Reconhecimento de Digitos Manuscritos em Python

Handwritten Digit Recognition in Python

Oleksandra Kukharska
Coimbra Business School
ISCAC, Polytechnic of Coimbra
Coimbra, Portugal
iscac16402@alumni.iscac.pt

Resumo — **O reconhecimento de manuscritos, é a capacidade de um computador receber e interpretar informações através de imagens. A dificuldade no reconhecimento deve-se ao fato de os caracteres, neste caso, dígitos, terem diferentes formas de serem representados e ao fato de que pessoas diferentes escrevem de maneiras diferentes. Neste artigo será realizado o reconhecimento de dígitos manuscritos, usando algoritmos de Machine Learning e Deep Learning, nomeadamente Convolutional Neural Network (CNN), Support Vector Machine (SVM) e Multilayer Perceptron (MLP). O principal objetivo é construír os três modelos de forma a obter uma previsão precisa para o reconhecimento de dígitos, e fazer comparação entre os resultados obtidos e selecionar o melhor.**

Palavras Chave – Deep Learning, Machine Learning; Redes Neurais; Convolutional Neural Network; Support Vector Machine; Multilayer Percentron; Previsão; Python.

Abstract — **Handwriting recognition is the ability of a computer to receive and interpret information through images. There are some difficulties in recognition due to the fact that the characters, in this case, digits, have different forms of differences, and the fact that people write in different ways. This article will perform handwritten digit recognition, using Machine Learning and Deep Learning features such as Convolutional Neural Network (CNN), Support Vector Machine (SVM) and Multilayer Perceptron (MLP). The main objective is to build the three models to obtain an accurate prediction of digit recognition, and to compare the results obtained and select the best one.**

Keywords – Deep Learning, Machine Learning; Neural Network; Convolutional Neural Network; Support Vector Machine; Multilayer Percentron; Prediction; Python.

I. INTRODUÇÃO

Estamos a viver numa época em que existem dados em abundância, através do uso algoritmos de autoaprendizagem no campo de Machine Learning, podemos transformar esses dados em conhecimento, graças às bibliotecas de código aberto que foram desenvolvidas para identificar padrões em dados e fazer previsões.[1]

As redes neurais, através de uma grande quantidade de dígitos manuscritos, conhecidos como treino, desenvolvem um sistema que, através desses dados de treino aprendam a identificar outros conjuntos de dataset do mesmo tipo, ou seja, a rede neural usa os treinos para inferir automaticamente regras para o reconhecimento de dígitos manuscritos. Quanto mais dados de treino, mais a rede pode aprender sobre a caligrafia, e assim melhorar a sua precisão.

II. METODOLOGIA

Foi utilizada a linguagem de programação Python, em Google Colab.

No campo das redes neurais, a biblioteca Tensorflow, oferece implementações específicas nesse domínio, camadas, otimizadores e funções de avaliação.

Além disso, para a execussão dos modelos, será utilizada a função keras, uma API que constrói e treina modelos no Tensorflow.

A. Dados

A base de dados encontra-se em formato *csv*, e é constituída por três ficheiros, **test.csv**, **train.csv** e **sample_submission.csv**, retirados do kaggle.com, e lidos no Python com recurso ao *Pandas.*

Os ficheiros **train.csv** e **test.csv**, correspondem ao conjunto de treino, composto por 42.000 imagens, e ao conjunto teste, composto por 28.000 imagens, ambos os ficheiros contêm imagens em escala cinza de dígitos manuscritos, do número zero até ao número nove.

Cada imagem tem 28 pixels de altura e 28 pixels de largura, para um total de 784 (28x28) pixels por imagem. A cada pixel é atribuído um único valor, de acordo com a claridade ou escuridão do pixel correspondente, em que, a tons mais escuros é atribuído um número mais alto, e vice-versa. Estes valores de pixel variam entre 0 e 255, inclusive.

O ficheiro **train.csv**, é composto por 785 colunas. A primeira coluna, denominada de 'label', corresponde ao dígito manuscrito. As restantes colunas contém os valores de pixel da imagem associada.

O ficheiro **test.csv** é igual ao ficheiro **train.csv**, exceto no facto de não conter a coluna 'label'.

A cada coluna de pixel é atribuído um nome, como pixelX, onde X é um número inteiro entre 0 e 783, inclusive. Para

localizar esse pixel na imagem, é utilizada a seguinte função, em que:

X = i * 28 + j, onde i e j são inteiros entre 0 e 27, inclusive.

Então o pixelX está localizado na linha i e na coluna j de uma matriz 28 x 28 (0-27). Por exemplo, pixel10 indica o pixel que está na segunda coluna e na primeira linha.

O ficheiro **sample_submission.csv** é compost por duas colunas, a primeira com o número de testes (0-28000) e a segunda com a *Label*, isto é, a resposta aos dados teste, apenas esta última coluna será utilizada, nomeadamente para definir **y_test**, como se explica mais a frente.

B. *Tratamento dos dados*

Os dados são compostos por dados de treino (*train*) e teste (*test*), em ambos os casos, são criadas *DataFrame* para demarcar as *features* e as *labels*, como se segue na tabela seguinte:

Tabela 1 Classificação dos dados de treino e teste

	train	test
features	**x_train**	**x_test**
labels	**y_train**	**y_test**

Para ambas as *features* é aplicada a função *to.numpy()* e é feito o redimensionamento, com a função *reshape()*, de 2 dimensões para 3 dimensões, passando para o formato (42000, 28, 28) no caso de **x_train** e (28000, 28, 28) para **y_train**, para se proceder à análise dos *dataset*.

Ao longo da execução dos três modelos selecionados para análise, dependendo de cada modelo, irá ser novamente necessário voltar ao redimensionamento de 2 dimensões.

C. *Seleção dos modelos*

O objetivo do Machine Learning supervisionado é construir um modelo, a partir dos dados de treino, faz previsões com dados futuros. O termo supervisionado refere-se a um conjunto de amostras onde os outputs já são conhecidos. Um algoritmo de aprendizagem supervisionada usa um conjunto conhecido de inputs e respostas conhecidas aos outputs e treina um modelo para gerar previsões razoáveis para a resposta a novos dados.[1]

A aprendizagem supervisionada usa técnicas de classificação e regressão para desenvolver modelos preditivos:

- Classificação: prevê respostas discretas;
- Regressão: prevê respostas contínuas.

Considerando o exemplo de reconhecimento de dígitos manuscritos, é treinado um modelo usando um algoritmo de Machine Learning supervisionado em um conjunto de dados, neste caso, imagens, que são estimados corretamente com dígitos entre 0-9, para prever se um novo conjunto de imagens pertence a qualquer um dos dígitos e se é corretamente identificado.

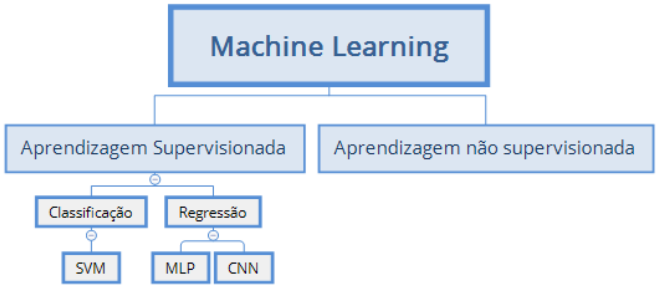

Figura 1 Tipos de aprendizagem de Machine Learning

Desta forma, foram selecionados três modelos de aprendizagem supervisionada, de classificação, SVM, e de regressão, MLP e CNN, este ultimo também pode ser considerado como classificação.

1) *Rede Neural Convolcional (CNN)*

CNN é um tipo de algoritmo de *Deep Learning* que considera uma imagem como input e aprende os vários recursos da imagem por meio de filtros. Permitindo identificar características importantes presentes na imagem, permitindo-lhes distinguir uma imagem da outra.[3]

Neste caso, a rede convolucional aprenderá as características específicas dos dígitos de zero a nove (0 a 9) que os diferenciam uns dos outros, para que, quando for fornecido um input, este seja corretamente reconhecido.

2) *Support Vector Machine (SVM)*

SVM é uma técnica de classificação baseada em *Machine Learning*, é um classificador linear binário que foi estendido para dados não lineares através de funções '*Kernel'* e dados multiclasse usando várias técnicas.[4]

SVM foi desenvolvida com o intuito de resolver problemas de classificação de padrões, e é eficiênte em trabalhar com dados de alta dimensionalidade, muitas vezes comparada as Redes Neurais. O *Kernel* é chamado de SVM não linear, podendo ser do tipo Polinomial, Gaussiano (ou RBF) ou Sigmoidal.[5]

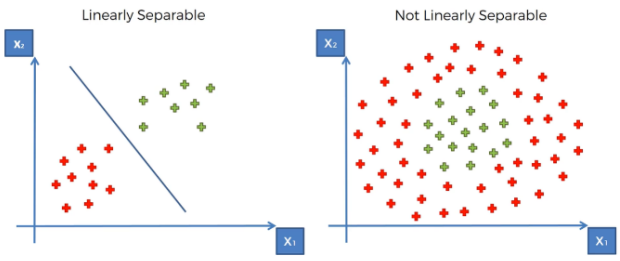

Figura 1 Gráfico linear e não linear da função Kernel do SVM[6]

3) *Multilayer perceptron (MLP)*

MLP é um método *de Deep Learning*, consiste numa rede neural artificial *feedforward* que gera um conjunto de saídas a partir de um conjunto de entradas. Um MLP é caracterizado

por várias camadas de nós de entrada conectadas como um grafo direcionado entre as camadas de entrada e saída. O MLP usa retropropagação para treinar a rede. MLP é um exemplo de rede neural artificial que é usada extensivamente para a solução de vários problemas diferentes, incluindo reconhecimento de padrões e interpolação.[7]

III. ANÁLISE

Cada imagem é constituída por pixels na escala de valores de 0-225.

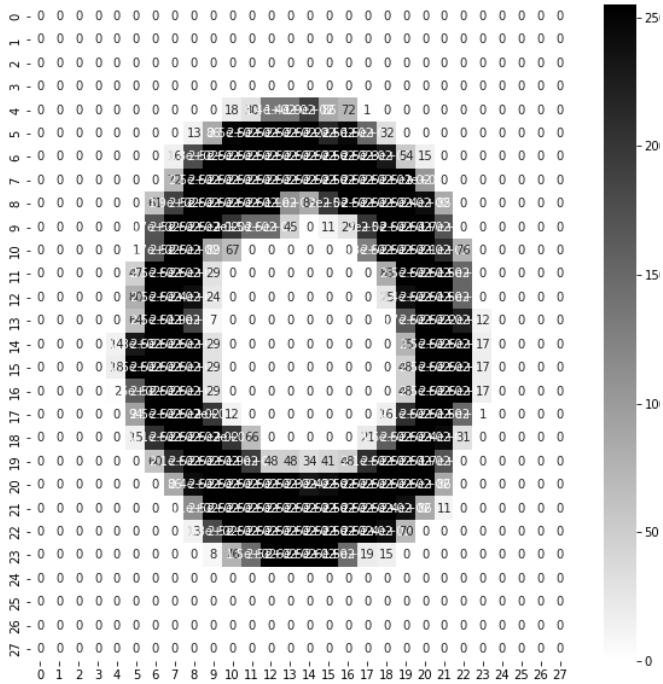

Figura 2 Segunda imagem dos dados de treino

No processamento de imagens, a normalização é um processo que altera a faixa de valores de intensidade dos pixels. O propósito desta alteração é mudar a imagem para uma faixa que seja mais familiar ou normal aos sentidos, tornando-a nítida, de forma a evitar distração mental ou fatiga. [8]

Com a normalização, a escala dos valores dos pixels passa a uma faixa de 0-1. A normalização foi feita de forma manual, através da divisão dos dados de treino por 225, obtendo-se o seguinte resultado, como mostra a Figure 4.

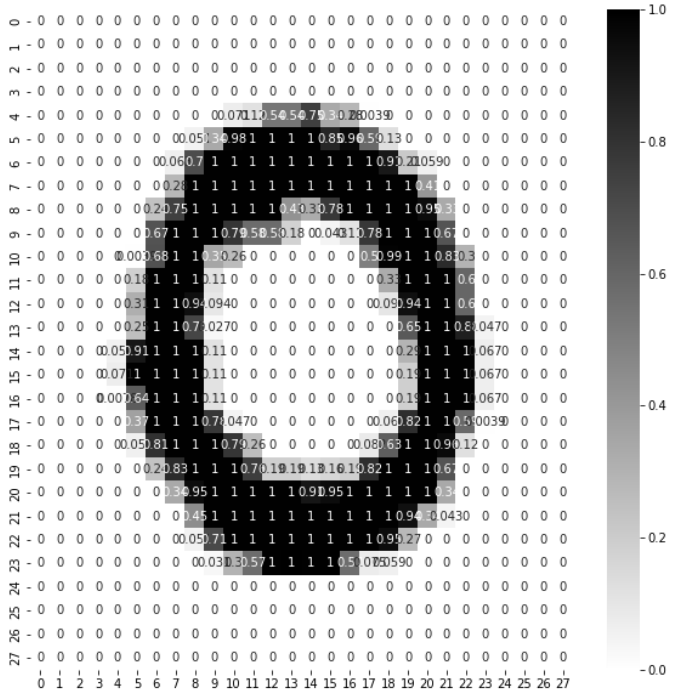

Figura 3 Segunda imagem dos dados de treino, após normalização

Este processamento de imagem é feito para todo o *dataset* de treino, em que através da análise dos valores dos pixels, obtem-se o seguinte conjunto de imagens.

Figura 4 Representação das 56 primeiras imagens de treino

Os dados de treino contêm 42.000 imagens, o dígito que aparece em maior quantidade, nomeadamente 4.684 vezes, é o dígito um (1), sendo o dígito cinco (5) que aparece em menor quantidade, nomeadamente, 3.795 vezes.

Tabela 2 Número de observações

```
Count of observations:
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
```

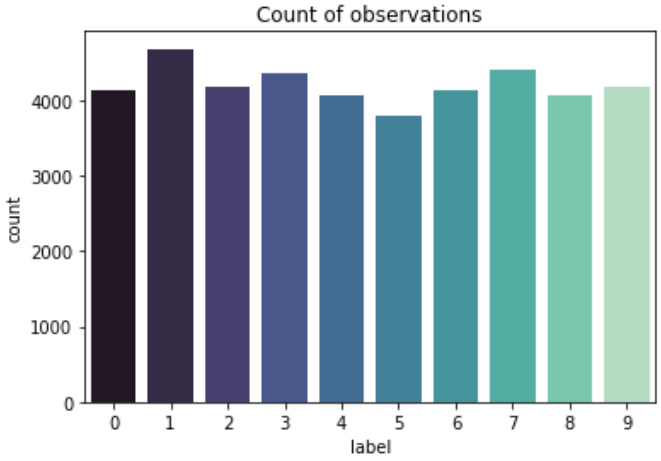

Figura 5 Número de observações

IV. IMPLEMENTAÇÃO

A. *CNN*

As bibliotecas a utilizer neste modelo, será o *sklearn* para preparação dos dados e plotagem e *keras* para a implementação do próprio modelo.

Para a execução do modelo CNN, teve que ser feito o redimensionamento dos dados para o formato (-1, 28, 28, 1), obtendo um **x_train** de (42000, 28, 28, 1) e um **x_test** de (28000, 28, 28, 1).

Os dados **y_train** foram transformados, através da função *to_categorical()*, para o formato categórico, o que permite que a representação dos dados seja mais expressiva, neste caso, são possíveis 10 respostas, entre 0-9, inclusive, e, cada valor inteiro é representado como um vetor binário que tem todos os valores zero, exceto o seu índice, que é representado por 1, como se apresenta no exemplo a seguir:

4 → [0 0 0 0 1 0 0 0 0 0]
9 → [0 0 0 0 0 0 0 0 0 1]

Para a validação, e consequente preparação para a previsão dos valores de **y_test**, será feita a separação dos dados de treino e teste, pela função *train_test_split()*.

A implementação do modelo CNN, requer a função *Sequential()*, que cria o modelo camada por camada, estas camadas, são:

- Dense
- Dropout
- Flatten
- Conv2D
- MaxPool2D

Após implementar o modelo, é feita a otimização, que é o processo onde treino do modelo que resulta na avaliação da função máxima e mínima, e que permite obter melhores resultados.

A métrica *compile()*, faz a configuração do modelo para os dados de treino.

O modelo requer a seleção de *epoch* e *batch_size*, que têm importância na precisão e execução do modelo:

- *epoch*: número de passagens completas pelo conjunto de dados de treino;

Não existe um *epoch* certo, inicialmente tinha sido estipulado um *epoch*=50, no entanto, a partir do *epoch*=30, a precisão veio a apresentar valores mais baixos e mais oscilantes. Uma razão para esta ocorrência é o modelo ter sido treinado por um longo *epoch*. Foi estabelecido um *epoch*=20.

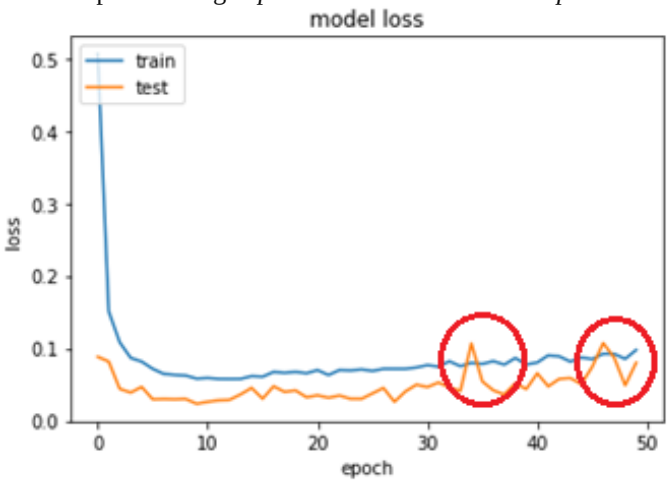

Figura 7 Oscilações na execução do modelo com epoch=50

- batch_size: número de amostras de treino trabalhadas antes de haver atualização dos parâmetros internos do modelo;

Por ultimo, é aplicada a função *fit()* que treina o modelo consoante o *epoch* estabelecido, e fornece a precisão do modelo.

Tabela 3 Sumário do modelo CNN

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        832

 conv2d_1 (Conv2D)           (None, 28, 28, 32)        25632

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 dropout (Dropout)           (None, 14, 14, 32)        0

 conv2d_2 (Conv2D)           (None, 14, 14, 64)        18496

 conv2d_3 (Conv2D)           (None, 14, 14, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 7, 7, 64)          0

 flatten (Flatten)           (None, 3136)              0

 dense (Dense)               (None, 256)               803072

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 887,530
Trainable params: 887,530
Non-trainable params: 0
_________________________________________________________________
```

Com a sumarização do modelo, pode-se observar que as camadas foram obtidas por:

- dense_1: 256 (dense) * 10 (dense_1) + 10 (*bias values*) = 2570 (o número de parâmetros);

O modelo CNN consegue prever os dados teste, com uma precisão de 98,9%.

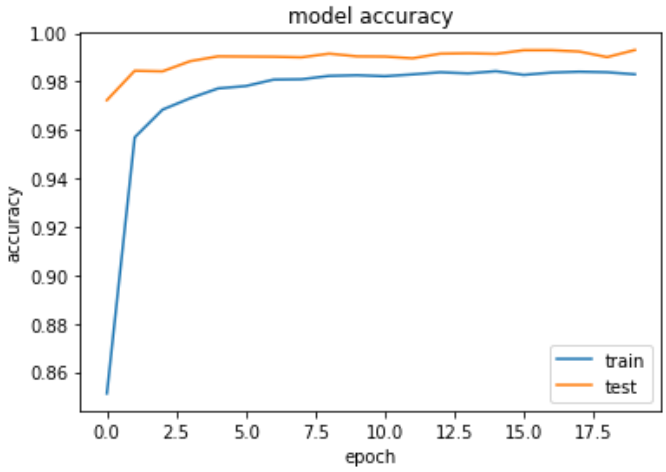

Figura 8 Histórico de precisão

Já quanto às perdas, é de 3,71%.

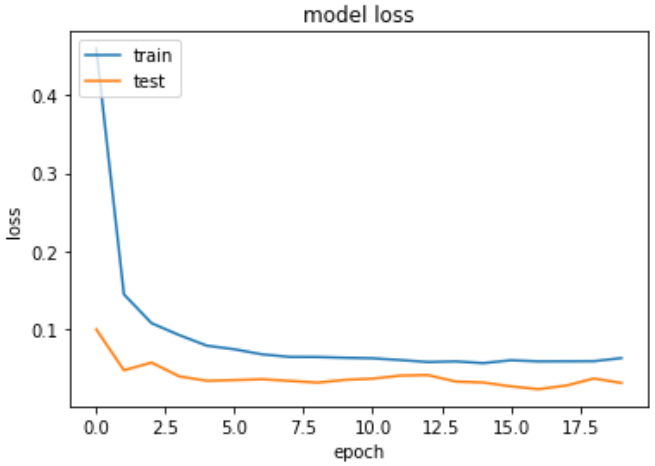

Figura 9 Histórico de perdas

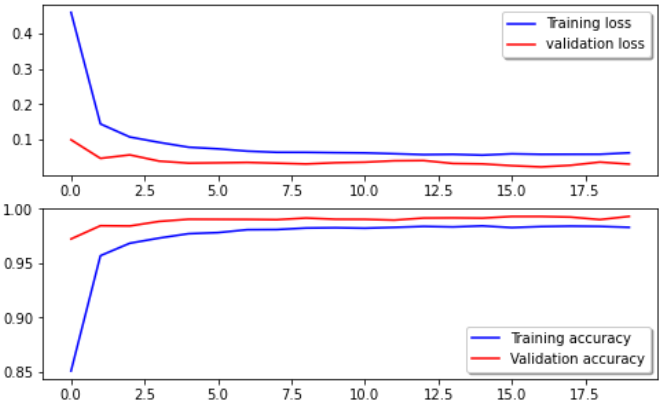

Figura 10 Curvas da perda e precisão para o treino e validação

A curva da precisão demonstra que à medida que aumenta o número de *epoch*, maior é a precisão do modelo, o mesmo acontece com a curva da perda, as perdas vão diminuíndo à medida que o *epoch* aumenta.

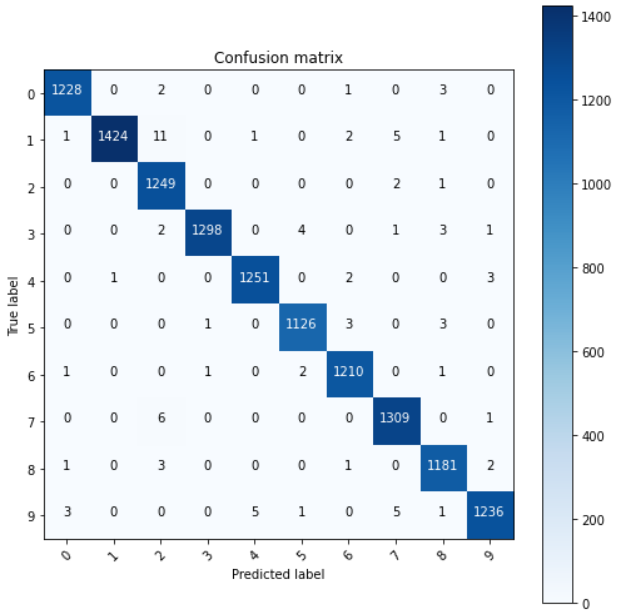

Figura 11 Confusion Matrix do modelo CNN

B. SVM

Ao conjunto treino irá ser realizada uma previsão linear e não linear, neste último será do tipo Gaussiano.

A biblioteca a utilizar será *sklearn.svm()*.

1) Linear

O modelo é implementado através da função *SVC(kernel='linear')*.

É implementado a função *fit()*, que treina o modelo.

Por ultimo é aplicada a função *predict()*, que fornece a previsão dos dados de teste, tendo o modelo conseguido prever com uma precisão de 90,9%.

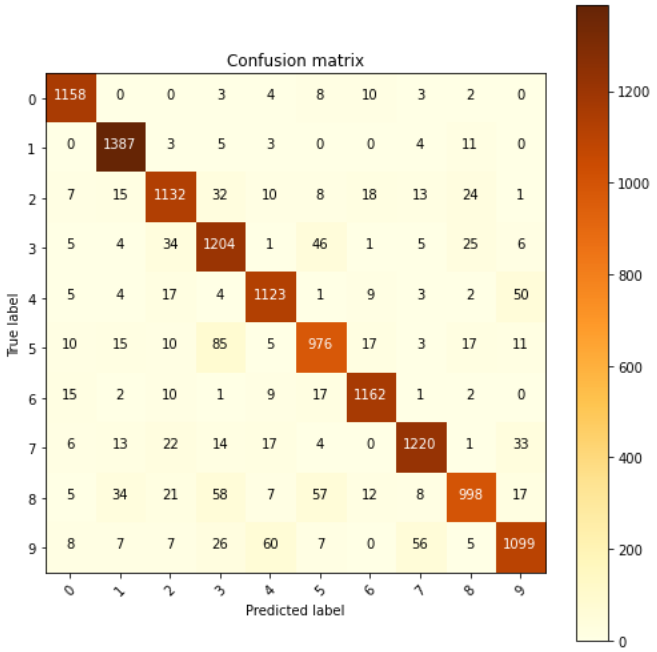

Figura 12 Confusion Matrix SVM Linear

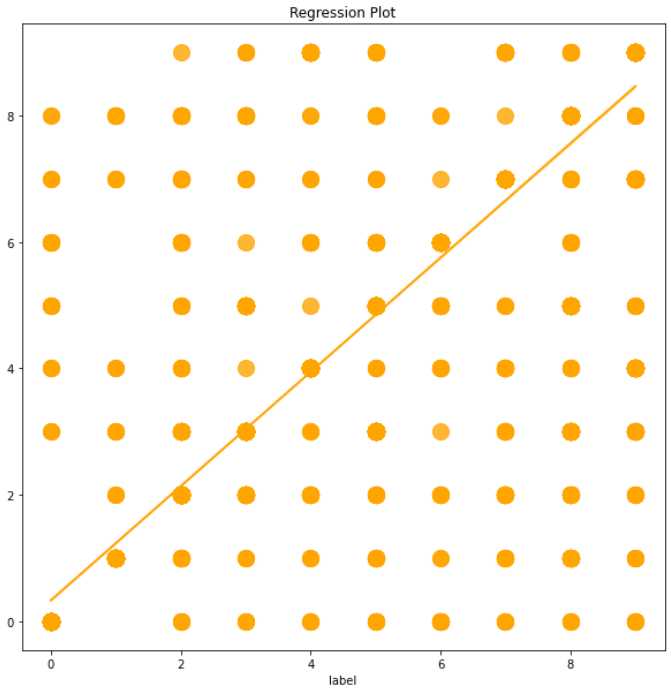

Figura 13 Regressão do modelo SVM Linear

2) Non Linear

O modelo é implementado através da função *SVC(kernel='rbf')*.

É implementado a função *fit()*, que treina o modelo.

Por ultimo é aplicada a função *predict()*, que fornece a previsão dos dados de teste, tendo o modelo conseguido prever com uma precisão de 96,0%.

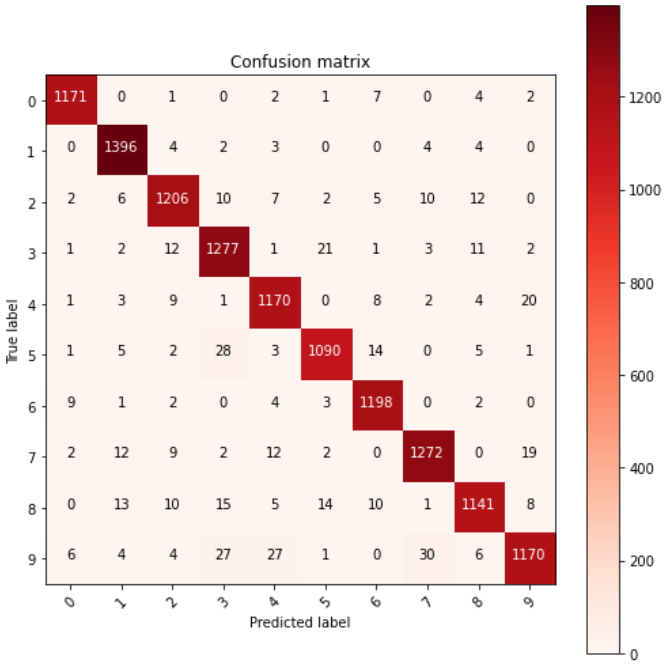

Figura 14 Confusion Matrix SVM não Linear

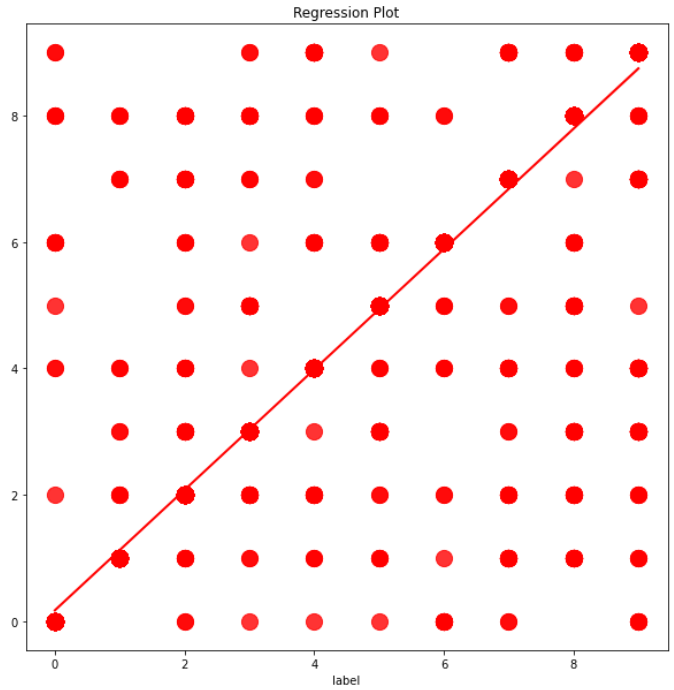

Figura 15 Regressão do modelo SVM não Linear

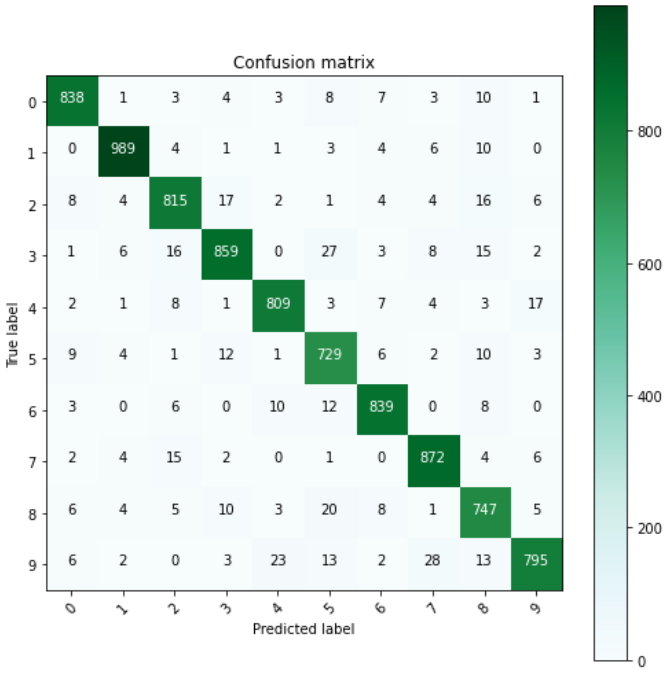

Figura 16 Confusion Matrix do modelo MLP

C. *MLP*

O modelo MLP é implementado através da função *MLPClassifier()*, pelas métricas de sklearn.

É implementada a função *fit()*, que treina o modelo.

Por ultimo é aplicada a função *predict()*, que fornece a previsão dos dados de teste, tendo o modelo conseguido prever com uma precisão de 96,2%.

Tabela 4 Tabela de previsão

	precision	recall	f1-score	support
0	0.98	0.99	0.98	1194
1	0.98	0.99	0.98	1414
2	0.95	0.97	0.96	1271
3	0.97	0.94	0.95	1326
4	0.97	0.95	0.96	1214
5	0.95	0.95	0.95	1125
6	0.96	0.98	0.97	1272
7	0.96	0.97	0.96	1345
8	0.95	0.95	0.95	1178
9	0.95	0.94	0.95	1261
accuracy			0.96	12600
macro avg	0.96	0.96	0.96	12600
weighted avg	0.96	0.96	0.96	12600

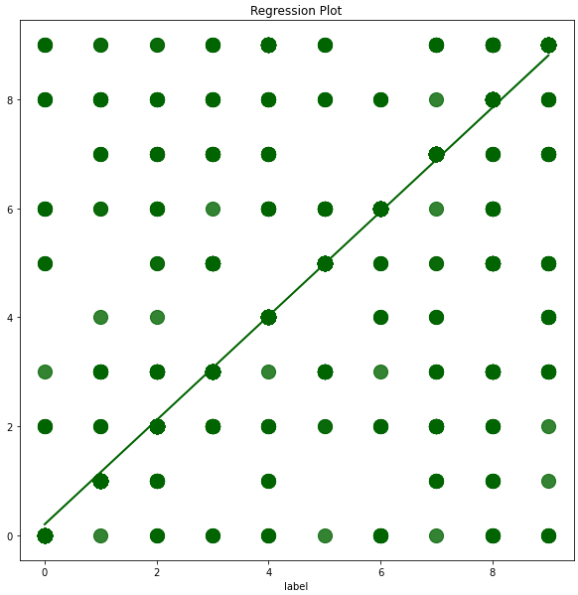

Figura 17 Regressão do modelo MLP

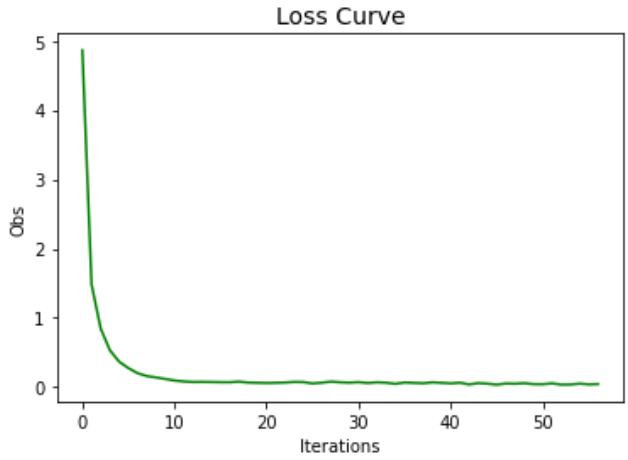

Figura 18 Curva da perda do modelo MLP

V. RESULTADOS

Dos três modelos em análise, o melhor que se adequa à previsão do *dataset*, é o modelo CNN, que permite uma precisão de previsão de 98,9%, este modelo, é no entanto, o mais complexo, visto que requer mais funcionalidades e demora Cerda de 1 hora, 19 minutos e 57 segundos para obter os resultados.

De referir também, que de entre os dois modelos SVM, linear e não linear, este último consegue uma melhor previsão, nomeadamente de 96,0%, não ficando muito atrás do modelo MLP, com 96,2%.

Tabela 5 Resultados

	Accuracy	Time
SVM Linear	0.909	19.24s
SVM non Linear	0.960	48.61s
MLP	0.962	1m:04s
CNN	0.989	1h:19m:57s

VI. CONCLUSÃO

Nesta análise, a partir de um conjunto de dados do Kaggle, foram implementados três modelos de reconhecimento de dígitos manuscritos, com base em algoritmos de Deep Learning e Machine Learning. O objetivo foi identificar o modelo mais preciso.

SVM é um dos classificadores básicos, é o mais rápido que a maioria dos algoritmos, mas devido à sua simplicidade, não é possível classificar imagens complexas e ambíguas com a maior precisão possível, como os algoritmos MLP e CNN.

O modelo CNN deu os resultados mais preciso para o reconhecimento de dígitos manuscritos, o que se conclui que é o modelo mais adequado para problemas de previsão em que o input corresponde a imagens.

REFERÊNCIAS

[1] S. Raschka, *Python machine learning: unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics*.

[2] R. Leenings *et al.*, “PHOTONAI-A Python API for rapid machine learning model development,” *PLoS ONE*, vol. 16, no. July, Jul. 2021, doi: 10.1371/journal.pone.0254062.

[3] R. Mu and X. Zeng, “A review of deep learning research,” *KSII Transactions on Internet and Information Systems*, vol. 13, no. 4, pp. 1738–1764, Apr. 2019, doi: 10.3837/tiis.2019.04.001.

[4] “Teoria e Aplicação de Support Vector Machines à Aprendizagem e Reconhecimento de Objetos.”

[5] A. R. Gonçalves, “Máquina de Vetores Suporte.” [Online]. Available: www.dca.fee.unicamp.br/~andreric

[6] Samet Girgin, “Kernel SVM (non Linear),” 2019.

[7] L. Noriega, “Multilayer Perceptron Tutorial,” 2005.

[8] “Matlab stem function figure Y = linspace(-2*pi,2*pi Normalization (image processing).” [Online]. Available: http://www.mathworks.com/matlabcentral/answers/26460-how-can-i-perform-gray-