CNNs* propostos em relação ao modelo de rede neural *back-propagation* otimizada por algoritmos genéticos proposto na literatura em ÖRKCÜ e BAL (2011) para problemas de classificação.

Primeiramente, foi necessário utilizar a rede neural CNN apresentada em Listing 7.1 como forma de ter uma base inicial dos resultados para o estudo em questão. Conforme as métricas estabelecidas e utilizando os datasets apresentados, montou-se uma tabela comparativa, no caso, a tabela 3, entre a rede neural convolucional (CNN) utilizada nesse trabalho e redes neurais por Back-propagation apresentadas por H. Hasan Örkcü e Hasan Bal (2011) no trabalho ”Comparing performances of backpropagation and genetic algorithms in the data classification”. Com isso, foi possível alcançar resultados iniciais satisfatórios, conforme exibido na tabela 2.

Tabela 2 – Tabela Comparativa entre a CNN Proposta e o trabalho de ÖRKCÜ e BAL (2011)

Dataset	Back Propagation	Binary Coded GA	Real- Coded GA	CNN
Breast Cancer	93.10;28.30	94.00;30.00	96.50;22.10	**97.34;64.90**
Bupa Live	69.70;11.40	70.40;13.60	71.50;6.80	**73.60;47.40**
Boston Housing	81.60;19.50	82.80;25.40	85.50;14.60	**91.00;85.20**
Pima Diabetes	73.80;14.60	74.80;17.40	**77.60;10.30**	73.60;68.00
Iris	96.80;3.80	97.50;3.70	97.80;1.70	**98.00;11.80**
Vehicle	74.80;55.20	76.00;42.00	76.90;35.80	**78.90;60.50**
Dermatology	90.70;39.40	92.00;32.30	95.00;18.80	**97.00;17.50**
Glass	63.60;8.90	65.50;9.20	67.60;4.90	**97.10;48.50**
E.coli	71.80;12.60	75.80;12.80	76.20;7.60	**91.80;25.70**
Yeast	67.00;62.60	68.00;65.80	7**2.80;45.20**	67.30;141.90

Legenda - cada coluna contém a Acurácia e Tempo em segundos dos respectivos métodos

De acordo com os resultados apresentados na tabela 2, verifica-se que a proposta inicial da CNN apresentou bons resultados quanto às taxas de classificação ou acurácia, possibilitando assim, a continuidade do trabalho fazendo uso de recursos mais sofisticados para testes e ajustes dos hiper-parâmetros.

Na tabela 3, apresenta-se para cada método considerado, a precisão (em percentual) e o tempo computacional em segundos, de acordo com o melhor resultado equivalente das arquiteturas utilizadas no trabalho original.

É possível perceber ao analisar os dados obtidos na tabela 3, que no âmbito geral, da precisão da rede neural convolucional (CNN) apresenta uma melhora em detrimento aos outros tipos de redes neurais (dados marcados em negrito), como nos datasets Breast Cancer, Boston housing, Iris, Dermatology e E.coli, além do dataset Glass em que houve uma variação positiva de trinta por cento a mais de precisão.

Tabela 3 – Tabela Comparativa entre a CNN Proposta utilizando o método K-fold e o trabalho de ÖRKCÜ e BAL (2011)

Dataset	Back Propagation	Binary Coded GA	Real- Coded GA	CNN K-fold
Breast Cancer	93.10;28.30	94.00;30.00	96.50;22.10	**98.50;80.80**
Bupa Live	69.70;11.40	70.40;13.60	71.50;6.80	**73.12;92.00**
Boston Housing	81.60;19.50	82.80;25.40	85.50;14.60	**96.50;69.60**
Pima Diabetes	73.80;14.60	74.80;17.40	**77.60;10.30**	73.90;72.30
Iris	96.80;3.80	97.50;3.70	97.80;1.70	**98.00;68.40**
Vehicle	74.80;55.20	76.00;42.00	76.90;35.80	**79.60;189.60**
Dermatology	90.70;39.40	92.00;32.30	95.00;18.80	**97.49;20.20**
Glass	63.60;8.90	65.50;9.20	67.60;4.90	**94.41;125.80**
E.coli	71.80;12.60	75.80;12.80	76.20;7.60	**83.94;44.00**
Yeast	67.00;62.60	68.00;65.80	**72.80;45.20**	69.90;100.00

Legenda - cada coluna contém a Acurácia e Tempo em segundos dos respectivos métodos

Porém, em alguns datasets nos quais não houve melhora e inclusive demonstraram rendimento inferior ao que foi usado como referência. No caso, os datasets Bupa Live, Vehicle, Pima Diabetes e Yeast, demonstrando assim que é necessário um estudo aprofundado sobre as possíveis variáveis que interferiram e causaram estes resultados negativos.

A performance das redes neurais quando associadas ao tempo, apesar de ser executada em uma máquina mais rápida, apresentou tempo de execução maior, devido à CNN ser um tipo de aprendizagem profunda (Deep Learning) onde se tem um número extenso de camadas e neurônios em cada uma delas, relação à rede neural multilayer perceptron do modelo Backpropagation do estudo da literatura utilizada para comparação.

A continuar a ideia e com o resultado previamente alcançado, inicia o pressuposto principal sobre a capacidade e a acurácia na classificação de grupos de dados com saídas binárias utilizando diferentes filtros de ativação durante a execução do algoritmo de rede neural.

De tal forma que analisar os erros cometidos na primeira etapa e restringir os parâmetros de pesquisa e estudo para datasets de saída binária em que demonstrassem a possibilidade de uma rede neural convolucional exibir uma performance relevante para a área da computação.

A tabela 4 demonstra a utilização das várias funções de ativação. Nos *datasets* especificados usa-se então, a métrica de acurácia e o tempo de execução em comparação entre os filtros escolhidos, de tal maneira, que exibe apenas os resultados mais satisfatórios com validação k-fold, em contraste aos mais diversos tipos de funções de ativação que poderiam ter sido escolhidos ou são mais comumente utilizados em outras situações.

A escolha dos grupos de dados por possuírem saídas binárias foi uma escolha categórica em relação as diversas possibilidades de tratamento de dados que a serem discutidos na área, em que o Bupa Live e o Pima diabetes haviam previamente demonstrado resultados negativos, assim, gera o motivo do estudo dos diversos filtros de ativação e suas implicações perante o tamanho dos datasets, a perfomance diante um sistema com capacidade computacional comum,

Tabela 4 – Tabela Resultados Variando as Funções de Ativação

Dataset	Sigmoid	Softmax	Sig. e Softmax	Selu e Softmax	Elu e Softmax
Boston Housing	**87.34/12.87**	85.36/12.31	87.17/98.91	85.76/10.56	85.40/74.37
Bupa Live	70.88/60.45	**71.96/95.50**	69.20/34.97	70.31/19.62	69.47/22.18
Breast Cancer	97.06/57.89	97.36/12.67	97.21/68.78	**94.71/38.03**	96.47/58.28
Pima Diabetes	**77.94/78.57**	74.10/19.95	74.61/20.16	74.61/50.01	74.21/56.62

Legenda - Cada coluna contém a Acurácia e Tempo em segundos dos respectivos métodos

análise do tempo gasto para a obtenção do resultado e se o mesmo é satisfatório diante os parâmetros de qualidade desejados.

Ao comparar a tabela dos resultados obtidos inicialmente (quinta coluna da tabela 3) com o resultado final (saídas em negrito da tabela 4) , é possível perceber que a priori, o Dataset Bupa Live apresenta uma melhora quando utiliza o filtro de ativação Softmax, num ambiente controlado de testes de validação k-fold, enquanto apresenta soluções menos desejadas com outros filtros, pode se deduzir então que o Softmax ao normalizar as saídas exponenciais das camadas densas entrega dados numa distribuição mais controlada diante do tipo específico e características do dataset em questão.

Boston housing exibe um comportamento positivo quando utilizado com as funções Sigmoid e o Sigmoid com Softmax em sua última camada, desta maneira, o fato do Sigmoid apresentar tamanha importância nesse grupo de dados, é devido ao fato de que o filtro ser uma derivada logarítimica não negativa, portanto, é capaz de interpretar números reais positivos de forma mais eficiente que o normal.

Breast câncer instiga a necessidade de analisar o comportamento da rede neural com as funções de ativação de forma ainda mais aprofundada, em virtude de que todas apresentaram um resultado relacionado a precisão inferior ao esperado, porém num tempo menor, o que leva em consideração a questão que com maior capacidade de processamento a máquina poderia apresentar uma perfomance mais aceitável nos padrões desejados. A função Relu que foi utilizada previamente é a mais difundida perante a área de rede neural convolucional, pois a taxa de ativação é mais esparsa e a escala apresenta boa compatibilidade para números não negativos.

O dataset Pima Diabetes apresenta a melhor qualificação ao perceber que em relação a precisão e tempo, é superior ao filtro Relu, além de se aproximar dos resultados vistos no Real Coded GA e no Binary Coded, portanto é o mais estável dentre os resultados com todos os específicos filtros de ativação, de tal forma que a possibilidade de alcançar um estágio de estudo da arte capaz de igualar e superar o Real Coded é uma visão necessária e abrangente se situar a constante evolução do campo de estudo.

A tabela 5 demonstra a comparação entre o avanço do estudo ao utilizar a hiperparametrização almejando assim encontrar valores que apresentassem uma maior performance quando utilizado nos datasets de exemplo. De tal modo que o número de épocas (*epochs*), o tamanho do batch (*batch_size*), a taxa de dropout (*dropout_rate*), a taxa de aprendizado (*learn_rate*) e a

Tabela 5 – Tabela Resultados da CNN com Hiperparametrização

Dataset	Melhor Resultado e Parâmetros	Tempo(s)
Breast Cancer	**98.52%** com 'batch size': 50, 'dropout rate': 0.1, 'epochs': 100, 'learn rate': 0.001, 'loss': 'categorical crossentropy'	5020.58
Bupa Live	**73.67%** com 'batch size': 10, 'dropout rate': 0.2, 'epochs': 100, 'learn rate': 0.001, 'loss': 'binary crossentropy'	2735.80
Boston Housing	**97.28%** com 'batch size': 10, 'dropout rate': 0.2, 'epochs': 500, 'learn rate': 0.001, 'loss': 'categorical crossentropy'	4156.39
Pima Diabetes	**77.48%** com 'batch size': 100, 'dropout rate': 0.1, 'epochs': 200, 'learn rate': 0.001, 'loss': 'binary crossentropy'	2412.41
Iris	**98.07%** com 'batch size': 50, 'dropout rate': 0.2, 'epochs': 500, 'learn rate': 0.01, 'loss': 'categorical crossentropy'	1464.82
Vehicle	**80.51%** com 'batch size': 10, 'dropout rate': 0.1, 'epochs': 500, 'learn rate': 0.001, 'loss': 'binary crossentropy'	5972.65
Dermatology	**98.30%** com 'batch size': 100, 'dropout rate': 0.2, 'epochs': 200, 'learn rate': 0.001, 'loss': 'binary crossentropy'	4238.20
Glass	**94.62%** com 'batch size': 10, 'dropout rate': 0.1, 'epochs': 100, 'learn rate': 0.01, 'loss': 'categorical crossentropy'	1875.09
E.coli	**85.60%** com 'batch size': 10, 'dropout rate': 0.1, 'epochs': 500, 'learn rate': 0.001, 'loss': 'categorical crossentropy'	2434.40
Yeast	**73.14%** com 'batch size': 10, 'dropout rate': 0.1, 'epochs': 500, 'learn rate': 0.001, 'loss': 'binary crossentropy'	9413.83

Legenda - A coluna do meio contém a Acurácia em negrito e os hiperparâmetros. Na útlima coluna contém o Tempo de execuação em segundos para cada Dataset

função de custo (*loss function*) foram vetorizados para possuirem mais de uma opção de testes por representarem as diversas possibilidades de configuração da estrutura de uma rede neural convolucional quando processada. Para os testes desenvolvidos, foram utilizados os seguintes valores para os parâmetros:

- *epochs* = [100, 200, 500]
- *batch_size* = [10, 50, 100]
- *dropout_rate* = [0.1, 0.2]
- *learn_rate* = [0.001, 0.01]
- *loss* = ['categorical_crossentropy', 'binary_crossentropy']

A ideia do algoritmo segue a realização de uma validação cruzada entre as diferentes combinações de parâmetros estabelecidos antecipadamente como pode ser percebido na tabela 5, de tal forma que uma execução extensiva é realizada, resultando num maior tempo de processamento.

Tabela 6 – Tabela Resultados da Hiperparametrização da CNN com Early Stopping

Dataset	Melhor Resultado e Parâmetros	Tempo(s)
Breast Cancer	**98.59%** com 'batch size': 100, 'dropout rate': 0.2, 'epochs': 500, 'learn rate': 0.01, 'loss': 'categorical crossentropy'	3936.06
Bupa Live	**73.08%** com 'batch size': 50, 'dropout rate': 0.2, 'epochs': 200, 'learn rate': 0.01, 'loss': 'binary crossentropy'	1266.56
Boston Housing	**99.20%** com 'batch size': 100, 'dropout rate': 0.2, 'epochs': 200, 'learn rate': 0.001, 'loss': 'categorical crossentropy'	3225.74
Pima Diabetes	**78.02%** com 'batch size': 50, 'dropout rate': 0.2, 'epochs': 500, 'learn rate': 0.001, 'loss': 'binary crossentropy'	4019.19
Iris	**98.10%** com 'batch size': 10, 'dropout rate': 0.1, 'epochs': 200, 'learn rate': 0.001, 'loss': 'categorical crossentropy'	1253.47
Vehicle	**80.54%** com 'batch size': 50, 'dropout rate': 0.1, 'epochs': 500, 'learn rate': 0.001, 'loss': 'binary crossentropy'	5912.56
Dermatology	**98.40%** com 'batch size': 10, 'dropout rate': 0.1, 'epochs': 500, 'learn rate': 0.001, 'loss': 'binary crossentropy'	2551.30
Glass	**95.29%** com 'batch size': 10, 'dropout rate': 0.1, 'epochs': 500, 'learn rate': 0.001, 'loss': 'categorical crossentropy'	1736.82
E.coli	**86.41%** com 'batch size': 10, 'dropout rate': 0.2, 'epochs': 500, 'learn rate': 0.001, 'loss': 'categorical crossentropy'	2136.77
Yeast	**73.18%** com 'batch size': 50, 'dropout rate': 0.1, 'epochs': 500, 'learn rate': 0.001, 'loss': 'binary crossentropy'	6985.68

Legenda - A coluna do meio contém a Acurácia em negrito e os hiperparâmetros. Na útlima coluna contém o Tempo de execuação em segundos para cada Dataset

A evolução do projeto se manteve ao utilizar os mesmos hiperparâmetros definidos anteriormente, porém acrescentando o parâmetro de early stopping com a escolha de como definir

sua paciência, sendo escolhido o valor de 100, para evitar iterações em demasia e favorecer o desempenho do código, economizando tempo ao executar e realizar os testes possíveis e possivelmente evitando o sobreajuste (overfitting).

Os resultados obtidos podem ser observados na tabela 6, nos quais apresentam certas melhoras na acurácia. Já, tempo de execução em todos os grupos de dados (*datasets*) foram menores com a utilização do *early stopping*.

Assim, ao estruturarmos os valores encontrados, comparandos-os desde o primeiro resultado obtido, é possível perceber ao observar a tabela 7 que a hiperparametrização encontrou um ponto de demonstrar uma maior performance em alguns datasets e se analisado no contexto geral e utilizar o early stopping em conjunto com a hiperparametrização (Hiperparametricação ES) apresenta resultados promissores em comparação a utilizar apenas a validação cruzada do método K-Fold.

Tabela 7 – Comparação dos Resultado de Acurácia das Várias Configuraçãoes de CNNs Testadas

Dataset	K-Fold	Hiperparametrização	Hiperparametrização ES
Breast Cancer	98.50	98.52	98.59
Bupa Live	73.12	73.67	73.08
Boston Housing	96.50	97.28	99.20
Pima Diabetes	73.90	77.48	78.02
Iris	98.00	98.07	98.10
Vehicle	79.60	80.51	80.54
Dermatology	97.49	98.30	98.40
Glass	94.41	94.62	95.29
E.coli	83.94	85.60	86.41
Yeast	69.90	73.14	73.18

Porém, o tempo de processamento aumenta em razão do alto número de valores a serem analisados e verificados pelo método *gridsearch*, exigindo tecnologias ainda mais potentes (como computação paralela e processamento em várias GPUs) para a execução da rede neural convolucional.

Logo o objetivo geral e específicos foram cumpridos ao econtrarmos resultados que se demonstram melhores que o trabalho proposto por ÖRKCÜ e BAL (2011) utilizando uma rede neural convolucional e definindo seus hiperparâmetros como pode ser observado ao longo de todas as etapas de testes e obtenção dos resultados.

6 CONCLUSÃO

Com tudo que foi realizado neste projeto de graduação, tendo em vista os resultados obtidos e seus respectivos aprofundamentos nas áreas, obteve-se as seguintes hipóteses para a resolução dos problemas encontrados durante a abordagem do trabalho.

Ao analisar e entender os resultados negativos encontrados nos datasets Bupa Live e Pima Diabetes, de modo que foi possível encontrar uma solução para o problema de performance da rede neural CNN nesses casos. O protótipo da rede neural convolucional sofreu algumas alterações ao longo de seu desenvolvimento, no entanto, sua versão final foi julgada satisfatória ao ser otimizado e de fácil manuseio.

Era esperado que com o desenvolvimento do sistema, fosse possível realizar uma análise da rede neural em questão de processamento, controle e predição de dados, relacionando com trabalhos já presentes na área porém que usam outros métodos para obtenção de resultados.

A princípio, desenvolver uma rede neural convolucional com uma alta taxa de precisão e suas possíveis variações era o objetivo a ser alcançado, porém com ramificações na problemática, alterações no tema provaram ser eficientes em relação a evolução da discussão.

Assim, sendo possível chegar a um veredito sobre a praticidade da aplicação e expectativas para a área de Redes neurais convolucionais na área de resolução de problemas de classificação de dados, tendo como foco saídas de dados binárias e seus diferentes caminhos.

Os diferentes tipos de filtros de ativação merecem uma gama de estudos mais desenvolvidos para que auxiliem a computação a alcançar um nível de perfeição do estudo da arte, no sentido que todos possuem, pontos positivos e negativos, sempre concomitante com o tipo de dados únicos de cada dataset, independente das saídas de dados serem binárias.

A expectativa para as etapas do projeto foi positiva diante dos percalços encontrados durante o caminho, analisando as possibilidades de se utilizar diferentes tipos de filtros de ativação com arquiteturas variadas durante a construção das camadas da rede neural convolucional, assim determina que a replicabilidade do projeto sem parâmetros específicos prejudica a capacidade de obter resultados ainda mais satisfatórios, além disso, constando a metanálise dos grupos de dados para tratar cada conjunto de forma direta e eficaz.

As soluções encontradas de se aproveitar a técnica de hiperparametrização e a de validação cruzada foram fatores que elevaram a necessidade por resultados ainda mais satisfatórios, independente se demonstrassem alto ou baixo desempenho, visto que são técnicas complexas que exigem um determinado grau de conhecimento.

A resposta positiva ao utilizar essas técnicas validam o desempenho do projeto, pois a hiperparametrização exibiu resultados com maior taxa precisão na maioria dos casos, corroborando a possibilidade de evolução das redes neurais convolucionais para as mais diversas áreas ao testar os mais diversos grupos de dados e demonstrar soluções com desempenhos considerados satisfatórios perante o que foi proposto ao longo de todo o estudo ao comparar o resultado

do material de estudo original proposto por ÖRKCÜ e BAL (2011).

REFERÊNCIAS

ANACONDA. **Anaconda Documentation**. 2017. <https://docs.anaconda.com/anaconda/>. Último acesso em 20/09/2019.

DUA, D.; GRAFF, C. **UCI Machine Learning Repository**. 2017. Disponível em: <http://archive.ics.uci.edu/ml>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

SALZBERG, S. L. On comparing classifiers: Pitfalls to avoid and a recommended approach. **Data mining and knowledge discovery**, Springer, v. 1, n. 3, p. 317–328, 1997.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. **Neural Networks**, Elsevier BV, v. 61, p. 85–117, Jan 2015. ISSN 0893-6080. Disponível em: <http://dx.doi.org/10.1016/j.neunet.2014.09.003>.

SILVA, M. da. Aplicação da ferramenta google colaboratory para o ensino da linguagem python. In: **Anais da IV Escola Regional de Engenharia de Software**. Porto Alegre, RS, Brasil: SBC, 2020. p. 67–76. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/eres/article/view/13717>.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. **Information Processing Management**, v. 45, p. 427–437, 07 2009.

VARGAS, A. C. G.; PAES, A.; VASCONCELOS, C. N. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. In: **Proceedings of the XXIX Conference on Graphics, Patterns and Images**. [S.l.: s.n.], 2016. p. 1–4.

ZEILER, M.; FERGUS, R. Visualizing and understanding convolutional neural networks. In: . [S.l.: s.n.], 2013. v. 8689.

ZEILER, M. D.; FERGUS, R. **Visualizing and Understanding Convolutional Networks**. 2013.

ÖRKCü, H.; BAL, H. Comparing performances of backpropagation and genetic algorithms in the data classification. **Expert Syst. Appl.**, v. 38, p. 3703–3709, 04 2011.

7 ANEXOS

7.1 Códigos utilizados

Protótipo de uma Rede Neural Convolucional

Esta seção apresenta um algoritmo de uma rede neural convolucional em Python em Listing 7.1 com aplicação no Jupyter Notebook e Anaconda, desenvolvido para reconhecer os três tipos de flores (classes) denominada do conjunto de dados Iris. Esse conjunto de dados Iris é padrão básico para teste de problemas de classificação, de modo a ter um código inicial básico que funcione para o respectivo problema.

```
1  import keras #biblioteca para Redes Neurais
2  import pandas as pd
3  import numpy as np
4  import sys
5  import keras
6  from keras.models import Sequential
7  from keras.layers import Dense, Flatten, Conv1D, Dropout
8  from keras.callbacks import ModelCheckpoint
9  from keras import backend as K
10
11
12 # Desenha curvas de aprendizado
13 def summarize_diagnostics(history):
14     from matplotlib import pyplot
15
16         # plota loss
17     pyplot.subplot(211)
18     pyplot.title('Cross Entropy Loss')
19     pyplot.plot(history.history['loss'], color='blue', label='train')
20     pyplot.plot(history.history['val_loss'], color='orange', label='test
           ')
21         # plota accuracy
22     pyplot.subplot(212)
23     pyplot.title('Classification Accuracy')
24     pyplot.plot(history.history['accuracy'], color='blue', label='train'
           )
25     pyplot.plot(history.history['val_accuracy'], color='orange', label='
           test')
26         # salva o plot para arquivo
27     filename = sys.argv[0].split('/')[-1]
28     pyplot.savefig(filename + '_plot.png')
29     pyplot.close()
30
31 def load_data() :
32     df=pd.read_csv("iris.data", header=0, names=['SepalLengthCm', '
           SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm', 'Species'])
33
34     df['labels']=df['Species'].astype('category').cat.codes
35     #print(df['labels'])
36
37     from sklearn.model_selection import train_test_split
38
39     X = df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', '
```

```
        PetalWidthCm']]
40      Y = df['labels']
41      x_train, x_test, y_train, y_test = train_test_split(np.asarray(X),
            np.asarray(Y), test_size=0.33, shuffle= True)
42
43      # numero de classes de saida.
44      num_classes = 3
45
46      # Dimensoes da entrada
47      input_shape = (4,1)
48
49      from keras.utils import np_utils
50
51      # Converte os vetores de classes para classes binarias. Usa 1 hot
            encoding.
52      y_train_binary = np_utils.to_categorical(y_train, num_classes)
53      y_test_binary = np_utils.to_categorical(y_test, num_classes)
54
55      size_train=len(x_train)
56      size_test=len(x_test)
57      x_train = x_train.reshape(size_train, 4,1)
58      x_test = x_test.reshape(size_test, 4,1)
59
60      return num_classes, x_train, x_test, y_train, y_test, y_train_binary
            , y_test_binary
61
62
63  def run_train(learn_rate=0.01, dropout_rate=0.2) :
64      #definicao da arquitetura da CNN
65      model = Sequential()
66      #primeira camada convolucional
67      model.add(Conv1D(32, (4), input_shape=(4,1), activation='relu'))
68      #segunda camada convolucional
69      model.add(Conv1D(32, (1), activation='relu'))
70      model.add(Dropout(dropout_rate+0.1))
71      model.add(Flatten())
72      model.add(Dense(32, activation='relu'))
73      model.add(Dropout(dropout_rate))
74      model.add(Dense(num_classes, activation='softmax'))
75
76      opt = keras.optimizers.RMSprop(learning_rate=learn_rate)
77      model.compile(optimizer=opt, loss='categorical_crossentropy',
            metrics=['accuracy'])
78
79      model.summary()
80
81      #earling stop
82      es = keras.callbacks.EarlyStopping(monitor='val_accuracy', mode='max
            ', verbose=1, patience=100, min_delta=0.001,restore_best_weights=
            True)
83      #salva o melhor modelo no arquivo best_model.h5
84      mc = keras.callbacks.ModelCheckpoint('best_model_iris.h5', monitor='
            val_accuracy', mode='max', verbose=1, save_best_only=True)
85
86
87      batch_size = 20
88      epochs = 1000
89
```

```
90      history = model.fit(x_train, y_train_binary, batch_size=batch_size,
            epochs=epochs, validation_data=(x_test, y_test_binary),verbose=0,
            callbacks=[es, mc])
91
92      # avalia o modelo criado
93      _, acc = model.evaluate(x_test, y_test_binary, verbose=1)
94      print('> %.3f' % (acc * 100.0))
95      # Curvas de aprendizado
96      summarize_diagnostics(history)
97
98  def run_test() :
99
100     from keras.models import load_model
101     # Carrega a rede salva
102     model = load_model("best_model_iris.h5")
103     # Mostra o modelo sumarizado
104     model.summary()
105
106     prediction=model.predict(x_test)
107     length=len(prediction)
108     y_label=np.argmax(y_test_binary,axis=1)
109     predict_label=np.argmax(prediction,axis=1)
110
111     accuracy=np.sum(y_label==predict_label)/length * 100
112     print("Accuracy do dataset de teste na CNN carregada",accuracy )
113
114     results01=[classes[x] for x in predict_label]
115     print("\nPredicao Obtida:",results01)
116
117     results02=[classes[x] for x in y_label]
118     print("\n\nPredicao Esperada:",results02)
119
120     classes=["setosa","versicolor","virginica"]
121     num_classes, x_train, x_test, y_train, y_test, y_train_binary,
            y_test_binary = load_data()
122
123 option=100
124 while option > 0 :
125     print("\n\nDigite (0) para Sair, (1) para Treinar ou (2) para Testar
            a CNN")
126     option=int(input())
127
128     if option == 1:
129         import datetime
130         start_time = datetime.datetime.now()
131
132         learn_rate=0.01
133         dropout_rate=0.2
134
135         run_train(learn_rate, dropout_rate)
136         end_time = datetime.datetime.now()
137         time_diff = (end_time - start_time)
138         execution_time = time_diff.total_seconds()
139         print("Tempo de execucao em segundos de relogio: %.20lf " % (
                execution_time))
140
141     if option == 2:
142
```

```
143         run_test()
```

Listing 7.1 – Protótipo de CNN para Classificação do Iris dataset

Seção dos códigos utilizados durante a execução do projeto. O seguinte código possui a função early stopping.

```
1 #Import required libraries
2 #import keras #library for neural network
3 import tensorflow as tf
4 from tensorflow import keras
5 import pandas as pd #loading data in table form
6 import numpy as np # linear algebra
7 import sys
8 import numpy
9 from keras.models import Sequential
10 from keras.layers import Dense, Flatten, Conv1D, Dropout
11 from keras.callbacks import ModelCheckpoint
12 from sklearn.model_selection import GridSearchCV
13 from sklearn.metrics import make_scorer
14 from sklearn.metrics import precision_score
15 from sklearn.metrics import accuracy_score
16 from sklearn.metrics import recall_score
17 from sklearn.metrics import fbeta_score
18 #from keras.models import model_from_json
19 from keras import backend as K
20
21 def load_data(filename) :
22
23     #Usar as 3 linhas abaixo no COLAB
24     from google.colab import drive
25     drive.mount("/content/drive")
26     df=pd.read_csv("/content/drive/My Drive/Colab Notebooks/"+filename)
27
28
29     #Usar linha abaixo no Jupyter
30     #df=pd.read_csv(filename)
31
32     numcols=len(df.columns)
33     lastcolumn=df.columns[numcols-1]
34
35     df[lastcolumn]=df[lastcolumn].astype('category').cat.codes #VER AQUI
           -ALTEREI
36
37     colsentrada=[]
38     for i in range(len(df.columns)-1): #-2):  #VER AQUI -ALTEREI
39         colsentrada.append(df.columns[i])
40
41     colsaida=df.columns[len(df.columns)-1]
42
43     numcols=len(df.columns)-1   #VER AQUI -ALTEREI
44
45     # The known number of output classes.
46     num_classes = len(df[colsaida].unique())
47
48     #classes of the problem
49     classes = df[colsaida].unique()
50
51     from sklearn.model_selection import train_test_split
```

```
52
53      X = df[colsentrada]
54      Y = df[colsaida]
55
56      #normaliza os dados de entrada
57      from sklearn.preprocessing import normalize
58      X=normalize(X, axis=0)
59
60      #nas duas linhas abaixo, está utilizando one-hot encoding na saída
61      from keras.utils import np_utils
62      Y = np_utils.to_categorical(Y, num_classes)
63
64      x_train, x_test, y_train, y_test = train_test_split(np.asarray(X),
            np.asarray(Y), test_size=0.3, stratify = Y, shuffle= True)
65      #x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size
            =0.3, stratify = Y, shuffle= True)
66
67      #VER AQUI - INCLUI AS DUAS LINHAS ABAIXO
68      x_train = x_train.reshape(len(x_train), numcols,1)
69      x_test = x_test.reshape(len(x_test), numcols,1)
70
71      print("Data loaded")
72
73      return num_classes, numcols, classes, x_train, x_test, y_train,
            y_test
74
75  def create_model(num_classes, numcols, learn_rate=0.01, dropout_rate
        =0.2):
76      model = Sequential()
77      #model.add(Conv1D(32, (4), input_shape=(numcols,num_classes),
            activation='relu'))
78      model.add(Conv1D(32, (4), input_shape=(numcols,1), activation='relu
            '))
79      model.add(Conv1D(32, (1), activation='relu'))
80      model.add(Dropout(dropout_rate+0.1))
81      model.add(Flatten())
82      model.add(Dense(32, activation='relu'))
83      model.add(Dropout(dropout_rate))
84      model.add(Dense(num_classes, activation='softmax'))
85      opt = tf.keras.optimizers.RMSprop(learning_rate=learn_rate) #keras.
            optimizers.Adam()
86      model.compile(optimizer=opt, loss='categorical_crossentropy',
            metrics=['accuracy'])
87
88      #model.compile(loss=keras.losses.categorical_crossentropy, optimizer
            =keras.optimizers.Adadelta(), metrics=['accuracy'])
89
90      return model
91
92  def run_train(num_classes, numcols, x_train, x_test, y_train, y_test,
        filename):
93      from keras.wrappers.scikit_learn import KerasClassifier
94
95      #ver depois a linha abaixo
96      #from scikeras.wrappers import KerasClassifier
97
98      # create model
99      model = KerasClassifier(build_fn=create_model, verbose=0)
```

```
100
101     # define the grid search parameters
102     num_classes= [num_classes]
103     numcols= [numcols]
104     batch_size = [10]#  , 50, 100] #, 40] #, 60, 80, 100]
105     epochs = [50]#, 100, 200] #, 30]
106     learn_rate = [0.001, 0.01]#, 0.1] #, 0.2, 0.3]
107     dropout_rate=[0.1]#, 0.2]
108
109     param_grid = dict(num_classes=num_classes,numcols=numcols,learn_rate
            =learn_rate, dropout_rate=dropout_rate, batch_size=batch_size,
            epochs=epochs)
110
111     #scoring_fit='accuracy'
112     #grid = GridSearchCV(estimator=model,param_grid=param_grid,scoring=
            scoring_fit,n_jobs=-1, cv=10)
113     grid = GridSearchCV(estimator=model,param_grid=param_grid,n_jobs=-1,
             cv=10)
114
115     # Early stopping
116     from keras.callbacks import EarlyStopping
117     stopper = EarlyStopping(monitor='accuracy', patience=50, verbose=0,
            restore_best_weights=True, mode="auto")
118
119     # Fitting parameters
120     fit_params = dict(callbacks=[stopper])
121
122     print("STARTING GRID SEARCH!!!")
123     #grid_result = grid.fit(x_train, y_train)
124     grid_result = grid.fit(x_train, y_train, **fit_params)
125
126     # summarize results
127     print("Best: %f using %s" % (grid_result.best_score_, grid_result.
            best_params_))
128     means = grid_result.cv_results_['mean_test_score']
129     stds = grid_result.cv_results_['std_test_score']
130     params = grid_result.cv_results_['params']
131     for mean, stdev, param in zip(means, stds, params):
132         print("%f (%f) with: %r" % (mean, stdev, param))
133
134     do_probabilities = False
135     fitted_model = grid_result
136
137     #pred variable store the predict result of the model to compare with
             y_test
138     if do_probabilities:
139         pred = fitted_model.predict_proba(x_test)
140     else:
141         pred = fitted_model.predict(x_test)
142
143     #print("fitted_model=",fitted_model)
144     #print("pred=",pred)
145
146     print("[INFO] evaluating the best model...")
147     bestModel = fitted_model.best_estimator_
148     accuracy = bestModel.score(x_test, y_test)
149     print("accuracy of test data: {:.2f}%".format(accuracy * 100))
150
```

```
151     print("bestModel=",bestModel)
152
153     # Save model
154     '''
155     name=filename.split(sep=".")
156     namefilesave="CNN-"+name[0]+".pickle"
157     import pickle
158     pickle.dump(bestModel, open(namefilesave, "wb"))
159     '''
160
161     tam=len(pred)
162     contequal=0
163     for i in range(tam):
164         if y_test[i,pred[i]]==1:
165             contequal=contequal+1
166
167     acc=(contequal/tam)*100
168     print("Accuracy 02 of test data: {:.2f}%".format(acc))
169
170     return fitted_model, pred
171
172 def run_test(num_classes, numcols, classes, x_train, x_test, y_train,
        y_test) :
173
174     from keras.models import load_model
175
176     # load model and architecture
177     model = load_model("best_model_housing.h5")
178     # summarize model.
179     model.summary()
180
181     prediction=model.predict(x_test)
182     length=len(prediction)
183     y_label=np.argmax(y_test,axis=1)
184     predict_label=np.argmax(prediction,axis=1)
185
186     accuracy=np.sum(y_label==predict_label)/length * 100
187     print("Accuracy of the test dataset on loaded ANN",accuracy )
188
189     #print("Obtained Prediction of Ann:",predict_label)
190     #print("Expected Prediction of Ann:",y_label)
191
192     results01=[classes[x] for x in predict_label]
193     print("\nObtained Prediction of Ann with labels:",results01)
194
195     results02=[classes[x] for x in y_label]
196     print("\n\nExpected Prediction of Ann with labels:",results02)
197
198 filename="yeastdataset.csv"   #<<=== NAME OF DATASET FILE
199
200 #num_classes, x_train, x_test, y_train, y_test, y_train_binary,
        y_test_binary = load_data()
201 num_classes, numcols, classes, x_train, x_test, y_train, y_test  =
        load_data(filename)
202
203 option=100
204 while option > 0 :
205     print("\n\nType (0) to Exit or (1) to Train or (2) to Test the CNN")
```

```
206     option=int(input())
207
208     if option == 1:
209         import datetime
210         start_time = datetime.datetime.now()
211
212         #=================================
213
214
215         #=================================
216
217         fitted_model, pred = run_train(num_classes, numcols, x_train,
                x_test, y_train, y_test, filename)
218
219         end_time = datetime.datetime.now()
220         time_diff = (end_time - start_time)
221         execution_time = time_diff.total_seconds()
222
223         print("Tempo de Treinamento em segundos de relógio: %.20lf " % (
                execution_time))
224
225     if option == 2:
226
227         run_test(num_classes, numcols, classes, x_train, x_test, y_train
                , y_test, filename)
```

Código sem early stopping:

```
1 #Import required libraries
2 #import keras #library for neural network
3 import tensorflow as tf
4 from tensorflow import keras
5 import pandas as pd #loading data in table form
6 import numpy as np # linear algebra
7 import sys
8 import numpy
9 from keras.models import Sequential
10 from keras.layers import Dense, Flatten, Conv1D, Dropout
11 from keras.callbacks import ModelCheckpoint
12 from sklearn.model_selection import GridSearchCV
13 from sklearn.metrics import make_scorer
14 from sklearn.metrics import precision_score
15 from sklearn.metrics import accuracy_score
16 from sklearn.metrics import recall_score
17 from sklearn.metrics import fbeta_score
18 #from keras.models import model_from_json
19 from keras import backend as K
20
21 def load_data(filename) :
22
23     #Usar as 3 linhas abaixo no COLAB
24     from google.colab import drive
25     drive.mount("/content/drive")
26     df=pd.read_csv("/content/drive/My Drive/Colab Notebooks/"+filename)
27
28
29     #Usar linha abaixo no Jupyter
30     #df=pd.read_csv(filename)
31
```

```
32      numcols=len(df.columns)
33      lastcolumn=df.columns[numcols-1]
34
35      df[lastcolumn]=df[lastcolumn].astype('category').cat.codes #VER AQUI
            -ALTEREI
36
37      colsentrada=[]
38      for i in range(len(df.columns)-1): #-2):  #VER AQUI -ALTEREI
39          colsentrada.append(df.columns[i])
40
41      colsaida=df.columns[len(df.columns)-1]
42
43      numcols=len(df.columns)-1   #VER AQUI -ALTEREI
44
45      # The known number of output classes.
46      num_classes = len(df[colsaida].unique())
47
48      #classes of the problem
49      classes = df[colsaida].unique()
50
51      from sklearn.model_selection import train_test_split
52
53      X = df[colsentrada]
54      Y = df[colsaida]
55
56      #normaliza os dados de entrada
57      from sklearn.preprocessing import normalize
58      X=normalize(X, axis=0)
59
60      #nas duas linhas abaixo, está utilizando one-hot encoding na saída
61      from keras.utils import np_utils
62      Y = np_utils.to_categorical(Y, num_classes)
63
64      x_train, x_test, y_train, y_test = train_test_split(np.asarray(X),
            np.asarray(Y), test_size=0.3, stratify = Y, shuffle= True)
65      #x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size
            =0.3, stratify = Y, shuffle= True)
66
67      #VER AQUI - INCLUI AS DUAS LINHAS ABAIXO
68      x_train = x_train.reshape(len(x_train), numcols,1)
69      x_test = x_test.reshape(len(x_test), numcols,1)
70
71      print("Data loaded")
72
73      return num_classes, numcols, classes, x_train, x_test, y_train,
            y_test
74
75  def create_model(num_classes, numcols, learn_rate=0.01, dropout_rate
        =0.2):
76      model = Sequential()
77      #model.add(Conv1D(32, (4), input_shape=(numcols,num_classes),
            activation='relu'))
78      model.add(Conv1D(32, (4), input_shape=(numcols,1), activation='relu
            '))
79      model.add(Conv1D(32, (1), activation='relu'))
80      model.add(Dropout(dropout_rate+0.1))
81      model.add(Flatten())
82      model.add(Dense(32, activation='relu'))
```

```
83      model.add(Dropout(dropout_rate))
84      model.add(Dense(num_classes, activation='softmax'))
85      opt = tf.keras.optimizers.RMSprop(learning_rate=learn_rate) #keras.
            optimizers.Adam()
86      model.compile(optimizer=opt, loss='categorical_crossentropy',
            metrics=['accuracy'])
87
88      #model.compile(loss=keras.losses.categorical_crossentropy, optimizer
            =keras.optimizers.Adadelta(), metrics=['accuracy'])
89
90      return model
91
92  def run_train(num_classes, numcols, x_train, x_test, y_train, y_test,
        filename):
93      from keras.wrappers.scikit_learn import KerasClassifier
94      #ver depois a linha abaixo
95      #from scikeras.wrappers import KerasClassifier
96
97      # create model
98      model = KerasClassifier(build_fn=create_model, verbose=0)
99
100     # define the grid search parameters
101     num_classes= [num_classes]
102     numcols= [numcols]
103     batch_size = [10 , 50, 100] #, 40] #, 60, 80, 100]
104     epochs = [50, 100, 200] #, 30]
105     learn_rate = [0.001, 0.01, 0.1] #, 0.2, 0.3]
106     dropout_rate=[0.1, 0.2]
107
108     param_grid = dict(num_classes=num_classes,numcols=numcols,learn_rate
            =learn_rate, dropout_rate=dropout_rate, batch_size=batch_size,
            epochs=epochs)
109
110     #scoring_fit='accuracy'
111     #grid = GridSearchCV(estimator=model,param_grid=param_grid,scoring=
            scoring_fit,n_jobs=-1, cv=10)
112     grid = GridSearchCV(estimator=model,param_grid=param_grid,n_jobs=-1,
            cv=10)
113
114     print("STARTING GRID SEARCH!!!")
115     grid_result = grid.fit(x_train, y_train)
116
117     # summarize results
118     print("Best: %f using %s" % (grid_result.best_score_, grid_result.
            best_params_))
119     means = grid_result.cv_results_['mean_test_score']
120     stds = grid_result.cv_results_['std_test_score']
121     params = grid_result.cv_results_['params']
122     for mean, stdev, param in zip(means, stds, params):
123         print("%f (%f) with: %r" % (mean, stdev, param))
124
125     do_probabilities = False
126     fitted_model = grid_result
127
128     #pred variable store the predict result of the model to compare with
            y_test
129     if do_probabilities:
130         pred = fitted_model.predict_proba(x_test)
```

```
131     else:
132         pred = fitted_model.predict(x_test)
133 
134     #print("fitted_model=",fitted_model)
135     #print("pred=",pred)
136 
137     print("[INFO] evaluating the best model...")
138     bestModel = fitted_model.best_estimator_
139     accuracy = bestModel.score(x_test, y_test)
140     print("accuracy of test data: {:.2f}%".format(accuracy * 100))
141 
142     print("bestModel=",bestModel)
143 
144     # Save model
145     '''
146     name=filename.split(sep=".")
147     namefilesave="CNN-"+name[0]+".pickle"
148     import pickle
149     pickle.dump(bestModel, open(namefilesave, "wb"))
150     '''
151 
152     tam=len(pred)
153     contequal=0
154     for i in range(tam):
155         if y_test[i,pred[i]]==1:
156             contequal=contequal+1
157 
158     acc=(contequal/tam)*100
159     print("Accuracy 02 of test data: {:.2f}%".format(acc))
160 
161     return fitted_model, pred
162 
163 def run_test(num_classes, numcols, classes, x_train, x_test, y_train,
        y_test) :
164 
165     from keras.models import load_model
166 
167     # load model and architecture
168     model = load_model("best_model_housing.h5")
169     # summarize model.
170     model.summary()
171 
172     prediction=model.predict(x_test)
173     length=len(prediction)
174     y_label=np.argmax(y_test,axis=1)
175     predict_label=np.argmax(prediction,axis=1)
176 
177     accuracy=np.sum(y_label==predict_label)/length * 100
178     print("Accuracy of the test dataset on loaded ANN",accuracy )
179 
180     #print("Obtained Prediction of Ann:",predict_label)
181     #print("Expected Prediction of Ann:",y_label)
182 
183     results01=[classes[x] for x in predict_label]
184     print("\nObtained Prediction of Ann with labels:",results01)
185 
186     results02=[classes[x] for x in y_label]
187     print("\n\nExpected Prediction of Ann with labels:",results02)
```

```
188
189 filename="iris.data"   #<<=== NAME OF DATASET FILE
190
191 #num_classes, x_train, x_test, y_train, y_test, y_train_binary,
    y_test_binary = load_data()
192 num_classes, numcols, classes, x_train, x_test, y_train, y_test  =
    load_data(filename)
193
194 option=100
195 while option > 0 :
196     print("\n\nType (0) to Exit or (1) to Train or (2) to Test the CNN")
197     option=int(input())
198
199     if option == 1:
200         import datetime
201         start_time = datetime.datetime.now()
202
203         #-----------------------------------
204         '''
205
206
207         #-----------------------------------
208
209         fitted_model, pred = run_train(num_classes, numcols, x_train,
                x_test, y_train, y_test, filename)
210
211         end_time = datetime.datetime.now()
212         time_diff = (end_time - start_time)
213         execution_time = time_diff.total_seconds()
214
215         print("Tempo de Treinamento em segundos de relógio: %.20lf " % (
                execution_time))
216
217     if option == 2:
218
219         run_test(num_classes, numcols, classes, x_train, x_test, y_train
                , y_test, filename)
```