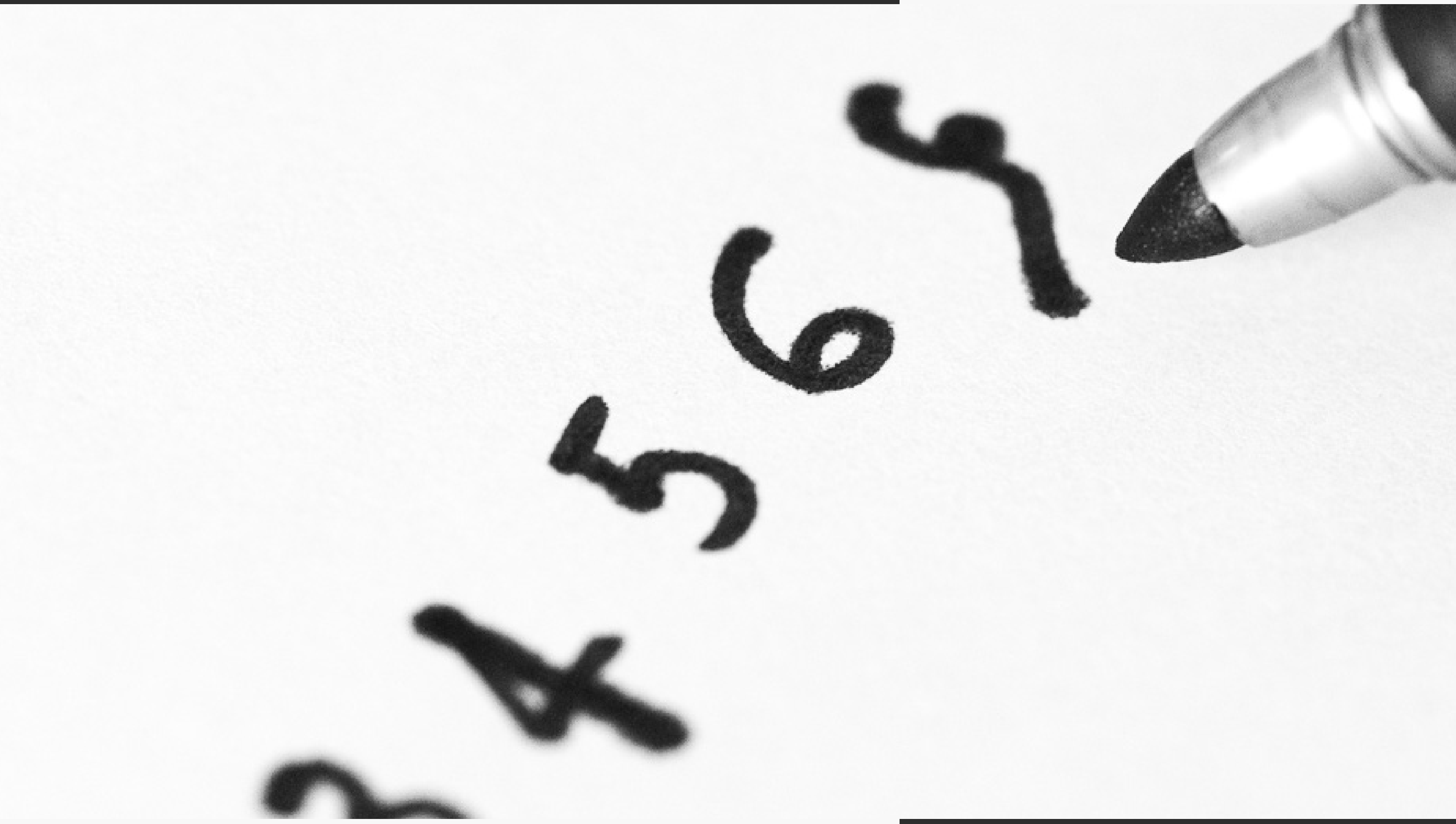
28 NOVEMBER 2022



Handwritten Digit Recognition

Data Mining Project Report

PREPARED FOR
Ms. Nassima DIF

PREBARED BY
BENDJEDDOU Randa Cheima
DJELLALI Fouad
MESSADI Said Abdesslem
BENAISSA Abir
HANANI Fetheddine

Table of contents

Table of figures

1. Abstract

This report presents a comparison of five classification algorithms that are “K Nearest Neighbor”, “Decision Tree”, “Support Vector Machine”, “Artificial Neural Network” and “Conventional Neural Network” using the combination of ARDIS dataset and MNIST dataset. The objective of this comparison is to find out the best classifier among the five ones that can give an acceptable accuracy rate using a minimum number of selected features.

The accuracy measurement parameter is used to assess the performance of each classifier individually, which is the accuracy. The results show that Conventional Neural Network (CNN) algorithm gives better recognition rate than others and it reached an accuracy of 98.12%.

2. Introduction

Handwritten digits recognition, which is the ability of computers to recognize human handwritten digits, is considered as a core to a diversity of emerging applications. It is used widely by computer vision and machine learning researchers for performing practical applications such as computerized bank check numbers reading.

However, executing a computerized system to carry out certain types of duties is not easy and it is a challenging matter. Recognizing the numeral handwriting of a person from another is a hard task because each individual has a unique handwriting way.

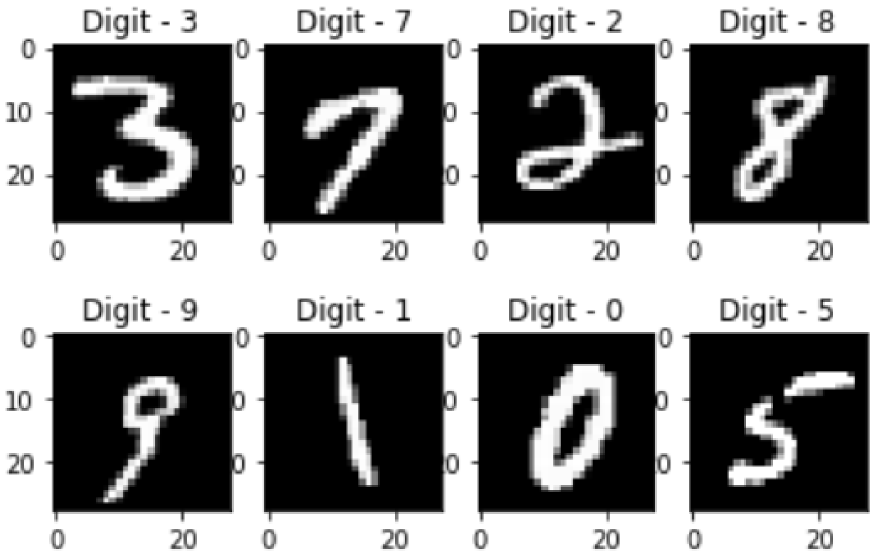


Figure 1 : Handwritten Digit Recognition

Many proposed system and classification techniques have been developed for this area, proper accuracy of predicting the pattern is still questionable.

So the comparison of proper techniques became a challenge and seems difficult to determine the best one because their performance is data-dependent. It also depends on many factors including high accuracy, low run time, low memory requirement and reasonable training time.

Thus, we aimed to study and compare five classifier techniques, K-Nearest Neighbor, Decision Tree, SVM, Artificial Neural Network and Conventional Neural Network. The performance evolution (accuracy, training time,) is done to analyze the various classification algorithms to select the proper classifier for handwritten digit recognition.

To do so, we took a Handwritten Digits Data Set from ARDIS but the data set was very tiny which cause problem for training the models, so we made a combination of the ARDIS dataset with the MINIST dataset .

This report is organized as follows: In next section, we present the ARDIS and MINIST handwritten digit dataset and how we prepared them. In Section 3, we have discussed the implementation of the five classification techniques (i.e. CNN, ANN, SVM, Decision Tree and K-Nearest Neighbor Classifier) used for recognizing handwritten digits. We also compare the classification accuracy among them and analyze the experimental result on Section 4. In section 5, we present the mobile application realized to make some examples of the predictions. Finally, Section 6 contains the conclusion.

Dataset

3. Dataset

3.1 ARDIS dataset

It is a new image-based handwritten historical digit dataset named ARDIS (Arkiv Digital Sweden). The images in the ARDIS dataset are extracted from 15.000 Swedish church records which were written by different priests with various handwriting styles in the nineteenth and twentieth centuries. The dataset contains 7600 single digit images from zero to nine in different color spaces with different shapes.

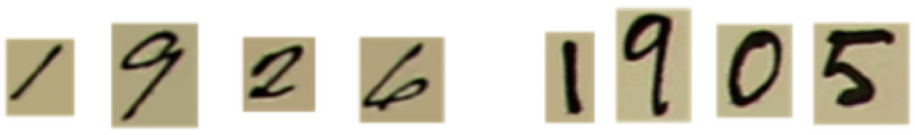


Figure 2 : ARDIS dataset

3.2. MNIST dataset

This is probably one of the most popular datasets among machine learning and deep learning enthusiasts. The MNIST dataset contains 60,000 training images of handwritten digits from zero to nine and 10,000 images for testing. So, the MNIST dataset has 10 different classes. The handwritten digits images are represented as a 28×28 matrix where each cell contains grayscale pixel value.

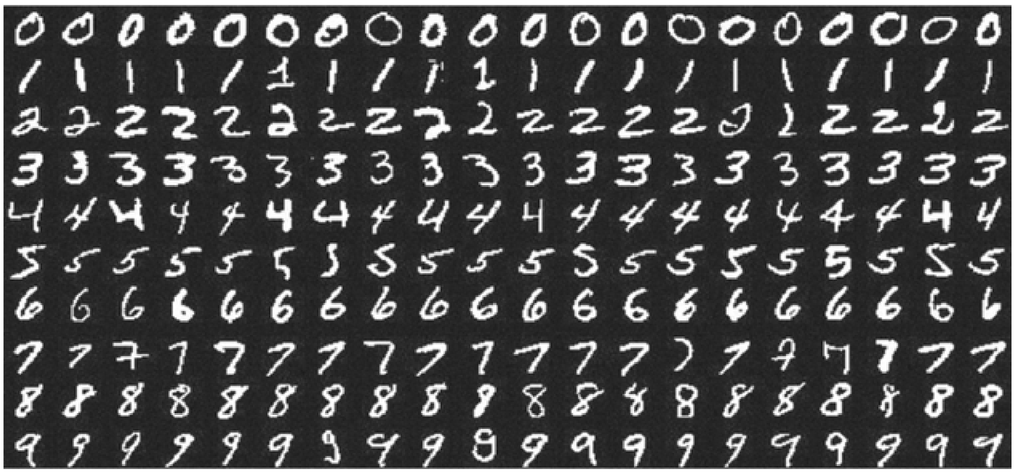


Figure 3 : MINIST dataset

4. Preprocess the data

4.1. General data preprocess

For the ARDIS dataset first we use the threshold function to binaries the images to cut off noise, after that we reshape each image to a 28 × 28 array of pixels in grayscale. The preprocess ARDIS dataset is loaded in a folder named “preprocessed”.

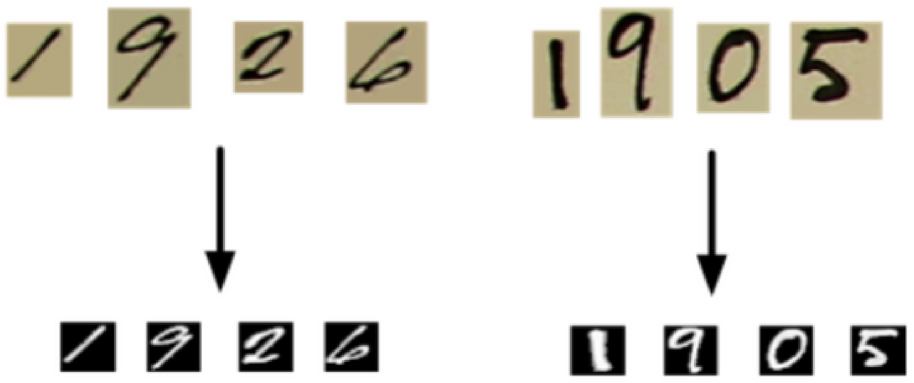


Figure 4 : ARDIS dataset Prepressing

We first used the ARDIS dataset, and we had a low accuracy because the dataset was tiny, only 7600 of samples for the training and test.
To augment the dataset we combine the ARDIS dataset with the MNIST dataset which contains 80000 samples.
The images of the MNIST dataset are already reshaped to a 28 × 28 array of pixels in grayscale.

The new dataset is loaded in a python programme named “load_ardis_mnist”, this data will be used in all the five classification algorithms, and in every one of them we will do an other preprocess for the data to adapt it with the features of the algorithm.

Models

5.1. Handwritten Digits Recognition using KNN

5.1.1. KNN model

KNN also called K- nearest neighbour is a supervised machine learning algorithm that can be used for classification and regression problems. K nearest neighbour is one of the simplest algorithms to learn. K nearest neighbour is non-parametric i,e. It does not make any assumptions for underlying data assumptions. K nearest neighbour is also termed as a lazy algorithm as it does not learn during the training phase rather it stores the data points but learns during the testing phase. It is a distance-based algorithm.

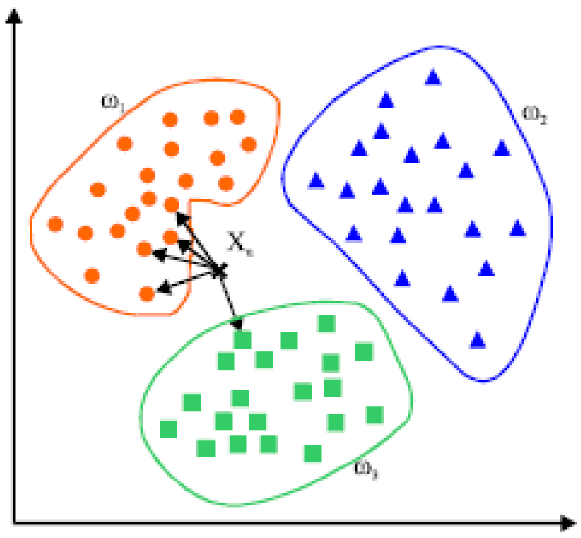


Figure 5 : K-NN Classification

5.1.2. Preprocess the data

To use our dataset of the handwritten digit in our KNN model, we need to make some preprocess on this data.
The images are represented as a matrix of 28x28, to use it as an input in the KNN, this images must be scaled to a size of 784 (28x28). After, we use StandardScaler which standardizes features by removing the mean and scaling it to unit variance. StandardScaler will transform the data such as its distribution will have a mean value of 0 and a standard deviation of 1.

5.1.3. Create the model

The scikit-learn library provides extensive tools for the k-Nearest Neighbors algorithm.

The module of the scikit-learn library used in creating our model is KNeighborsClassifier with the parameters k equal to five (5), and metric equal to euclidean.
After initializing the model with its parameters, we fit our training dataset in it.

As we know the KNN algorithm do not have classic learning phase, because the classifier is “lazy”, so we do not save any parameters or model for this algorithm, which mean that for predicting we need to calculate the distance between the new sample with all the samples of the training dataset, and that make the execution time for predicting in the KNN algorithm very slow.

5.1.4. Evaluate the model

We have 11000 images in our dataset which will be used to evaluate how good our model works. The testing data was not involved in the training of the data therefore, it is new data for our model.
The function was called for k = 5 neighbors, with uniform weights for each distance and obtained a decent accuracy of **92.65%**.

5.2. Handwritten Digits Recognition using Decision Tree

5.2.1. Decision Tree model

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too.
The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data).

In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

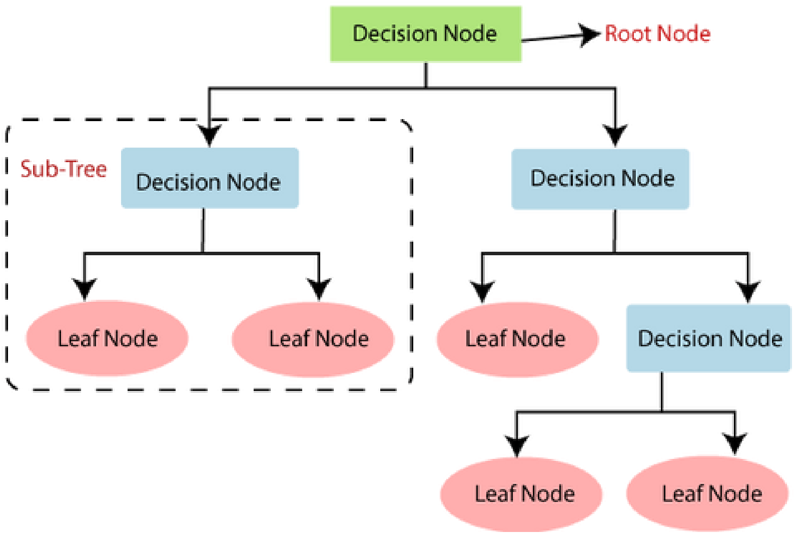


Figure 6 : Decision Tree Classification

Important Terminology related to Decision Tree

- **Root Node**: It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
- **Splitting**: a process of dividing a node into two or more sub-nodes.
- **Decision Node**: When a sub-node splits into further sub-nodes, then it is called the decision node.
- **Leaf/Terminal Node**: Nodes that do not split are called Leaf or Terminal nodes.
- **Pruning**: When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.
- **Branch / Sub-Tree**: A subsection of the entire tree is called a branch or sub-tree.
- **Parent and Child Node**: A node divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

Assumptions while creating a Decision Tree

Below are some of the assumptions we make while using the Decision tree:

- In the beginning, the whole training set is considered as the root.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized before building the model.
- Records are distributed recursively based on attribute values.
- Orderplacecing attributes as root or an internal tree node are done using some statistical approach.
- Decision Trees follow the Sum of Product (SOP) representation. The Sum of the product (SOP) is also known as a Disjunctive Normal Form. For a class, every branch from the root of the tree to a leaf node having the same class is a conjunction (product) of values, different branches ending in that class form a disjunction (sum).
- The primary challenge in the decision tree implementation is to identify which attributes we need to consider as the root node and each level. Handling this is known as the attributes selection. We have different attribute selection measures to identify the attribute which can be considered as the root note at each level.

How do Decision Trees work?

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria are different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases concerning the target variable. The decision tree splits the nodes on all available variables and then selects the split which results in are most homogeneous sub-nodes.

5.2.2. Preprocess the data

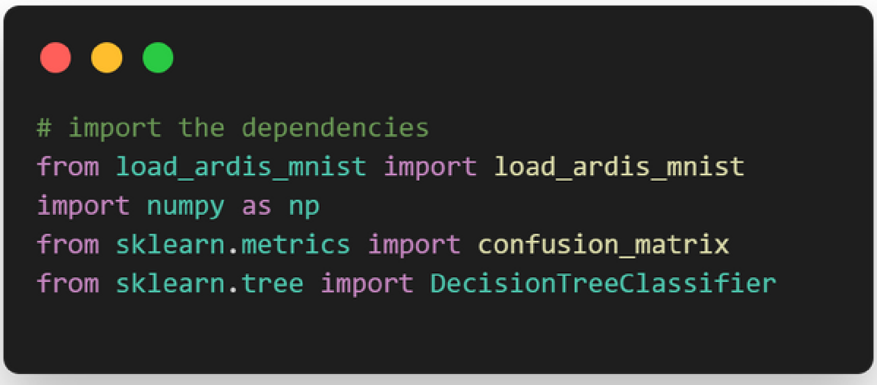
One of the benefits of decision trees is that ordinal (continuous or discrete) input data does not require any significant preprocessing. In fact, the results should be consistent regardless of any scaling or translational normalization, since the trees can choose equivalent splitting points. The best preprocessing for decision trees is typically whatever is easiest or whatever is best for visualization, as long as it doesn't change the relative order of values within each data dimension.

So, just to make the comparison with the rest of the algorithms fairer, it's best to prepare the data in the same way that is used before.

- normalize and make the black-and-white values between -0.5 and 0.5 to facilitate the model training.
- perform some operations and process the data to make it ready.
- normalize the value of pixels to be between -0.5 and 0.5 to facilitate the model training.

5.2.3. Create the model

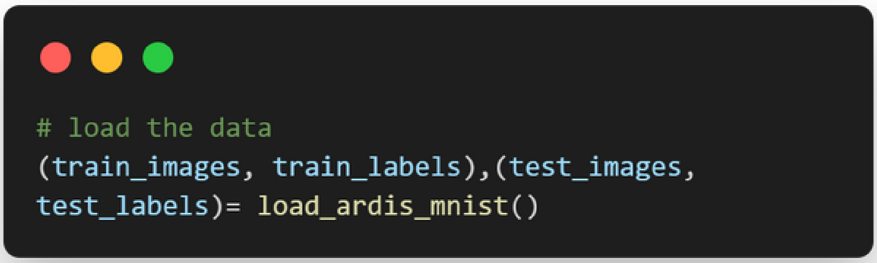
Import the dependencies



```python
# import the dependencies
from load_ardis_mnist import load_ardis_mnist
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
```

Figure 7 : Decision Tree Dependencies

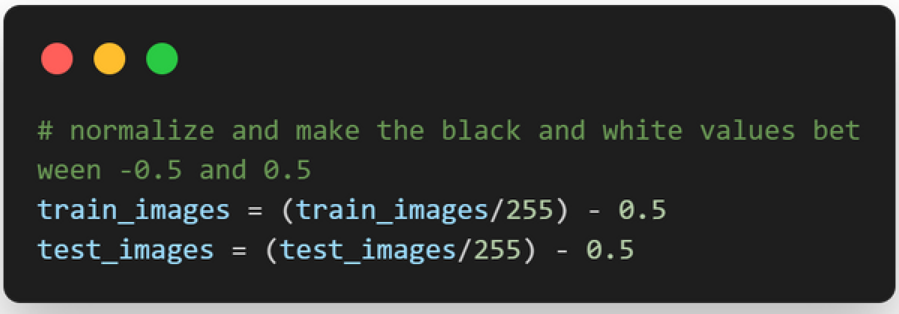
Load the data



```
# load the data
(train_images, train_labels),(test_images,
test_labels)= load_ardis_mnist()
```

Figure 8 : Decision Tree Data Loading

Normalization

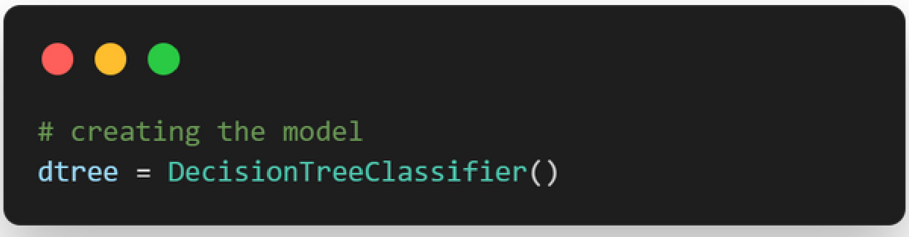


```
# normalize and make the black and white values bet
ween -0.5 and 0.5
train_images = (train_images/255) - 0.5
test_images = (test_images/255) - 0.5
```

Figure 9 : Decision Tree Data Normalization

Creating the model

A machine learning model determines the output you get after running a machine learning algorithm on the collected data.
In this case, our model is the "DecisionTreeClassifier"

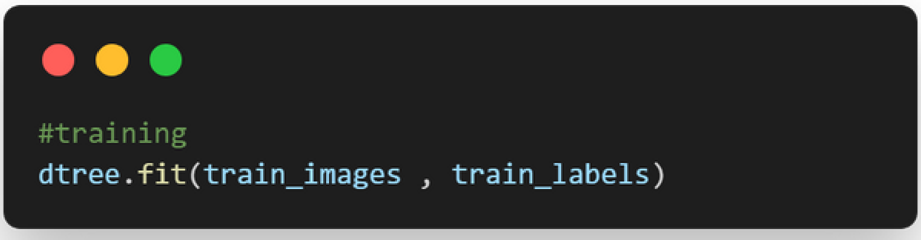


```
# creating the model
dtree = DecisionTreeClassifier()
```

Figure 10 : Decision Tree Model

Training the model

Training is the most important step in machine learning. In training, you pass the prepared data to your machine-learning model to find patterns and make predictions. It results in the model learning from the data so that it can accomplish the task set. Over time, with training, the model gets better at predicting.



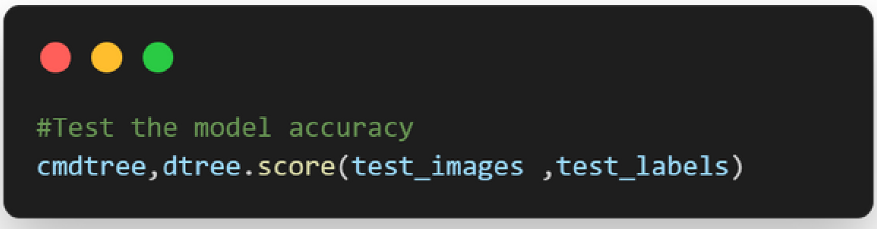
```
#training
dtree.fit(train_images , train_labels)
```

Figure 11 : Decision Tree Fitting Data

5.2.4. Evaluate the model

Accuracy

Model accuracy is defined as the number of classifications a model correctly predicts divided by the total number of predictions made. It's a way of assessing the performance of a model, but certainly not the only way.



```
#Test the model accuracy
cmdtree,dtree.score(test_images ,test_labels)
```

Figure 12 : Decision Tree Accuracy Score

Result: 84.1 %

Optimizing the Decision Tree Classifier

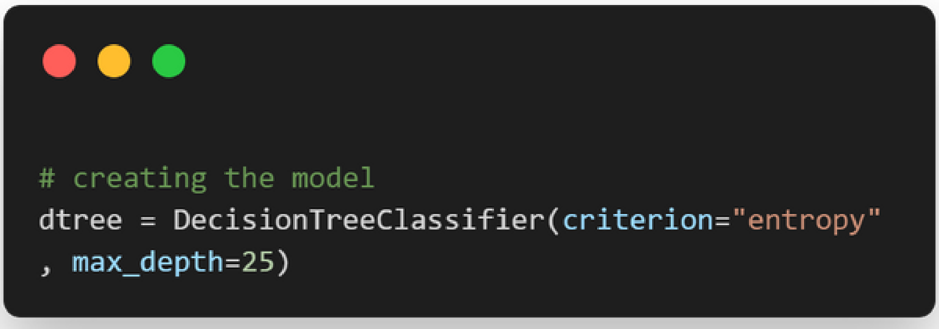
Criterion: optional (default="gini") or Choose attribute selection measure: This parameter allows us to use the different-different attribute selection measure. Supported criteria are "gini" for the Gini index and "entropy" for the information gain.

Splitter: string, optional (default="best") or Split Strategy: This parameter allows us to choose the split strategy. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

Max_depth: int or None, optional (default=None) or Maximum Depth of a Tree: The maximum depth of the tree. If None, then nodes are expanded until all the leaves contain less than min_samples_split samples. The higher value of maximum depth causes overfitting, and a lower value causes underfitting (Source).

In Scikit-learn, optimization of the decision tree classifier is performed by only pre-pruning. The maximum depth of the tree can be used as a control variable for pre-pruning.

Changing parameters



```
# creating the model
dtree = DecisionTreeClassifier(criterion="entropy"
, max_depth=25)
```

Figure 13 : Decision Tree Model with parameters

Result: 86 %

5.3. Handwritten Digits Recognition using SVM

5.3.1. SVM model

Support Vector Machine or SVM is a Supervised Learning algorithm, which is used for Classification and Regression problems. but it is mostly used for Classification problems in Machine Learning.
an SVM separates data across a decision boundary (plane) determined by only a small subset of the data (feature vectors). The data subset that supports the decision boundary is called the support vector. and This best decision boundary is called a hyperplane. Consider the below diagram in which two different categories are classified using a decision boundary or hyperplane:

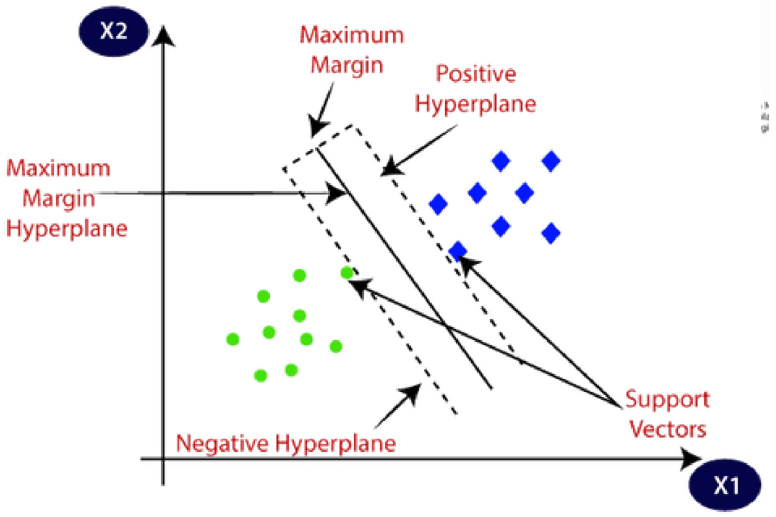


Figure 13 : SVM Classification

The distance between the decision-boundary and the closest data points is called the margin.
SVM algorithm can be used for image classification, Face detection, text categorization, etc.

SVM can be of two types:

- Linear SVM: if a dataset can be classified into two classes by using a single straight line, then such data is linearly separable data, and the classifier is used called a Linear SVM classifier. and the kernel is also used “linear”
- Non-linear SVM: if a dataset cannot be classified by using a straight line, then such data is non-linear data, and the classifier used is called a Non-linear SVM classifier. and the kernel is used “poly” or “rbf”.

5.3.2. Preprocess the data

To use the images in the SVM algorithm, first this images must be scaled to a size of 28x28 (784 cells in a one-dimensional representation). Then, each pixel value of the images need to be normalized by converting it into 'float32' and then dividing by 255.0 so that the input features will range between 0.0 to 1.0 and this help to make model training less sensitive to the scale of features.

For the output of our dataset, we encode them - digits from 0 to 9 - using one-hot encoding. The result is a vector with a length equal to the number of categories. The vector is all zeroes except in the position for the respective category.

5.3.3. Create the model

For implementing the Support Vector Machine algorithms, We used the TENSORFLOW(Keras) library.
To train a Keras model that approximates a Support Vector Machine (SVM), the key idea is to stack a RandomFourierFeatures layer with a linear layer.

The RandomFourierFeatures layer can be used to "kernelize" linear models by applying a non-linear transformation to the input features and then training a linear model on top of the transformed features. Depending on the loss function of the linear model, the composition of this layer and the linear model results to models that are equivalent (up to approximation) to kernel SVMs (for hinge loss), kernel logistic regression (for logistic loss), kernel linear regression (for MSE loss), etc.

Our SVM model contain only two layers, the first is the input layer with 784 nodes, and the second is the output layer with 10 nodes (0-9). For the RandomFourierFeatures, the kernal is gaussian.

5.3.4. Evaluate the model

To evaluate our SVM, The accuracy measurement parameter used is the accuracy which reached an accuracy of **96.61%** for 11000 training samples.

5.4. Handwritten Digits Recognition using ANN

5.4.1. ANN model

Artificial neural networks (ANNs), usually simply called neural networks (NNs) or neural nets, are computing systems inspired by the biological neural networks that constitute animal brains.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain.

The ANN model is made of layers of neurons that makes a network we define 3 types of layers:

- **Inupt layer:** the first layer of our model where we input the data.
- **Output layer**: is the layer of our results and it is the last layer.
- **Hidden layers**: are the layers between the input and the output layer their are the one responsible for determining the different features of our data.

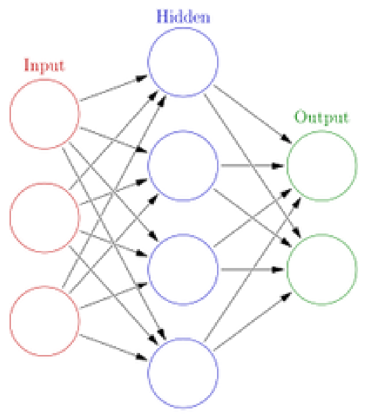


Figure 14 : ANN Classification

5.4.2. Preprocess the data

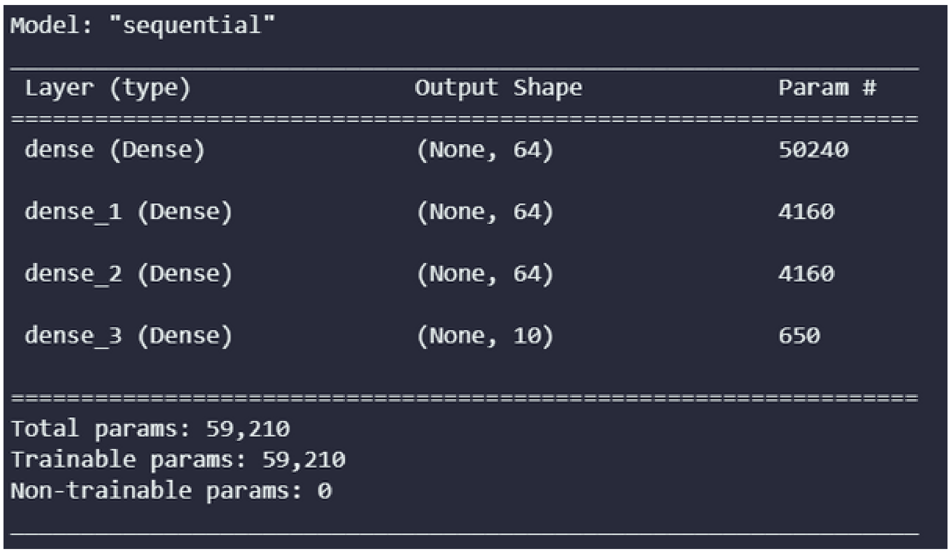
In our data set we have images with dimensions of 28 by 28 and are not normalized so we need to normalize the images by dividing their values by 255 and taking away 0.5 to make our images array values between -0.5 and 0.5 (positive and negative intervals helps with the model training), afterwards we need to flatten the images wich means make theses last into a one demensional array for each image.

So our dataset dimension after theses preprocessing is (66600,784) which we will input into our model.

5.4.3. Create the model

Our model consists of four Dense layers one input layer two hidden layers and one output layer, the first three layers uses the relu activation function, while the output layer uses the softmax activation functions for outputing the results.

Below you can find the summary of our model:



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                50240

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 64)                4160

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 59,210
Trainable params: 59,210
Non-trainable params: 0
_________________________________________________________________
```

Figure 15 : ANN Summary

5.4.4. Evaluate the model

For evaluating our model we use a test dataset that consists of 11000 elements that never were presented to our model in the training phase, after running the evaluation we had an accuracy of **95.51%**.

5.5. Handwritten Digits Recognition using CNN

5.5.1. CNN model

CNN is a type of deep learning model for processing data that can recognize and classify particular features from images and is widely used for analyzing visual images. The term Convolution is a mathematical construct that is typically composed of three types of layers (or building blocks): convolution, pooling, and fully connected layers. The first two, convolution and pooling layers, perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output, such as classification.

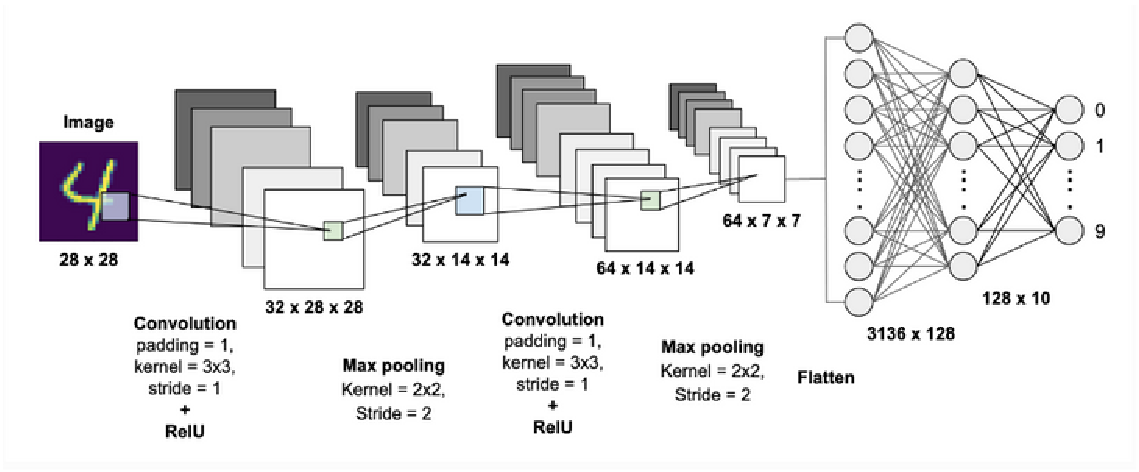


Image
28 x 28
32 x 28 x 28
32 x 14 x 14
64 x 14 x 14
64 x 7 x 7
3136 x 128
128 x 10
0
1
9
Convolution padding = 1, kernel = 3x3, stride = 1 + RelU
Max pooling Kernel = 2x2, Stride = 2
Convolution padding = 1, kernel = 3x3, stride = 1 + RelU
Max pooling Kernel = 2x2, Stride = 2
Flatten

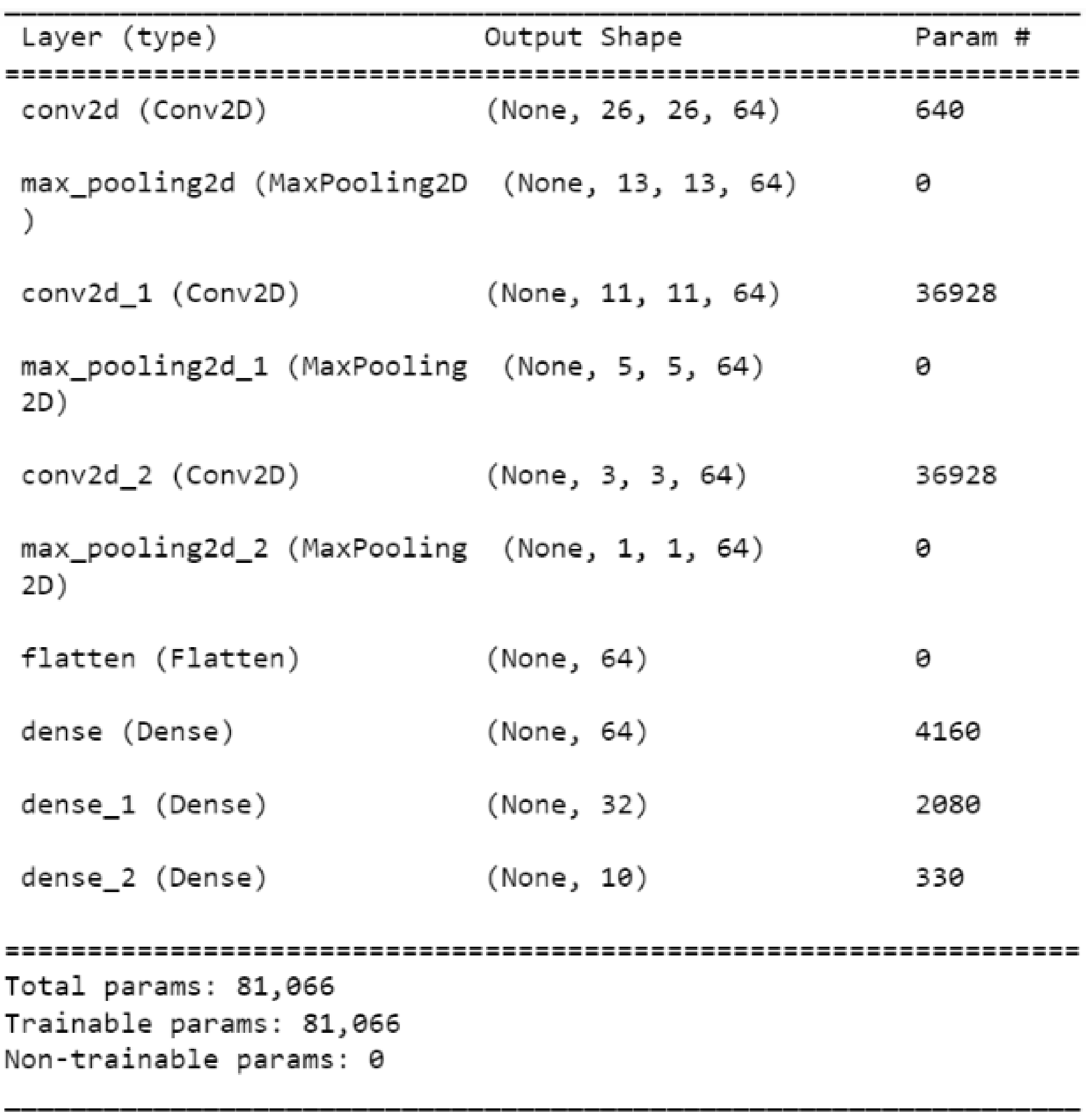
Figure 16 : CNN Classification

5.5.2. Preprocess the data

The image data cannot be fed directly into the model so we need to perform some operations and process the data to make it ready for our neural network.
First, we need to normalize by dividing the pixel values of the images by 255 and make the black and white values between -0.5 and 0.5 to facilitate the model training and help to speed up the training.
Also, The dimension of the training data is (66600,28,28) and the CNN model will require one more dimension so we reshape the matrix to shape (66600,28,28,1)

5.5.3. Create the model

In the creation of our CNN model we used the TENSORFLOW library.
For the first part of the model in which we extract features we have three convolution layers, and three pooling layers. For the second part, Classification part, we have three dense layers, the first two are hidden, and they have 64, 32 neurons successively and they use the "relu" function as an activation function, the last layer is the output of our model, it have 10 neurons each one for a specific number (from 0 to 9), in this we used the "softmax" function as an activation function.
this is a simple summary of the CNN model:



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 64)        640

 max_pooling2d (MaxPooling2D  (None, 13, 13, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

 max_pooling2d_2 (MaxPooling  (None, 1, 1, 64)         0
 2D)

 flatten (Flatten)           (None, 64)                0

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 10)                330

=================================================================
Total params: 81,066
Trainable params: 81,066
Non-trainable params: 0
_________________________________________________________________
```

Figure 17 : CNN Summary

5.5.4. Evaluate the model

We have 11000 images in our dataset which will be used to evaluate how good our model works. The testing data was not involved in the training of the data therefore, it is new data for our model.
With the CNN model we got **98.12%** accuracy.

Models Comparison

6.1. Comparing models

As we mentioned above, the objective of our work is to find the best model classifier of Handwritten Digit Recognition based on several measurement parameters that are the accuracy and time of prediction new samples.

The best algorithm with the highest accuracy and the fastest in the prediction is the Conventional Neural Network (CNN) with an accuracy of **98.12%.** The second algorithm is the Support Vector Machine (SVM) with an accuracy of **96.61%**, The third is the Artificial Neural Network (ANN) with an accuracy of **95.51%**, The forth is the K Nearest Neighbor (KNN) with an accuracy of **92.65%** and the worst algorithm is the Decision Tree with the lowest accuracy of **86%**.

For time of prediction, the KNN algorithm is the worst because it calculate the distance between the new sample and all the samples of the training dataset. And for the rest of the algorithms (ANN, SVM, and decision Tree) the time seems same.

To conclude, the Conventional Neural Network (CNN) is the best algorithm for Handwritten Digit Recognition.

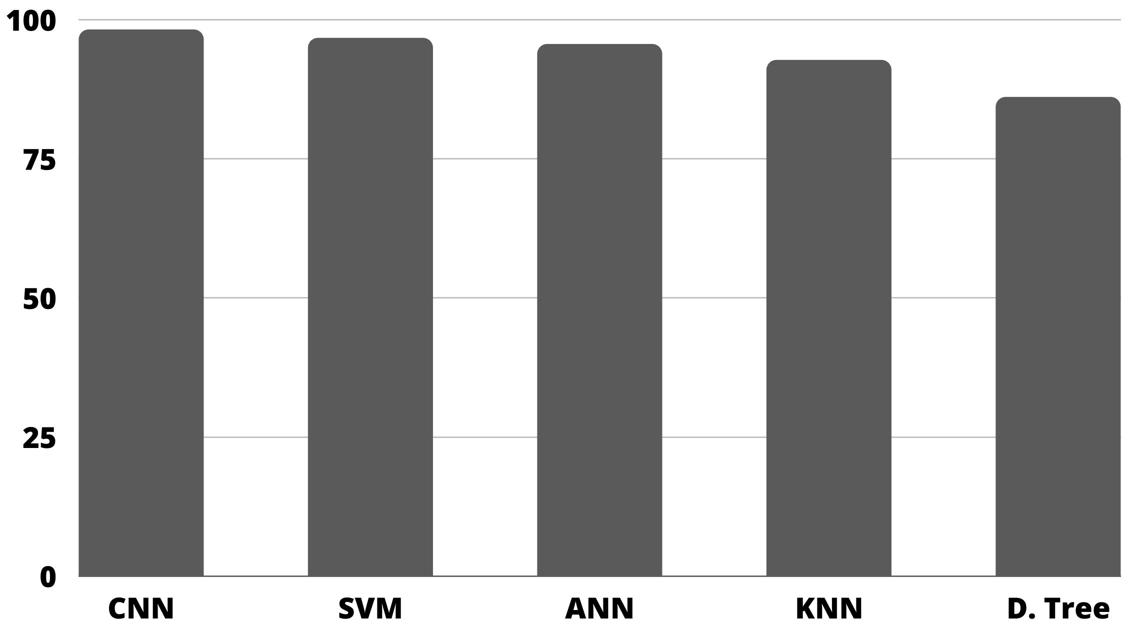


Figure 18 : Algorithms Comparison

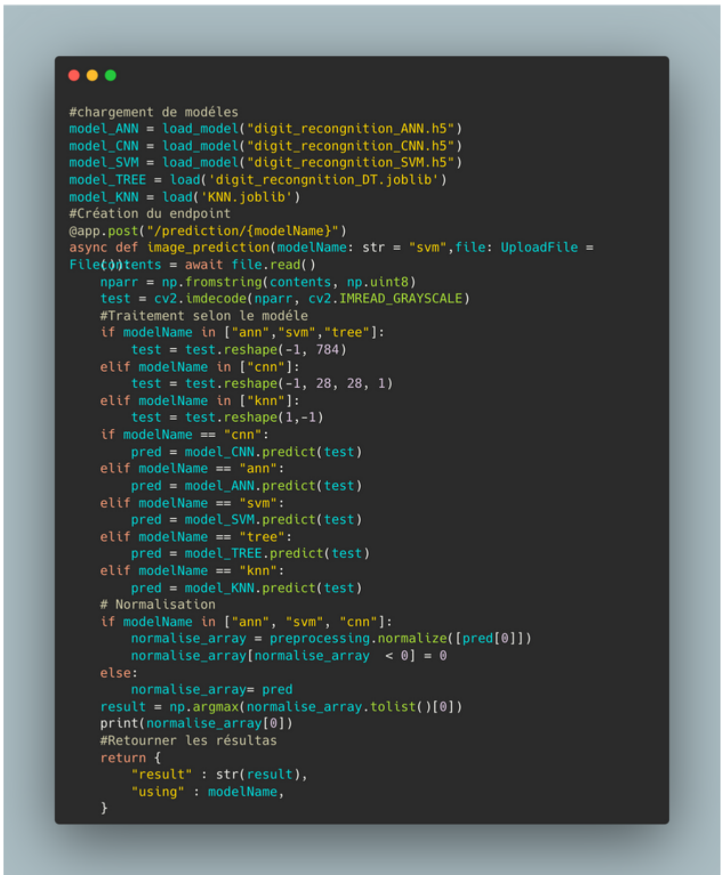
Mobile Application

7.1. Back-end

7.1.1. FastAPI

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints.

After creating the model we use them to create a rest API.
the image bellow show the code of the back-end of our mobile application.



```python
#chargement de modéles
model_ANN = load_model("digit_recongnition_ANN.h5")
model_CNN = load_model("digit_recongnition_CNN.h5")
model_SVM = load_model("digit_recongnition_SVM.h5")
model_TREE = load('digit_recongnition_DT.joblib')
model_KNN = load('KNN.joblib')
#Création du endpoint
@app.post("/prediction/{modelName}")
async def image_prediction(modelName: str = "svm",file: UploadFile =
File()):contents = await file.read()
    nparr = np.fromstring(contents, np.uint8)
    test = cv2.imdecode(nparr, cv2.IMREAD_GRAYSCALE)
    #Traitement selon le modéle
    if modelName in ["ann","svm","tree"]:
        test = test.reshape(-1, 784)
    elif modelName in ["cnn"]:
        test = test.reshape(-1, 28, 28, 1)
    elif modelName in ["knn"]:
        test = test.reshape(1,-1)
    if modelName == "cnn":
        pred = model_CNN.predict(test)
    elif modelName == "ann":
        pred = model_ANN.predict(test)
    elif modelName == "svm":
        pred = model_SVM.predict(test)
    elif modelName == "tree":
        pred = model_TREE.predict(test)
    elif modelName == "knn":
        pred = model_KNN.predict(test)
    # Normalisation
    if modelName in ["ann", "svm", "cnn"]:
        normalise_array = preprocessing.normalize([pred[0]])
        normalise_array[normalise_array  < 0] = 0
    else:
        normalise_array= pred
    result = np.argmax(normalise_array.tolist()[0])
    print(normalise_array[0])
    #Retourner les résultas
    return {
        "result" : str(result),
        "using" : modelName,
    }
```

Figure 19 : Back-end mobile application

7.1.2. Library used

- **FastAPI**: to create the rest API
- **UploadFile, File**: to process the images sent from the mobile application.
- **Cors**:to allow the mobile application access to the API.
- **Cv2**: image processing.
- **Numpy**: for the calculation.
- **Preprocessing**: for normalization.

7.2. Front-end

7.2.1. Flutter

Flutter is an open-source user interface software development kit (SDK) created by Google. It is used to develop applications for Android , iOS , Linux , Mac , Windows , Google Fuchsia and the web from a single code base.

7.2.2. Used packages

- **painter**: for drawing the hand written number
- **get**: for state management
- **http**: to make requests to the API

7.2.3. Application overview

The mobile application consists of 1 screen where we have an area to draw the digit we want to predict it's value, 3 buttons to undo last line drawn, a button to clear the canvas and a button to capture the data in the canvas and sends it to the server to predict the results, also we have a drop down menu to select the wanted algorithm.

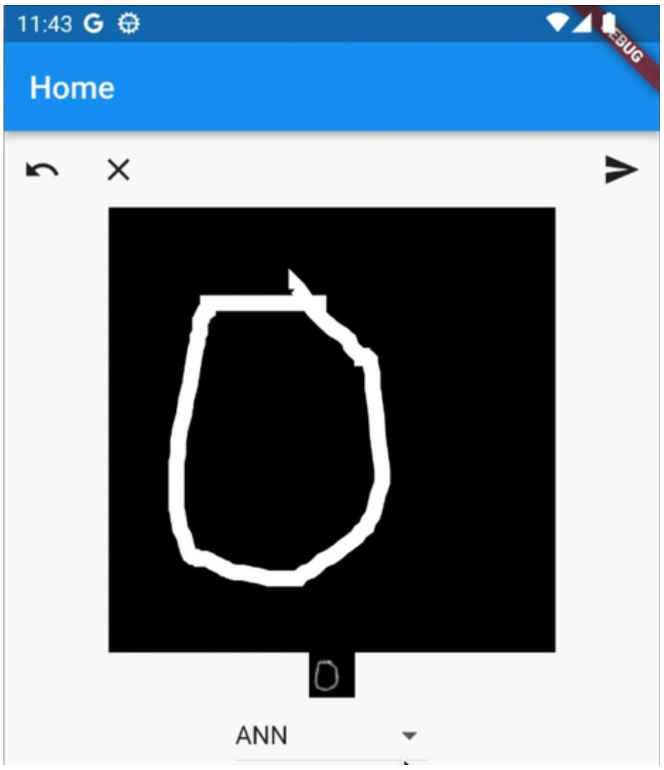
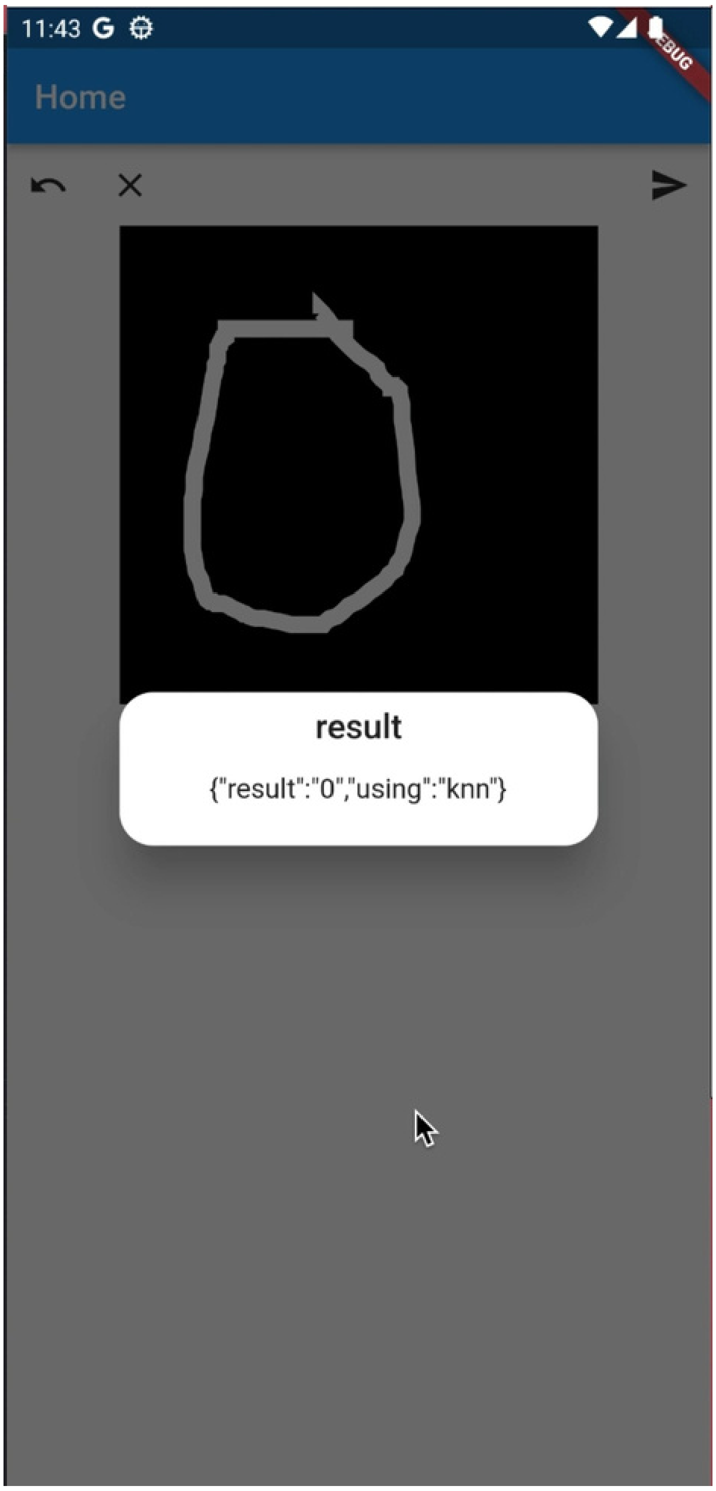


Figure 20 : Application Interface

After clicking the send button the canvas is transformed into an Uint8List and stored in memory then it gets sent to the API through the appropriate route with a multipart request (for image), then we wait for the response and we show the prediction results in a popup as shown below



```
{"result":"0","using":"knn"}
```

Figure 21 : ANN Execution Example

Conclusion

8. Conclusion

In this study, we developed some models namely Support Vector Machine Classifier, KNN Classifier, Decision Tree Classifier, Artificial Neural Network Classifier and Convolutional Neural Network for handwritten digit recognition using MNIST and ARDIS datasets. It compared them based on their working accuracy.

After a simple setup, all these algorithms demonstrated almost the same accuracy of handwritten digit recognition, differing within +1%.

It was found that CNN gave the most accurate results for handwritten digit recognition, but the only drawback is that it took an exponential amount of computing time.

After building recognition models using all the algorithms mentioned above, the recognition accuracy of all handwritten digits on the test program turned out to be in the range of 86-98%.