
HANDWRITTEN DIGITS RECOGNITION USING MACHINE LEARNING



Abstract

Our project aims to build an AI agent to recognize handwritten digits using machine learning algorithms. The agent will extract the handwritten text from an image and classify it into their respective digit labels using a machine-learning module trained upon the MNIST dataset by the SVM algorithm.

Table of Contents

1. Introduction.....	7
1.1 problem selection.....	8
1.2 Why is it an appropriate topic for AI development?.....	9
2. Knowledge representation.....	10
2.1 Dataset.....	11
2.2 Image process.....	12
2.2.1 Grey-scaling.....	12
2.2.2 Binarization.....	13
2.2.3 Inversion.....	14
2.2.4 Cropping and Reshaping.....	15
3. Methodology.....	15
3.1 Equipment and dataset.....	16
3.2 Classification.....	17
3.3 SVM algorithm.....	17
3.4 Coding.....	18
3.4.1 Data Preparation.....	19
3.4.2 create & train the module.....	19
3.4.3 create handwritten application.....	20
4. Testing and evaluation.....	21
4.1 Testing on the MNIST dataset.....	21
4.2 Measuring the accuracy.....	23
4.3 Testing the whole system.....	26
5. Appendices.....	28
5.1 SVC module code.....	28
5.2 handwritten application code.....	29
6. Conclusion.....	31
7. Bibliography.....	32

1.Introduction

Handwritten recognition is a field of research in artificial intelligence, computer vision, and pattern recognition. Where a computer or devices become able to acquire and detect text in paper documents, pictures, touch-screen devices, which are interpreted as text, is widely used. An example is smartphones or tablets with a touchscreen that can use handwriting (by finger or stylus) input as text or number input, and other sources and convert them into machine-encoded form. Its application found in optical character recognition and more advanced intelligent character recognition systems. Most of these systems nowadays implement machine-learning mechanisms such as neural networks. Machine learning is a branch of artificial intelligence that deals with learning from a set of data and can be applied to solve a wide spectrum of problems. A supervised machine learning module is given instances of data specific to a problem domain and an answer that solves the problem for each instance. When learning is complete, the module is able not only to provide answers to the data it has learned on but also to yet unseen data with high precision.

1.1 problem selection

In this project, we chose handwritten digits recognition as our problem topic to be solved by artificial intelligence, which is a subarea within optical character recognition OCR. There are many OCR applications that have been developed that solve the problems of text information extraction, automatic recognition of car license plates.

This project, 'Handwritten Digits Recognition' is a software algorithm project to recognize any hand written digits efficiently on computer, this type of problem is a classification task that includes recognizing a set of characters in an image, dividing them into 10 classes each class represents a digit from 0 to 9, the classification problems are concerned with learning modules which in our case used to distinguish pre-segmented handwritten digits to be able to predict the class of new handwritten digits.

In terms of performing this task we developed a software problem solving agent as shown in figure 1 that uses a machine learning module written by the programming language python, it was developed using the scikit-learn library that provides a built-in classification algorithm like SVC and the MNIST digit dataset which we will use to solve the problem of extracting the number from an image to be transformed into digital form and achieving

our project objective as what we will discuss in knowledge representation and methodology.

AI in OCR



Figure 1: classification agent

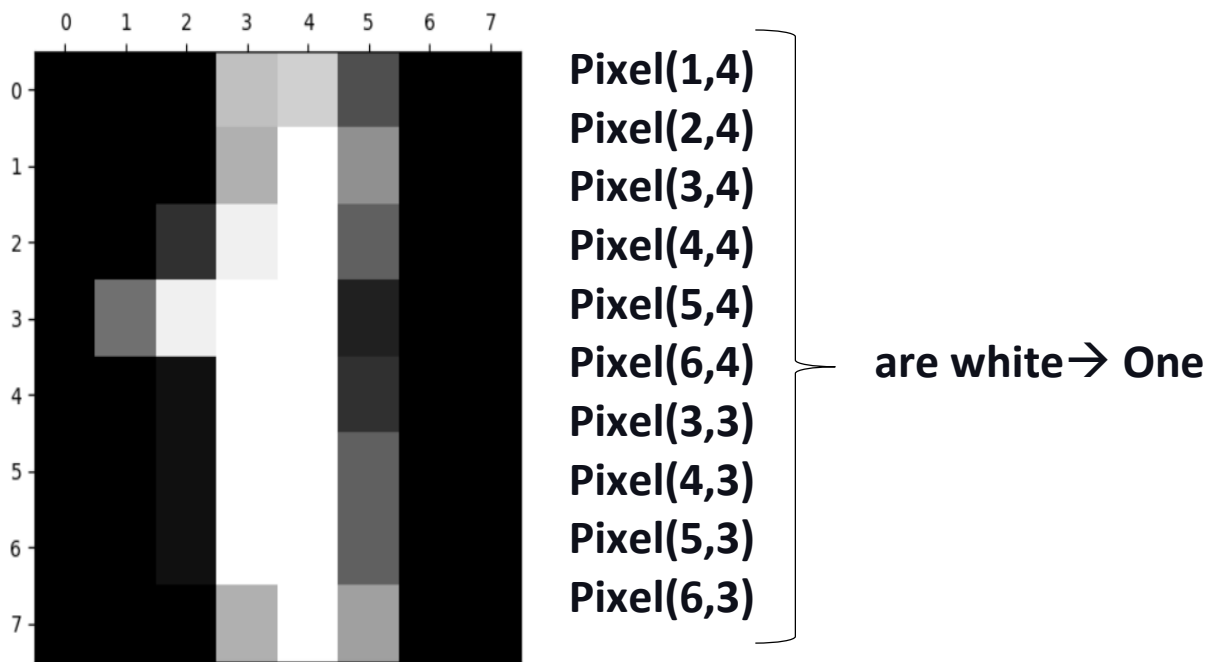
1.2 Why is it an appropriate topic for AI development?

Artificial intelligence gives an OCR boost and amplified OCR utility due to its higher accuracy and greater speed addition to the benefit of AI, human supervision isn't needed at every step owing to, AI high performing in the OCR it becomes a concerned field of research in pattern recognition, artificial intelligence, and machine vision, Most of the OCR systems nowadays implement AI Techniques such as machine learning mechanisms, in hence we elect 'Handwritten Digits Recognition' as an appropriate topic for AI development Specifically the machine learning mechanisms since we deal with a

classification task as what we mentioned in the problem selection part 1.1

2. Knowledge representation

In terms of making our AI agent able to recognize a handwritten digit we need to train the ML module in previous data that contain a pre-segmented digit image then, the agent will process the image extract the features and predict which class does the handwritten digit belongs relying on his prior knowledge. For instance to predict the one digit class then several pixels should be white and so on for the rest of the digits:



2.1 Dataset

We use the MNIST 8x8 dataset of handwritten digits, which has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The data set contain gray-scale images of hand-drawn digits, from zero through nine. Each image is eight pixels in height and eight pixels in width, for 64 pixels in total. Each pixel has a single pixel value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning lighter. This pixel-value is an integer between 0 and 255, Inclusive. The training data set has 784 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image. Each pixel column in the training set has a name like a pixel x, where x is an integer between 0 and 783, inclusive.

Figure 2 show the detail of the dataset where each data-point is an image 8x8 of a digit

Classes	10
Samples per class	~180
Samples total	1797
Dimensionality	64
Features	integers 0-16

Figure 2: MNIST 8x8-dataset information

Visually, if we omit the "pixel" prefix, the pixels make up the image like figure 3:

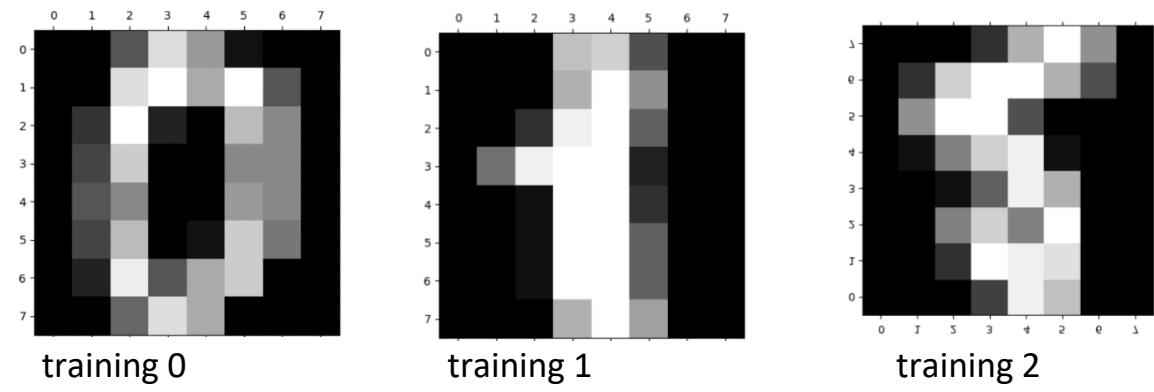


Figure 3: training image from the dataset

2.2 Image process

Image preprocessing is essential in the recognition pipeline for accurate digit prediction. Image preprocessing helps make feature extraction a smoother process, while feature extraction is necessary for correct classification. These methods typically involve noise removal, image segmentation, cropping, scaling, and more.

In our project we will use :

- Grey-scaling
- Binarization
- Inversion
- Cropping and Reshaping

2.2.1 Grey-scaling

Greyscaling is the most crucial process since we need to convert the image into an array it must convert first to a 2D form in order to do that we require to transform the image from an RGB image to a grey-scaling image its process by which an RGB image is converted into a black and white image after that become next stage Binarization where only shades of grey remain in the image, binarization of such image is efficient.

2.2.2 Binarization

Binarisation of an image transforms it into an image that only has clear black and clear white pixel values in it. During binarization of a grey-scale image, pixels with intensity lower than half of the full intensity value get a zero value converting them into black ones. Moreover, the remaining pixels get a full intensity value converting it into white pixels.

2.2.3 Inversion

Inversion is a process in which each pixel of the image gets a color, which is the inverted color of the previous one. This process is the most important one because any character on a sample image can only be extracted efficiently if it contains only one color which is distinct from the background color. Note that it is only required if the objects we have to identify are of darker intensity on a lighter background.

Figure 4 illustrates the physical meaning of the processes that are mentioned above:

RGB → Grey-scaling → Binarization → Inversion

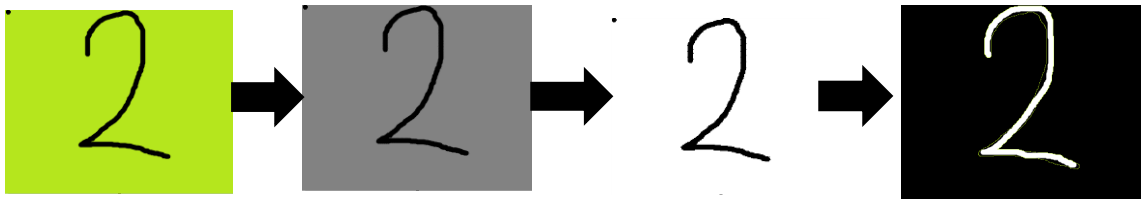


Figure 4: the physical of the image process

2.2.4 Cropping and Reshaping

then we will Cropping and Reshaping the image after that the digit image array is resized to form an 8x8 2D matrix pixelated image. This is done because an image array can only be defined with all images of fixed size figure 5 show the this process.

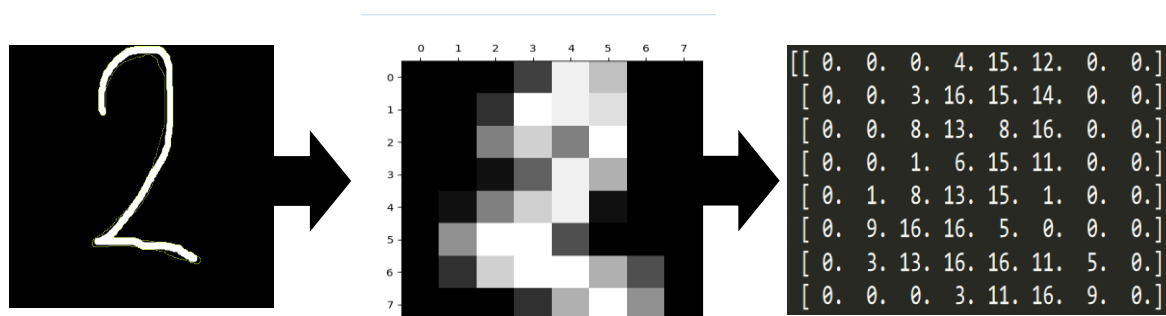


Figure 5: the image after Cropping , Reshaping & transform to 2D array

3. Methodology

3.1 Equipment and methodology

In this project, we will be using the programming language python and scikit-learn library in order to build and train our AI agent and we will use the classification method in terms to reach for the most efficient outcome, the classification will be illustrated further in the following part

3.2 Classification

Classification is designated as the task of assigning labels, classes to further unseen observations (instances of data). In machine learning, this is done on the basis of training an algorithm on a set of training examples.

Classification is a supervised learning problem, where a supervisor links a label to every instance of data. A label is a discrete number that identifies the class a particular instance belongs to. There are many machine learning modules that implement classification which is known as classifiers. The purpose of classifiers is to fit a decision boundary as given in figure 6 in the feature space that separates the training examples so that the class of a new observation instance can be correctly labeled in this project we select the SVM method as a classifier.

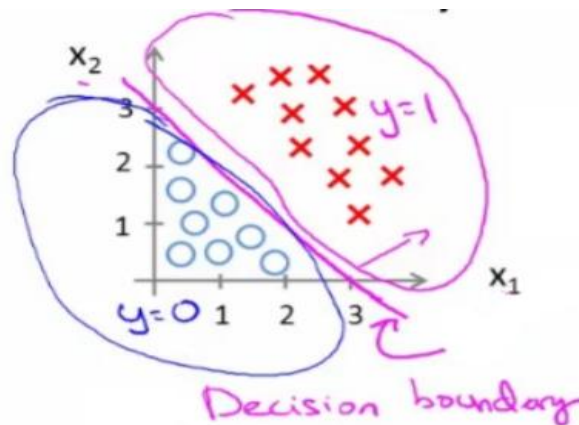


Figure 6: Visualization of a decision boundary. In feature-space given by x_1 and x_2 , a decision boundary is plotted between two linearly separable classes.

3.3 SVM algorithm

Support vector machines (SVM) are a set of related supervised learning methods used for classification and regression. They relate to a group of generalized linear classifiers. In other term, Support Vector Machine (SVM) is a classification and regression prediction tool that uses machine learning theory to maximize predictive accuracy while automatically avoiding over-fit to the data. Support Vector Machines can be described as systems that use hypothesis space of a linear function in a high dimensional feature space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory. The goal of the SVM is to find the hyper-plane 'margin' maximizes the minimum distance between any data point, as shown in figure 7.

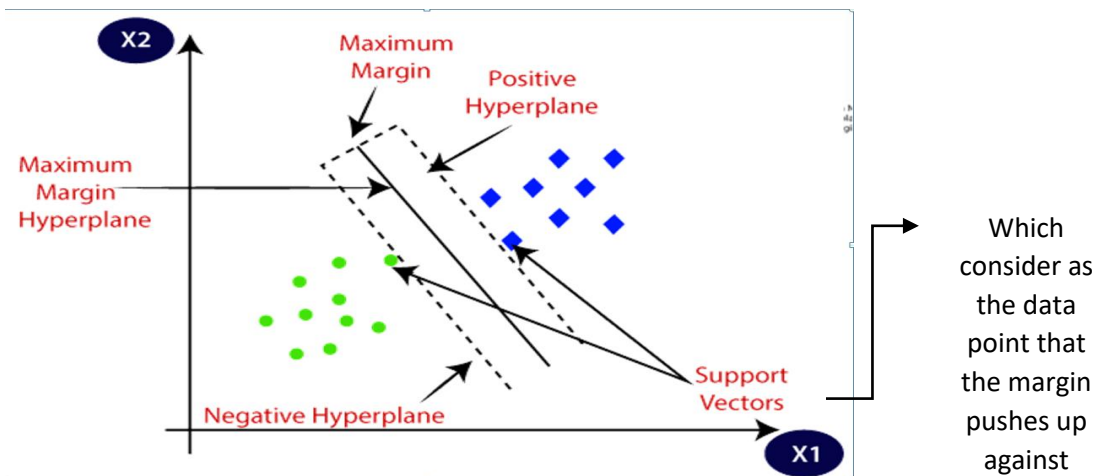


Figure 7: Illustration of SVM Linear the simplest kind of SVM

3.4 Coding

3.4.1 Data Preparation

for the sake of making the module get the most accurate result we must prepare the data which can be done as the following :

- Data cleaning

Where we remove all the NAN values and zeros and fill them with the mean of a specified column or row which can be done by a building module in the scikit-learn library called SimpleImputer:

```
SimpleImputer(missing_values = np.nan, strategy ='mean')
```

- Feature selection

We use the selectpercentile tool to select the most significant features that are connected to the result depending on the given percentage, therefore to determine the significant features the chi2 tool used. In the digit dataset, we have 64 features by using the selectpercentile and chi2 we make the module use only 10% of them, which mean the most seven important features:

```
SelectPercentile(score_func=chi2,percentile=10).fit_transform(X,y)
```

- Data Split

The digit dataset consists of data and the target of that data in data splitting we will split the data(mainData) before training into two parts one for the train (will use only [1500] column and their target), the rest of the data we will use it for the test (the test data will be discussed in the Testing and evaluation part 4) :

```
mainData=digits['data']  
targets=digits['target']  
svc.fit(mainData[:1500], targets[:1500])
```

3.4.2 Creat & train the module

After preparing the data we will create and fit the module to the data and as we mentioned previously we use python in order to code our module.

First we load the svm.SVC module wich will applay the svc algorithem and train the module :

```
svc=svm.SVC(gamma=0.001, C=100)
```

then we fit the data and their targets:

```
svc.fit(mainData[:1500], targets[:1500])
```

secondaly saving the trained module by joblib tool

```
joblib.dump(svc, "svmRmodule")
```


3.4.3 create handwritten application

We developed an application in python using pygame library and skimage library which facilitates users to write numbers on the screen to be predicted by the module. We make the application has a black background and the font has a white color (as shown in figure 8) in order to achieve the image processing steps that have been discussed in part 2.2 (the image processing), then we will take an image from the pygame window screen and matched the image with the module input scale by applying the flowing process to it:

- read and resize the image to 8x8

```
resize(imread((img)), (8,8))
```

- rescale the intensity in the range of 16

```
rescale_intensity(img, out_range=(0, 16))
```



Figure 8: pygame window

then we will pass the image as in input to the module and get the predicting result as in output :

The predicted digit is [2]

4. Testing and evaluation

In the matter of testing and evaluation our agent and measure the effectiveness of our module , we will divide the testing phase into three parts, first testing the module on the MNIST dataset, secondly we will measuring the accuracy of the system by different metrics module tools, eventually testing the whole system and the application.

4.1 Testing on the MNIST dataset

We have split the dataset in the Data Preparation part 3.4.1 at (data splinting point) In order to use the rest of the data to test the module and evaluate by matching the module prediction value to the target value.

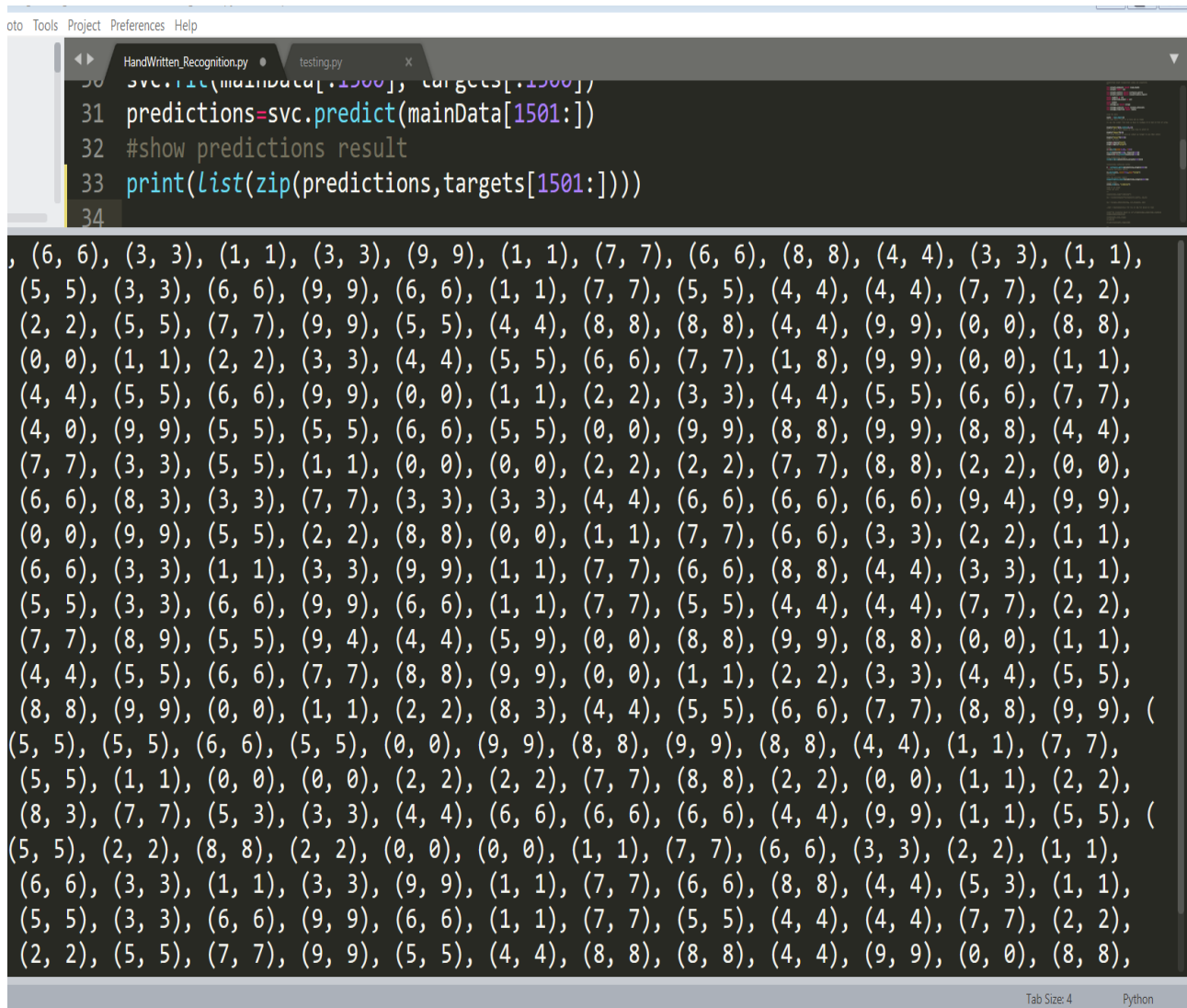
feed the rest of the mainData to the module (svc) o be predict it :

```
predictions=svc.predict(mainData[1501:])
```

print the result as a list consisting of (y,x) where y is the predicted value(the output of the svc module) and x is the target value (the correct value of the input) :

```
print(list(zip(predictions,targets[1501:])))
```

as what shown in figure 9 the y value match the x value in pretty much a lot of cases but still, there are some wrong predictions notably between 4 and 9 also 3 and 8



The screenshot shows a Python IDE with a file named `testing.py`. The code defines a function `svc_fit` and uses it to fit a model. It then predicts on a dataset `mainData` and prints the results. The output is a list of tuples, each containing a predicted digit and a target digit. The digits are from 0 to 9. The output shows a high degree of accuracy, with many correct predictions (e.g., (6, 6), (3, 3), (1, 1)) and some incorrect ones (e.g., (1, 8), (8, 3), (5, 3)).

```
HandWritten_Recognition.py  testing.py
30 svc_fit(mainData[:1500], targets[:1500])
31 predictions=svc.predict(mainData[1501:])
32 #show predictions result
33 print(list(zip(predictions,targets[1501:])))
34
```

(6, 6), (3, 3), (1, 1), (3, 3), (9, 9), (1, 1), (7, 7), (6, 6), (8, 8), (4, 4), (3, 3), (1, 1),
(5, 5), (3, 3), (6, 6), (9, 9), (6, 6), (1, 1), (7, 7), (5, 5), (4, 4), (4, 4), (7, 7), (2, 2),
(2, 2), (5, 5), (7, 7), (9, 9), (5, 5), (4, 4), (8, 8), (8, 8), (4, 4), (9, 9), (0, 0), (8, 8),
(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (1, 8), (9, 9), (0, 0), (1, 1),
(4, 4), (5, 5), (6, 6), (9, 9), (0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7),
(4, 0), (9, 9), (5, 5), (5, 5), (6, 6), (5, 5), (0, 0), (9, 9), (8, 8), (9, 9), (8, 8), (4, 4),
(7, 7), (3, 3), (5, 5), (1, 1), (0, 0), (0, 0), (2, 2), (2, 2), (7, 7), (8, 8), (2, 2), (0, 0),
(6, 6), (8, 3), (3, 3), (7, 7), (3, 3), (3, 3), (4, 4), (6, 6), (6, 6), (6, 6), (9, 4), (9, 9),
(0, 0), (9, 9), (5, 5), (2, 2), (8, 8), (0, 0), (1, 1), (7, 7), (6, 6), (3, 3), (2, 2), (1, 1),
(6, 6), (3, 3), (1, 1), (3, 3), (9, 9), (1, 1), (7, 7), (6, 6), (8, 8), (4, 4), (3, 3), (1, 1),
(5, 5), (3, 3), (6, 6), (9, 9), (6, 6), (1, 1), (7, 7), (5, 5), (4, 4), (4, 4), (7, 7), (2, 2),
(7, 7), (8, 9), (5, 5), (9, 4), (4, 4), (5, 9), (0, 0), (8, 8), (9, 9), (8, 8), (0, 0), (1, 1),
(4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9, 9), (0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5),
(8, 8), (9, 9), (0, 0), (1, 1), (2, 2), (8, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9, 9), (
(5, 5), (5, 5), (6, 6), (5, 5), (0, 0), (9, 9), (8, 8), (9, 9), (8, 8), (4, 4), (1, 1), (7, 7),
(5, 5), (1, 1), (0, 0), (0, 0), (2, 2), (2, 2), (7, 7), (8, 8), (2, 2), (0, 0), (1, 1), (2, 2),
(8, 3), (7, 7), (5, 3), (3, 3), (4, 4), (6, 6), (6, 6), (6, 6), (4, 4), (9, 9), (1, 1), (5, 5), (
(5, 5), (2, 2), (8, 8), (2, 2), (0, 0), (0, 0), (1, 1), (7, 7), (6, 6), (3, 3), (2, 2), (1, 1),
(6, 6), (3, 3), (1, 1), (3, 3), (9, 9), (1, 1), (7, 7), (6, 6), (8, 8), (4, 4), (5, 3), (1, 1),
(5, 5), (3, 3), (6, 6), (9, 9), (6, 6), (1, 1), (7, 7), (5, 5), (4, 4), (4, 4), (7, 7), (2, 2),
(2, 2), (5, 5), (7, 7), (9, 9), (5, 5), (4, 4), (8, 8), (8, 8), (4, 4), (9, 9), (0, 0), (8, 8),

Figure 9: the rusel of the predictions

4.2 Measuring the accuracy

After getting the prediction value we will measure the accuracy by two metrics module tools:

- Confusion Matrix

Confusion Matrix is a performance measurement for machine learning classification problem where a matrix consisting of an organized number of correct predictions and incorrect predictions as follows:

- Expected down the side: Each row of the matrix corresponds to a predicted class.
- Predicted across the top: Each column of the matrix corresponds to an actual class.

Then the counts of correct and incorrect classification are filled into the table as shown in figure 8 where the classes from 0 to 9 are labeled top and down.

#Calculating Confusion Matrix

CM = confusion_matrix(predictions,targets[1501:])

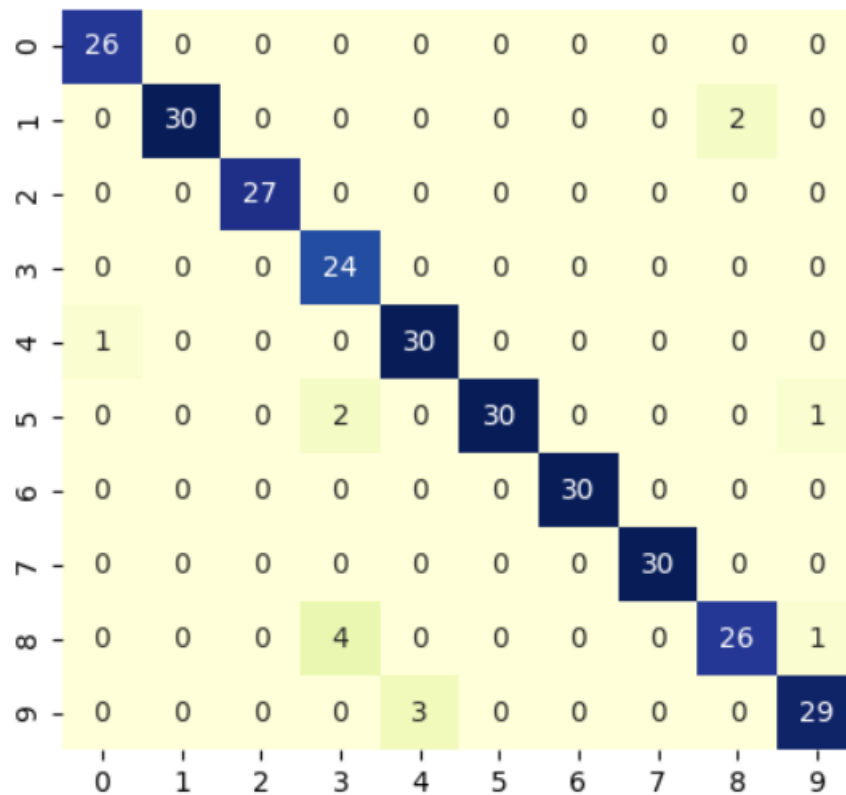


Figure 10: the Confusion Matrix

Figure 10 illustrates that most of the system predictions were correct for example all the one's values have been classified correctly whereas four of the 3 values have been classified as 8 and three of the four values are classified as 9 and so on

- accuracy report

The accuracy report is one of the most efficient metrics tool which provides a summary by calculating the precision, recall, f1score , support for every value, and the accuracy as shown in figure 11.

#show the accuracy report

```
print(classification_report(predictions,targets[1501:]))
```

	precision	recall	f1-score	support
0	0.96	1.00	0.98	26
1	1.00	0.94	0.97	32
2	1.00	1.00	1.00	27
3	0.80	1.00	0.89	24
4	0.91	0.97	0.94	31
5	1.00	0.91	0.95	33
6	1.00	1.00	1.00	30
7	1.00	1.00	1.00	30
8	0.93	0.84	0.88	31
9	0.94	0.91	0.92	32
accuracy			0.95	296
macro avg	0.95	0.96	0.95	296
weighted avg	0.96	0.95	0.95	296

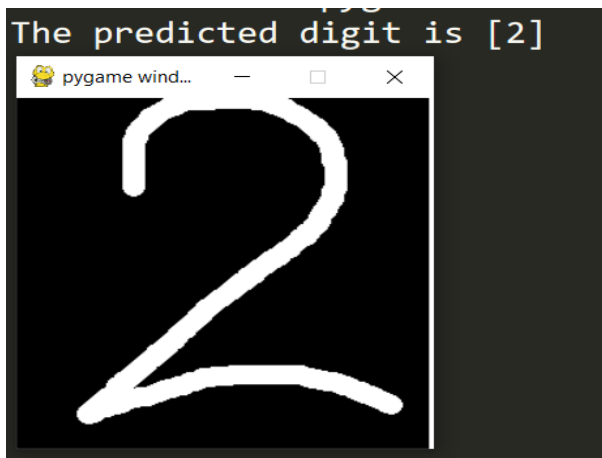
Figure 11: accuracy report

We can now perceive that the accuracy of our system is 95%

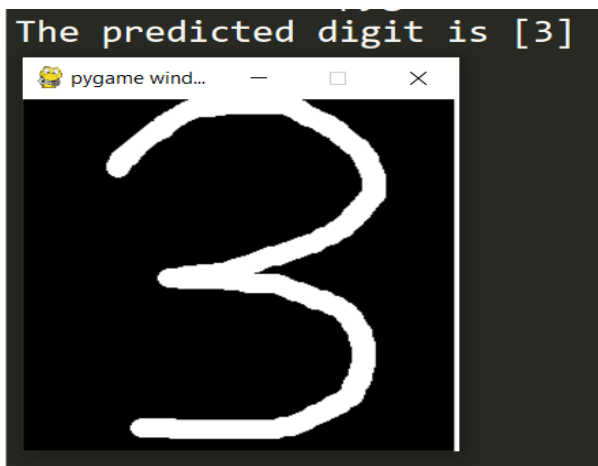
4.3 Testing the whole system

In this phase, we will pass the input to the module through the pygame application and get the prediction result.

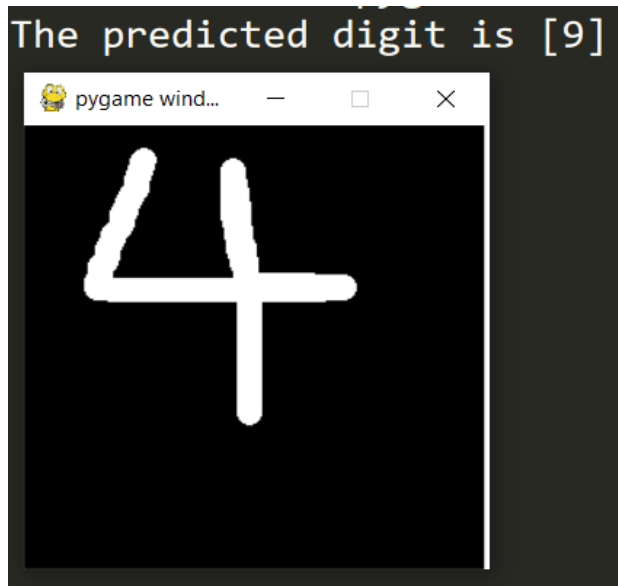
- test with number 2



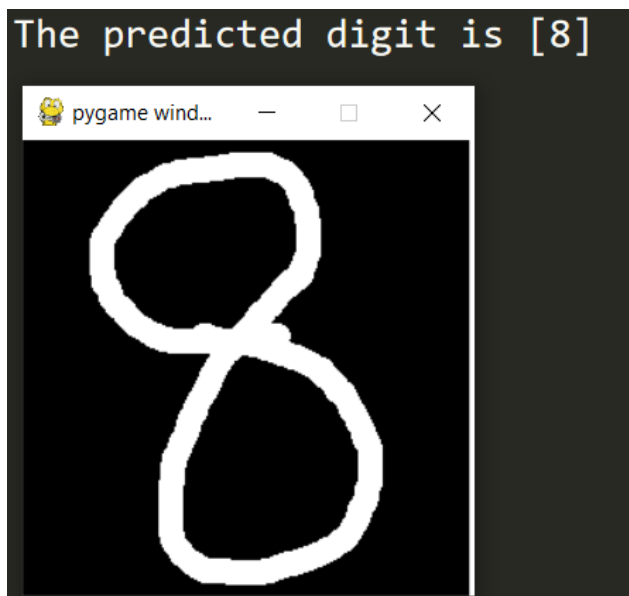
- Test with number 3



- Test with number 4



- Test with number 8



5. Appendices

5.1 SVC module code

```
1  '''
2  handwritten digit recognition using svm algorithm
3  '''
4  from sklearn.datasets import load_digits
5  from sklearn import svm
6  from sklearn.metrics import confusion_matrix
7  from sklearn.metrics import classification_report
8  import seaborn as sns
9  import matplotlib.pyplot as plt
10 import joblib
11 from skimage.io import imread
12 from skimage.exposure import rescale_intensity
13 from skimage.transform import resize
14 #load the data
15 digits = load_digits()
16 mainData=digits['data']
17 targets=digits['target']
18 #train
19 svc=svm.SVC(gamma=0.001, C=100)
20 svc.fit(mainData[:1500], targets[:1500])
21 predictions=svc.predict(mainData[1501:])
22 #show predictions result
23 (list(zip(predictions,targets[1501:])))
24 #Calculating Confusion Matrix
25 CM = confusion_matrix(predictions,targets[1501:])
26 # drawing confusion matrix
27 sns.heatmap(CM, annot=True,cmap="YlGnBu")
28 plt.show()
29 #show the accuracy report
30 (classification_report(predictions,targets[1501:]))
31 #saving model
32 joblib.dump(svc, "svmRmodel")
```

5.2 handwritten application code

```
1 import pygame
2 import joblib
3 from skimage.io import imread
4 from skimage.exposure import rescale_intensity
5 from skimage.transform import resize
6 # loading pre trained model
7 model=joblib.load('svmRmodel')
8
9 #predict function
10 def predict_digit(img):
11     img = resize(imread((img)), (8,8))
12     img = rescale_intensity(img, out_range=(0, 16))
13     x_test = [sum(pixel)/3.0 for row in img for pixel in row]
14     return (model.predict([x_test]))
15
16
17 # pre defined colors, pen radius and font color
18 black = [255, 255, 255]
19 white = [0, 0, 0]
20 red = [255, 0, 0]
21 green = [0, 255, 0]
22 draw_on = False
23 last_pos = (0, 0)
24 color = (255, 128, 0)
25 radius = 7
26 font_size = 500
27
28 #image size
29 width = 256
30 height = 256
31
32 # initializing screen
33 screen = pygame.display.set_mode((width, height))
34 screen.fill(white)
35 pygame.font.init()
36
37
38
39 def roundline(srf, color, start, end, radius=1):
40     dx = end[0] - start[0]
41     dy = end[1] - start[1]
42     distance = max(abs(dx), abs(dy))
43     for i in range(distance):
44         x = int(start[0] + float(i) / distance * dx)
45         y = int(start[1] + float(i) / distance * dy)
46         pygame.draw.circle(srf, color, (x, y), radius)
```

```

49 def draw_partition_line():
50     pygame.draw.line(screen, black, [width, 0], [width,height ], 8)
51
52 try:
53     while True:
54         # get all events
55         e = pygame.event.wait()
56         draw_partition_line()
57
58         # clear screen after right click
59         if(e.type == pygame.MOUSEBUTTONDOWN and e.button == 3):
60             screen.fill(white)
61
62         # quit
63         if e.type == pygame.QUIT:
64             raise StopIteration
65
66         # start drawing after left click
67         if(e.type == pygame.MOUSEBUTTONDOWN and e.button != 3):
68             color = black
69             pygame.draw.circle(screen, color, e.pos, radius)
70             draw_on = True
71
72         # stop drawing after releasing left click
73         if e.type == pygame.MOUSEBUTTONUP and e.button != 3:
74             draw_on = False
75             fname = "out.png"
76             img = screen
77             pygame.image.save(img, fname)
78             print("The predicted digit is {}".format(predict_digit("out.png")))
79
80
81
82         # start drawing line on screen if draw is true
83         if e.type == pygame.MOUSEMOTION:
84             if draw_on:
85                 pygame.draw.circle(screen, color, e.pos, radius)
86                 roundline(screen, color, e.pos, last_pos, radius)
87                 last_pos = e.pos
88
89         pygame.display.flip()
90
91 except StopIteration:
92     pygame.quit()

```

6. Conclusion

building an AI agent to Classification a handwritten digit was done in this project. The result was correct up to more than 95% of the cases. This work was focused on SVC methods that can efficiently classify the digit correctly and give high performance.

7. Bibliography

- scikit-learn user guide Release 0.18.2
- [Srivastava, Durgesh, Bhambhu, Lekha. \(2010\). Data classification using support vector machine. Journal of Theoretical and Applied Information Technology.](#)
- [Handwritten Digits Recognition Using SVM, KNN, RF and Deep Learning Neural](#)
- [SVM algorithm](#)
- [SVM Methods in Image Segmentation](#)
- [Real time digit recognition](#)