Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

LEONARDO DE SOUZA RODRIGUES

Reconhecimento de caracteres manuscritos com Redes Neurais Convolucionais

Assis/SP
2020

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

LEONARDO DE SOUZA RODRIGUES

Reconhecimento de caracteres manuscritos com Redes Neurais Convolucionais

Projeto de pesquisa apresentado ao curso de Ciência da Computação do Instituto Municipal de Ensino Superior de Assis – IMESA e a Fundação Educacional do Município de Assis – FEMA, como requisito parcial à obtenção do Certificado de Conclusão.

Orientando:Leonardo de Souza Rodrigues
Orientador: Dr. Luiz Carlos Begosso

Assis/SP
2020

FICHA CATALOGRÁFICA

R696r RODRIGUES, Leonardo de Souza

Reconhecimento de caracteres manuscritos com redes neurais convolucionais / Leonardo de Souza Rodrigues. – Assis, 2020.

47p.

Trabalho de conclusão do curso (Ciência da Computação). – Fundação Educacional do Município de Assis-FEMA

Orientador: Dr. Luiz Carlos Begosso

1.Redes Neurais 2.Neurônio

CDD006.32

Reconhecimento de caracteres manuscritos com Redes Neurais Convolucionais

LEONARDO DE SOUZA RODRIGUES

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, avaliado pela seguinte comissão examinadora:

Orientador: _____
Dr. Luiz Carlos Begosso

Examinador: _____
Diomara M. R. Barros

Assis/SP
2020

DEDICATÓRIA

Dedico este trabalho a Deus, a minha família e aos meus amigos por toda ajuda e incentivo nestes últimos anos.

AGRADECIMENTOS

Agradeço a Deus por tudo o que tem feito nesses últimos anos de minha vida, tenho a certeza de que se não fosse por Ele, eu não estaria aqui. Também agradeço a minha família que sempre apoiou nos meus estudos e na busca pela felicidade. Não posso esquecer de agradecer a todos que passaram pela minha vida nesses anos de faculdade, agradeço a todos que estagiaram ou conviveram comigo na faculdade, vocês foram pessoas enviadas por Deus em muitas vezes para me fazer prosseguir. Aos meus amigos de infância, aos meus irmãos que seguem os passos de Jesus ao meu lado, aos meus amigos da faculdade, obrigado por me socorrerem em muitos momentos de necessidade e a todos os professores.

RESUMO

Redes neurais artificiais simulam uma rede neural biológica. O cérebro possui capacidade limitada para determinadas tarefas, por consequência podemos utilizar máquinas, que se saem melhor na execução e agilidade, além de automatizar tarefas que são repetitivas. As redes neurais artificiais convolucionais são modelos ideais para problemas relacionados a imagem. Este trabalho tem como objetivo desenvolver uma rede neural convolucional que realize o reconhecimento de caracteres numéricos manuscritos e assim contribuir para a automatização da digitalização e com a redução de manuscritos.

Palavras-chave: Redes Neurais Artificiais; Redes Neurais Convolucionais; Reconhecimento de caracteres;

ABSTRACT

Artificial neural networks simulate a biological neural network. The brain has limited capacity for certain tasks, so we can use machines, which do better in execution and agility, in addition to automating tasks that are repetitive. Convolutional artificial neural networks are ideal models for image-related problems. This work aims to develop a convolutional neural network that performs the recognition of handwritten numeric characters and thus contribute to the automation of digitization and the reduction of manuscripts.

Keywords: Artificial neural networks; Convolutional Neural Networks; Character recognition;

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

Com o surgimento do computador as informações que antes eram documentadas em livros, passaram a ser digitadas e armazenadas em dispositivos de armazenamento. Desde o início da Internet podemos enxergar um grande volume de dados que foram e estão sendo produzidos a cada segundo, ainda que conseguimos automatizar o processo de digitalização, existe uma persistência de manuscritos em uma era digital (SILVA;THOMÉ,2006).

Ensinar computadores a reconhecer os manuscritos pode ser uma solução para automatizar a digitalização de documentos. Para esse tipo de tarefa as Redes Neurais Artificiais (RNAs) podem desempenhar uma importante contribuição.

As RNAs se baseiam na rede neural de um ser humano, o neurônio artificial é um conjunto de operações matemáticas que tem como finalidade mostrar o resultado de um cálculo, uma transformação linear, em outras palavras “uma rede neural é uma máquina que é projetada para modelar a maneira como o cérebro realiza uma tarefa particular”(HAYKIN, 2001, p.28).

O procedimento que a rede neural artificial utiliza para aprender é denominado algoritmo de aprendizagem, que é “um conjunto de procedimentos bem-definidos para adaptar os parâmetros de uma rede neural artificial (RNA) para que a mesma possa aprender uma determinada função” (BRAGA; CARVALHO; LUDERMIR, 2000, p.15).

O cérebro humano, normalmente, possui capacidade limitada para outras tarefas nas quais as máquinas se saem melhor, como resolver uma conta matemática, identificar um objeto em imagens, reconhecer padrões. Contudo existe uma grande diferença de RNAs e programas convencionais, um programa convencional requer a inserção manual de diversas regras e já as RNAs aprendem as regras sozinhas.

Encontram-se na literatura muitos estudos que abordam o reconhecimento de padrões. Oliveira (2018) desenvolveu um trabalho para o reconhecimento de caracteres numéricos em imagens. Menezes et al. (2014) trabalhou no reconhecimento de caracteres manuscritos. Dutra (2011) trabalhou no reconhecimento de padrões de texto. Silva e Thomé (2006) trabalharam no reconhecimento de letras manuscritas e até sistemas feitos por Souza (2013) e Campestrini (2000) para o reconhecimento de caracteres.

As RNAs se aplicam em tudo: auxílio ao diagnóstico médico, mineração de dados, veículos autônomos, avaliações de imagens, classificações de rentabilidade futura de empresas, análise técnica no mercado de ações, agricultura, controle de aparelhos eletrônicos, predições de doenças em medicina, controle para tratamento de água (SILVA; SPATTI; FLAUZINO, 2010). E também classificam padrões onde podem aprender a diferenciar um gato de um cachorro, diferenciar as vozes de pessoas e classificar músicas automaticamente pelos seus gêneros musicais. A aproximação de funções onde podem aprender qual é a relação da função de entrada com a função de saída (HAYKIN,2001). A segmentação em classes, elas tem a capacidade de separar um grande volume de dados em grupos diferentes, baseados em algum critério de semelhança entre eles, nisso elas possuem a capacidade de encontrar padrões totalmente ocultos aos olhos humanos e a predição de séries temporais, dado um conjunto de dados que ocorreram no passado as redes neurais podem tentar predizer o futuro (HAYKIN,2001).

O presente trabalho está situado no contexto de reconhecimento de caracteres manuscritos. Para isso serão utilizadas técnicas de RNA para o reconhecimento de tais caracteres, a linguagem Python para o desenvolvimento do trabalho e as bibliotecas Keras e OpenCV.

1.1 OBJETIVO

O objetivo principal do presente trabalho é desenvolver uma rede neural convolucional que realize o reconhecimento de caracteres numéricos manuscritos. Especificamente, a rede neural convolucional será capaz de identificar uma imagem, previamente digitalizada, de número manuscrito.

O trabalho pretende contribuir com o desenvolvimento da área para o reconhecimento de padrões em textos.

1.2 JUSTIFICATIVA

As RNAs têm uma grande contribuição em pesquisas de diversas áreas como medicina, biologia, química, ecologia, finanças e economia. Em cada artigo cientifico conseguimos observar que a RNA pode ser muito eficaz na resolução de problemas, e também na automatização de tarefas (SILVA; SPATTI;FLAUZINO,2010).

Analisar e processar o grande volume de dados gerados pela Internet, tem sido um grande desafio, pois são muitas informações que podem contribuir para um determinado problema, mas para esta finalidade as informações necessitam de uma investigação feita pelos processos de reconhecimento de caracteres, manuscritos e textos.

O presente trabalho tem como objetivo contribuir para novas pesquisas e para o crescimento da área de reconhecimento de padrões em RNAs, visando ajudar no entendimento e aumentar o interesse no assunto.

1.3 MOTIVAÇÃO

Um dos temas abordados nesse trabalho é o aprendizado de máquina, a maneira como o computador pode aprender tarefas que os seres humanos executam, que muitas vezes são tarefas monótonas e cansativas, aplicando essas tarefas em máquinas tornam-se mais eficientes e rápidas.

Uma das principais motivações para o desenvolvimento desse trabalho é que as RNAs possuem a capacidade fantástica do aprendizado de máquina, podendo realizar previsões, tarefas monótonas, reconhecimento de padrões, automatização de assistência, oferecendo mais benefícios na qualidade de vida, na experiência do usuário e no desenvolvimento tecnológico.

1.4 CONTRIBUIÇÃO

A contribuição efetiva deste trabalho está no desenvolvimento de uma RNA que reconheça caracteres numéricos manuscritos, e também facilitar o entendimento da área. Espera-se que este trabalho possa fomentar o interesse social na área de RNAs, tratamento de imagens, e também na análise e processamento de dados utilizando RNAs.

Desta forma, após a conclusão do trabalho, espera-se que o mesmo possa contribuir para trabalhos vindouros que utilizem a mesma abordagem em reconhecimento de imagens, padrões, manuscritos, caracteres e textos.

2. REDES NEURAIS

De acordo com Haykin (2001), o cérebro é um computador altamente complexo, não-linear e paralelo, que tem a capacidade de organizar seus neurônios de forma a realizar certos processamentos muito mais rapidamente que um computador. A visão humana é um exemplo de tarefa de processamento de informação realizada pelo cérebro, "que realiza rotineiramente tarefas de reconhecimento perceptivo" (HAYKIN, 2001, p.27).

Inspirando-se no sistema nervoso de seres vivos, McCulloch e Pitts criaram em 1943 o primeiro modelo artificial de um neurônio biológico, com a capacidade de aprender, recebendo e processando informações, por meio das RNAs interligadas por um grande número de interconexões (BRAGA; CARVALHO; LUDERMIR, 2000).

As RNAs podem resolver problemas, esse procedimento inicialmente recebe informações na sua entrada e posteriormente gera respostas. E como anteriormente apresentamos, as redes neurais possuem a capacidade de generalizar a informação aprendida.

De acordo com Braga, Carvalho e Ludermir (2000), a generalização está associada a capacidade de a rede aprender através de um conjunto reduzido de exemplos e posteriormente dar respostas coerentes para dados não conhecidos, ou seja, a partir do treinamento realizado com os nossos exemplos a RNA consegue produzir saídas que não estavam presentes no treinamento.

Podemos citar outras características na aplicação de redes neurais, uma delas é adaptação por experiência ou adaptabilidade, que recebe esse nome por ter a capacidade de adaptar os seus pesos sinápticos a partir da apresentação sucessiva de amostras ou exemplos. A classificação de padrões aliada a adaptação por experiência, se torna uma ferramenta muito útil para a classificação adaptativa de padrões, processamento adaptativo de sinais e controle adaptativo (HAYKIN, 2001).

Outra característica é a tolerância a falhas, que segundo Braga, Carvalho e Ludermir (2000), devido ao elevado nível de interconexões entre os neurônios artificiais a rede neural torna-se um sistema tolerante a falhas quando parte de sua estrutura interna é sensivelmente corrompida. O desempenho da rede neural se altera suavemente em condições de operações adversas, seu neurônio é danificado a recuperação de um

padrão armazenado, a princípio o dano é apresentado suavemente pela rede neural ao invés de apresentar um dano bem maior (HAYKIN, 2001).

2.1 NEURÔNIO BIOLÓGICO

A capacidade de sentir e reagir a situações que acontecem ao nosso redor, devem-se ao sistema nervoso que é responsável por captar processar e gerar respostas a estímulos (SANTOS,2020).

A principal unidade de funcionamento do cérebro é uma célula chamada neurônio, podemos defini-la como a estrutura básica do nosso cérebro, que possui aproximadamente de 10 bilhões de neurônios. Um único neurônio não é capaz de sentir, pensar ou lembrar de nada, mas pode ter 60 trilhões de conexões com outros, então o seu poder está na conexão com outros neurônios, formando a tão famosa Rede Neural (HAYKIN, 2001).

A Figura 1 ilustra um neurônio biológico.

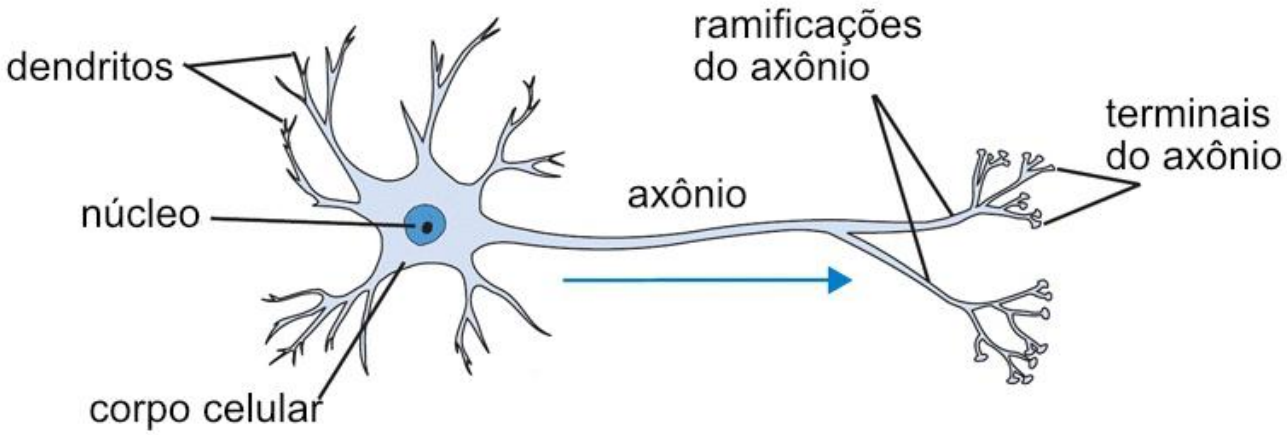

Figura 1: Neurônio Biológico.
Fonte: extraído e adaptado de http://cs231n.github.io/neural-networks-1/

O neurônio parece uma árvore, ele é uma unidade de comunicação, pois recebe e envia informações. As informações chegam pelos dendritos, que seriam os galhos dessa árvore, através de substâncias químicas e sendo chamados de neurotransmissores.

Segundo Pimenta (2018), neurotransmissores são definidos como mensageiros químicos que transportam, estimulam e equilibram os sinais entre os neurônios ou células nervosas e outras células do corpo.

Os dendritos possuem membranas, que tem características bioquímicas que reagem a essas substâncias de duas possíveis formas, algumas excitam as membranas fazendo que disparos elétricos ocorram ou aumentando disparos que já estão ocorrendo, outras inibem a membrana reduzindo a frequência de disparos elétricos (PIMENTA,2018). Podemos observar que há uma conversão de informações químicas para informações elétricas. Quando o neurônio dispara esse impulso elétrico percorre o axônio, o tronco dessa árvore, os impulsos são transmitidos a outros neurônios, passando por outros dendritos e refazendo o mesmo ciclo (BRAGA; CARVALHO; LUDERMIR, 2000).

Os axônios são as linhas de transmissão, e como dito anteriormente sua função é conduzir os impulsos para outros neurônios (HAYKIN, 2001). De acordo com Braga, Carvalho e Ludermir (2000), o ponto de contato entre a terminação axônica de um neurônio e o dendrito de outro é chamado de sinapse, cuja função é atuar como válvulas capazes de controlar a transmissão de impulsos.

2.2 NEURÔNIO ARTIFICIAL

As RNAs são compostas por neurônios artificiais (unidades de processamento de informação) que realizam cálculos de funções matemáticas, são modelos simplificados baseados nos neurônios biológicos. Tais neurônios estão em uma ou mais camadas interligadas por muitas conexões, um neurônio recebe uma informação, processa e passa para outro neurônio (BRAGA; CARVALHO; LUDERMIR, 2000).

Para que as informações sejam processadas, no modelo do neurônio artificial encontramos seis elementos básicos que são:

- Sinais de entrada ou terminais de entrada;
- Pesos sinápticos;
- Combinador linear ou somador;
- Bias ou limiar de ativação;
- Função de Ativação;
- Saída.

A Figura 2 ilustra o modelo de neurônio artificial.

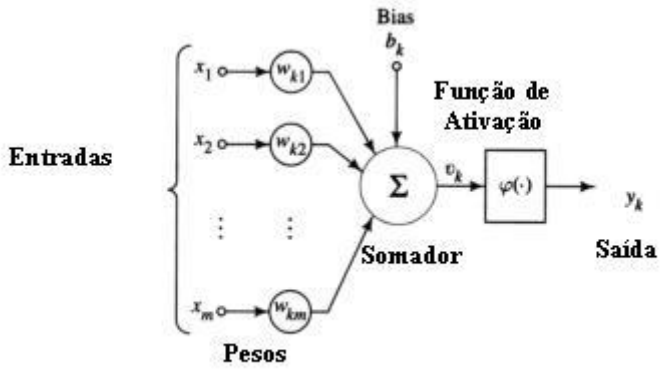

Figura 2: Modelo de um Neurônio Artificial.

Fonte:https://www.gsigma.ufsc.br/~popov/aulas/rna/neuronio_artificial/index.html.

A partir da figura 2, podemos observar que as informações são recebidas pelos terminais de entrada, que representam os dendritos, x1,x2,...,xn. Os pesos, por sua vez representam o comportamento das sinapses que podem ser positivas ou negativas, pois as reações bioquímicas podem excitar ou inibir os disparos elétricos. Braga, Carvalho e Ludermir (2000) destacam que os pesos são multiplicados com os valores dos terminais de entrada, e que eles determinam em que grau o neurônio deve considerar sinais de disparo que ocorrem naquela conexão.

O combinador linear realiza a soma dos valores dos terminais de entrada ponderados pelos pesos sinápticos. O *bias* pode aumentar ou diminuir a entrada líquida da função de ativação, sendo positivo ou negativo (HAYKIN,2001). A função de ativação, segundo Silva, Spatti e Flauzino (2010), tem o objetivo de limitar a saída do neurônio dentro de um

intervalo de valores razoáveis a serem assumidos pela sua própria imagem funcional e a saída é o resultado produzido pelo neurônio em relação ao conjunto de sinais de entrada.

Podemos citar alguns modelos de funções de ativação que são derivados no modelo proposto por McCulloch e Pitts. Na figura 3, podemos observar a função de ativação linear.

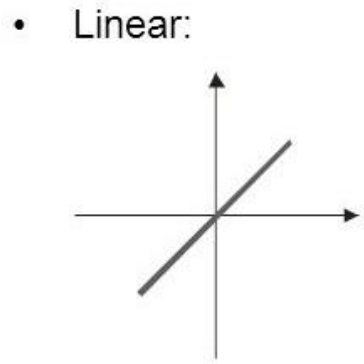

Figura 3: Função de ativação linear.
Fonte: https://slideplayer.com.br/slide/292589/.

Esta função é definida pela equação 1:

$$y = \delta x$$

O delta(δ) é um número real que define a saída linear para os valores de entrada, Y é a saída e X é a entrada (BRAGA; CARVALHO; LUDERMIR, 2000).

Na função degrau, se a saída da sua função for maior que zero, então sua classificação é positiva, caso contrário é negativo.

A figura 4 ilustra a função de ativação degrau.

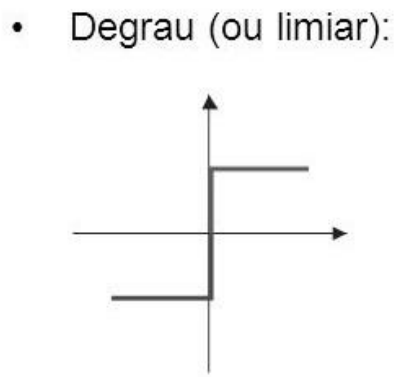

Figura 4: Função de ativação degrau.
Fonte: https://slideplayer.com.br/slide/292589/.

Segundo Haykin (2001, p. 40), a função sigmoide “é definida como uma função estritamente crescente que exibe um balanceamento adequado entre comportamento linear e não linear. Um exemplo de função sigmoide é a função logística”.

A figura 5 ilustra a função de ativação sigmoide ou sigmoidal.

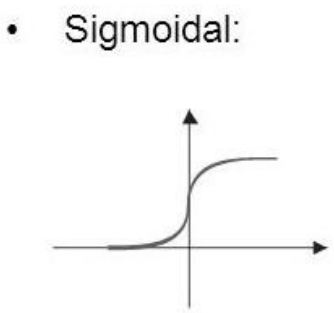

Figura 5: Função de ativação sigmoide.
Fonte: https://slideplayer.com.br/slide/292589/.

2.3 APRENDIZAGEM DE REDES NEURAIS

As RNAs possuem a habilidade de aprender por meio de exemplos e melhorar o seu desempenho através de aprendizagem. Segundo Haykin (2001, p.76), “um conjunto pré-estabelecido de regras bem definidas para solução de um problema de aprendizagem é

denominado algoritmo de aprendizagem", e não há um único algoritmo de aprendizagem, mas temos uma variedade de algoritmos de aprendizagem dentro de um conjunto de ferramentas, cada uma tem a sua vantagem.

Quando utilizamos uma RNA para solução de um problema, ela passa por um processo de aprendizado onde a rede neural identifica padrões, ou seja, características das informações apresentadas a RNA. Podemos encontrar diversos métodos para treinamento de RNAs, construídos em duas formas de aprendizado: aprendizado supervisionado e aprendizado não supervisionado (BRAGA; CARVALHO; LUDERMIR, 2000).

Inicialmente, abordaremos a aprendizagem supervisionada, como o próprio nome já diz a forma de aprendizagem é feita por meio de uma supervisão, ou seja, existe um gestor por trás do processo de aprendizagem regrando e ensinando a RNA. Esse processo é semelhante ao papel dos pais, da mesma forma são gestores que devem ensinar os caminhos que a criança deve seguir, dizendo o certo e o errado (SILVA; SPATTI;FLAUZINO,2010).

Nesse método de aprendizado, a rede tem sua saída comparada com uma saída desejada, que é informada pelo gestor. Toda vez que uma informação é apresentada na camada de entrada, a rede compara essa informação com a resposta desejada, por sua vez, os pesos sinápticos são ajustados para minimização do erro (BRAGA; CARVALHO; LUDERMIR, 2000).

No aprendizado não supervisionado, ao contrário da supervisionada, não existe a presença de um gestor ou supervisor para regrar e induzir a RNA para resposta desejável, que segundo Haykin (2001, p. 90), "são dadas as condições para realizar uma medida independente da tarefa da qualidade da representação que a rede deve aprender, e os parâmetros livre da rede são utilizados em relação a esta medida".

A RNA terá que tentar encontrar sozinha semelhanças e diferenças nos dados que foram informados. Assim, quando não há uma referência, uma regra para a RNA seguir, identificamos que o método utilizado para aprendizagem é o não supervisionado.

2.4 MODELOS DE REDES NEURAIS

Após McCulloch e Pitts em 1943 apresentarem a ideia de RNAs como máquinas computacionais, várias pesquisas foram surgindo ao longo dos anos, como é o caso da aprendizagem *hebbiana*, proposta por Hebb em 1949, que ficou conhecida como a primeira regra de aprendizado auto-organizada (HAYKIN, 2001).

Acrescenta-se também o fato de que o perceptron, idealizado por Rosenblatt em 1958, que foi o primeiro modelo para aprendizagem supervisionada e segundo Silva, Spatti e Flauzino (2010), “é a forma mais simples de configuração de uma rede neural artificial, cujo propósito focava em implementar um modelo computacional inspirado na retina, objetivando-se então um elemento de percepção eletrônica de sinais” e possui apenas uma camada.

A figura 6 ilustra o Perceptron de Rosenblatt.

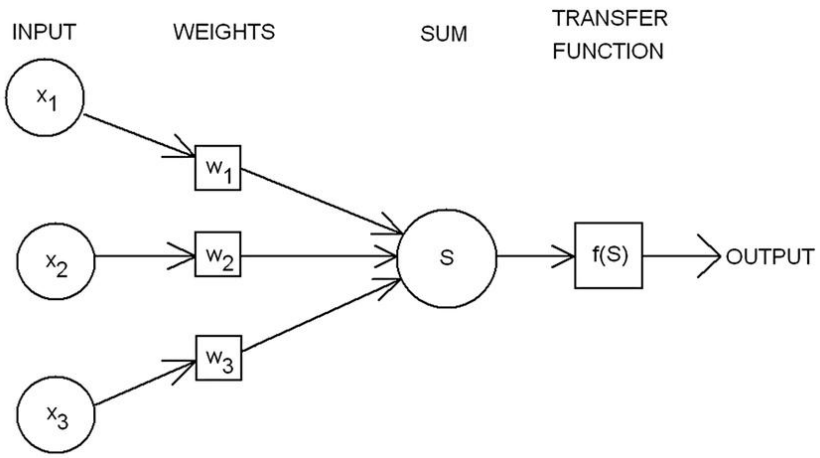

Figura 6: Perceptron de Rosenblatt.

Fonte: https://www.researchgate.net/figure/Rosenblatts-perceptron_fig1_226190708.

Como abordamos anteriormente um neurônio tem maior poder quando é conectado a outros neurônios, e por essa razão o perceptron multi-camadas, ou MLP (*multilayer perceptron*), apresenta um poder computacional maior, e é uma ótima opção para

problemas complexos, onde o perceptron de uma camada não consegue atender a necessidade.

De acordo com Moreira (2018), o MLP é uma rede neural com uma ou mais camadas ocultas com o número indeterminado de neurônios e a camada oculta recebe esse nome por não conseguir prever a saída desejada nas camadas intermediárias.

Em resumo, a rede possui um conjunto de unidades sensoriais o que constitui a camada de entrada, uma ou mais camadas ocultas e uma camada de saída. O treinamento para aprendizagem utilizado é do método supervisionado, mas com um algoritmo, que é o algoritmo de retropropagação do erro ou *backpropagation* (HAYKIN, 2001).

A figura 7 ilustra o Perceptron Multi-Camadas.

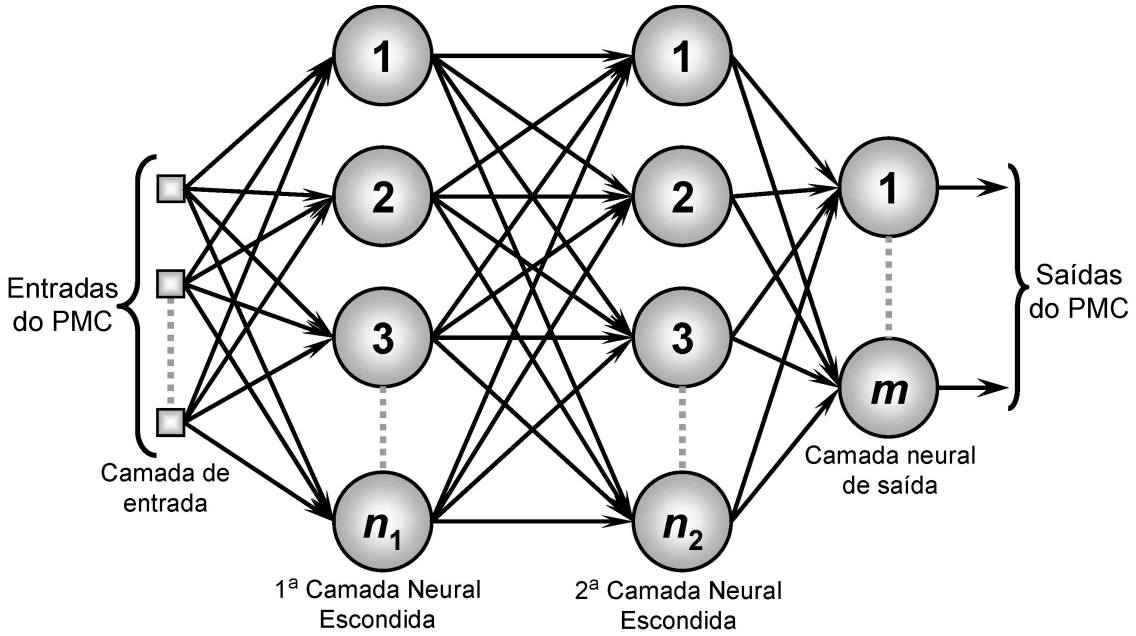

Figura 7: Diagrama do Perceptron Multi-camadas.

Fonte: https://medium.com/ensina-ai/rede-neural-perceptron-multicamadas-f9de8471f1a9.

Podemos observar que a rede neural possui uma camada de entrada com N valores, também com N1 neurônios na primeira camada neural escondida e também N2 neurônios na segunda camada neural escondida e por fim N valores na camada neural de saída.

No processo de treinamento do perceptron multi-camadas utilizando o algoritmo *backpropagation*, o algoritmo utiliza duas fases para realizar o treinamento. Na primeira fase que é aplicada a camada de propagação adiante (*foward*), em que a rede é alimentada na sua camada de entrada, essas informações percorrem toda a rede até a

camada de saída para que possamos obter respostas da rede. Durante a execução dessa fase, os pesos sinápticos permanecem inalterados. E por fim os valores da camada de saída são comparados a saída desejada (SILVA; SPATTI; FLAUZINO,2010).

A partir daí, a segunda fase do método *backpropagation*, chamada de propagação reversa (*backward*) é executada e são feitas alterações nos pesos sinápticos de todos os neurônios ao decorrer da fase (BRAGA; CARVALHO; LUDERMIR, 2000).

3. REDE NEURAL CONVOLUCIONAL

O computador enxerga uma imagem ou um objeto, de uma maneira diferente em relação aos seres humanos, pois para ele uma imagem deve ser representada como uma matriz bidimensional de números, essa matriz é chamada de pixels. Para que o algoritmo possa reconhecer os objetos em imagens a gente utiliza um tipo específico de rede neural, que é a rede neural convolucional (CORNELISSE, 2018).

A RNA convolucional ou *Convolutional Neural Networks* (CNN), é o modelo ideal para os problemas relacionados a imagem, também pode ser utilizada para sistemas de recomendação e processamento de linguagem natural. O interessante da CNN, é que ela aprende características distintas para cada classe, sem qualquer supervisão humana (DERTAT, 2017).

A figura 8 ilustra a arquitetura de uma de rede neural convolucional.

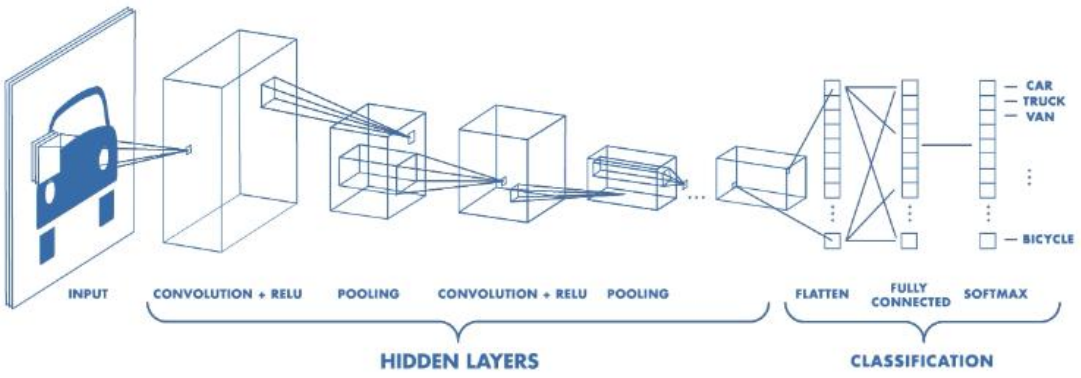

Figura 8: Arquitetura de uma CNN.

Fonte: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html.

A RNA está dividida em três sessões: camadas de entrada, camadas de extração de recursos e a camadas de classificação.

As camadas de entrada são todas as informações que passamos para a CNN, que no caso serão figuras de números manuscritos. Nas camadas de extração de recursos, encontramos justamente a camada que dá nome a RNA, a camada de convolução, ela é conhecida por esse nome, pois é nessa camada que ocorre uma operação chamada convolução. Segundo Cornelisse (2018), o termo convolução refere-se à combinação matemática de duas funções para produzir uma terceira função. Na CNN a convolução é aplicada nos dados de entrada usando o filtro de convolução ou kernel para produzir um mapa de recursos.

Como podemos observar nas figuras abaixo, a convolução é executada ao movermos o filtro (o quadrado verde) nessa matriz de entrada (o quadrado azul), e tal ação realiza uma multiplicação de matrizes gerando o mapa de recursos (o quadrado vermelho).

As figuras 9 e 10 ilustram a operação de convolução.

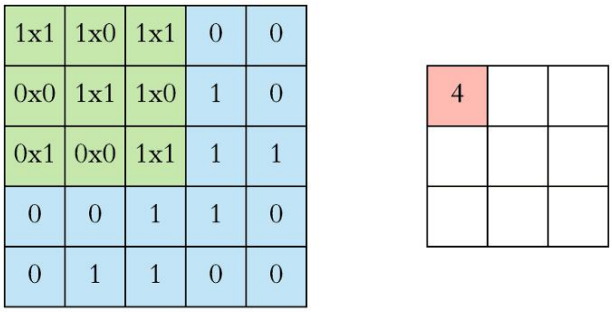

Figura 9: Operação de convolução.

Fonte: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/.

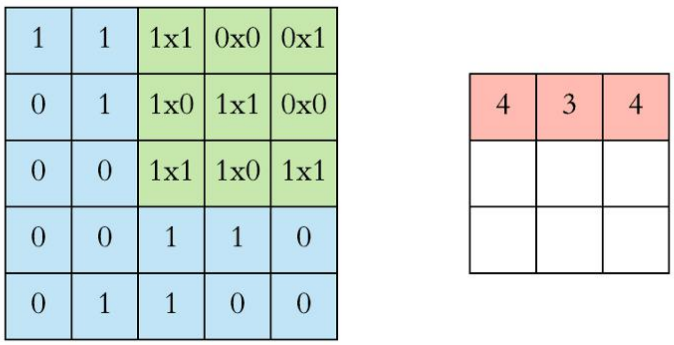

Figura 10: Operação de convolução.
Fonte: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/.

Em outras palavras, observamos que no lado esquerdo existem os valores de entrada dentro de uma matriz, que pode ser uma imagem de entrada, já a direita temos o filtro de convolução e essa convulsão é chamada de combinação 3x3 devido a forma do filtro (DERTAT, 2017).

Na operação de convolução também ocorre uma transformação linear, que pode ser "uma função de ativação do tipo ReLU (*Rectified Linear Unit*) comumente aplicada com o objetivo de promover a não linearidade ao modelo" (GONÇALVES, 2017).

A camada de *pooling* ou agrupamentos, é executada após a operação de convolução, e permite reduzir a dimensionalidade, que reduz o número de parâmetros, o tempo de treinamento e controla o ajuste excessivo (DERTAT, 2017).

A CNN precisa também de uma função de regularização, para isso utilizaremos uma técnica de regularização chamada *Dropout*. A regularização é uma forma de evitar o *overfitting*, que é quando o modelo se ajusta demais aos dados de exemplo e perde a capacidade de generalizar em dados novos que não estavam no conjunto de exemplos. O *Dropout* é uma técnica muito utilizada, que consiste em deliberadamente, de uma forma aleatória, desligar alguns neurônios da rede, informando uma probabilidade que o *Dropout* utilizará para desligar alguns neurônios. Em uma nova iteração do algoritmo, os neurônios que foram desligados aprenderão uma nova característica. Isso evita erros de classificação e melhora a generalização do modelo de redes neurais artificiais (BROWNLEE, 2019).

4. PROPOSTA DE TRABALHO

Para atender os objetivos estabelecidos, o presente trabalho tem por finalidade expor o estudo e o desenvolvimento de uma rede neural artificial convolucional, que possa contribuir para futuros trabalhos relacionados a reconhecimento de padrões, reconhecimento de objetos, reconhecimento de caracteres em RNAs.

Para desenvolvermos a CNN utilizaremos a linguagem Python e as bibliotecas Keras e Opencv. A RNA desenvolvida tem o intuito de reconhecer os caracteres manuscritos que serão informados de entrada. Para variarmos o projeto, estudaremos os seus resultados, alterando parâmetros e observando a sua acurácia.

Dividiremos o capítulo em três seções: tecnologia utilizadas, classes importantes e desenvolvimento do trabalho.

4.1 TECNOLOGIAS UTILIZADAS

A figura 11 ilustra as tecnologias utilizadas no projeto:

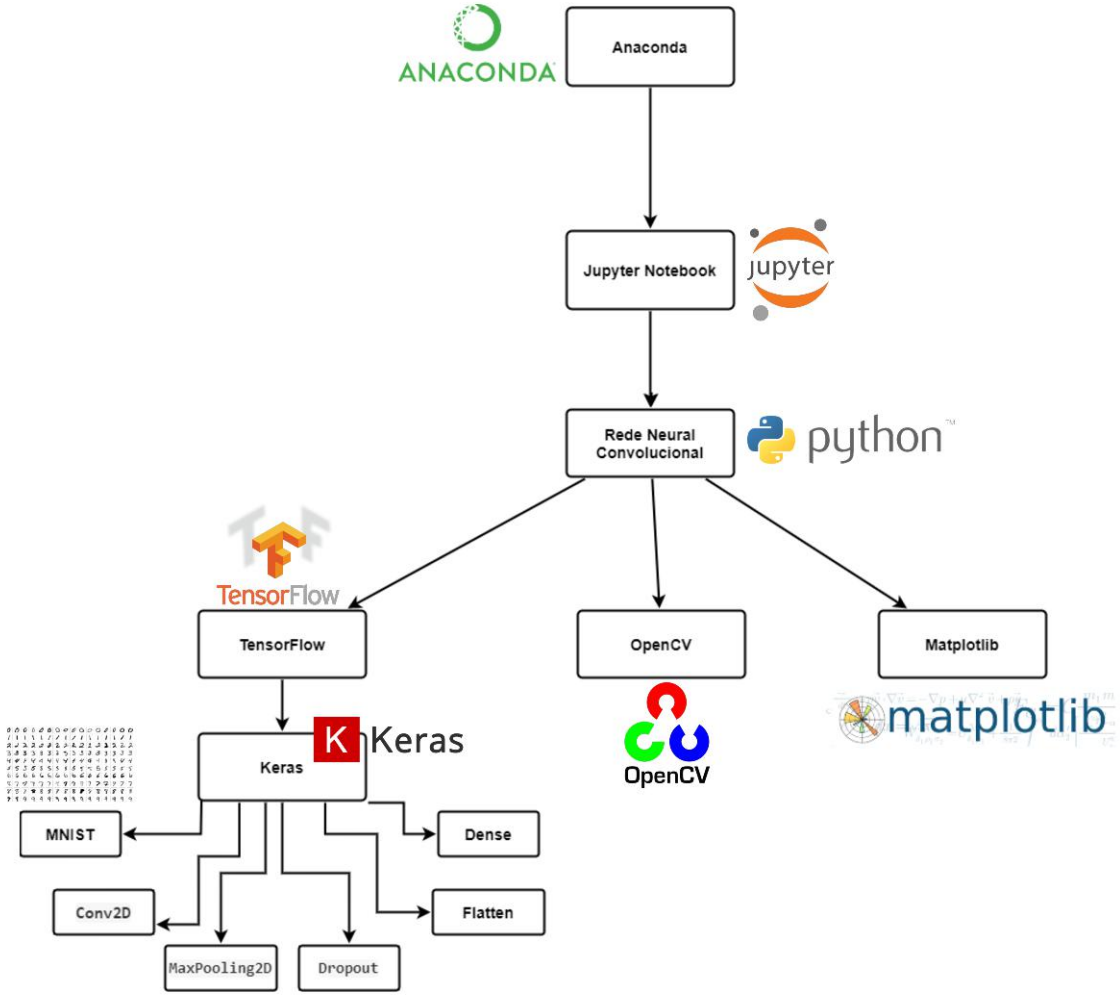

Figura 11: Tecnologias Utilizadas.

Fonte: Próprio autor.

O Anaconda é uma ferramenta utilizada por profissionais de análise de dados, ciências de dados, ciência da computação, estatística, entre outros. No Anaconda pode-se criar facilmente um ambiente para gerenciar versões dos pacotes e programas necessários para o trabalho, agrupando várias ferramentas necessárias para profissionais da área, facilitando o desenvolvimento de projetos de ciências de dados e *machine learning* (ANACONDA, 2020).

Python é uma linguagem de programação, interpretada, de alto nível, orientado a objetos, de *script*, funcional, de tipagem dinâmica e forte. A linguagem tem o intuito de ser fácil e amigável no aprendizado, também é desenvolvida sobre uma licença de código aberto e é

utilizada em várias aplicações como desenvolvimento web, ciências de dados e *scripting* (PYTHON, 2020).

Júpiter Notebook é uma aplicação que permite criar e visualizar documentos que contêm equações, imagens, textos e códigos, a vantagem é que podemos executar o código no próprio documento. O Júpiter é um projeto de código aberto, sem fins lucrativos, da mesma forma que o Anaconda, busca oferecer um suporte, ciência de dados e a computação científica em todas as linguagens de programação (JUPYTER, 2020).

Opencv é uma biblioteca construída para fornecer uma infraestrutura para visão computacional e aprendizado de máquina. Opencv é *open source*, tem mais de 2.500 algoritmos otimizados, algoritmos que podem ser utilizados para detectar e reconhecer rostos, identificar objetos, classificar ações humanas em vídeos, rastrear movimentos de câmera e dentre outras funções (OPENCV, 2020).

O Keras é uma API projetada para seres humanos, pois oferece APIs consistentes e simples, minimiza o número de ações do usuário necessárias, além de fornecer mensagens claras. A ferramenta tem o intuito de tornar fácil a realização de novos experimentos. A API foi construída com base no TensorFlow 2.0, e pode ser escalonada para grandes *clusters* de GPUs (KERAS, 2020).

TensorFlow, é uma plataforma completa de código aberto, para desenvolvimento de *machine learning*. Ela possui várias ferramentas, bibliotecas e recursos. Permite a criação fácil de modelos, com uma produção robusta de *machine learning* em qualquer lugar, pois tem opção de implantar projetos na nuvem, *localhost,* pelo navegador ou em algum dispositivo, e também em qualquer linguagem (TENSORFLOW, 2020).

O MNIST é um banco de dados, com uma grande quantidade de imagens de dígitos manuscritos que são utilizados para vários sistemas de processamento de imagem, por esse motivo foi utilizado nesse projeto. O banco de dados MNIST contém 60000 imagens de treinamento e 10000 imagens de teste (WIKIPEDIA, 2020).

Matplotlib é uma biblioteca utilizada na visualização em Python para criação de visualizações estatísticas, animadas e interativas (MATPLOTLIB, 2020).

O Numpy é um pacote para ciências de dados, que fornece uma estrutura de dados *array* que contém alguns benefícios sobre listas Python, benefícios como ser fácil de usar, tirar acesso mais rápido na leitura e escrita de itens e ser mais eficiente (WILLEMS, 2019).

4.2 CLASSES IMPORTANTES

Detalharemos algumas classes que foram essenciais para o desenvolvimento do presente projeto de CNN.

A figura 12, ilustra as classes importantes para o desenvolvimento da CNN.

```
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
```

Figura 12: Classes importantes.

Fonte: Próprio autor.

A Conv2D cria um filtro convolucional, que convolve com a matriz dos dados de entrada para produzir um tensor de saída.

A MaxPooling2D aplica uma operação do tipo *pooling*, que retira uma amostra do *feature map* (mapa de recursos).

Precisaremos também de uma função de regularização, para isso utilizaremos uma técnica de regularização chamada *Dropout*. A regularização é uma forma de evitar o *overfitting* (Sobreajuste), que é quando o modelo se ajusta demais aos dados de exemplo e por essa razão perde a capacidade de generalizar em dados novos. (WIKIPEDIA, 2017).

A classe *Flatten*, permite converter a saída das camadas convolucionais em um vetor de características, de apenas uma dimensão. Isso precisa ser feito antes da camada densamente conectada. Utilizamos o mapa de recursos antes de entrar na rede densamente conectada, o mapa precisa passar por alteração estrutural, essa alteração consiste em achatar a matriz transformando-a em um único vetor, de uma única dimensão, com todos os valores que estão nessa matriz, este vetor entrará na rede densamente conectada como uma transposta (GONÇALVES, 2017).

Por fim a classe *Dense*, que é a camada densamente conectada, e permite criar uma rede neural regular densamente conectada.

4.3 DESENVOLVIMENTO DO TRABALHO

Para o desenvolvimento do presente projeto foi utilizado o ambiente Anaconda com o Jupyter Notebook para a codificação da CNN. O código no Jupyter Notebook é dividido em células, o que facilita quando queremos executar apenas uma parte do código.

A seguir apresentaremos figuras que representam o código da CNN.

A figura 13, ilustra a célula de importação das bibliotecas.

```
In [1]: import tensorflow as tf
        from keras.datasets import mnist
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.layers import Dropout
        from keras.layers import Flatten
        from keras.layers.convolutional import Conv2D
        from keras.layers.convolutional import MaxPooling2D
        import numpy as np

        import matplotlib
        from matplotlib import pyplot as plt
        from keras.utils import np_utils
        import cv2
        import os

        from keras import backend as K

        #K.tensorflow_backend.set_image_dim_ordering('th')
        K.set_image_data_format('channels_first')

        %matplotlib inline

        seed = 7
        np.random.seed(seed)
```

Figura 13: Célula de importação.

Fonte: Próprio autor.

Nessa célula estão todas as classes que utilizamos para o desenvolvimento da CNN, já detalhamos algumas classes e explicamos algumas bibliotecas como o Keras, Matplotlib, Tensorflow, OpenCV e Numpy. No *dataset* MNIST, as dimensões das imagens estão centradas de 28 x 28. Neste projeto não foram utilizadas as dimensões padrões do TensorFlow e sim as dimensões da biblioteca Theano, que por parâmetro tem (1, 28, 28).

Essas informações são aplicadas nas imagens, um canal na escala de cinza com as dimensões do MNIST 28 x 28.

A figura 14, ilustra a célula de definição dos subconjuntos.

```
In [2]: def get_available_gpus():
            if K.tensorflow_backend._LOCAL_DEVICES is None:
                devices = tf.config.list_logical_devices()
                K.tensorflow_backend._LOCAL_DEVICES = [x.name for x in devices]
            return [x for x in K.tensorflow_backend._LOCAL_DEVICES if 'device:gpu' in x.lower()]

        def load_data():
            (x_train, y_train), (x_cnn, y_cnn) = mnist.load_data()

            x_train = x_train.reshape(x_train.shape[0], 1, 28, 28).astype('float32')

            x_cnn = x_cnn.reshape(x_cnn.shape[0], 1, 28, 28).astype('float32')

            x_train = x_train / 255
            x_cnn = x_cnn / 255

            y_train = np_utils.to_categorical(y_train)
            y_cnn = np_utils.to_categorical(y_cnn)

            return x_train, y_train, x_cnn, y_cnn
```

Figura 14: Definição dos subconjuntos.

Fonte: Próprio autor.

Na segunda célula, dividiremos o *dataset* em dois subconjuntos, o subconjunto de treino e o subconjunto de teste, o subconjunto de treino está definido como o *train* e o subconjunto de teste está definido como cnn. Trabalharemos com rótulos e *features* (recursos).

Estamos utilizando o aprendizado supervisionado na CNN, por isso precisamos de rótulos para dizer a CNN o que os dados informados representam. Um exemplo, é apresentado a imagem do número 5, então utilizamos os rótulos para dizer que essa imagem representa o número 5.

Lembrando que, no MNIST as imagens foram centradas no tamanho 28 x 28, por isso precisamos redimensionar utilizando o *reshape*. Para padronizarmos os valores de saída entre 0 e 1, dividiremos as variáveis rótulos x_train e x_cnn por 255. As variáveis features y_*train* e y_cnn são vetores com valores numéricos, por isso devemos convertê-los para matrizes de valores multi-classes usando a função *categorical*.

A figura 15, ilustra a estrutura das camadas da CNN.

```
In [3]: x_train, y_train, x_cnn, y_cnn = load_data()
num_classes = y_cnn.shape[1]

# O model será exportado para este arquivo
filename='mnistneuralnet.h5'


def model():
    model = Sequential()

    model.add(Conv2D(30, (5, 5), input_shape=(1, 28, 28), activation='relu'))

    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(15, (3, 3), input_shape=(1, 28, 28), activation='relu'))

    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Dropout(0.2))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))

    model.add(Dense(64, activation='relu'))

    model.add(Dense(32, activation='relu'))

    model.add(Dense(num_classes, activation='softmax', name='predict'))

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

    return model
```

Figura 15: Estrutura das camadas da CNN.

Fonte: Próprio autor.

Nessa célula vamos estruturar as camadas da CNN, instanciando uma variável *model* a classe *sequencial*, que receberá as camadas do modelo.

No modelo, adicionaremos duas camadas de convolução, nessa camada, nos parâmetros informamos a quantidade de mapa de recursos que a camada de convolução deve gerar, o tamanho da nossa janela de filtro de convolução, o formato dos dados de entrada e por fim informamos a função de ativação ReLu.

Quando a convolução gerar o mapa de recursos, precisaremos de uma camada *pooling*, que vai retirar uma amostra de cada mapa de recursos, conforme o *pooling* avança nas camadas, reduzirá a dimensionalidade dos dados. Nesse modelo foram adicionadas duas camadas de *pooling*.

Também precisamos incluir a nossa técnica de regularização, a camada *Dropout*, pois precisaremos desligar alguns neurônios da rede de maneira aleatória, para que eles não se ajustem demais aos dados de treino.

Adicionaremos a camada de achatamento, *Flatten*, e também a camada densamente conectada, *Dense*. No modelo adicionamos três camadas densamente conectadas um com 128 neurônios, outra com 64 e outra com 32, todas com função de ativação ReLu.

A última camada também é uma camada densamente conectada, mas com um classificador, uma distribuição de probabilidades e a função de ativação softmax, que servirá para obter a distribuição de probabilidades.

Por fim precisamos consolidar a arquitetura, para isso usaremos o método *compile* para que o Keras junte e forme uma rede, ou seja, ele deverá compilar essa arquitetura. Também reduziremos os erros de classificação com uma função de perda, essa função calcula a diferença entre o que a rede previu, a saída, e o valor verdadeiro, essa diferença deve ser minimizada a cada iteração na rede.

A figura 16, ilustra o treinamento do modelo.

```
In [5]: model.fit(x_train, y_train, validation_data=(x_cnn, y_cnn), epochs=20, batch_size=200)
        model.save_weights(filename)
```

Figura 16: Treinamento do modelo.

Fonte: Próprio autor.

Nesse método é feito o treino e também testaremos a performance do modelo e da generalização. O parâmetro *epochs*, informa a quantidade de iterações de treinos que serão realizados, ou seja, durante X vezes os dados de exemplo serão expostos ao modelo. Também definimos o tamanho do lote, essa função divide os conjuntos de dados em lotes para serem processados, isso melhora a performance de processamento, ao invés de processarmos todos os dados.

A figura 17, ilustra a acurácia do modelo.

```
In [6]: scores = model.evaluate(x_cnn, y_cnn, verbose=0)
        print("\nacc: %.2f%%" % (scores[1]*100))

        acc: 99.34%
```

Figura 17: Acurácia do modelo.

Fonte: Próprio autor.

Mediremos a acurácia do modelo e para termos uma visão mais precisa da exatidão dos valores encontrados, utilizaremos um método do Keras chamado *evaluate*. A varável *scores* receberá a avaliação do *evaluate* sobre os dados de validação x_cnn e y_cnn, e exibiremos na tela a acurácia geral do modelo.

A figura 18, ilustra o teste com dados inéditos.

```
In [7]: img_pred = cv2.imread("number-five.png", 0)
        plt.imshow(img_pred, cmap='gray')

        if img_pred.shape != [28,28]:
            img2 = cv2.resize(img_pred, (28, 28))
            img_pred = img2.reshape(28, 28, -1)
        else:
            img_pred = img_pred.reshape(28, 28, -1)


        img_pred = img_pred.reshape(1, 1, 28, 28).astype('float32')

        img_pred = img_pred/255.0



        pred = model.predict_classes(img_pred)
        pred_proba = model.predict_proba(img_pred)
        pred_proba = "%.2f%%" % (pred_proba[0][pred]*100)
        print(pred[0], " com confiança de ", pred_proba)
```

Figura 18: Teste com dados inéditos.

Fonte: Próprio autor.

Na última célula testaremos a precisão do modelo ao predizer dados inéditos, pois até aqui o modelo testou sobre o conjunto de dados de teste e exemplos, mas queremos testar o modelo em valores novos, por isso nessa última célula informaremos essas figuras de números manuscritos como dados de entrada.

É nessa célula que utilizamos novamente a biblioteca de visão Computacional, OpenCV, para carregar e ler a figura, depois vamos plotar, ou seja, gerar em um gráfico, com a biblioteca Matplotlib e por fim poderemos ver a predição do modelo baseado na figura informada.

.

5. RESULTADOS OBTIDOS

Neste capítulo apresentaremos os resultados obtidos pelo modelo de CNN que desenvolvemos.

Na figura 16 apresentamos a função *fit, e* o *epoch* é um dos parâmetros da função, onde alteramos a quantidade de iteração do treino, será muito importante para a análise de precisão do modelo. A predição e probabilidade são variáveis que também serão importantes para a análise dos resultados.

A figura 19 ilustra o conjunto inicial de figuras que denominaremos de figuras 1 para testes.

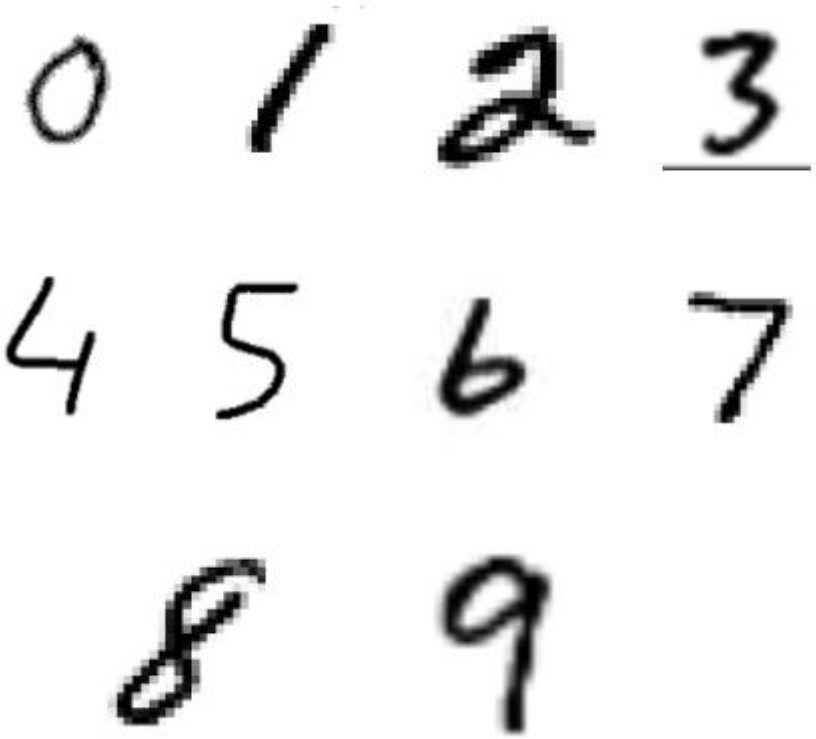

Figura 19: Conjunto de figuras 1 para teste.

Fonte: Próprio autor.

A figura 20 ilustra um outro conjunto de figuras que denominaremos de figuras 2 para testes.

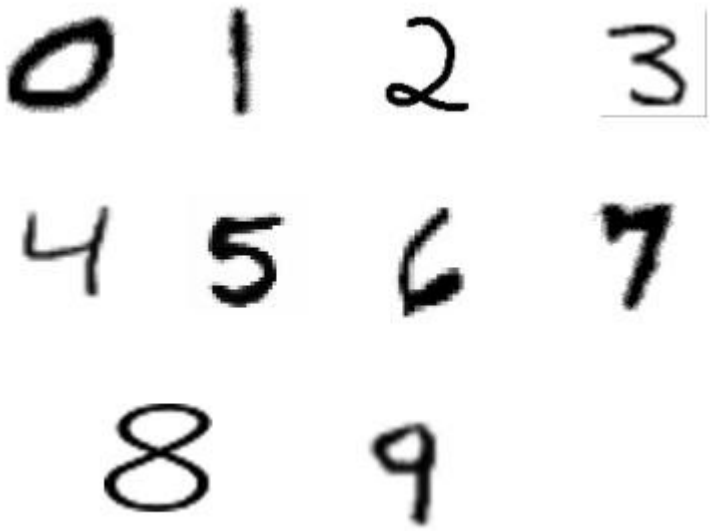

Figura 20: Conjunto de figuras 2 para teste.

Fonte: Próprio autor.

O conjunto de figuras 1 e conjunto de figuras 2 serão utilizados no teste do modelo com dados inéditos, mas as figuras dos conjuntos são apenas uma representação das figuras que utilizaremos, pois devemos informar uma figura individual de cada número para o modelo.

Para observarmos os resultados de uma forma mais adequada, anotamos em uma planilha o que o modelo apontou de predição e probabilidade das figuras informadas.

A figura 21 ilustra o treino com apenas uma iteração.

Treino com 1 Iteração no Conjunto de figuras 1										
Números	0	1	2	3	4	5	6	7	8	9
Reconheceu	8	4	8	8	8	8	8	8	8	8
Probabilidade %	98.74	59.73	94.89	99.30	97.94	97.34	95.86	99.31	83.50	64.60
								Acurácia do modelo		97.41%

Treino com 1 Iteração no Conjunto de figuras 2										
Números	0	1	2	3	4	5	6	7	8	9
Reconheceu	3	8	8	8	8	8	8	8	8	8
Probabilidade %	62.83	97.06	59.73	99.31	83.50	94.89	56.19	70.43	55.79	80.80
								Acurácia do modelo		97.41%

Figura 21: Treino com uma iteração.

Fonte: Próprio autor.

Podemos observar que mesmo com a acurácia do modelo sendo de 97.41%, a predição em figuras que não estavam na fase de treino teve uma taxa de erros quase em 100% das figuras apresentadas, reconhecendo a maioria das figuras como número oito.

A figura 22 ilustra o treino com dez iterações.

Treino com 10 Iterações no Conjunto de figuras 1										
Números	0	1	2	3	4	5	6	7	8	9
Reconheceu	0	1	8	8	4	5	5	7	8	2
Probabilidade %	64.66	48.78	59.59	99.32	83.99	90.06	34.98	99.94	96.74	52.08
								Acurácia do modelo		99.06%

Treino com 10 Iterações no Conjunto de figuras 2										
	0	1	2	3	4	5	6	7	8	9
Reconheceu	3	8	2	8	7	5	6	7	8	9
Probabilidade %	95.24	56.19	96.45	47.96	67.48	99.20	70.43	94.14	55.79	55.79
								Acurácia do modelo		99.06%

Figura 22: Treino com dez iterações.

Fonte: Próprio autor.

Podemos observar que a acurácia do modelo aumentou para 99.06%, e as predições tiveram uma melhora, errando apenas em alguns números: no conjunto 1 os números dois, três, seis e nove, e no conjunto 2 os números zero, um, três e quatro. Estas variações de erros e acertos são esperadas, uma vez que temos dois conjuntos de figuras diferentes.

A figura 23, apresenta um exemplo de predição baseada no número zero. Podemos observar que o modelo obteve sucesso na predição do número, com um grau de confiança na casa de 95.59%.

```
In [15]: img_pred = cv2.imread("zero1.png", 0)
plt.imshow(img_pred, cmap='gray')

if img_pred.shape != [28,28]:
    img2 = cv2.resize(img_pred, (28, 28))
    img_pred = img2.reshape(28, 28, -1)
else:
    img_pred = img_pred.reshape(28, 28, -1)


img_pred = img_pred.reshape(1, 1, 28, 28).astype('float32')

img_pred = img_pred/255.0



pred = model.predict_classes(img_pred)
pred_proba = model.predict_proba(img_pred)
pred_proba = "%.2f%%" % (pred_proba[0][pred]*100)
print(pred[0], " com confiança de ", pred_proba)
```

```
0  com confiança de  95.59%
```

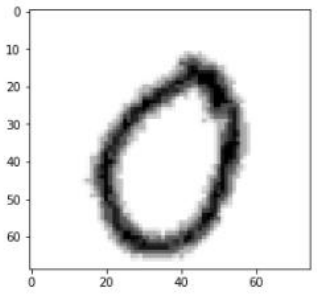

Figura 23: Exemplo de teste com o número zero.

Fonte: Próprio autor.

A figura 24, apresenta outro exemplo de predição, mas é baseada no número dois. Podemos observar que o modelo novamente obteve sucesso na predição do número, e, dessa vez, o grau de confiança na resposta apresentada foi de 99.92%.

In [14]:
```
img_pred = cv2.imread("dois2.png", 0)
plt.imshow(img_pred, cmap='gray')

if img_pred.shape != [28,28]:
    img2 = cv2.resize(img_pred, (28, 28))
    img_pred = img2.reshape(28, 28, -1)
else:
    img_pred = img_pred.reshape(28, 28, -1)


img_pred = img_pred.reshape(1, 1, 28, 28).astype('float32')

img_pred = img_pred/255.0



pred = model.predict_classes(img_pred)
pred_proba = model.predict_proba(img_pred)
pred_proba = "%.2f%%" % (pred_proba[0][pred]*100)
print(pred[0], " com confiança de ", pred_proba)
```

```
2  com confiança de  99.92%
```

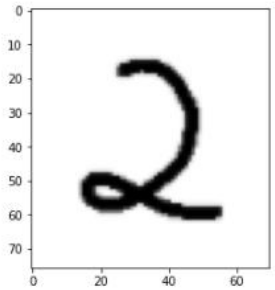

Figura 24: Exemplo de teste com o número dois.

Fonte: Próprio autor.

5.1 TRABALHOS FUTUROS

Com base no estudo e no modelo apresentado, espera-se aproveitar o conteúdo para estudar novas estruturas de modelo. Também sugere-se estudar novas aplicações para o modelo, por exemplo, desenvolver um aplicativo que utilize redes neurais convolucionais para reconhecer e digitalizar receitas ou um aplicativo para reconhecer textos das letras do alfabeto, palavras ou até redações e artigos manuscritos.

Apesar do trabalho abordar redes neurais, o conteúdo está circunscrito como uma subárea da inteligência artificial, e hoje em dia há muitas possibilidades e assuntos para serem estudados.

6. CONCLUSÃO

Após o estudo realizado, e o modelo treinado, podemos concluir que objetivo do modelo de rede neural convolucional reconhecer caracteres manuscritos foi atingido, assim como o reconhecimento de padrões e objetos.

Os resultados apresentados foram feitos com duas variações de iterações. Uma rede neural aprende por meio de uma série de repetições, portanto quanto mais iterações a rede neural possui, maior será sua predição e acurácia.

As redes neurais artificiais foram criadas com o intuito de simular uma rede neural biológica, da mesma forma que a biológica resolve problemas, podemos utilizar as artificiais para resolver os problemas do dia a dia.

O modelo de rede neural convolucional utilizado neste trabalho não segue uma padronização, as camadas podem ser alteradas acrescentando mais camadas convolucionais, camadas de *pooling*, camadas densas ou até retirando as mesmas.

Python, Tensorflow, Keras e OpenCv são tecnologias novas e que estão em crescimento, assim como utilizar o Anaconda como ambiente de desenvolvimento de máquinas de aprendizado e ciência de dados, essas tecnologias são uma ótima alternativa nas aplicações futuras de trabalhos, pois são ágeis, tem uma ótima performance e facilitam muito no desenvolvimento.

7. REFERÊNCIAS

ANACONDA. **Anaconda: The World's Most Popular Data Science Platform**, 2020. Página inicial. Disponível em: <https://www.anaconda.com/>. Acesso em: 19, ago. de 2020.

BABBAR, Sarthak. **"Understanding Dropout and implementing it on MNIST dataset"**; Data Science Blog. Disponível em: <https://data-science-blog.com/blog/2019/05/20/understanding-dropout-and-implementing-it-on-mnist-dataset/>. Acesso em 03 set. 2020.

BRAGA, Antônio de Pádua; CARVALHO, André Ponce de Leon F.; LUDERMIR, Teresa Bernarda . **Redes Neurais Artificiais: Teoria e Aplicações**, 1. Ed. Rio de Janeiro: Editora LTC, 2000.

BROWNLEE, Jason. **A Gentle Introduction to Dropout for Regularizing Deep Neural Networks**; Machine Learning Mastery. Disponível em: <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>. Acesso em 01 ago. 2020.

CAMPESTRINI, Márcio Rodrigo. **Protótipo de um Sistema de Reconhecimento de Caracteres Baseado em Redes Neurais**. 2000. 45p. Trabalho de Conclusão de Curso - Centro de Ciências Exatas e Naturais - Universidade Regional de Blumenau, Santa Catarina, Blumenau, 2000.

CORNELISSE, Daphne. **An intuitive guide to Convolutional Neural Networks**; FreeCodeCamp. Disponível em: <https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/>. Acesso em 21 mar. 2020.

DERTAT, Arden. **Applied Deep Learning - Part 4: Convolutional Neural Networks**; Medium. Disponível em: <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>. Acesso em 21 mar. 2020.

DUTRA, Vagner Pinto. **Redes Neurais e o Reconhecimento de Padrões de texto**. 2011. 78p. Monografia - Universidade São Francisco, São Paulo, Itatiba, 2011.

GONÇALVES, Luís. **Reconhecimento de escrita manual com Redes Neurais Convolucionais**; Medium. Disponível em: <https://medium.com/luisfredgs/reconhecimento-de-escrita-manual-com-redes-neurais-convolucionais-6fca996af39e/>. Acesso em 10 nov. 2019.

HAYKIN, Simon. **Redes Neurais: Princípios e prática**, 2. Ed. Tradução de Paulo Martins Engel., Porto Alegre: Editora Bookman, 2001.

JUPYTER, Project. **Projeto Jupyter: Home**, 2020. Pagina inicial. Disponível em: <https://jupyter.org/>. Acesso em: 19, ago. de 2020.

KERAS. **Keras: the Python deep learning API**, 2020. About. Disponível em: <https://keras.io/>. Acesso em: 19, ago. de 2020.

MARCONDES, José Sérgio. **Conceito de Cronograma: Que é? Definição, Aplicações, Exemplos**. Disponível em: <https://www.gestaodesegurancaprivada.com.br/conceito-de-cronograma-que- edefinicao/>. Acesso em 05 nov. 2019.

MATPLOTLIB. **Matplotlib: Python plotting - Matplotlib 3.3.1 documentation**, 2020. Pagina inicial. Disponível em: <https://matplotlib.org>. Acesso em: 19, ago. de 2020.

MENEZES, C. S.; ALMEIDA, L. L.; SILVA, F. A. DA; PAZOTI, M. A.; ARTERO, A. O. **Redes Neurais Classe Modular Aplicadas No Reconhecimento De Caracteres Manuscritos**. In: COLLOQUIUM EXACTARUM. 6, dezembro, 2014, p. 170-183.

MOREIRA, Sandro. **Rede Neural Perceptron Multicamadas**; Medium. Disponível em: <https://medium.com/ensina-ai/rede-neural-perceptron-multicamadas-f9de8471f1a9>. Acesso em 23 mar. 2020.

NEUROVOX. **Como funciona o cérebro? | Parte 1: Neurônios | PEDRO CALABREZ | NeuroVox 009.** 2016. (10m09s). Disponível em: <https://www.youtube.com/watch?time_continue=8&v=c-RUQPw9rss&feature=emb_title>. Acesso em: 02 mar. 2020.

OLIVEIRA, Eric Lau. **Redes Neurais Artificiais e Reconhecimento de Caracteres Numéricos em Imagens**. 2018. 105p. Trabalho de Conclusão de Curso - Instituto de Ensino Superior - Fundação Educacional do Município de Assis, São Paulo, Assis, 2018.

OPENCV. **OpenCV: About**, 2020. About. Disponível em: <https://opencv.org/about/>. Acesso em: 19, ago. de 2020.

PIMENTA, Tatiana. **Conheça todos os tipos de neurotransmissores e saiba porque eles são importantes para sua saúde**; Vittude. Disponível em: <https://www.vittude.com/blog/neurotransmissores/>. Acesso em 21 mar. 2020.

PYTHON. **About Python: Python.org**, 2020. About. Disponível em: <https://www.python.org/about/>. Acesso em: 19, ago. de 2020.

SANTOS, Vanessa Sardinha dos. **"Sistema nervoso"**; Brasil Escola. Disponível em: <https://brasilescola.uol.com.br/biologia/sistema-nervoso.htm>. Acesso em 21 mar. 2020.

SILVA, Ivan Nunes; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. **Redes Neurais Artificiais para engenharia e ciências aplicadas**, 2. Ed. São Paulo: Editora Artliber, 2016.

SILVA, Viviane S. R.; THOMÉ, Antônio C. G.. **Um Comitê de Redes Neurais para o Reconhecimento de Letras Manuscritas**. 2006. 10p. Monografia. Núcleo de Computação Eletrônica - Universidade Federal do Rio de Janeiro, Rio de Janeiro, Ilha do Fundão, 2006.

SOUZA, Sheila. **Sistema de reconhecimento de caracteres numéricos manuscritos baseado nas redes neurais artificiais paraconsistentes**. 2013. 194p. Dissertação(mestrado) - Faculdade de Medicina da Universidade de São Paulo, São Paulo, São Paulo, 2013.

SOBREAJUSTE. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2017. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Sobreajuste&oldid=50033939>. Acesso em: 03 set. 2020.

TENSORFLOW. **Por que usar o TensorFlow**, 2020. About. Disponível em: <https://www.tensorflow.org/about>. Acesso em: 19, ago. de 2020.

WIKIPEDIA. **"MNIST database"**; Wikipedia. Disponível em: <https://en.wikipedia.org/w/index.php?title=MNIST_database&oldid=962009282>. Acesso em 19 ago. 2020.

WILLEMS, Karlijn. **"Python Numpy Array Tutorial"**; DataCamp. Disponível em: <https://cutt.ly/lfn0Bst>. Acesso em 19 ago. 2020.