REVISTA UIIPS

INSTITUTO POLITÉCNICO DE SANTARÉM

9 - 4 | 2021

Classification of Handwritten Digits

Classificação de dígitos manuscritos

Clasificación de dígitos manuscritos

Ana Rodrigues | Bernardo Rhodes | Daniel Cambinda | Pedro Lourenço | Ricardo Mota | Tiago Aires

Electronic version
URL: https://revistas.rcaap.pt/uiips/ ISSN: 2182-9608

Publisher
Revista UI_IPSantarém

Printed version
Date of publication: 31st December 2021 Number of pages: 14
ISSN: 2182-9608

Electronic reference

Rodrigues, A. Rhodes, B. Cambinda, D. Lourenço, P. Mota, R. & Aires, T. (2021). *Classification of Handwritten Digits.*Revista da UI_IPSantarém. *Edição Temática: Ciências Exatas e das Engenharias*. Número especial: Conferência Internacional Cooperação Internacional, multiculturalidade, trabalho colaborativo e ambientes mais inclusivos, sustentáveis e resilientes. 9(4), 44-57. https://revistas.rcaap.pt/uiips/

revistauiips@ipsantarem.pt
Complexo Andaluz,
Apartado 279
2001-904 Santarém

Revista da UI_IPSantarém – Unidade de Investigação do Instituto Politécnico de Santarém,
Vol. 9, N. º 4, 2021, pp. 44-57, Santarém
ISSN: 2182-9608
https://revistas.rcaap.pt/uiips/

CLASSIFICATION OF HANDWRITTEN DIGITS

Classificação de dígitos manuscritos

Clasificación de dígitos manuscritos

Ana Sofia Oliveira Rodrigues
Instituto Politécnico de Santarém, Portugal
190100383@esg.ipsantarem.pt

Bernardo Duarte Rodrigues Fragoso de Rhodes
Instituto Politécnico de Santarém, Portugal
190100124@esg.ipsantarem.pt

Daniel Alexandre Pires Correia Cambinda
Instituto Politécnico de Santarém, Portugal
190100190@esg.ipsantarem.pt

Pedro Jorge Ferreira Lourenço
Instituto Politécnico de Santarém, Portugal
190100117@esg.ipsantarem.pt

Ricardo Jorge Marcos Mota
Instituto Politécnico de Santarém, Portugal
190100138@esg.ipsantarem.pt

Tiago Correia Aires
Instituto Politécnico de Santarém, Portugal
190100114@esg.ipsantarem.pt

ABSTRACT

This article aims to introduce some concepts related to Artificial Intelligence, more specifically Machine Learning, and to present an example of a project related to the classification of handwritten digits in Python. To do so, this article follows a project studied in one of our classes of «AI», complemented by literature on the subject. Finally, the article presents results obtained by testing the algorithms in question, a corresponding interpretation, along with a discussion on the applicability and viability of this approach in real-life conditions.

Keywords: Machine Learning, Neural Network, Supervised Learning, Training set, Test set.

1 INTRODUCTION

In this era of digitalization, Artificial Intelligence (AI) is progressing rapidly into diverse areas. Society gradually sees the necessity for machines to be able to learn and mimic our actions, as humans. The use of Machine Learning has grown exponentially as it is used in our everyday life. It can be seen, for example, in image and speech recognition, medical diagnosis, advertising and many other fields of the industry.

This article intends to present one example of Machine Learning (ML). It introduces some related concepts. This investigation was guided by classes of Artificial Intelligence, with guidance from Professor Artur Marques, from the Polytechnic Institute of Santarém, and some online research apart from classes. As such, throughout this article, to better understand the concepts and theories presented, segments from relevant books and papers will be quoted. This article will also possess illustrations of some results and include images regarding the project's source code.

The project in this article - “Classification of Handwritten Digits" - was created and discussed during Artificial Intelligence classes. It is a generic example of how Machine Learning can happen and how a machine can learn to correctly classify handwritten digits. In short, the machine receives a series of pictures of handwritten digits and is expected to return their correct corresponding value or classification. Before mentioning, in detail, how the project works, some concepts need to be established for better understanding of its scope.

This is an article on supervised Machine Learning which reaches into related concepts, also mentioned in the source code itself, for an overall understanding of the whole project.

2 METHODOLOGY

2.1 Machine Learning

Machine Learning is precisely what one may think it is - it is about the Machine Learning something and can be achieved through unsupervised or Supervised Learning. This article will focus on the latter. Essentially, in Supervised Learning, the machine learns by analysing training data on a specific subject and having access to the correct classification for each sample in the data. Thus, the dataset is a set of examples for the machine to “learn” from. In the end, it is expected that the machine will be able to make correct classifications when presented with new data - data that was not included in these examples. Usually, the more data we give the machine, the more chances we create for the machine to learn, making it more capable of correctly classifying samples predicting their corresponding classes.

2.2 Supervised Learning VS Unsupervised Learning

For many projects, including the one at hand, Supervised Learning is used, as stated before. This term is a subset of Machine Learning and is based on training. Its concept revolves around using

algorithms that learn from supervised pairs (input and desired output), “it is defined by its use of labelled datasets that classify data or predict outcomes accurately” (IBM, 2020A). After giving the machine such labelled datasets, another dataset, different from training one, is used for testing, to measure the accuracy of the model the machine built, while learning.

Data for testing is also labelled, but the algorithm is only fed with inputs, not classifications. After running the algorithm, one can check if the computed outputs match the correct labels, or not. If the accuracy is too low, one solution might be to increase the amount of data available for training, or change its quality, or change the entire model complexity.

What if the entire data in a dataset is used only for training, and nothing is reserved for testing? This is where the terms overfitting and underfitting enter. If give all data is given to training, one may risk "overfitting", where the model learns the detail and noise of the training data to the extent that it might negatively affect its performance on new data. If most of the data was used for testing, at the cost of not enough examples being used, that might cause "underfitting", where the model is lacking in capturing the relationship between the input and output variables accurately, resulting in a low accuracy algorithm.

Being Supervised Learning key concept regarding this project, there is still a need to explain what Unsupervised Learning is and what it stands for.

Unsupervised Learning, unlike Supervised Learning, uses unlabelled data. For IBM (2020B) “It uses machine learning algorithms to analyse and cluster” those same unlabelled datasets mentioned earlier. This method allows the machine to work on its own to discover patterns and information, making it able to process complex tasks when comparing to its counterpart. Unsupervised Learning is more suitable for problems that require the algorithms to find and extract similarities between the inputs, so that those can be categorised together. There are two fundamental types of Unsupervised Learning methods: clustering and density estimation. The former involves problems where we need to group the data into specific categories while the latter involves summarizing the distribution of the data.

2.3 Supervised Learning Algorithms

Now that the concepts behind Machine Learning have been introduced, this section explores what algorithms and methods compose Supervised Learning. A variety of them can be found, all having datasets for training and testing but used for different purposes. Even though this project only uses one of these algorithms (Neural Networks), it is still required to explore other methods and explain them, thus comprehending why Neural Networks is the most suitable for the project at hand.

2.3.1 Neural Networks and Deep Learning

Neural Networks is a very well-known term that was inspired in the real human brain. This algorithm utterly refers to networks composed of neurons that transform data from an input to an output.

Furthermore, since Neural Networks are used on the project explored ahead, it is essential to understand them concisely before being able to understand the project. This article will be quoting some fundamental segments of a book called “Neural Networks: Methodology and Applications” to help explaining some basic concepts as: neuron, Neural Networks, activation function, neural network training, feedforward, and feedback networks.

According to the author of the book in question, “a neuron is a nonlinear, parameterized, bounded function” (Dreyfus, 2005, p.2). For instance, a linear parameterized function is called a linear neuron. Their variables are often called inputs of the neuron and its value is its output. A neuron being nonlinear means that it works with an activation function. This activation function is a function that transforms the summed weighted input from the node to an output value to be fed to the output layer. There can be several activation functions, such as sigmoid, that takes any real value as input and gives an output value in the range of 0 to 1. Another relevant activation function is called *tanh*

function, also known as hyperbolic tangent, that is remarkably like the previous one but instead of having a range of 0 to 1, it has a range of -1 to 1.

Now that these concepts have been established, it seems simple to understand what a network of neurons is: the composition of the nonlinear functions of two or more neurons. As such, Neural Networks come in two classes: feedforward networks and recurrent (or feedback) networks. Feedforward networks refer to when the neurons are only allowed to travel one way, from the input to the output. The recurrent networks use the previous output from the neuron as an input of the current one, forming a cycle. In this next image (figure 1), the process of how feed-forward and recurrent networks perform is illustrated.

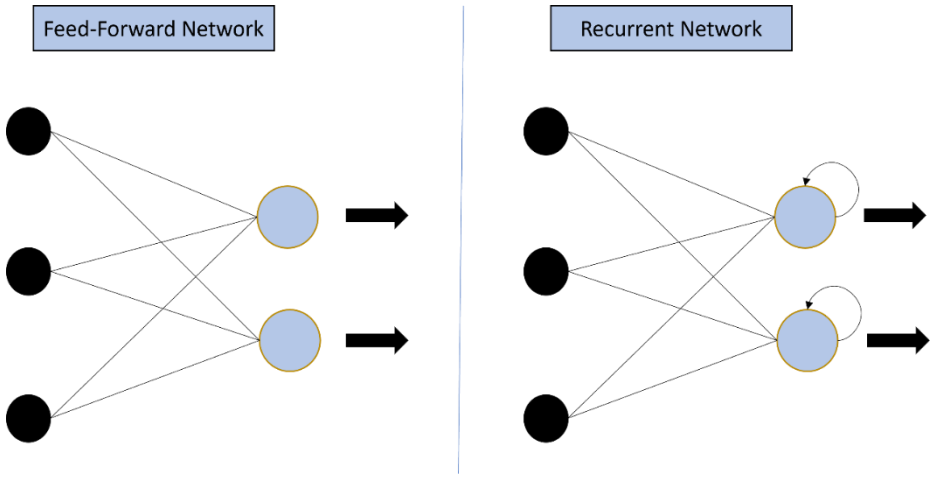

Figure 1 - Feed-Forward and Recurrent NN

To help understand how Neural Networks work, a sketch that represents an example of a neural network was drawn (figure 2), being, in this case, a feed-forward neural network. In any neural network tree, different layers subsist, as can be seen below: the input, the hidden layer, and the output. The first layer of a neural network is responsible for accepting a certain type of data and passing it to the rest of the network. The hidden layer oversees all sorts of computation to transform the data using an Activation function. As the name of the layer indicates, the processes that occur in the layer are not directly observable from the inputs and outputs of the system. Finally, the last layer has the purpose of giving the output of the transformed data.

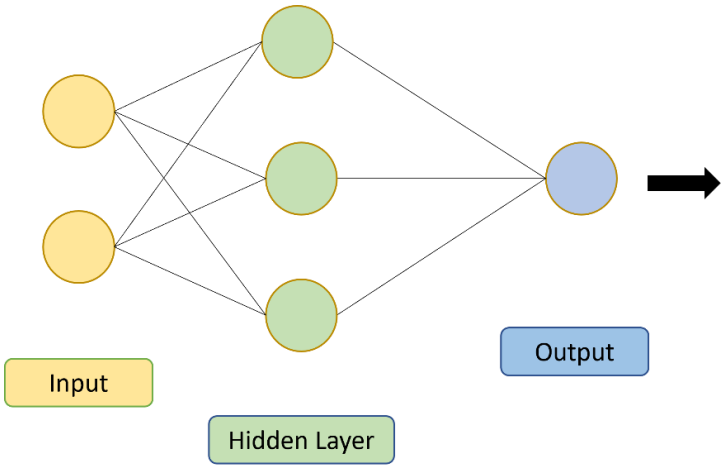

Figure 2 – Representation of a Neural Network

When a neural network has more than one hidden layer, it is a deep neural network, which makes it Deep Learning. This concept is a form of Machine Learning that trains machines to execute tasks just like humans do. However, while traditional Machine Learning algorithms are linear, Deep Learning algorithms are stacked in a layered hierarchy of concepts, more specifically, of increasing complexity and abstraction. Given its extended architecture, deep Neural Networks can process more data, resulting in the ability to extract higher-level and more efficient predictions from the raw input. As a result of using more hidden layers, Deep Learning algorithms can be supervised, semi-supervised and unsupervised.

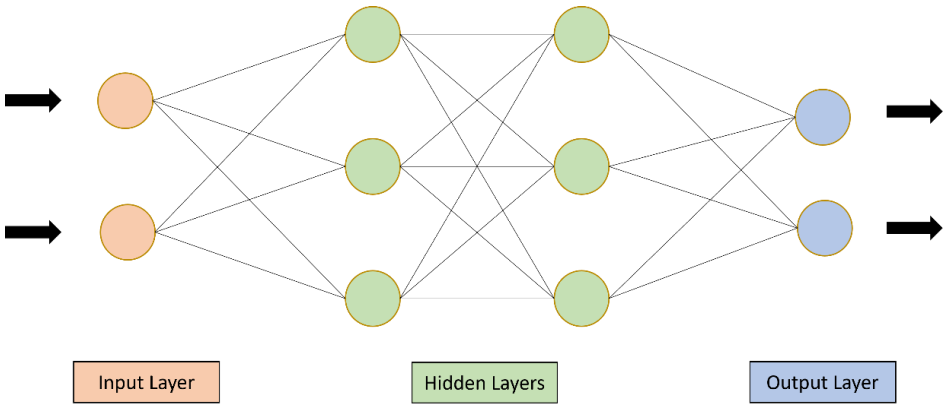

Figure 3 - Deep Neural Network

2.3.2 K-Nearest Neighbours

Another relevant Supervised Learning algorithm is K-Nearest Neighbours, also known as KNN. This designation refers to “a non-parametric algorithm that classifies data points based on their proximity and association to other available data” (IBM, 2020A). This algorithm assumes that similar data points will be found near each other. In other words, it will calculate the distance between data points, and then assign them a category based on the most frequent one. Typically, the KNN algorithm calculates the distance between data points through the Euclidean distance.

This kind of algorithm is mostly used for recommendation engines and image recognition, since as the test dataset grows, the processing time lengthens, making it less appealing for classification tasks. On the other hand, its easy use and low calculation time makes it a preferred algorithm by data scientists.

2.3.3 Linear Regression

According to IBM (2020A), “Linear Regression is used to identify the relationship between a dependent variable and one or more independent variables”. It is commonly used to make predictions about future outcomes. It is known as simple Linear Regression when there is only one of each variable. On the other hand, as the number of independent variables increases, it is referred to as multiple Linear Regression.

For each type of Linear Regression, a line of best fit is calculated through the method of least squares.

3 CLASSIFICATION OF HANDWRITTEN DIGITS

The algorithm presented next is an example of how to do image classification of handwritten digits as mentioned before. In this software one could submit his own handwritten decimal digit (0...9) image and the algorithm will predict the class of the input with ten predefined classes labelled with

the numbers 0 to 9. The predictions will be, for example, if a picture of a handwritten digit is a 1 or a 2.

To be able to make this project some generic steps must be performed. Firstly, as any other program, one needs to have all the entities necessary to proceed with coding, such as importing all the modules required and creating variables representing the data and some other operations in the project.

In a second phase data needs to be pre-processed, which refers to the transformations applied to data before feeding it to the algorithm. It converts the raw data, not feasible for the analysis, into clean data, ready to be modelled.

After pre-processing the data, a mode is ready to be created. The latter is, essentially, a file that has been trained to recognize certain types of patterns. Initially, the base of the model is created and is then given training data.

Lastly, a model evaluation is performed with test data (to avoid overfitting), in other words, the model's accuracy is evaluated to make predictions. Afterwards, the model is ready to work on its own predictions.

4 METHOD

4.1 Imports and Variables

As stated before, this project uses Python as its programming language. In the following images, some pieces of code will be presented by order while explaining the methods used to implement them.

First, as can be seen in Figures 4, 5 and 6, some imports are required such as the database holding all the pictures of the handwritten digits used for training and testing the machine called MNIST (Modified National Institute of Standards and Technology database). The database contains 70.000 pictures, being 60.000 for training and 10.000 for testing the data. This database is imported from «TensorFlow», that is a very high-levelled library related to Machine Learning and, on top, «Keras», a high-level API from «TensorFlow» used to create and train models for Deep Learning.

It is also imported the display module from «IPython», a multilanguage command shell for interactive computing created for Python. This display module from this command shell is used to display data, being, in this case, an image that summarizes what will happen. Next, «Pyplot», naming it plt, from «Matplotlib» library, which is used for creating static, animated and interactive visualizations in Python. «Pyplot» is a state-based interface that supplies a way of plotting. It can also open figures on the screen and act as a figure GUI (graphical user interface) manager.

Additionally, «NumPy» is imported, an open-source project that enables numerical computing with Python. This library offers comprehensive mathematical functions, random number's generators, linear algebra routines, and more. More importantly, this library is imported to be able to use arrays with the data needed for this project. The last three imports consist of a function called «to_categorical» that is used to convert class vectors(integers) to a binary class matrix, a class called sequential that groups a linear stack of layers into an object with training and inference features and a regular densely connected neural network layer named dense.

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from IPython import display
from matplotlib import pyplot as plt
import numpy as np #if a shorter syntax is wanted
import numpy #if the explicit usage of numpy is preferred
```

Figure 4 - Print of code (imports)

```
140 from tensorflow.keras.utils import to_categorical
```

Figure 5 - Print of code (imports)

```
210 from tensorflow.keras.models import Sequential
211 from tensorflow.keras.layers import Dense
```

Figure 6 - Print of code (imports)

In the next section (figure 7), the variables needed for the rest of the code are declared. Initially, two variables that store the data from the MNIST database are created, then two arrays, using the «NumPy» library, that saves the data of the images used for training and the labels associated. A tuple (list of elements) is created that keep the shape of the images and labels. Finally, the same variables are declared again, only this time for the testing dataset.

```
#display an image that summarizes what will happen
display.Image("./1_1.png")

tupleForTraining, tupleForTesting = mnist.load_data()

INDEX_OF_IMAGES : int = 0
INDEX_OF_LABELS : int = 1
npaImagesForTraining : numpy.ndarray = tupleForTraining[INDEX_OF_IMAGES]
npaLabelsForTraining : numpy.ndarray = tupleForTraining[INDEX_OF_LABELS]

tuple3ShapeOfTrainingImages : tuple = npaImagesForTraining.shape #(60000, 28, 28) #60K 3-tuple
tuple3ShapeOfTrainingLabels : tuple = npaLabelsForTraining.shape #(60000,) #60K 1-tuple uint8

npaImagesForTesting = tupleForTesting[INDEX_OF_IMAGES]
npaLabelsForTesting = tupleForTesting[INDEX_OF_LABELS]
tuple3ShapeOfTestingImages : tuple = npaImagesForTesting.shape
tuple3ShapeOfTestingLabels : tuple = npaLabelsForTesting.shape
```

Figure 7 - Print of code (Variables)

4.2 Pre-processing

The next images in this phase (figure 8 and 9) show a crucial step in the code, pre-processing the data. This essential step in any Machine Learning project is when one gets its data ready for modelling.

Most Machine Learning tutorials and tools require preparation of the data before it can be fit to a particular Machine Learning model. For this reason, one needs to prepare this data using one-hot encoding. One-hot encoding plays a big part in the computer system's world. It allows verification for

data leaks, which makes it easier for the computer systems to manipulate and store data. This terminology is a process by which categorical variables are converted into a form that could be provided to Machine Learning algorithms, to help them do a better job in prediction. This process works by converting categorical values into a new column and values itself into binary (0 and 1).

In the following piece of code (figure 8), it is represented the “one-hot encoding” for the labels of the train and test sets. The function «to_categorical», imported in the beginning, is used to transform the data into binary vectors, so that they can be provided to Machine Learning algorithms to improve predictions, as mentioned earlier.

```
#will do the "one hot encoding" from both the labels of the train AND test sets
npaTrainLabelsAs10DVectors = y_train_encoded = to_categorical(npaLabelsForTraining)
npaTestLabelsAs10DVectors = y_test_encoded = to_categorical(npaLabelsForTesting)
```

Figure 8 - Print of code (one-hot encoding)

At the beginning of figure 9, the reshaping of the images for training and testing is made, making each one of them a fixed image of twenty-eight-by-twenty-eight pixels. Using 60.000 pictures for training and 10.000 for testing. This chosen amount for the number of images that is being used for training and testing data were not chosen by chance. They were divided in a way that can possibly bring the best results to prevent the machine from over or underfitting, a concept covered before.

After reshaping the data to the pretended size, one must calculate the mean and the standard deviation (a measure of how dispersed the data is in relation to the mean) of the dataset for training. With these two values one can do the normalization, an important step in pre-processing the data. The data can be normalized by subtracting the mean of each feature and a division by the standard deviation. This formula is used to convert the source data to another format that allows processing data to effectively occur. This process allows to minimize any duplicated data.

```
npaImagesForTrainingReshapedLinear = np.reshape(npaImagesForTraining, (60000, 28 * 28))
npaImagesForTestingReshapedLinear = np.reshape(npaImagesForTesting, (10000, 28 * 28))
print('npaImagesForTrainingReshapedLinearized shape: ', npaImagesForTrainingReshapedLinear)
print('npaImagesForTestingReshapedLinearized shape: ', npaImagesForTestingReshapedLinear)

#normalize using mean and standard deviation
fMeanOfTrainingImages = np.mean(npaImagesForTrainingReshapedLinear)
fStdOfTrainingImages = np.std(npaImagesForTrainingReshapedLinear)
#define a small value const
SMALL = 1e-10
trainingImagesNormalized = \
    (npaImagesForTrainingReshapedLinear - fMeanOfTrainingImages) / (fStdOfTrainingImages + SMALL)
#using the mean and the std obtained from the training data - the is a pattern in data science
testingImagesNormalized = \
    (npaImagesForTestingReshapedLinear - fMeanOfTrainingImages) / (fStdOfTrainingImages + SMALL)
```

Figure 9 - Print of code (Reshaping and Normalizing Data)

4.3 Model Creating

The figure below represents the creation of the model used in this project, which is a neural network, by using the API sequential, imported before. To create this model, a list of layers is passed to the sequential constructor. The first layer passed to the model is the hidden layer that has a shape of twenty-eight-by-twenty-eight pixels. The second value passed is the input layer, that contains the input values and the input shape, already implied in the hidden layer. And the last parameter is the output layer that holds ten classes.

Then, the model is compiled, defining the loss function, the optimizer, and the metrics. This step is needed to train the model, as it uses these three parameters, although not being necessary to compile a model for predicting.

```
iUnits = 16
model = Sequential([
    Dense(iUnits, activation="relu", input_shape=(28*28,)),
    Dense(iUnits, activation="relu"),
    Dense(10, activation="softmax")
])
compileRes = model.compile(
    optimizer="sgd", #stochastic gradient descent
    loss="categorical_crossentropy",
    metrics=['accuracy']
)
#the loss is the func that needs to be minimized, for ALL the samples, for accurate outputs
summaryRes = model.summary()
```

Figure 10 - Print of code (Creation of a model)

4.4 Model Training

Now, as it is represented in figure 11, it is time to train the model. In another words, the machine needs to learn through the given training dataset, both the images and labels received for each image. The more data is given to the model to train, the longer it takes.

The function fit receives three parameters: the images of the handwritten digits, their labels, and the epochs. The parameter “epoch” corresponds to the number of times that the training set will run through the samples, to train the model.

```
fitRes = model.fit(trainingImagesNormalized,npaTrainLabelsAs10DVectors,epochs=3)
```

Figure 11 - Print of code (Model Training)

4.5 Model Evaluation

Model evaluation is an integral phase of the model development process. It helps to quantify the quality of a system’s predictions.

In the image below (figure 12) the model is tested, and it determines loss and accuracy of the predictions. Loss is the sum of the errors made for each example and the accuracy is the measure of the algorithm performance. So, the higher the loss, the worse it is for the model because it means the machine is giving a greater number of false predictions.

```
fLoss, fAccuracy = model.evaluate(
    npaImagesForTestingReshapedLinear,
    npaTestLabelsAs10DVectors
)
# if accuracy is less than acc from the final epoch, that might mean the model just memorized
print('Test col accuracy: ', fAccuracy * 100)
print('Test col loss: ', fLoss)
```

Figure 12 - Print of code (Model Evaluation)

4.6 Model Prediction

In the figure below (figure 13), the samples were inserted into the model, for the machine to start processing them and predict a value for the received image, printing it.

```
npaPredictions = model.predict(npaImagesForTestingReshapedLinear)
print('Shape of npaPredictions: ', npaPredictions.shape)
```

Figure 13 - Print of code (Model Predictions)

4.7 Model Plotting

The model plotting is the final step of this project. To simply put it, it shows a graphical representation that verifies if the prediction value will be the same as the real value of the image in question. If they are the same, it will display a green coloured sentence. Otherwise, it will display the predicted value with a red colour.

```
plt.figure(figsize=(16, 16))
for i in range(25):
    #rows, cols, index starting at 1 in the upper-left corner
    plt.subplot(5, 5, i+1)
    plt.grid(False)

    predictedClass = np.argmax(npaPredictions[i])
    actualObservedClass = npaLabelsForTesting[i]

    colorGreenForOKRedForFailedPredictions = 'green'
    if (predictedClass != actualObservedClass):
        colorGreenForOKRedForFailedPredictions = 'red'

    strFormat = "i=%d, prediction=%d, observed=%d"%(i+1, predictedClass, actualObservedClass)
    plt.xlabel(strFormat,color=colorGreenForOKRedForFailedPredictions)
    plt.imshow(npaImagesForTesting[i],cmap="binary")

plt.show()
plt.close()
```

Figure 14 - Print of code (Model Plotting)

5 RESULTS

After compiling the code, a graphic representation of the handwritten digits is obtained, as well as the labels associated to each of them. These labels include the position of every single image, as well as the prediction and the actual number in each of them. In figure 15, one can observe the output of executing the code - this output may vary for each iteration. In this case, the machine correctly predicts the values for every single number except for the number 3, in the position 19 (i = 19), since the machine predicted the number 2.

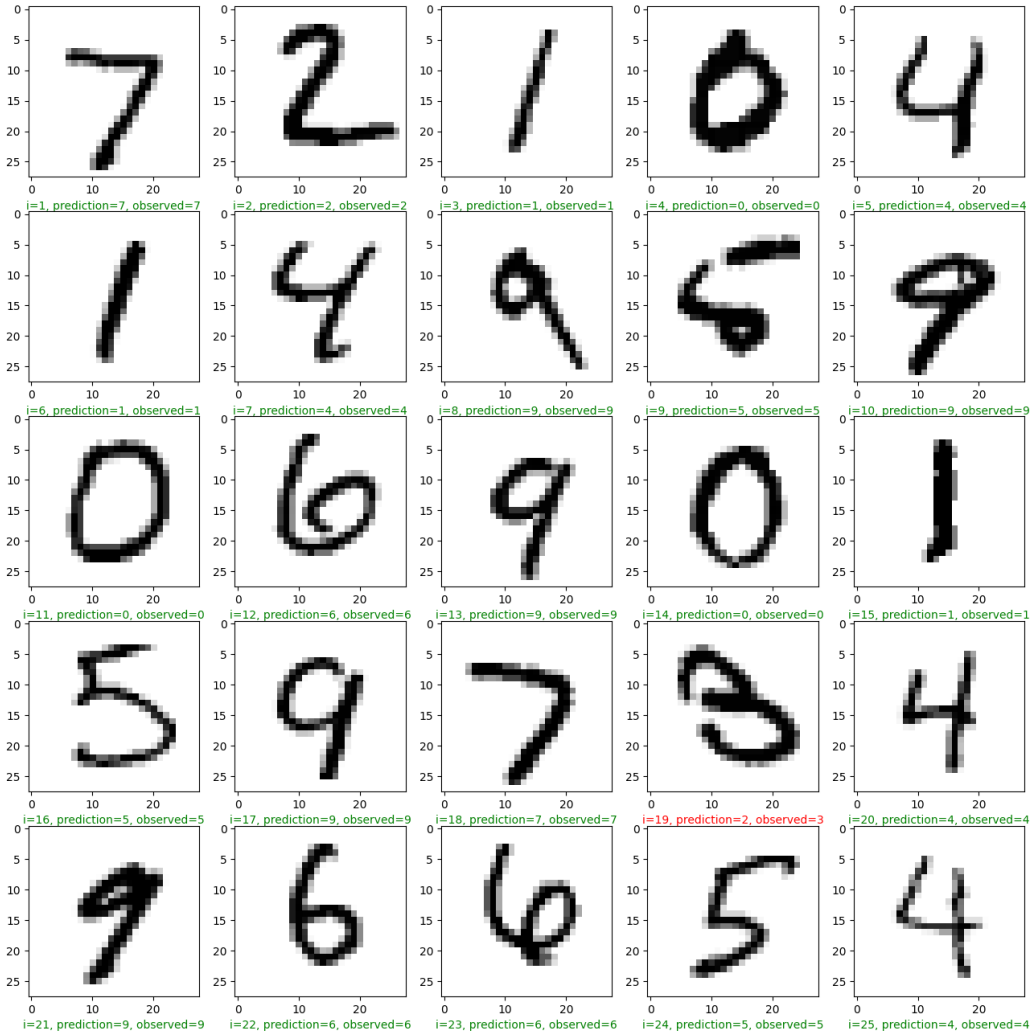

Figure 15 - Results

Below there is a table that represents ten iterations of executing the algorithm, in the same environment. Each of these trials are accounted for in the first column, as well as the failed predictions and their corresponding position for each one of them, along with the percentage of correctly predicted images.

# Test	# Failed predictions	Position (i) of failed images	Correctly predicted images (%)
1	1	i = 19	96%
2	1	i = 9	96%
3	3	i = 7 ; i = 9 ; i = 19	88%
4	2	i = 9 ; i = 19	92%
5	2	i = 9 ; i = 16	92%
6	3	i = 5 ; i = 9 ; i = 19	88%

7	4	i = 9 ; i = 16 ; i = 19 ; i = 24	84%
8	2	i = 9 ; i = 19	92%
9	2	i = 9 ; i = 16	92%
10	1	i = 9	96%

Table 1: Outcome of ten iterations of the algorithm

6 DISCUSSION OF RESULTS

As stated before, the algorithms were iterated ten times in a row, to obtain a significant sample that would allow an examination of results.

Firstly, the overall accuracy of the algorithm rounds to 91.6%, which is a desirable value given the environment on which it was tested. It is also possible to observe that the lowest value was 84%, occurring in only one of the ten iterations (10%), as well as 96% being the highest value obtained, observed in three of the ten iterations (30%) - never obtaining 100% accuracy. The mode of this sample would be 92%, since it is the most common value obtained.

Secondly, it is possible to observe that the machine failed to predict the number in 9^{th} position (i = 9) 90% of the trials (9 in 10) - which corresponds to the number “5”. This recurring failed prediction could be explained simply by looking at the graphical representation of the image, on figure 15, where even to the human eye, it is difficult to recognize the image in question as the number 5.

Lastly, a hypothesis can be formed that human error is at cause on the failure to recognize both numbers 3 and 5 mentioned before, being i = 19 and i = 9 respectively. Since this algorithm is based on Supervised Learning, and the machine is being given less explicit examples, there is a higher probability that the machine will fail, as it is following a human behaviour.

7 CONCLUSION

Over the course of this article, some fundamental Machine Learning concepts and algorithms were explored in detail, mainly the most relevant ones regarding this project, such as Supervised and Unsupervised Learning. The theory and functionality behind Supervised vs Unsupervised Learning were analysed, and simple explanations of each learning algorithm were given in hope to illustrate its usability and why they differ. Since Neural Networks played such an important part and were the main algorithm used in this project, a more in-depth explanation was required, and some visual examples were displayed. It was also relevant to mention other fundamental algorithms, such as Linear Regression and K-Nearest Neighbours, regarding Supervised Learning, as well as understanding that one algorithm can be supervised, semi-supervised and unsupervised, as is the case for deep Neural Networks.

As for the methods used to conduct this project, those were provided in class with resort to third-party applications and resources, such as the MNIST database for training of the model and the «plotlib» library. The implementation of this methods was documented in a way to allow its reproduction using screen-captures of relevant code and should provide an easy understanding of the process.

Additionally, after the observation of the results on the project, it is possible to deduce that the poor quality and illegible handwriting was the most probable reason for inaccurate handwritten digits recognition, with the overall accuracy being high for the given context.

No process goes without its setbacks, as was the case for this, where some required packages were missing from the IDE and its installation was not as simple as it should be, requiring some workarounds.

Conducting this project provided a first contact with Artificial Intelligence and its capabilities along with its usability, proving itself of immense value for us as students for future work in the matter.

8 REFERENCES

Al-Shehari, T., & Alsowail, R. A. (2021). An Insider Data Leakage Detection Using One-Hot Encoding, Synthetic Minority Oversampling and Machine Learning Techniques. Entropy, 23(10), 1258. https://doi.org/10.3390/e23101258

Dreyfus, G. (2005). Neural Networks: Methodology and Applications. Springer Science & Business Media.

el Kessab, B.-E., Cherki, D., Belaid, B., Fakir, M., & Moro, K. (2013). Extraction Method of Handwritten Digit Recognition Tested on the MNIST Database.

Hodnett, M., & Wiley, J. F. (2018). R Deep Learning Essentials: A step-by-step guide to building deep learning models using TensorFlow, Keras, and MXNet, 2nd Edition. Packt Publishing Ltd.

Keras | TensorFlow Core. (n.d.). TensorFlow. Retrieved 7 December 2021, from https://www.tensorflow.org/guide/keras?hl=pt

Marques, A. (n.d.). IA. Retrieved 7 December 2021, from https://arturmarques.com/edu/ia/

Module: Tf.keras.datasets.mnist | TensorFlow Core v2.7.0. (n.d.-a). TensorFlow. Retrieved 7 December 2021, from https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist?hl=pt

Module: Tf.keras.models | TensorFlow Core v2.7.0. (n.d.-b). TensorFlow. Retrieved 7 December 2021, from https://www.tensorflow.org/api_docs/python/tf/keras/models?hl=pt

Parmelee, D. (2021, January 22). How to classify handwritten digits in python. Medium. https://towardsdatascience.com/how-to-classify-handwritten-digits-in-python-7706b1ab93a3

Pashine, S., Dixit, R., & Kushwah, R. (2020). Handwritten Digit Recognition using Machine and Deep Learning Algorithms. International Journal of Computer Applications, 176(42), 27–33. https://doi.org/10.5120/ijca2020920550

Seth, A. (2021). Handwritten Digit Classification. 8(7), 20.

Siddique, F., Sakib, S., & Siddique, A. B. (n.d.). Recognition of Handwritten Digit using Convolutional Neural Network in Python with Tensorflow and Comparison of Performance for Various Hidden Layers. 6.

Tf.keras.layers.Dense | TensorFlow Core v2.7.0. (n.d.). TensorFlow. Retrieved 7 December 2021, from https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense?hl=pt

Tf.keras.utils.to_categorical | TensorFlow Core v2.7.0. (n.d.). TensorFlow. Retrieved 7 December 2021, from https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categorical?hl=pt

Tibshirani, S., & Friedman, H. (n.d.). Valerie and Patrick Hastie. 764.

What is Supervised Learning? (2020A, August 19). https://www.ibm.com/cloud/learn/supervised-learning

What is Underfitting? (2021, March 23). https://www.ibm.com/cloud/learn/underfitting

What is Unsupervised Learning? (2020B, September 21). https://www.ibm.com/cloud/learn/unsupervised-learning