UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUIS FERNANDO DE OLIVEIRA UZAI

RECONHECIMENTO ÓPTICO DE CARACTERES MANUSCRITOS USANDO REDES NEURAIS CONVOLUCIONAIS

BAURU
2018

LUIS FERNANDO DE OLIVEIRA UZAI

RECONHECIMENTO ÓPTICO DE CARACTERES MANUSCRITOS USANDO REDES NEURAIS CONVOLUCIONAIS

Trabalho de Conclusão de Curso do Curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Associado Aparecido Nilceu Marana

BAURU
2018

Luis Fernando de Oliveira Uzai RECONHECIMENTO ÓPTICO DE CARACTERES MANUSCRITOS USANDO REDES NEURAIS CONVOLUCIONAIS/ Luis Fernando de Oliveira Uzai. – Bauru, 2018- 45 p. : il. (algumas color.) ; 30 cm.
Orientador: Prof. Associado Aparecido Nilceu Marana
Trabalho de Conclusão de Curso – Universidade Estadual Paulista "Júlio de Mesquita Filho" Faculdade de Ciências
Ciência da Computação, 2018.
1. Visão Computacional 2. Inteligência Artificial 3. Redes Neurais Convolucionais 4. Reconhecimento Óptico de Caracteres 5. Aprendizado de Máquina

Luis Fernando de Oliveira Uzai

RECONHECIMENTO ÓPTICO DE CARACTERES MANUSCRITOS USANDO REDES NEURAIS CONVOLUCIONAIS

Trabalho de Conclusão de Curso do Curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Associado Aparecido Nilceu Marana
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Profª. Drª. Simone das Graças Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Profª. Associada Roberta Spolon
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Bauru, _____ de _____ de _____.

Dedico este trabalho a Deus, que me proporcionou a honra de cursar a graduação.

Agradecimentos

Agradeço primeiramente à Deus, que me sustentou e inspirou até aqui. Agradeço à toda minha família, por todo seu amor. Aos meus amigos e colegas de curso, por momentos tão especiais passados juntos a mim. Ao meu orientador, Prof. Aparecido Nilceu Marana, pelo apoio e ajuda durante o desenvolvimento deste projeto. Aos professores, por todo o conhecimento transmitido. Aos funcionários, por manterem a escola um local agradável e organizado. E à minha namorada, por me apoiar.

Nós somos aquilo que fazemos repetidamente.
Excelência, então, não é um modo de agir, mas um hábito.

Aristóteles

Resumo

O ser humano por natureza possui uma grande capacidade de reconhecer padrões visuais com diversos níveis de complexidade - é capaz de identificar formas, linhas, curvas, etc - e com isso, diferenciar as letras e os dígitos uns dos outros torna-se uma tarefa simples sendo esta uma das primeiras coisas que se aprende quando começa a frequentar a escola. Mas, o computador, diferente do ser humano, não possui essa capacidade. Deste modo, este trabalho teve como objetivo a implementação de um modelo de aprendizado profundo, usando redes neurais convolucionais, para o reconhecimento óptico de caracteres manuscritos. Foram utilizadas bibliotecas recentes como numpy, pandas, matplotlib, Keras. O conjunto de dados usado para o treinamento da rede foi o EMNIST, que contém todas as classes de caracteres. Com a finalização do projeto, os resultados obtidos mostraram-se válidos, levando em conta as acurácias do modelo implementado comparadas às existentes na literatura. Também foi desenvolvido um aplicativo para testar o classificador de caracteres manuscritos, implementado em QT e QML.

Palavras-chave: Redes Neurais Convolucionais, Aprendizado de Máquina, Aprendizado Profundo, reconhecimento Óptico de Caracteres, EMNIST.

Abstract

The human being by nature has a great capacity to recognize visual patterns with varying degrees of complexity - we are able to identify shapes, lines, curves, etc - and thus differentiating letters and digits from each other becomes a simple task this being one of the first things we learned when we started attending school. But the computer, unlike the human being, does not have that capacity. In this way, this work proposes the implementation of a deep learning model, using convolutional neural networks, for the optical recognition of handwritten characters. Recent libraries such as numpy, pandas, matplotlib, Keras, have been used. The data set used for network training was EMNIST, which contains all character classes. With the completion of the project, the results obtained were valid, taking into account the accuracy of the implemented model compared to those in the literature. Also developed was an application for testing the handwritten character classifier, implemented in QT and QML.

Keywords: Convolutional Neural Networks, Machine Learning, Deep Learning, Optical Character Recognition, EMNIST.

Lista de figuras

Figura 1 – Exemplo de um caractere manuscrito. 15
Figura 2 – Imagem em tons de cinza e Imagem em tons de cinza representada em números. 16
Figura 3 – Rede neural simples e Rede Neural de Aprendizado Profundo. 20
Figura 4 – Efeito da aplicação da técnica *Dropout* em uma rede neural. 21
Figura 5 – Exemplo de Convolução 2-D. 23
Figura 6 – Comparação de redes com conexões esparsas e redes totalmente conectadas. . . 24
Figura 7 – CNN com duas camadas de convolução (Ilustração). 24
Figura 8 – Camada de Convolução. 25
Figura 9 – ReLU. 25
Figura 10 – *Max Pooling*. 26
Figura 11 – Frequência dos itens do conjunto de dados EMNIST separado por classes. . . . 28
Figura 12 – Popularidade/Uso dos Frameworks de *Deep Learning*. 29
Figura 13 – Arquitetura LeNet 5 33
Figura 14 – Arquitetura da Rede Neural Convolucional 33
Figura 15 – Ilustração da Arquitetura gerada pelo Keras 35
Figura 16 – Treinamento em relação às épocas 36
Figura 17 – Primeira passada no conjunto de teste 37
Figura 18 – Segunda passada no conjunto de teste 37
Figura 19 – Terceira passada no conjunto de teste 38
Figura 20 – Tela inicial do Aplicativo Mobile 41
Figura 21 – Tela com um “M” desenhado 42
Figura 22 – Tela com “M” submetido ao servidor e com resposta 43

Lista de tabelas

Tabela 1 – Resultados de algoritmos aplicados ao conjunto de dados EMNIST. 38
Tabela 2 – Comparação entre acurácia do artigo do EMNIST e a deste trabalho. . . . 44

Lista de códigos

1 Função que rotaciona *numpy images* 30
2 Carregando o Conjunto de Dados com pandas 31
3 Iterando sobre o conjunto de dados 31
4 Preparação dos dados para o treinamento 31
5 Arquitetura da Rede Neural Convolucional 34
6 Treinamento da rede 36
7 Servidor para predição de caractere 38

Lista de abreviaturas e siglas

ML	*Machine Learning*
IA	*Inteligência Artificial*
DL	*Deep Learning*
CNN	*Convolutional Neural Network*
OCR	*Optical Character Recognition*

Lista de símbolos

$\int$	Símbolo da Integral
$\sum$	Letra Grega *Sigma*

Sumário

Lista de códigos **11**

1 INTRODUÇÃO **15**
1.1 Problema **16**
1.2 Justificativa **17**
1.3 Objetivos **17**
1.3.1 Objetivos Gerais 17
1.3.2 Objetivos Específicos 18
1.3.3 Organização da monografia 18

2 FUNDAMENTAÇÃO TEÓRICA **19**
2.1 Processamento de Imagem **19**
2.2 Aprendizado de Máquina **19**
2.2.1 Aprendizado de Máquina Supervisionado 19
2.2.2 Aprendizado de Máquina Não Supervisionado 19
2.2.3 Aprendizado Profundo 20
2.3 Redes Neurais Convolucionais **21**
2.3.1 Camada de Convolução 24
2.3.2 Função de Ativação ReLU 24
2.3.3 Função de *Pooling* 25

3 DESENVOLVIMENTO **27**
3.1 Base de Dados EMNIST **27**
3.2 Keras **28**
3.3 Desenvolvimento da Rede Neural Convolucional **30**
3.3.1 Preparação para o Treinamento 30
3.3.2 Arquitetura 32
3.3.3 Treinamento 35
3.3.4 Teste 37
3.4 Servidor **38**
3.5 QT **40**

4 CONCLUSÃO **44**
4.1 Trabalhos Futuros **44**

REFERÊNCIAS **45**

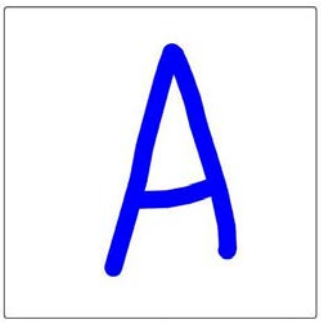
1 Introdução

O sentido humano de visão é inacreditavelmente avançado. Em menos de um segundo, o ser humano consegue identificar objetos em seu campo de visão, sem muita atenção ou hesitação. Não apenas consegue nomear os objetos que está olhando, ele também percebe a profundidade, distingui perfeitamente os contornos, e separa os objetos do fundo. De alguma forma os olhos dos seres humanos trazem apenas uma imagem em forma de dados, mas o seu cérebro transforma essa informação em uma que tem um significado maior - linhas, curvas, e formas (HUBEL; DAVID; WIELSEL, 1968).

"Reconhecimento Ótico de Caractere" que tem origem do inglês *"Optical Character Recognition"* (OCR) é a tecnologia que tem como objetivo reconhecer texto que está inserido em uma imagem digital. Um dos usos comuns é para reconhecer texto em documentos escaneados, mas tem muitos outros usos práticos no dia a dia (CHRISTENSSON, 2018).

OCR foi originalmente criado para reconhecer textos impressos (caracteres oriundos de computador), mas hoje em dia o uso foi estendido para caracteres manuscritos (escritos à mão). Um dos exemplos que a maioria das pessoas usam é o Google Tradutor, que possui uma função de escrever as palavras que eventualmente serão traduzidas. (CHRISTENSSON, 2018). Um exemplo de um caractere manuscrito feito em um aplicativo é dado na Figura 1.

Figura 1 – Exemplo de um caractere manuscrito.



Fonte: Elaborada pelo Autor.

Inteligência Artificial (IA) tem como foco fazer as máquinas "pensarem" e se comportarem de maneira inteligente. Essas máquinas são controladas pelo software que existe dentro delas, então a IA tem de ser feita com algoritmos inteligentes, que as controlam da maneira mais eficiente possível. Inteligência Artificial é a ciência de descobrir teorias e metodologias que ajudam máquinas a entenderem o mundo e dada uma situação, agirem da mesma maneira que os humanos (JOSHI; PRATEEK, 2017).

Aprendizado Profundo é um subconjunto de um campo da Inteligência Artificial chamado Aprendizado de Máquina, o qual é baseado na ideia de aprender de exemplos. No

aprendizado de máquina, ao invés do computador ter uma lista massiva de regras para resolver o problema, tem-se um modelo que pode estimar a partir de exemplos, e um pequeno conjunto de instruções para modificar e otimizar o modelo quando ele apresenta erros. O que se espera é que, ao longo do tempo, um bom modelo possa ser capaz de resolver problemas com alta precisão (BUDUMA, 2017).

Como outros tipos de redes neurais artificiais, uma rede neural convolucional tem uma camada de entrada, uma camada de saída e várias camadas ocultas. Algumas dessas camadas são convolucionais e os seus resultados são passados para as camadas sucessivas, simulando, desse modo, algumas ações realizadas pelo córtex visual humano (TECHOPEDIA, 2018).

1.1 Problema

Quando o ser humano é criança e está aprendendo a ler, a primeira coisa que se aprende é a identificar as letras do alfabeto e os dígitos. Depois disso, independente da forma que está escrita (levando em conta grafia entendível) ou mostrada, a criança pode dizer qual letra é sem muita dificuldade. Mas, para um computador identificar um caractere a partir de uma imagem é muito difícil, pois a imagem é representada para ele por meio de uma matriz de *pixels* $n \times m$, quando a imagem é em tons de cinza. Já, quando a imagem é colorida, a representação é *RGB* (*Red, Green e Blue*) e a matriz de *pixels*, por ser profunda, é: $n \times m \times 3$.

Como mostrado na Figura 2, a representação de imagens (em tons de cinza) em um computador é apenas em números, e é um grande desafio transformar essa matriz de *pixels* em algo que o computador possa analisar e tomar alguma decisão.

Figura 2 – Imagem em tons de cinza e Imagem em tons de cinza representada em números.



Fonte: <https://goo.gl/syC2Pp>.

Em relação aos caracteres manuscritos oriundos de imagens, a tarefa de classificação/identificação fica ainda mais difícil, pois as pessoas escrevem de maneiras totalmente diferentes uma das outras e a mesma pessoa muito possivelmente escreve diferente na maioria das vezes. Além do que, o custo computacional para processar uma imagem é grande, pois para se ter

uma visão humana melhor da imagem, a resolução precisa ser grande, aumentando o tamanho da matriz, que chega na casa dos *MegaBytes*.

1.2 Justificativa

Nos últimos tempos, por meio da tecnologia e ciênca, a humanidade tem avançado de uma maneira muito rápida, e isto sem dúvidas é por conta da grande evolução da tecnologia, tanto física, quanto teórica. E um dos fatos que vem acontecendo é que a distância entre homem e máquina está cada vez menor, ou seja, o computador está cada vez mais realizando de forma autônoma, usando algoritmos de Inteligência Artificial, tarefas que antes somente humanos faziam.

O campo de trabalho dos computadores vai além de tarefas manuais (montar carros, por exemplo). Atualmente, existem computadores capazes de simular os sentidos humanos, como a fala, audição e visão. A Computação está sendo usada para verificar, por meio de imagens, as emoções das pessoas, por exemplo, e a partir disso, otimizar a eficácia dos anúncios e propagandas.

A possibilidade de um computador analisar e conseguir ler um documento manuscrito oriundo de uma imagem, abre um leque de possibilidades na inteligência artificial, remetendo à diminuição da distância entre humano e máquina. A possibilidade do computador ler e abstrair, deixa-o mais parecido com o ser humano, tornando possível a realização de trabalhos mais difíceis e que antes não eram realizados por máquinas. Um dos exemplos é a capacidade de um computador ler documentos e tomar decisões que antes eram tomadas por humanos, mas com uma rapidez muito maior.

Para a leitura de um texto manuscrito a partir de uma imagem digital, é necessário o Reconhecimento Óptico de Caracteres Manuscritos que compõem este texto.

1.3 Objetivos

Este trabalho visa aplicar técnicas da Inteligência Artificial, em específico, Redes Neurais Artificiais Convolucionais, para o problema de classificação de caracteres manuscritos.

1.3.1 Objetivos Gerais

Desenvolver um modelo de Inteligência Artificial, capaz de classificar caracteres manuscritos de **0**-**9**, **a**-**z** e **A**-**Z**, com a implementação de uma Rede Neural Convolucional e um aplicativo para testes.

1.3.2 Objetivos Específicos

a) Estudar Visão Computacional e Reconhecimento Óptico de Caracteres;

b) Estudar Aprendizado Profundo e Redes Neurais Convolucionais;

c) Desenvolver a arquitetura da Rede Neural Convolucional para a classificação de caracteres manuscritos;

d) Treinar a Rede Neural Convolucional;

e) Desenvolver um aplicativo para experimentos;

f) Avaliar Resultados.

1.3.3 Organização da monografia

Esta monografia está organizada da maneira que se segue.

O Capítulo 2 contém a fundamentação teórica, apresentando vários conceitos sobre Aprendizado de Máquina e Redes Neurais Convolucionais utilizados.

O Capítulo 3 relata todo o processo de desenvolvimento, desde a obtenção, tratamento de dados, a execução dos algoritmos, as ferramentas utilizadas e os resultados obtidos.

Por fim, o Capítulo 4 apresenta a conclusão e sugestão de trabalhos futuros baseados nesta monografia.

2 Fundamentação Teórica

2.1 Processamento de Imagem

Uma imagem pode ser definida como uma função bidimensional, $f(x, y)$, onde x e y são coordenadas no espaço, e a amplitude de f em qualquer par de coordenadas (x, y) é denominado intensidade ou "nível de cinza" naquele ponto. Quando os valores das coordenadas e da intensidade são valores finitos e discretos pode haver o processamento digital da imagem (GONZALES; WOODS, 2008)

2.2 Aprendizado de Máquina

Aprendizado de Máquina é o ramo da Ciência da Computação que utiliza experiências passadas para aprender e usa o conhecimento adquirido para tomar decisões. Aprendizado de Máquina é a interseção da Ciência da Computação, Engenharia e Estatística. A finalidade do Aprendizado de Máquina é criar um modelo genérico que detecta padrões e regras não conhecidas a partir de exemplos conhecidos (DANGETI, 2017).

O Aprendizado de Máquina é separado em duas categorias, sendo elas o **Aprendizado de Máquina Supervisionado** e **Aprendizado de Máquina Não Supervisionado**.

2.2.1 Aprendizado de Máquina Supervisionado

No Aprendizado de Máquina Supervisionado, dentro do conjunto de dados, tem-se os dados e as classes de cada item. E posteriormente (depois do treinamento), dado um item, o modelo ser capaz de colocá-lo em alguma das classes disponíveis (BUDUMA, 2017). O Aprendizado de Máquina Supervisionado é separado em classificação e regressão.

Classificação: Quando o problema analisado tem a resposta em classes bem definidas e discretas. Exemplo: dado tamanho e idade do tumor, classificar em câncer ou não câncer.

Regressão: O problema submetido não tem classes bem definidas e a resposta é sempre um número real que pode estar em um intervalo muito grande, ou seja, problema contínuo. Exemplo: predizer o preço de uma casa, dadas as características (bairro, tamanho, etc).

2.2.2 Aprendizado de Máquina Não Supervisionado

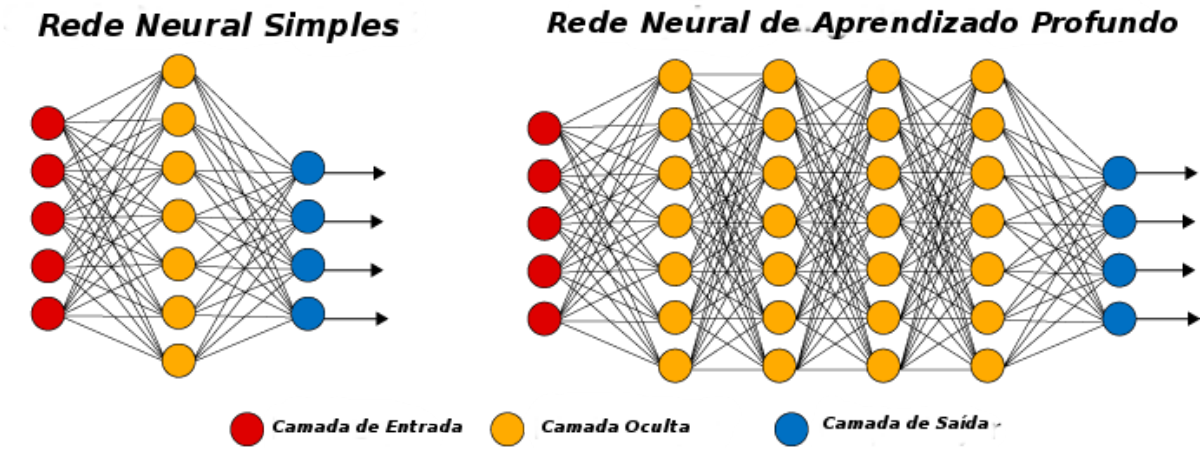
Com o Aprendizado de Máquina Não Supervisionado, a máquina é treinada com dados sem informações de saída, e o objetivo é agrupar os elementos baseados em características

similares ou em características que os tornam únicos. Esses grupos são chamados de *clusters*. Com o Aprendizado Não Supervisionado, o objetivo não é procurar por uma resposta única, aproximada, específica ou certa. Ao invés disso, é relacionar cada elemento do grupo pela similaridade das características ou do comportamento entre os membros, e também pelas diferenças com os outros grupos (CÁNEPA, 2016).

2.2.3 Aprendizado Profundo

O aprendizado profundo é um conjunto de algoritmos de aprendizado de máquina que tentam assimilar em vários níveis, correspondendo a diferentes graus de abstração. Os níveis nesses modelos estatísticos aprendidos correspondem a graus distintos de conceitos, em que os de alto nível são definidos a partir de níveis inferiores, e os mesmos conceitos de nível inferior podem ajudar a definir muitos conceitos de nível superior. Aprendizado Profundo envolve a aprendizagem de múltiplos níveis de representação e abstração que ajudam a dar sentido aos dados, como imagens, sons e textos. (NETO et al., 2017). Na Figura 3 são apresentadas duas arquiteturas de redes neurais. A arquitetura da esquerda refere-se a uma rede neural simples, com uma camada de entrada, uma de saída e apenas uma camada oculta. A arquitetura da direita refere-se a uma rede neural de Aprendizado Profundo, composta por uma camada de entrada, uma camada de saída e quatro camadas ocultas.

Figura 3 – Rede neural simples e Rede Neural de Aprendizado Profundo.



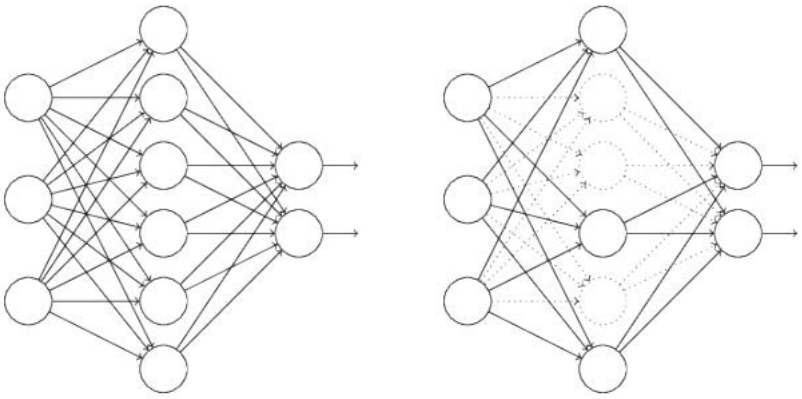
Fonte: <http://deeplearningbook.com.br>.

Dropout

Dropout é uma técnica aplicável ao treinamento de uma rede neural para reduzir o risco de *overfitting* (vício da rede nos dados do treinamento). Na fase de propagação adiante, antes de um padrão ser apresentado à camada intermediária de índice k, cada entrada de um vetor binário m(k) é gerada por meio de uma distribuição uniforme com parâmetro p (p=50% é um

valor usual). Esse vetor é então usado como uma máscara para desativar algumas das unidades da camada k (NETO et al., 2017).

Figura 4 – Efeito da aplicação da técnica *Dropout* em uma rede neural.



(a) Estágio inicial (b) Estágio final

Fonte: (NETO et al., 2017)

Como observado na Figura 4, alguns neurônios da camada da rede neural foram desativados durante uma fase do treinamento com probabilidade de 50%. Ao fazer isso, a rede fica consideravelmente mais genérica, pois o contexto de treinamento não é o mesmo em todas as etapas. Em todas as etapas, a rede é reinicializada com todos os neurônios ativados e então a técnica de *Droupout* é novamente aplicada.

2.3 Redes Neurais Convolucionais

Redes Neurais Convolucionais ou CNN (*Convolutional Neural Networks*) é um tipo de rede neural especializada em processar dados que tem como característica topologia em grades, ou seja, dados representados em 1-D, 2-D, ..., n-D. O nome "Redes Neurais Convolucionais" vem por conta da operação matemática chamada **convolução** que é aplicada nas camadas da rede. Convolução é um tipo de operação linear. CNN é uma rede que usa convolução no lugar da matriz comum de multiplicação, em pelo menos uma de suas camadas (GOODFELLOW; BENGIO; COURVILLE, 2016).

Convolução é um operador linear que, a partir de duas funções dadas, resulta numa terceira que mede a soma do produto dessas funções ao longo da região subentendida pela superposição delas em função do deslocamento existente entre elas:

$$s(t) = \int x(a)w(t-a)da. \tag{2.1}$$

A operação de Convolução geralmente é representada com um asterisco:

$$s(t) = (x * w)(t). \tag{2.2}$$

Na terminologia de Redes Neurais Convolucionais, o primeiro argumento (no exemplo, a função x) para a convolução é frequentemente chamado de entrada, e o segundo argumento (no exemplo, a função w) é chamado de `Kernel`. A saída, muitas vezes, é chamada de `feature map`.

Para intervalos discretos, a definição pode ser representada por meio de somatório, ao invés da integral, como segue:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{2.3}$$

No aprendizado de máquina aplicado à imagens, os intervalos, além de serem discretos, têm dimensão maior que 1-D. Para uma imagem de duas dimensões I como entrada, e também um *Kernel* K também bi-dimensional, a fórmula fica:

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{2.4}$$

A convolução é comutativa, o que significa que de maneira equivalente, pode-se escrever,

$$S(i,j) = (K * I)(i,j) = \sum_{m}\sum_{n} I(i-m, j-n)K(m,n) \tag{2.5}$$

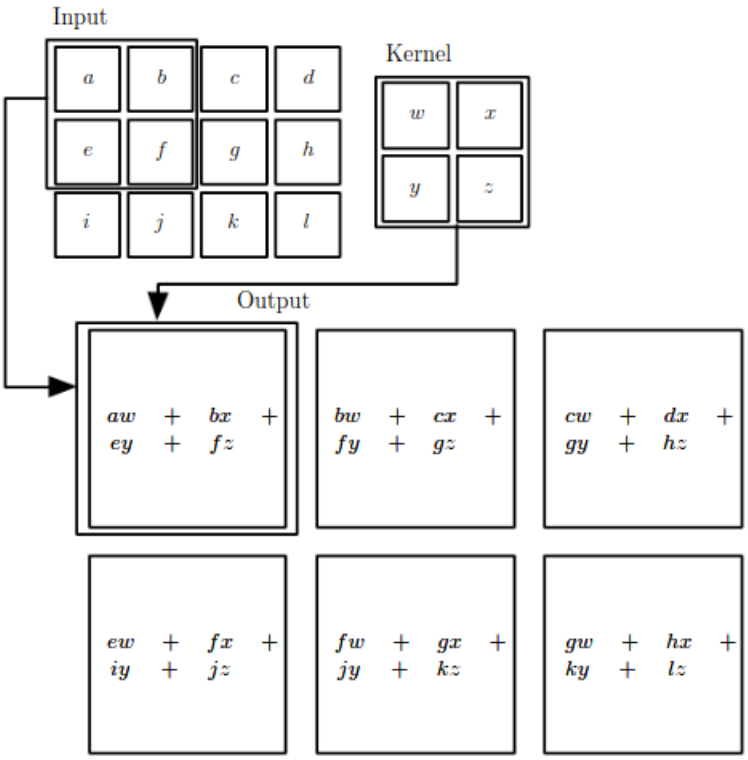
Usualmente, a última fórmula, por ser mais simples, é frequentemente implementada nas bibliotecas de Aprendizado de Máquina, porque há pequena variação nos intervalos dos valores de m e n.

Na Figura 5 vemos o exemplo de uma entrada com duas dimensões, um *Kernel* de tamanho 2×2 e os *feature maps* (saídas) da convolução. Também é mostrado um exemplo de como a convolução funciona. Têm-se quadrados com setas para indicar a ordem das operações, ou seja, a multiplicação da entrada pelo *Kernel*.

Uma importante característica das Redes Neurais Convolucionais, e que a deixa mais veloz, é a de ter **interações esparsas** entre as camadas.

Interações esparsas (também conhecida como conexões esparsas ou pesos esparsos) fazem com que o *Kernel* seja menor do que a entrada. Por exemplo, quando uma imagem é processada, a entrada tem milhares ou milhões de pixels, mas, com a CNN, consegue-se detectar características como bordas, porém, com *kernels* que ocupam dezenas ou centenas de pixels. Isso significa que precisa-se armazenar menos parâmetros, o que reduz a quantidade necessária de memória do modelo e melhora a eficiência estatística. Também significa que

Figura 5 – Exemplo de Convolução 2-D.



Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

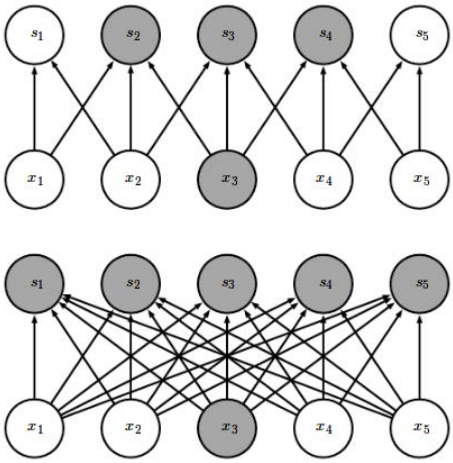
computar a saída requer menos operações. As melhorias na eficiência são bem grandes. Se têm m entradas e n saídas, então, a matriz de multiplicação necessita de $m \times n$ parâmetros, e a execução na prática tem como complexidade de $O(m \times n)$. Se limitar os números de conexões, tem-se k saídas, então, as conexões esparsas requerem apenas $k \times n$ operações e a complexidade fica $O(k \times n)$ (GOODFELLOW; BENGIO; COURVILLE, 2016). Uma ilustração das conexões esparsas é dada na Figura 6.

Na Figura 6 vê-se que a entrada x_3 tem conexão apenas com os neurônios s_2, s_3 e s_4 da camada S. Ao contrário da rede neural comum que dada uma entrada qualquer x_n, ela está conectada com todos os neurônios da camada S. Visualmente, o número de operações na CNN vai ser menor do que em uma rede neural comum.

Uma outra característica das CNN que vem das conexões esparsas é que ao longo das camadas, toda entrada vai estar conectada indireta ou diretamente à um neurônio específico da camada.

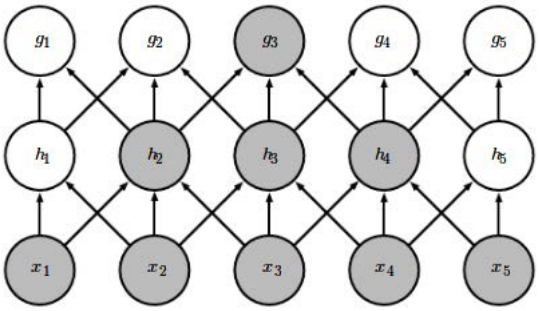
Vê-se na Figura 7, que em uma das operações, a entrada excita 3 neurônios inicialmente, que excitam apenas um. Com isso, visualmente pode-se concluir que se a rede tiver duas ou mais camadas, a entrada vai estar conectada a pelo menos um neurônio específico de uma das camadas mais profunda.

Figura 6 – Comparação de redes com conexões esparsas e redes totalmente conectadas.



Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

Figura 7 – CNN com duas camadas de convolução (Ilustração).



Fonte: (GOODFELLOW; BENGIO; COURVILLE, 2016)

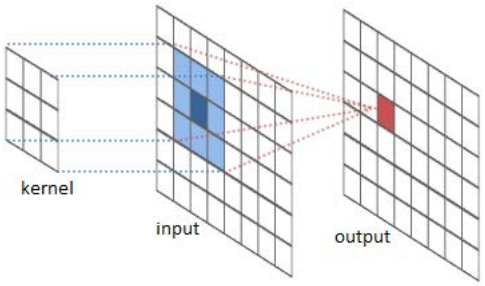
2.3.1 Camada de Convolução

Na camada de convolução cada neurônio está associado a uma janela de kernel que será convoluída na imagem de entrada. Esse kernel de convolução é composto pelos valores dos pesos de ligação do neurônio. A partir da imagem original são geradas n imagens, que resultam das operações em cada um dos n neurônios (NETO et al., 2017). Um exemplo é descrito na Figura 8.

2.3.2 Função de Ativação ReLU

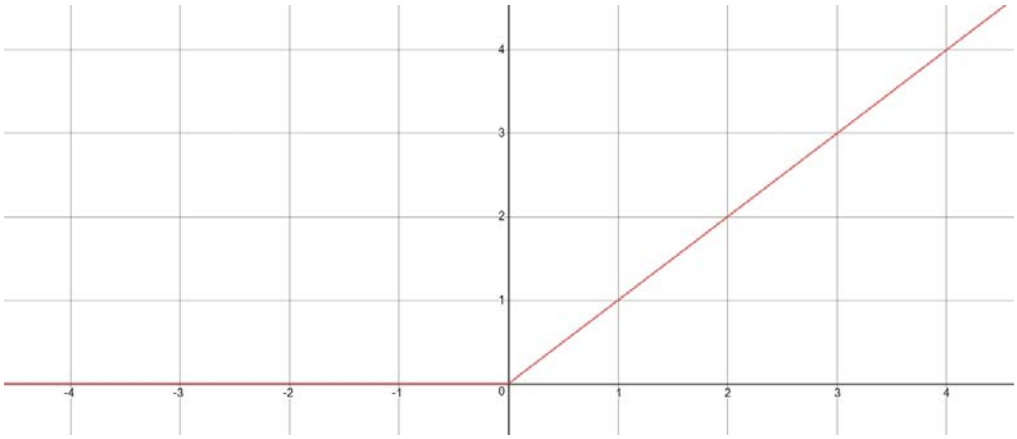
Devido às operações feitas no treinamento, o valor dos pesos pode vir a ficar negativo. Como em uma Rede Neural Convolucional isso não é permitido, pois, a cada camada, uma imagem nova é gerada e não existem imagens que contém pixels com valores negativos, a função de ativação ReLU resolve esse problema. A fórmula da função é $max(0, x)$, ou seja dado um valor, o máximo entre ele e 0 prevalece. A Figura 9 mostra o gráfico da função ReLU.

Figura 8 – Camada de Convolução.



Fonte: <http://colah.github.io/>.

Figura 9 – ReLU.



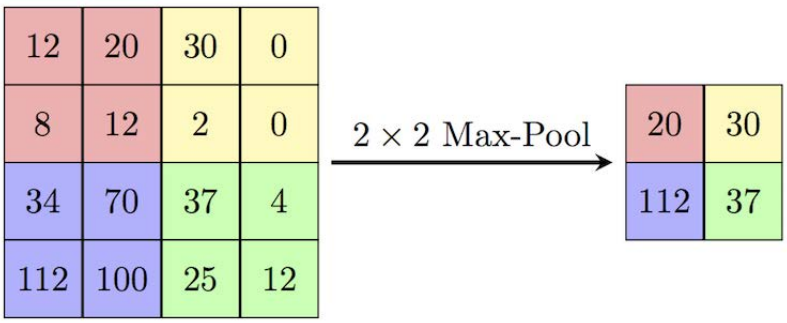
Fonte: <https://www.jefkine.com/general/2016/08/24/formulating-the-relu/>.

2.3.3 Função de *Pooling*

Após a camada de ReLU é comum ser aplicada uma camada de agrupamento (*pooling*). A importância dessa etapa é a redução na dimensionalidade dos mapas de características (*feature map*), acelerando o tempo necessário de treinamento da rede (NETO et al., 2017).

Dentre as funções de *Pooling* usadas existe uma chamada *Max Pooling*, que dada uma dimensão $m \times n$, o maior valor é escolhido para compor a entrada da próxima camada. Exemplo é dado na Figura 10.

Figura 10 – *Max Pooling*.



Fonte: <https://computersciencewiki.org/index.php/Max-pooling_/_Pooling>.

Como pode-se observar, as dimensões escolhidas são 2×2 e o valor que é passado como entrada para a próxima camada é o máximo. Têm-se uma diminuição considerável da dimensão da próxima camada e não perde-se o desempenho.

3 Desenvolvimento

Nessa seção serão apresentados a metodologia utilizada neste trabalho, bem como a base de dados, a biblioteca de DL em Python utilizada, a rede neural implementada (arquitetura e parâmetros usados), QT (framework utilizado para o desenvolvimento do aplicativo) e os resultados.

3.1 Base de Dados EMNIST

O conjunto de dados MNIST se tornou um padrão para quando se vai aprender problemas de classificação e de visão computacional. O que contribuiu para o MNIST se tornar tão difundido foi que o conjunto é de fácil entendimento, é natural e intuitivo para os seres humanos, é relativamente pequeno para armazenar e processar, e é acessível e fácil de usar em banco de dados. O MNIST veio de um conjunto de dados maior chamado *NIST Special Database 19* que contém dígitos, e letras maiúsculas e minúsculas, de forma manuscrita. O EMNIST é uma variante do conjunto de dados NIST e seu nome veio de *Extended MNIST (EMNIST)*, que segue o mesmo paradigma de conversão usado para criar o conjunto de dados MNIST. O resultado é um conjunto de vários conjuntos de dados que tem um desafio maior para problemas de classificação envolvendo letras e dígitos manuscritos, ele também tem a mesma estrutura e parâmetros das imagens originais do MNIST, dando total compatibilidade com todos os sistemas e classificadores existentes (COHEN et al., 2017).

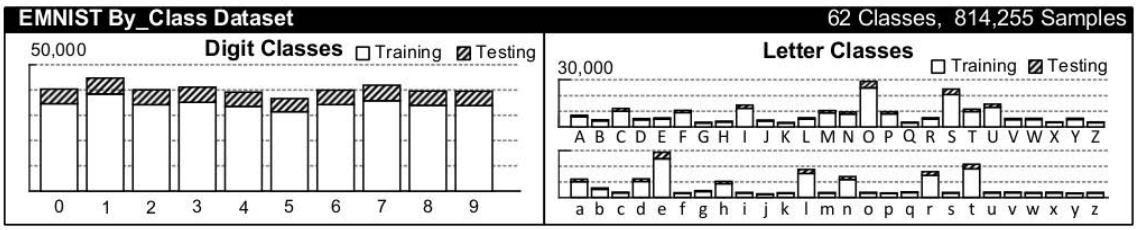
Características Importantes do EMNIST

Desenvolvido na *Western Sydney University* da Austrália em 2017, o EMNIST contém 6 conjuntos de dados dentro dele: *By Class*, *By Merge*, *Balanced*, *Digits*, *Letters* e *MNIST*. Para comportar o problema proposto, a escolha foi utilizar o conjunto de dados *By Class*, que é separado e rotulado pelas classes (dígitos, letras maiúsculas e letras minúsculas).

O conjunto de dados do EMNIST escolhido contém 62 classes, sendo os dígitos de **0**-**9**, as letras maiúsculas de **A**-**Z** e as minúsculas de **a**-**z**. Contém 814.255 imagens rotuladas, com resolução de 28×28, com 697.932 para o conjunto de treinamento e 116.323 para o conjunto de testes, sendo preciso separar uma parte do conjunto de treinamento para a validação do modelo.

Na Figura 11 é observado que a frequência de cada item é diferente, e a frequência dos dígitos é bem maior do que a das letras, isso implica no treinamento, sendo que os itens com maior frequência tem uma acurácia maior, pois, uma das premissas do aprendizado de máquina é: quanto mais dados para o treinamento, melhor a precisão.

Figura 11 – Frequência dos itens do conjunto de dados EMNIST separado por classes.



Fonte: (COHEN et al., 2017)

3.2 Keras

Keras é uma biblioteca de redes neurais artificiais escrita em *Python* e de código aberto. É capaz de rodar no seu *backend* bibliotecas como: **TensorFlow**, **Microsoft Cognitive Toolkit** ou **Theano**. O seu foco é permitir um desenvolvimento fácil de redes neurais profundas, com foco em ser amigável ao usuário, modular e extensiva. Um dos focos do Keras é ser capaz de ir da ideia ao resultado de forma rápida (KERAS. . . , 2018).

Keras é recomendado se você precisa de uma biblioteca que:

- Permita fácil e rápida prototipagem (através da interface amigável, modularidade e extensibilidade do Keras);
- Suporte para Redes Neurais Artificiais e Redes Neurais Recorrentes, ou uma combinação das duas;
- Permite rodar perfeitamente na CPU ou GPU.

Princípios do Keras:

- **Amigável ao Usuário**: Keras é uma API feita para humanos e não máquinas. Isso coloca na frente e no centro a experiência do usuário. Keras segue boas práticas para reduzir carga cognitiva: Oferece uma API simples e consistente, o que minimiza o número de ações que um usuário precisa fazer em casos comuns de desenvolvimento, e isso faz com que se previna erros, com um feedback recorrente e ativo através dos erros do usuário.
- **Modularidade**: Um modelo é entendido como uma sequência ou um grafo, que é totalmente configurável e os módulos podem ser ligados uns aos outros com poucas restrições. No Keras, as redes neurais, funções de custo, funções de otimização, inicializadores de esquema, funções de ativação, esquemas de regularização são todos feitos de maneira independente uns aos outros e se pode combiná-los e criar novos modelos a partir deles.

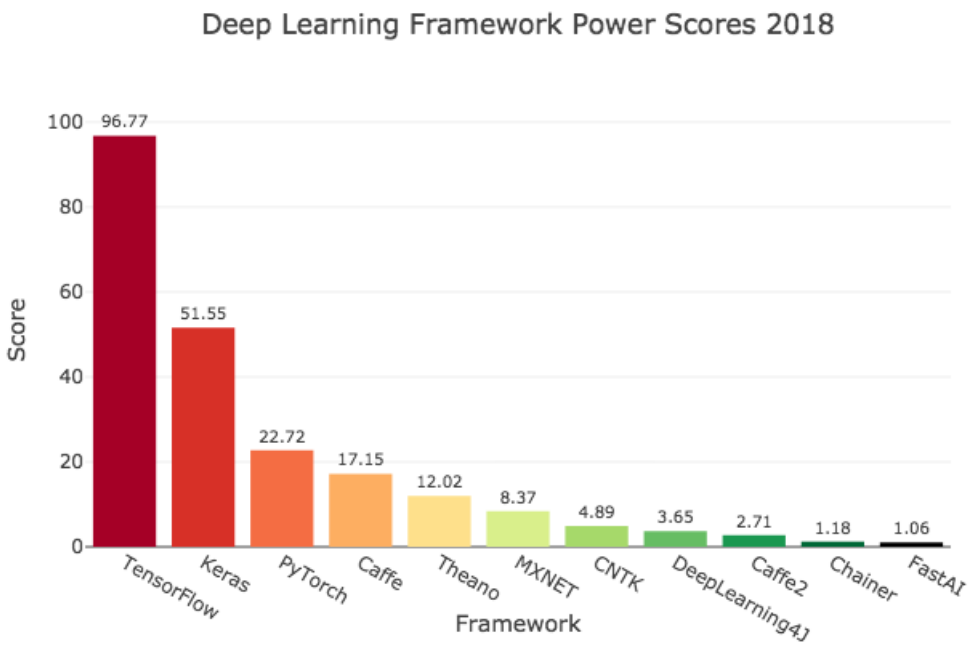
- **Fácil extensibilidade**: É simples adicionar novas classes e funções para os novos modelos, e existem muitos módulos de exemplos. Ser capaz de criar e estender módulos é totalmente expressivo, permitindo o Keras ser uma boa escolha para pesquisas avançadas.

- **Python**: Não precisa de modelos de configurações em arquivos diferentes. Modelos são descritos em Python, de modo compacto, fácil para debugar e permite facilidade para extensibilidade. O Keras tem compatibilidade com as versões de **Python 2.7 - 3.6**.

Motivo para usar o Keras:

Com mais de 250.000 usuários individuais na metade de 2018, Keras tem uma forte adoção, tanto na comunidade científica tanto na indústria. As pessoas ao redor do mundo interagem diariamente com aplicações de *Deep Learning* que usam o Keras, ele é usado: no Netflix, Uber, Yelp, Instacart, Zocdoc, Square, dentre outras. É muito popular em *startups* que pretendem usar *Deep Learning* como o seu produto principal (KERAS. . . , 2018). A Figura 12 ilustra um gráfico que compara os Frameworks de *Deep Learning* entre si e é observado que o Keras está em segundo, após o TensorFlow.

Figura 12 – Popularidade/Uso dos Frameworks de *Deep Learning*.



Fonte: (HALE, 2018)

3.3 Desenvolvimento da Rede Neural Convolucional

Como dito anteriormente, para este trabalho foi utilizado o método de Rede Neural Convolucional. Nesta subseção será apresentada a o desenvolvimento da arquitetura da rede, bem como os resultados de acurácia obtidos.

3.3.1 Preparação para o Treinamento

Imagens do Conjunto de Dados rotacionadas

Um primeiro desafio encontrado foi que ao usar a base de dados EMNIST, os itens vieram rotacionados 180°. Então, foi feita uma função que rotaciona todas as imagens do conjunto de dados e as deixa na forma correta, como o Código 1 mostra.

Código 1 – Função que rotaciona *numpy images*

```
import numpy as np

# Rotaciona a imagem
def rotate(image):
    # A imagem eh um vetor de 784 elementos,
    # entao, a transforma em uma 28x28.
    image = image.reshape([28, 28])
    image = np.fliplr(image)
    image = np.rot90(image)

    # Retorna com a imagem jah rotacionada
    # e restaura para um vetor de 784 elementos.
    return image.reshape([28 * 28])
```

Toda estrutura de dados usado pelo Keras são *numpy arrays* (*arrays* da biblioteca *numpy*, que é muito usada cientificamente, pois possui alto desempenho e muitas funções que auxiliam o desenvolvimento), existem funções como as ***fliplr()*** e ***rot90()*** que fazem todo o trabalho de rotacionar a imagem de maneira precisa e rápida.

Grande dimensão do conjunto de dados

Outro desafio encontrado, é que o conjunto de dados EMNIST é grande, e com isso, foi usada uma função da biblioteca ***pandas*** que além de permitir ler arquivos *.csv*, também permite que o treinamento seja feito por etapas, ou seja, separar o treinamento em intervalos de x elementos por vez. O gargalo obtido por conta da grande dimensão do conjunto de dados,

foi resolvido depois da aplicação da função. O código para carregar os dados dessa forma está descrito no Código 2.

Código 2 – Carregando o Conjunto de Dados com pandas

```
import pandas as pd
# Pega o caminho do HOME do sistema
HOME = path.expanduser("~")
# Carregando o conjunto de dados
train = pd.read_csv(filepath_or_buffer=
                    HOME + r'/dev/datasets/emnist-byclass-train.csv',
                    header=None, chunksize=40000)
```

Para percorrer todos os itens do conjunto de dados, na fase de treinamento e teste, faz-se um loop em cima dos dados carregado pelo *pandas*, conforme o Código 3 mostra.

Código 3 – Iterando sobre o conjunto de dados

```
# Iterando
for chunk in train:
    # codigo para o treinamento ou teste
    pass
```

Preparação dos dados para o treinamento da Rede Neural Convolucional

Os dados precisam ser preparados, para assim, ser a entrada do treinamento da rede neural. Foram feitas várias operações para tornar isso possível, como: estudo prévio de como o conjunto de dados é organizado, manipulação da estrutura, rotação das imagens, transformação das imagens em intervalos convenientes, entre outros.

O primeiro passo foi salvar e retirar os rótulos do conjunto de dados, que estavam na primeira posição do vetor. O segundo passo foi rotacionar as imagens usando a função descrita no Código 1. O terceiro passo foi converter o vetor de rótulos para a matriz de rótulos, e assim, usar a probabilidade como medida, por exemplo em um conjunto de dados que existem apenas dois rótulos, 90% de probabilidade de ser a primeira opção e 10% de probabilidade de ser a segunda opção: [0.9, 0.1]. O quarto passo foi converter o conjunto de treinamento à estrutura de dados que tem o formato que o *TensorFlow* (que é a biblioteca usada como *backend* do Keras) recebe como entrada. O último passo foi deixar as imagens em intervalos de 0 à 1 (como as imagens são em escala de cinza, otimiza as operações). Todas as etapas estão descritas no Código 4.

Código 4 – Preparação dos dados para o treinamento

```
from keras.utils import to_categorical
```

```
import numpy as np

# Treinamento
for chunk in train:
    # Labels
    train_labels = chunk[0].values

    # Tirando a primeira coluna (que era os labels)
    names = chunk.columns.values
    image = chunk.drop(names[0], axis=1)

    # Vetor de imagens gerado
    train_images = image.values

    # Rotacionar Imagens
    train_images = np.apply_along_axis(rotate, 1,
                                       train_images)/255

    # Conjunto de Dados de Treinamento
    train_labels = to_categorical(train_labels)

    # Criando os Tensors
    train_images = train_images.reshape((len(train_images),
                                       28, 28, 1))

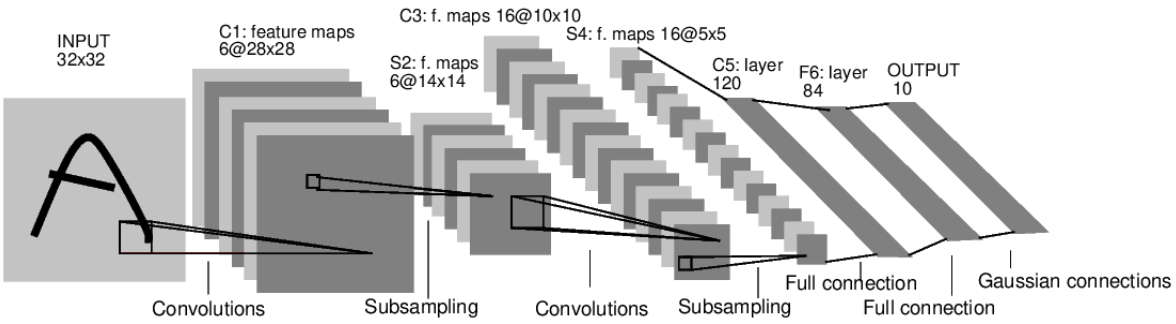
    # converte os dados para escala de [0, 1]
    train_images = train_images.astype('float32') / 255
```

Com todas as etapas feitas, os dados estão preparados para serem submetidos à rede neural e o treinamento pode ser feito.

3.3.2 Arquitetura

A arquitetura escolhida foi baseada na desenvolvida por LeCun em 1998 (LECUN et al., 1998). Essa arquitetura contém uma primeira camada de convolução, outra camada de convolução subsequente, duas camadas totalmente conectadas e em seguida a saída, conforme a Figura 13 mostra. A arquitetura de LeCun aplicada ao conjunto de dados MNIST teve 0,8% de taxa de erro no conjunto de testes e com isso mostrou-se uma ótima arquitetura para reconhecimento de dígitos manuscritos e é base de várias outras que usam redes neurais convolucionais.

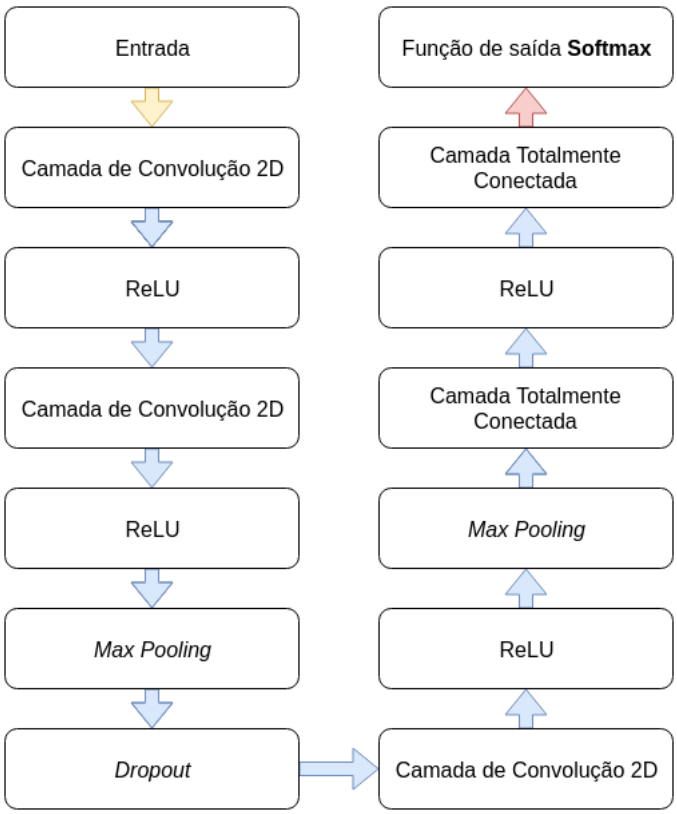
Figura 13 – Arquitetura LeNet 5



Fonte: (LECUN et al., 1998).

A arquitetura da rede neural convolucional do projeto ficou: Duas camadas de convolução 2D com função de ativação ReLU, *Maxpooling*, *Dropout*, camada de convolução 2D com função de ativação ReLU, *Maxpooling*, camada totalmente conectada com função de ativação ReLU e camada totalmente conectada com função de saída ***Softmax***. A Figura 14 mostra a arquitetura e a sua ordem.

Figura 14 – Arquitetura da Rede Neural Convolucional



Fonte: Elaborada pelo autor.

Para criar a arquitetura usando o Keras, se instancia o modelo com o objeto ***Sequen-***

cial(). A partir disso, para adicionar as camadas e as funções de ativação, se usa o método da classe chamado ***add(...)***, que contém os parâmetros necessários para adicionar as camadas. A arquitetura do projeto contida na Figura 14 é implementada conforme o Código 5.

Código 5 – Arquitetura da Rede Neural Convolucional

```
from keras.layers import Dense, Dropout,
Activation, Flatten, Conv2D,
MaxPooling2D, BatchNormalization

from keras.models import Sequential

# Criando o modelo
my_model = Sequential()

my_model.add(Conv2D(32, (3, 3), input_shape=(28, 28, 1)))
my_model.add(Activation('relu'))

my_model.add(Conv2D(32, (3, 3)))
my_model.add(Activation('relu'))
my_model.add(MaxPooling2D(pool_size=(2, 2)))
my_model.add(Dropout(0.2))

my_model.add(Conv2D(64, (3, 3)))
my_model.add(Activation('relu'))
my_model.add(MaxPooling2D(pool_size=(2, 2)))

# Camadas totalmente conectadas
my_model.add(Flatten())
my_model.add(Dropout(0.2))
my_model.add(Dense(128))
my_model.add(Activation('relu'))
my_model.add(Dense(62))

#Camada de saida
my_model.add(Activation('softmax'))
```

Com Código 5, o modelo é criado e está pronto para receber a entrada de dados e para o treinamento. O Keras gera uma figura da arquitetura conforme a Figura 15 ilustra. Na figura, se tem informações sobre as saídas das camadas, o tipo da camada, a quantidade de parâmetros em cada camada e o total de parâmetros da rede.

Figura 15 – Ilustração da Arquitetura gerada pelo Keras

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 26, 26, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 24, 24, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 10, 10, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1600)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               204928
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 62)                7998
_________________________________________________________________
activation_5 (Activation)    (None, 62)                0
=================================================================
Total params: 240,990
Trainable params: 240,990
Non-trainable params: 0
_________________________________________________________________
```

Fonte: Elaborada pelo autor.

3.3.3 Treinamento

Para o treinamento da rede foi selecionada como função de otimização, a função ***Adam*** que é um algoritmo que igualmente ao gradiente descendente, usa o gradiente para otimizar o modelo. A função de otimização é moderadamente de fácil implementação, computacionalmente eficiente, requer pouco uso de memória e funciona muito bem para problemas com grandes quantidades de dados e/ou parâmetros (KINGMA; BA, 2015).

Como função para calcular o erro, foi utilizada a ***categorical crossentropy*** que é utilizada quando a função de saída retorna um vetor de probabilidades, calculando assim o erro para cada classe.

Para o treinamento, é necessário informar à biblioteca quanto das imagens do treinamento serão usadas para a validação e neste projeto 15% do conjunto de treinamento é destinado à validação. O treinamento tem 5 épocas, ou seja, nesta etapa a mesma imagem será visitada 5 vezes, com um lote de 250 imagens para então fazer a otimização.

O código de como estabelecer os parâmetros e as funções está descrito no Código 6.

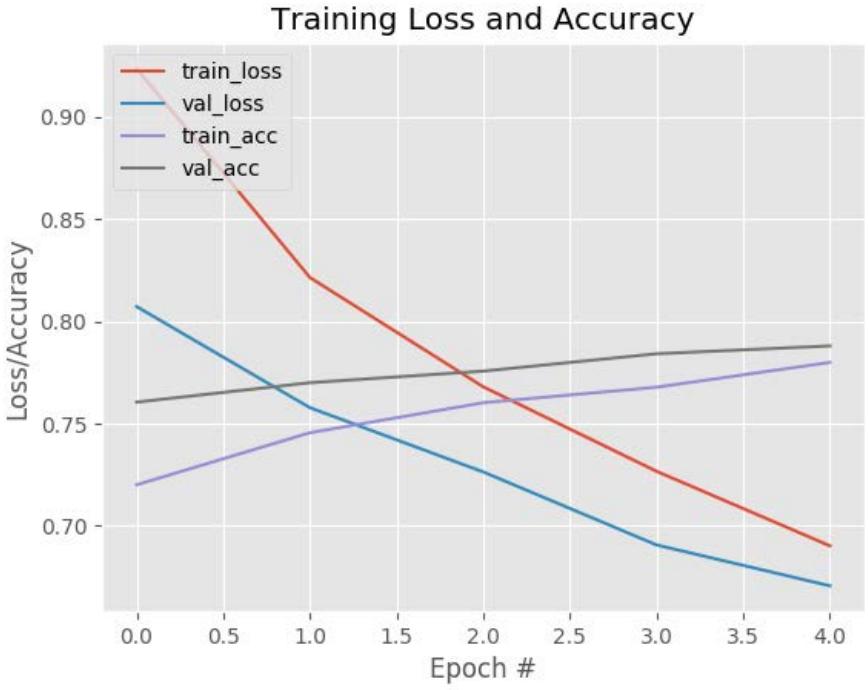
Código 6 – Treinamento da rede

```
# CNN Loss e Optimizer
compile_model = my_model.compile(optimizer='Adam',
                                 loss='categorical_crossentropy',
                                 metrics=['accuracy'])

# CNN Treinamento
fit_model = my_model.fit(x=train_images,
                         y=train_labels, batch_size=250,
                         epochs=5, validation_split=0.15)
```

Após o treinamento, foi obtido uma acurácia de 78% no conjunto de treinamento e 79% na validação, na última passagem dos dados conforme a Figura 16 ilustra.

Figura 16 – Treinamento em relação às épocas



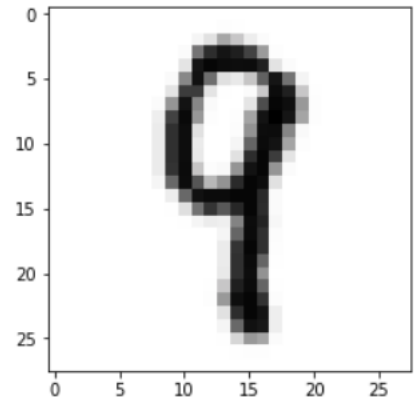
Fonte: Elaborada pelo autor.

O modelo é salvo usando a função **save(filepath=*LOCAL_ONDE_QUER_SALVAR*)**.

3.3.4 Teste

No conjunto de teste foram feitas três medidas, com 40.000 imagens em duas e 36.323 em uma, para abranger todo o conjunto. Foi obtida acurácia que ficou entre 76.5% e 77.5%. A Figura 17, Figura 18 e Figura 19 mostram as passadas pelo conjunto de teste (somando as três passadas se tem 116.323 imagens) e contém um teste aleatório que pega um caractere oriundo do conjunto de testes.

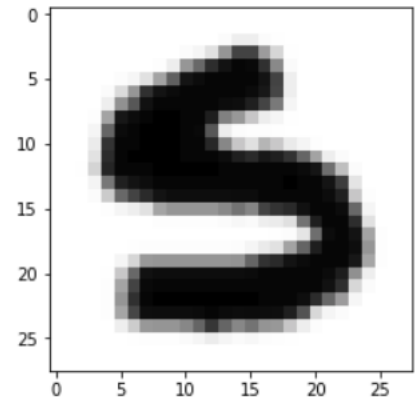
Figura 17 – Primeira passada no conjunto de teste



```
O modelo prevê: q
40000/40000 [==============================] - 13s 337us/step
Acurácia no conjunto de teste:: 77.08%
```

Fonte: Elaborada pelo autor.

Figura 18 – Segunda passada no conjunto de teste

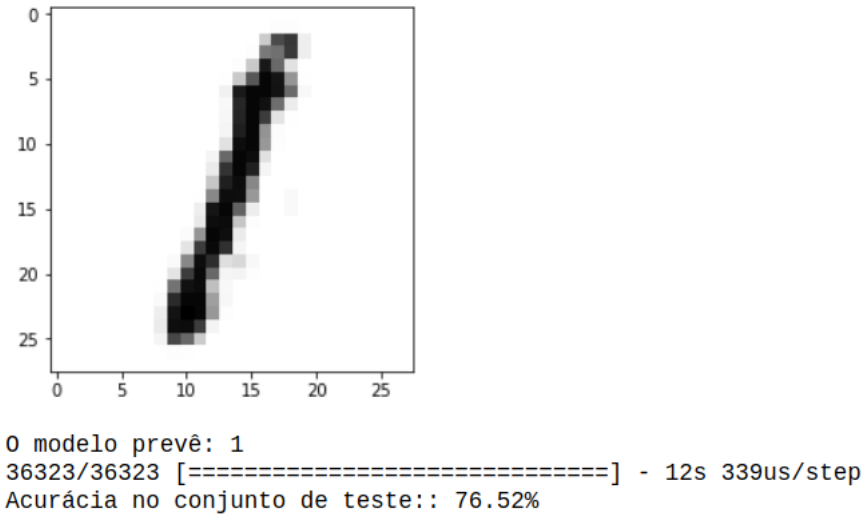


```
O modelo prevê: S
40000/40000 [==============================] - 15s 367us/step
Acurácia no conjunto de teste:: 76.40%
```

Fonte: Elaborada pelo autor.

No artigo original do EMNIST (COHEN et al., 2017), a acurácia obtida no conjunto que utiliza as classes (conjunto também usado neste trabalho) foi de 69,71%±1,47%, conforme

Figura 19 – Terceira passada no conjunto de teste



Fonte: Elaborada pelo autor.

a Tabela 1. Este valor de acurácia é coerente, pois há grande dificuldade em problemas de caracteres manuscritos com tantas classes, que ao todo são 62.

Tabela 1 – Resultados de algoritmos aplicados ao conjunto de dados EMNIST.

	Linear Classifier	**OPIUM Classifier**
Balanced	50.93%	78.02%±0.98%
By Merge	50.51%	72.57%±1.18%
By Class	51.80%	69.71%±1.47%
Letters	55.78%	85.15%±0.12%
EMNIST MNIST	85.11%	96.22%±0.14%
Digits	84.70%	95.90%±0.40%

Fonte: (COHEN et al., 2017).

3.4 Servidor

O servidor foi desenvolvido usando uma biblioteca de Python chamada **Flask**. Ele funciona através de requisições, semelhantemente à um site.

O servidor, depois que ligado, espera uma requisição na ***url***:<*IP_LOCAL*:8800/predict>, onde o IP_LOCAL é o IP da máquina na rede. O servidor recebe uma imagem em Base64, a transforma em PNG, converte em escala de cinza, usa o modelo para fazer a predição e retorna o resultado. O código do servidor é descrito no Código 7.

Código 7 – Servidor para predição de caractere

```
from flask import Flask, jsonify, request, abort
import numpy as np
```

```
import io
import base64
from keras.models import load_model
from keras import backend
from os import environ
from PIL import Image
from skimage.transform import resize
import scipy.misc

environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

app = Flask(__name__)

# Conversao de RGB para escala Preto e Branco
def rgb2gray(rgb):
        r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
        gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
        return gray

#-------------------------------
@app.route('/predict', methods=['POST'])
def receive_image():
    if not request.data:
        abort(400)
    pass
    # Preparando a imagem vinda em Base64 para a transformacao
    data = request.json['image'].split(',')[-1]
    data = base64.b64decode(data)

    # Salvando a imagem no disco
    filename = 'temp.png'
    with open(filename, 'wb') as f:
        f.write(data)
    temp_data= open("temp.jpg", "wb")
    temp_data.write(data)
    temp_data.close()

    # Abrindo a imagem e transformando ela em um numpy array
    aux = Image.open("temp.jpg")
```

```
    to_gray = np.array(aux)

    # Transformando de RGB para escala Preto e Branco
    to_gray = rgb2gray(to_gray)

    # Escala real da CNN
    to_gray = resize(to_gray, (28,28))

    # Carregando o modelo
    my_model = load_model(filepath=
                          """/home/luisuzai/
                          dev/app-tcc/server
                          /model_save.h5""")

    # Classificando a imagem
    pred = my_model.predict(to_gray.reshape(1, 28, 28, 1),
                            batch_size=1)

    # Dicionario das classes
    label_dictionary ="""0123456789ABCD
                        EFGHIJKLMNOP
                        QRSTUVWXYZab
                        cdefghijklmn
                        opqrstuvwxyz"""

    # Dando refresh na sessao
    backend.clear_session()

    # Retornando a classificacao
    return label_dictionary[np.argmax(pred)]
# ---------------------------------
if __name__ == '__main__':
        app.run(host='0.0.0.0', port=8800)
```

3.5 QT

Para testar o modelo e o servidor, foi implementado um aplicativo usando o *framework* **QT**. O QT usa C++ como *backend* e como interface uma linguagem própria conhecida como *QML*.

Em QML foi feito uma caixa para o desenho do caractere manuscrito e dois botões, um de envio e um de limpar a tela. Quando se clica no botão de envio, é feita uma requisição no servidor e uma resposta com o caractere classificado pelo modelo é obtida. Um exemplo é dado pelas figuras, Figura 20, Figura 21 e Figura 22, que contém todo o fluxo do aplicativo.

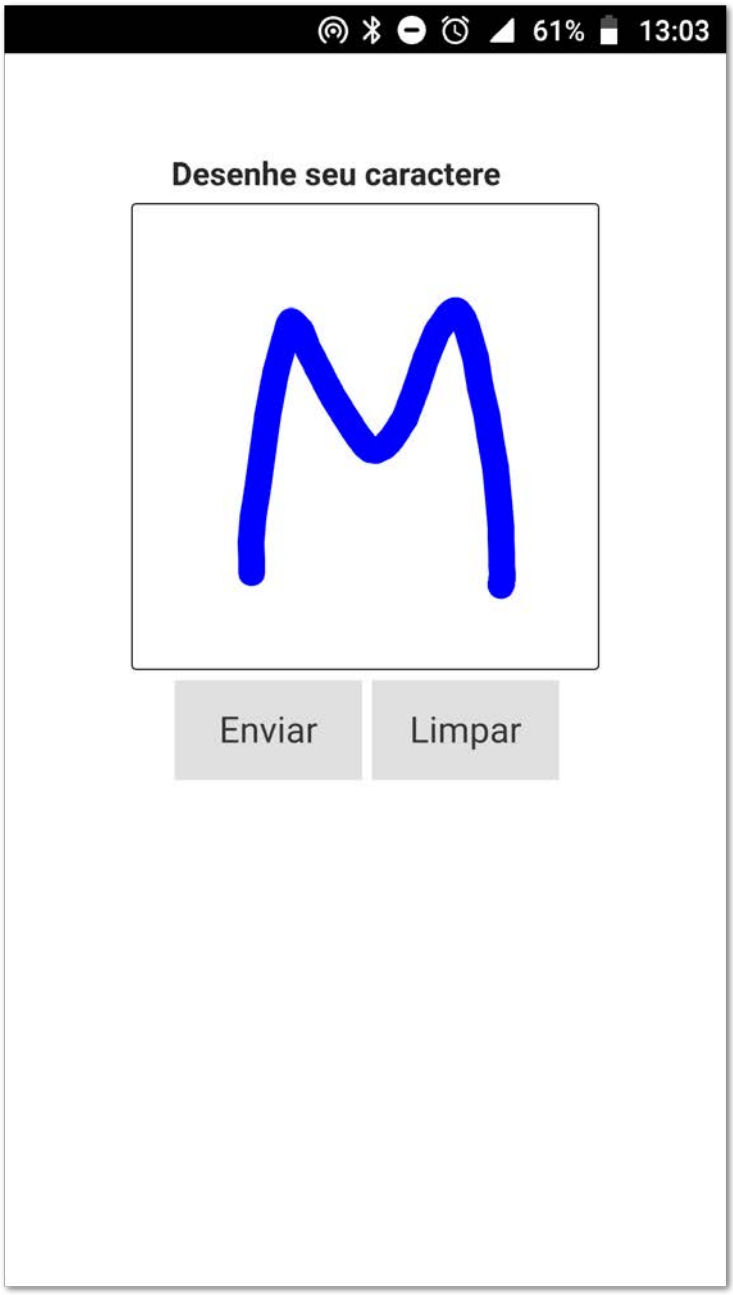
O desenvolvimento do aplicativo é simples e como foi feito apenas para testes e para verificar a funcionalidade do modelo, o código não está contido neste trabalho.

Figura 20 – Tela inicial do Aplicativo Mobile



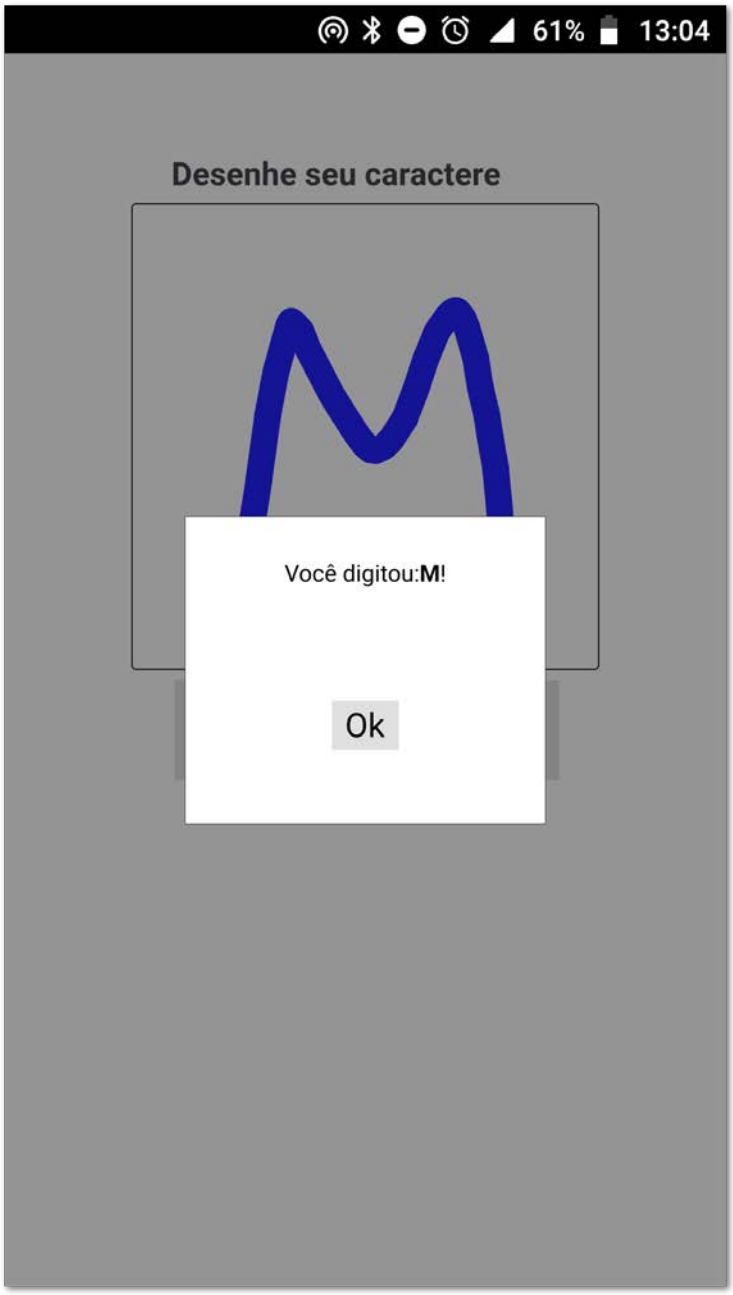
Fonte: Elaborada pelo autor.

Figura 21 – Tela com um “M” desenhado



Fonte: Elaborada pelo autor.

Figura 22 – Tela com “M” submetido ao servidor e com resposta



Fonte: Elaborada pelo autor.

4 Conclusão

Este projeto teve como objetivo desenvolver e modelar uma rede neural convolucional para a classificação de caracteres manuscritos. Para o aprendizado, desenvolvimento e modelamento, foi necessário um estudo árduo sobre a matemática, álgebra linear e estatística envolvidas no aprendizado profundo, mais especificamente em redes neurais convolucionais. Para isso, foram usados, livros, artigos, *frameworks* e bibliotecas mais atuais para o desenvolvimento, como: Keras, QT, Tensorflow, flask, jupyter notebook, linguagem Python, etc.

Trabalhar com um conjunto de dados complexo foi desafiante, normalmente quando se vai desenvolver para este tipo de problema, o conjunto de dados usados é o MNIST que tem apenas 10 classes (dígitos). Mas, como a proposta era classificar caracteres manuscritos, a quantidade de classes foi de 62 (digitos e letras maíusculas e minúsculas).

As redes neurais convolucionais se mostraram um método eficiente para a classificação de caracteres manuscritos, conforme a Tabela 2 mostra. A melhor acurácia obtida no artigo original do EMNIST para o caso de 62 classes foi de 69,71%, enquanto a implementação com redes neurais convolucionais contidas neste trabalho, teve como acurácia 76,67%.

Tabela 2 – Comparação entre acurácia do artigo do EMNIST e a deste trabalho.

Método	**Fonte**	**Acurácia**
Classificador Linear	Artigo EMINIST (COHEN et al., 2017)	69,71%
Redes Neurais Convolucionais	Este trabalho	76,67%

Fonte: Elaborada pelo autor.

Todo o estudo, desenvolvimento, modelagem contidos neste projeto são semelhantes do que seria um estudo de aprendizado profundo para aplicações práticas reais, além de que, existem muitos conceitos matemáticos e estatísticos estão por trás dos métodos utilizados, e o trabalho contribuiu para a formação do aluno e para a aplicação de conhecimentos obtidos ao longo da graduação.

4.1 Trabalhos Futuros

Como sugestão de trabalhos futuros, pode-se considerar a melhoria dos resultados alcançados por este trabalho. Para isso, é possível estudar o ajuste dos hiper-parâmetros e a tentativa de modelar a arquitetura com modelos mais complexos, observando o comportamento da rede. Além da tentativa de combinar este método com outros métodos, aplicar transformações no conjunto de dados. Também, considerar treinar a rede em computadores mais robustos, explorando o paralelismo obtido por meio das GPU's, para otimizar e não ter como problema o tempo de processamento.

Referências

BUDUMA, N. *Fundamentals of Deep Learning*. [S.l.]: O'Reilly, 2017. 4 p.

CÁNEPA, G. *What You Need to Know about Machine Learning*. [S.l.]: Packt, 2016.

CHRISTENSSON. *OCR Definition*. 2018. Disponível em: <https://techterms.com/definition/ocr>. Acesso em 01 de Outubro de 2018.

COHEN, G.; AFSHAR, S.; TAPSON, J.; SCHAIK, A. van. Emnist: an extension of mnist to handwritten letters. *The MARCS Institute for Brain, Behaviour and Development*, 2017.

DANGETI, P. *Statistics for Machine Learning*. [S.l.]: Packt, 2017. 8 p.

GONZALES; WOODS. *Digital Image Processing*. [S.l.]: Pearson Education Inc., 2008.

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

HALE, J. *Deep Learning Framework Power Scores*. 2018. Disponível em: <https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a>. Acesso em 22 de Outubro de 2018.

HUBEL; DAVID, H.; WIELSEL, N. Receptive fields and functional architecture of monkey striate cortex. the journal of physiology. *The Journal of Physiology*, p. 215–245, 1968.

JOSHI; PRATEEK. *Artificial Intelligence with Python*. [S.l.]: Packt, 2017. 8–14 p.

KERAS Documentation. 2018. Disponível em: <https://keras.io/>. Acesso em 22 de Outubro de 2018.

KINGMA, D. P.; BA, J. L. *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*. 2015.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

NETO, F. G. M.; JÚNIOR, R. F. P.; ROCHA, M. G. O.; JUNIOR, J. J. de M. S.; JÚNIOR, I. C. de P. Aprendizado profundo: conceitos, técnicas e estudo de caso de análise de imagens com java. *III Escola Regional de Informática do Piauí. Livro Anais - Artigos e Minicursos*, 2017.

TECHOPEDIA. *Convolutional Neural Network (CNN)*. 2018. Disponível em: <https://www.techopedia.com/definition/32731/convolutional-neural-network-cnn>. Acesso em 02 de Outubro de 2018.