CLASSIFICATION REPORT

Sharjeel Akram.... BSCS-2019-60
Azeem Akhtar.... BSCS-2019-20

December 12, 2021

ARTIFICIAL INTELLIGENCE
CS-340
Dr. Junaid Akhtar

Acknowledgement

I would like to express the deepest gratitude to Professor Dr. Junaid Akhtar, my teacher, for his help, guidance and encouragement in my work and far beyond. Without his numerous suggestions and immense knowledge, this would never have been completed. Without his open-mindedness, my hard-work would not have been so rewarding.
I appreciate the many inspiring discussions with Ali Raza Khan, Student of 4th year Computer Science Department, T.A to Dr Juniad, who gave me the eye-opening opportunity to try out this evolutionary algorithm.
It gives me great pleasure to have the opportunity to thank my class fellows who helped me in developing the algorithm at Namal University Mianwali.

Table of Contents

List of Figures

Abstract

Handwritten Roman numeral's classification and recognition is an essential advance move for optical character recognition and others. Although there are several handwritten numeral's classification algorithms have been developed, but we have also implemented an algorithm to train the numerals for identification. A lot of searches have been done in this active area of characters recognition in several fields. In the field of pattern classification and identification, the handwritten recognition has been an open problem. By experimenting and several studied shows that Neural Network has a tremendous overall performance in classification. This report is to provide efficient and dependable techniques for training of handwritten numerals by comparing numerous current category models. The technique of handwritten classification is widely used in numerous fields like reading postal addresses, bank check, amounts and forms etc.

1 Problem Statement:

In the course of Artificial Intelligence, we were given some training data and some data for validation. It means we have to train a machine that can learn data then validate another data. And the challenging part was this that we had to use only MLPClassifier of Sci-kit library of python.

2 Introduction:

The automatic recognition of roman or any digits on images has wide commerce importance. It has applications in automatic pin code recognition, reading cheque and data collecting from filled in forms. But there are concerns about the digits recognition that human handwritten numerals are various in size, translation, thickness, rotation and deformation of the image on which it is contained. The main reason is that the handwritten digits or numerals are written by different users and their writing style is different from one another.
Let's discuss the Artificial Neural Network in details that how the concept of ANN can help us to solve this problem.

3 Artificial Neural Network:

What is an ANN? The simply ANN is a machine learning method evolved from the idea and intelligence of a human brain or simply we can say that it is inspired a human brain. It means we want to train a machine that can work like a human brain. The most important functionalities of an ANN are that it can recognize correlated patterns between datasets (inputs and output). ANN has huge value in prediction, pattern recognition, classifications, decision making and data comprehension. It consists of artificial neurons. Here are the details of biological neurons
These artificial neurons are copy of human brain neurons. They are trained artificially such that they have tendency to work like a human brain. As we know that all the movements and actions are controlled and managed by our brain. In brain, there are several components of brain called as neurons. Whenever, our hands touch the hot things, the signals are generated and passed to our brain. Then, it takes reaction whether we should keep on our hands or raise our hands. The final decision is taken by the brain. Similarly,

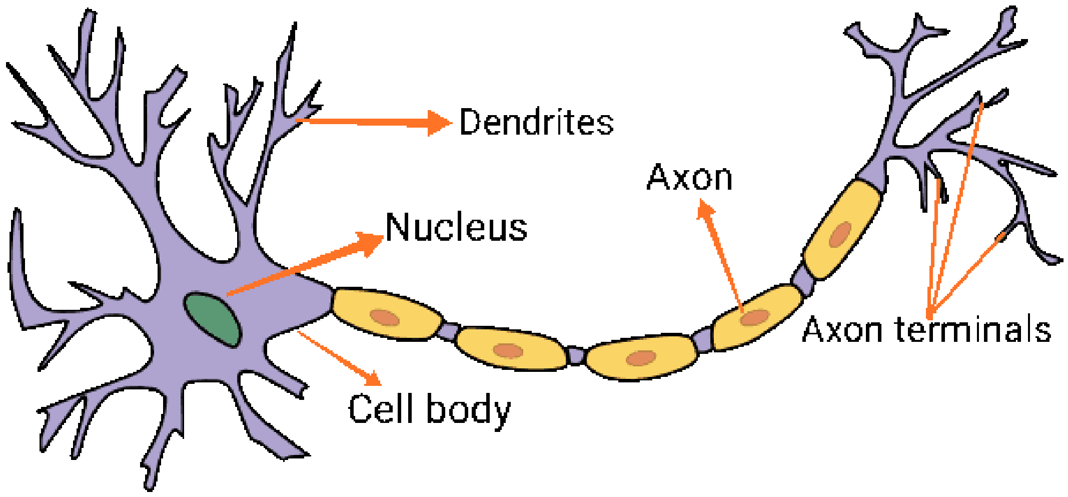


Figure 1: Biological Neurons

we need to build such neurons and train them best. An artificial neural network has several artificial neurons who can work like the biological neurons. It is a connection point in an ANN.

4 Perceptron:

The perceptron was first introduced by American psychologist, Frank Rosenblatt in 1957 at Cornell Aeronautical Laboratory. A perceptron is a type of neuron and it is a binary classifier. In the field of machine learning, it also works as binary classifier. Actually, binary classifier is a function that can decide (acts as a decider).

4.1 Types:

There are two types of perceptron. Single Layer Perceptron and Multi-Layer Perceptron. The primary distinction between these two perceptrons is that solitary layer perceptron can just adapt directly distinguishable patterns while multi-facet perceptrons with at least two layers have the more prominent handling power. Single layer perceptron can just arrangement with the straight issues. Multi-facet perceptron has middle layer known as "Hidden Layer".

In the above figure part(a), this shows a Single Layer Perceptron while

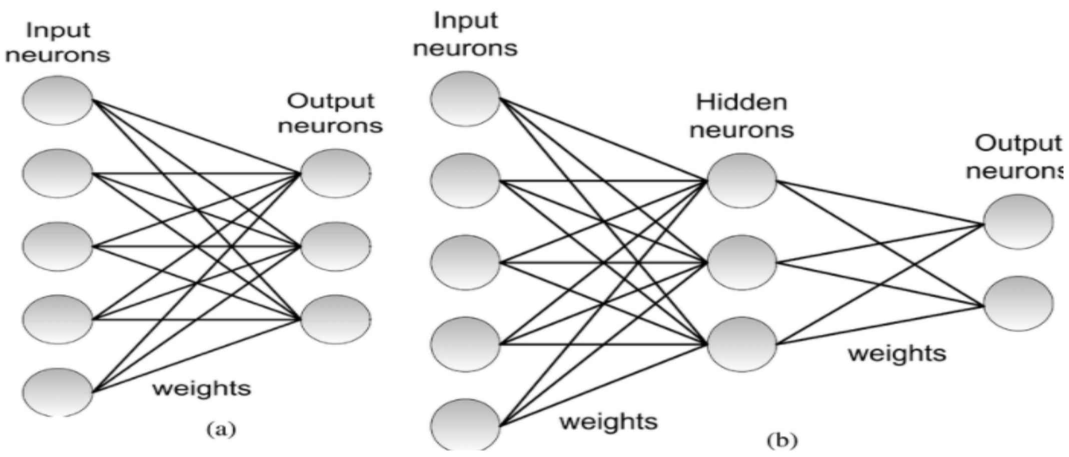


Figure 2: Single vs Multi-Layer Perceptron

on the other hand there is Multi-layer Perceptron containing Hidden Layer and Neurons. We will not go in deep here, first need to understand what is Perceptron and will go in detail later.
A perceptron is a function that takes several inputs and produces a single output. It is a building block of ANN. First of all, let's take a simple example.

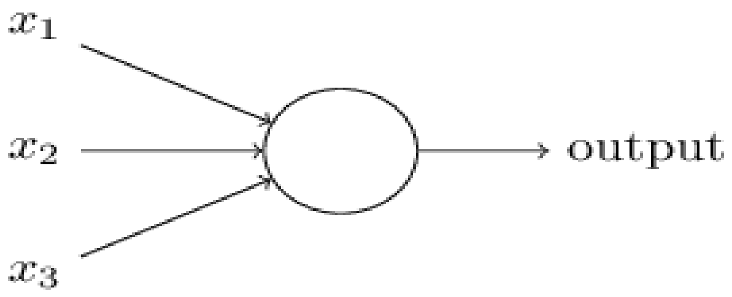


Figure 3: A Building block of ANN

In the above example, the perceptron ha three inputs such as x1, x2, x3. In general, this perceptron might have more or less than 3 inputs as an example we just take 3 inputs to understand. There is a simple rule for the calculation of the output. Weights were introduced w1, w2,... (real numbers), they express the value of the respective inputs to output. The neuron's value 0,1 are dependent on the threshold value. Threshold is also a real number which is a parameter to the neuron. The weighted sum j wjxj might be less or greater than the value of threshold. That's why output is determined by comparing with threshold.

In Algebric terms $$output = \begin{cases} 0 \text{ if } \sum_j w_j x_j \leq \text{threshold} \\ 1 \text{ if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

Figure 4: Equation

5 Main Components of Perceptron:

There are three main components of a perceptron. It is binary classifier;

1. Input Layer: (In the input layer, the info is taken into the framework for additional handling and estimations. This info may be shifted with various numbers. Be that as it may, it should be a genuine number. The job of Bias is as similar as intercept in a linear equation.)
2. Weights and Bias: (Weight boundaries address the strength of the association between units)
3. Activation Function: (the weighted amount of info is changed into a result from nodes in a layer of the network)

Three important steps take place in a single iteration of a Neural Network as forward, back propagation and optimization

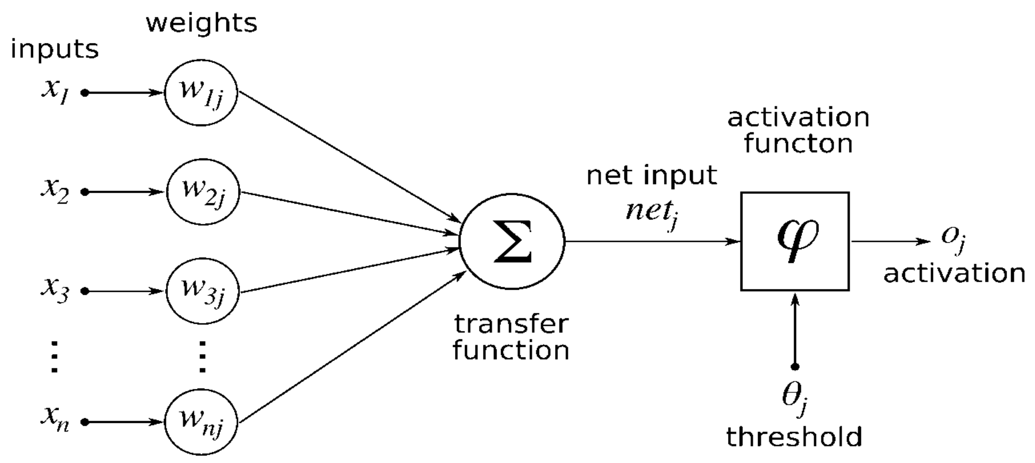


Figure 5: Equation

5.1 Forward Propagation:

Forward propagation is the method to move from input layer (left) to the output layer (right). It is used to calculate output from input and when we are working in ANN, we need to use both algorithms.

5.2 Back Propagation:

Back Propagation is the method to move from output (left) to the input layer (right). Back propagation is the method of adjusting weights respective to the inputs to reduce the error and increase correctness. In other words, back propagation is used to control the loss function. And loss function mostly is used to calculate overall error. Back propagation is an algorithm to train the weights.

5.3 Why we need?

- Back propagation is simple, fast algorithm and easy to program
- It is a flexible method
- It does not need any special features of the function

6 Methodology and Implementation:

Once a problem is given, its first step is to define the problem. What is real problem or what we have to develop? and how to find the optimal solution for that specific problem. Here it was given a project of image classification using sci-kit library of machine learning. When the problem has been identified then next step is to adopt optimal method and techniques to solve the problem. Before implementation, we have to make plans because when the plans are cleared then it is easy to implement the project. On the other side we have to set up specific tools on which the project is to be implemented.
Following methods and tools are used for image classification;

6.1 Visual Stdio Code:

For image classification we have to write code. For this we had installed Visual stdio code. Visual Studio Code is a lightweight yet capable source code editor for Windows, mac-OS, and Linux that runs on desktop. It features JavaScript support built-in, as well as a large ecosystem of extensions for other languages (such as C++, C, Java, and Python).

6.2 Python and its Libraries:

Python is a programming language which is almost being used in every field of computer science. Python is used for web development, game development, machine learning, implementation of GUI's and to solve scientific and numerical problems. For image classification, we selected python language. Python has open source libraries. sci-kit library is machine learning library for artificial neural network. To use this library we have to install it using pip install command in VS code IDE of python. Once the library has been installed then you have to make a folder and a file in it with the extension of ".py" which means that we are writing the code for python. Once you start using the file of python, it is ensures that specific directory should be opened in VS code.
We have to place train and validation data in the same directory. To access data from directories we have to use OS library. Once we accessed the data that was images then we used matplotlib for image reading.

7 Functions in Code:

We used different kind of terms and function for image classification in our code;

7.1 Train-x-final:

In this we placed all number of zero's from each feature extraction of each image separately for learning data. Basically, this is an array of our training data which we pass to the machine for learning.

7.2 Train-y:

This is the array for labeling each image with its number. For example, if the image is from "i" directory then the label of the image will be "i". This is also for our training data. We also pass this array to the machine.

7.3 val-x-final:

In this we placed all number of zero's from each feature extraction of each image separately for validation data. Basically, this is an array of our validation data.

7.4 val-y:

This is the array for labeling each image with its number. For example, if the image is from "i" directory then the label of the image will be "i". This is also for our validation data.

7.5 MLP Classifier:

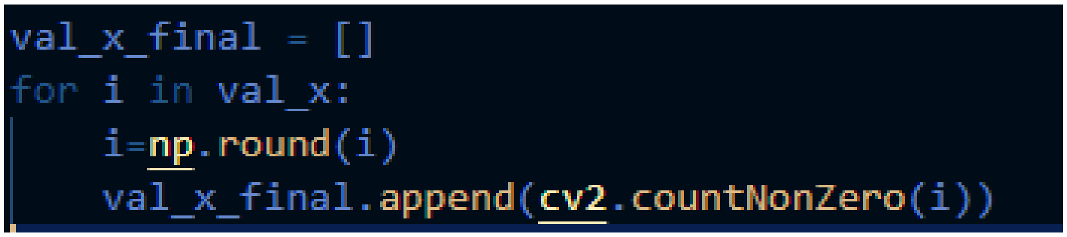
MLP classifier is the function of sci-kit library in python. It is used for multiple data classification. It has certain parameters;
Hidden layers which means how many neurons we are using for classification.
Learning rate which should be minimum. It is used for linearly so that it should not jump over the data.
Maximum Iteration which means in how many iteration the data should be learnt.

Activation function which transforms the sum of weights into output.
verbose which prints each iteration with the loss in learning.
Fit function which maps the data of train-x and train-y.
Predict function which predicts the result of validation data mapping with training data.

8 Testing and Analysis:

8.1 Count no of zero's in each image:

First of all We counted no of zero's in each image of train data and validation data using given below code and cv2 library.

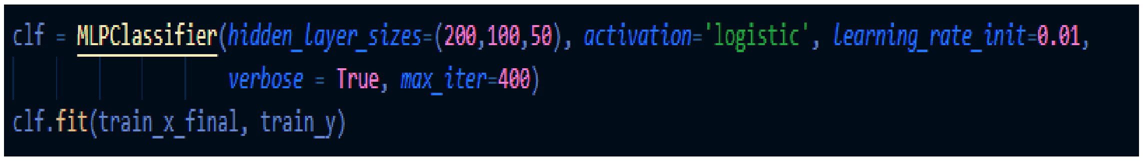


```
val_x_final = []
for i in val_x:
    i=np.round(i)
    val_x_final.append(cv2.countNonZero(i))
```

Figure 6: Counting the no of zero's

8.1.1 Setting MLP Classifier:

We set the MLP classifier with certain parameters;
Hidden-Layer-Sizes = 3
Learning Rate = 0.01
max-iteration = 400
activation = logistic

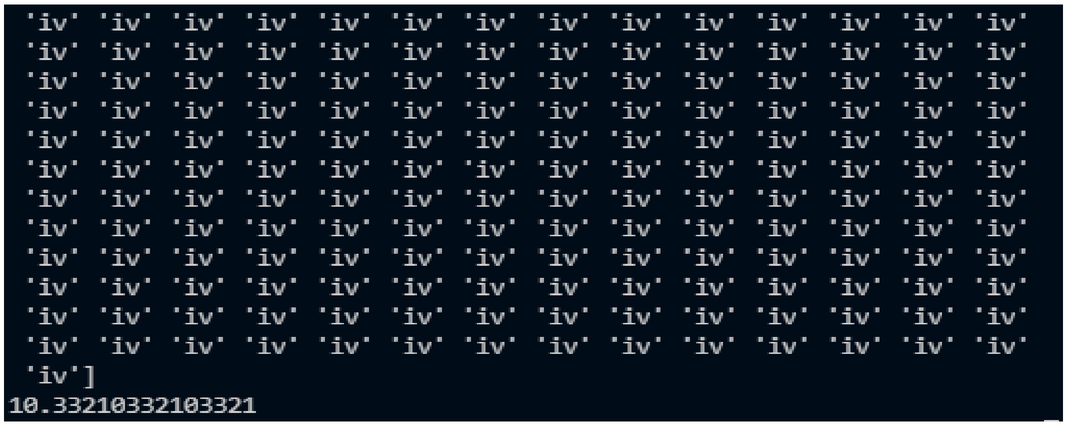


```
clf = MLPClassifier(hidden_layer_sizes=(200,100,50), activation='logistic', learning_rate_init=0.01,
                    verbose = True, max_iter=400)
clf.fit(train_x_final, train_y)
```

Figure 7: setting of MLP

8.1.2 Result:

This is not giving us a proper learning through our validation data. This gives us only 10 percent of total leaning.

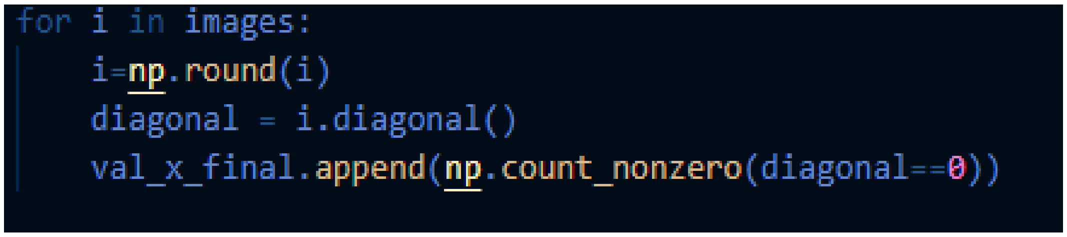


```
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv' 'iv'
 'iv']
10.33210332103321
```

Figure 8: Total Learning

8.2 Count no of zero's in each diagonal of image:

In second experiment, we found diagonal of each image and then counted the no of zero's in each diagonal of image and stored them in an array and that is array is passed to MLP Classifier.

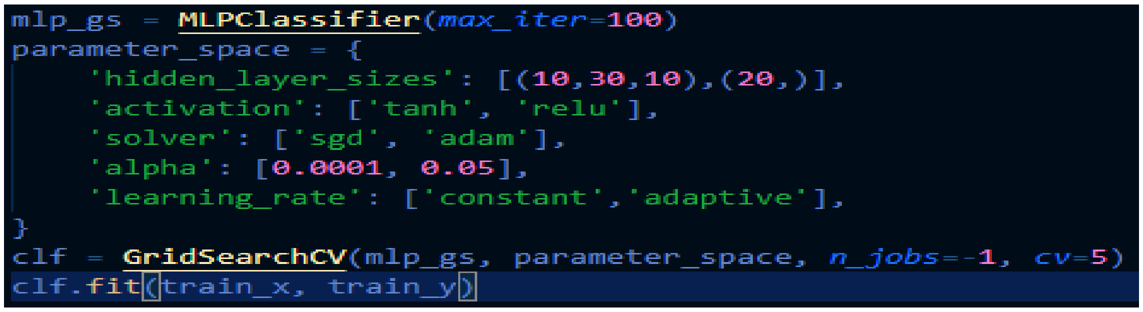


```
for i in images:
    i=np.round(i)
    diagonal = i.diagonal()
    val_x_final.append(np.count_nonzero(diagonal==0))
```

Figure 9: Counting the no of zero's in diagonal

8.2.1 Setting MLP Classifier:

This time I used an other version of MLP classifier which is using GridSearchCV. We set this classifier as given below;
Hidden-Layer-Sizes = 4
Learning Rate = 0.0001
max-iteration = 100
activation = tanh, relu

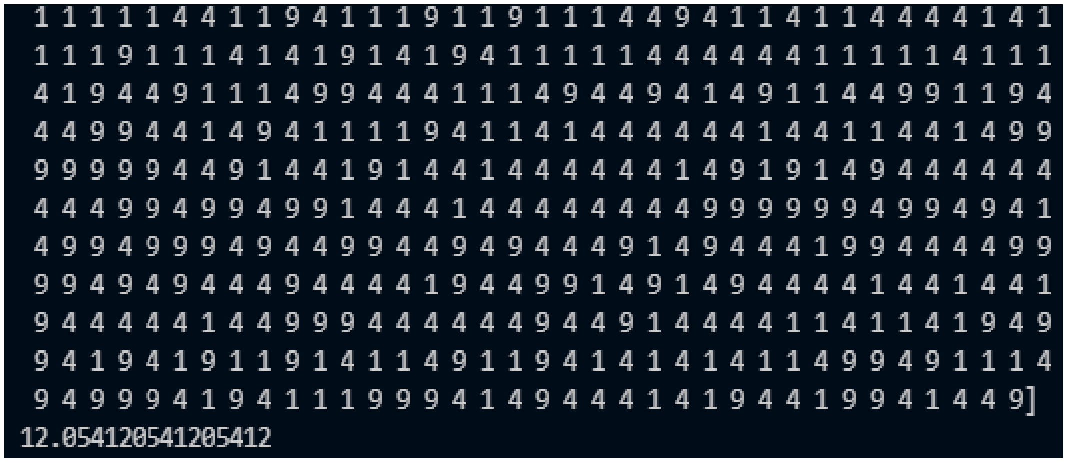


```
mlp_gs = MLPClassifier(max_iter=100)
parameter_space = {
    'hidden_layer_sizes': [(10,30,10),(20,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
clf = GridSearchCV(mlp_gs, parameter_space, n_jobs=-1, cv=5)
clf.fit(train_x, train_y)
```

Figure 10: setting of MLP

8.2.2 Result:

This is not giving us a proper learning through our validation data. This gives us only 12 percent of total leaning. This time 2 percent has been increased from the previous learning.

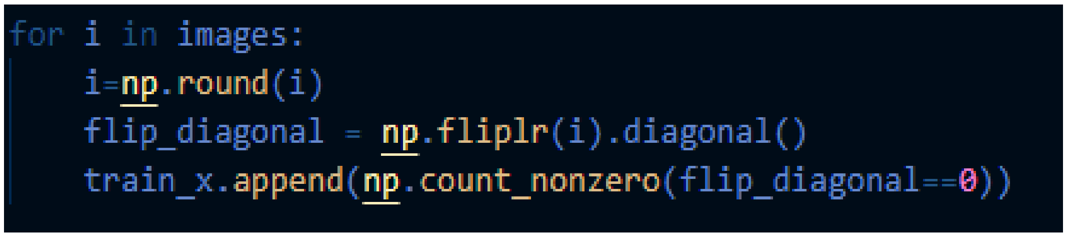


```
 1 1 1 1 1 4 4 1 1 9 4 1 1 1 9 1 1 9 1 1 1 4 4 9 4 1 1 4 1 1 4 4 4 4 1 4 1
 1 1 1 9 1 1 1 4 1 4 1 9 1 4 1 9 4 1 1 1 1 1 4 4 4 4 4 4 1 1 1 1 1 4 1 1 1
 4 1 9 4 4 9 1 1 1 4 9 9 4 4 4 1 1 1 4 9 4 4 9 4 1 4 9 1 1 4 4 9 9 1 1 9 4
 4 4 9 9 4 4 1 4 9 4 1 1 1 1 9 4 1 1 4 1 4 4 4 4 4 4 1 4 4 1 1 4 4 1 4 9 9
 9 9 9 9 9 4 4 9 1 4 4 1 9 1 4 4 1 4 4 4 4 4 4 1 4 9 1 9 1 4 9 4 4 4 4 4 4
 4 4 4 9 9 4 9 9 4 9 9 1 4 4 4 1 4 4 4 4 4 4 4 4 9 9 9 9 9 9 4 9 9 4 9 4 1
 4 9 9 4 9 9 9 4 9 4 4 9 9 4 4 9 4 9 4 4 4 9 1 4 9 4 4 4 1 9 9 4 4 4 4 9 9
 9 9 4 9 4 9 4 4 4 9 4 4 4 4 1 9 4 4 9 9 1 4 9 1 4 9 4 4 4 4 1 4 4 1 4 4 1
 9 4 4 4 4 4 1 4 4 9 9 9 4 4 4 4 4 4 9 4 4 9 1 4 4 4 4 1 1 4 1 1 4 1 9 4 9
 9 4 1 9 4 1 9 1 1 9 1 4 1 1 4 9 1 1 9 4 1 4 1 4 1 4 1 1 4 9 9 4 9 1 1 1 4
 9 4 9 9 9 4 1 9 4 1 1 1 9 9 9 4 1 4 9 4 4 4 1 4 1 9 4 4 1 9 9 4 1 4 4 9]
12.054120541205412
```

Figure 11: Total Learning

8.3 Count no of zero's in each flip diagonal of image:

In third experiment, we found flip diagonal of each image and then counted the no of zero's in each flip diagonal of image and stored them in an array and that is array is passed to MLP Classifier.

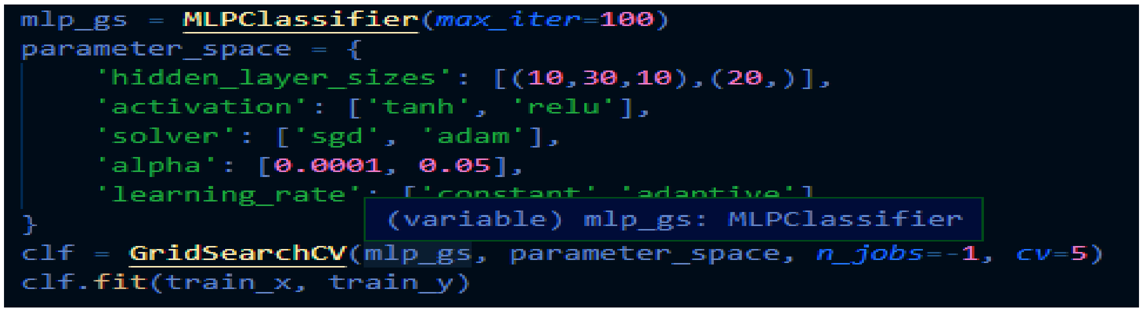


```
for i in images:
    i=np.round(i)
    flip_diagonal = np.fliplr(i).diagonal()
    train_x.append(np.count_nonzero(flip_diagonal==0))
```

Figure 12: Counting the no of zero's in flip diagonal

8.3.1 Setting MLP Classifier:

This time I used previous version of MLP classifier which is using GridSearchCV. We set this classifier as given below;
Hidden-Layer-Sizes = 4
Learning Rate = 0.0001
max-iteration = 100
activation = tanh, relu



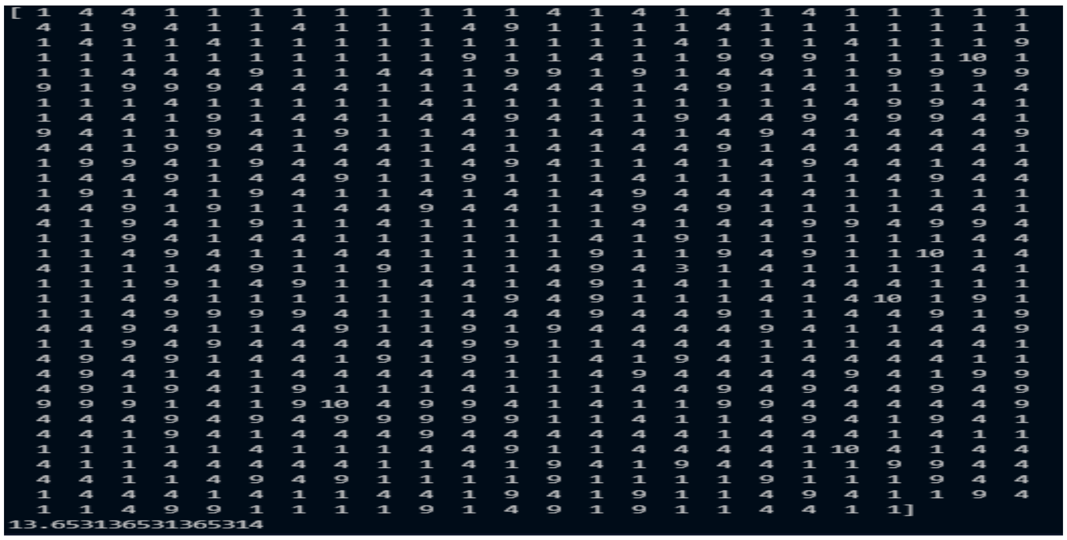
```
mlp_gs = MLPClassifier(max_iter=100)
parameter_space = {
    'hidden_layer_sizes': [(10,30,10),(20,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
clf = GridSearchCV(mlp_gs, parameter_space, n_jobs=-1, cv=5)
clf.fit(train_x, train_y)
```

(variable) mlp_gs: MLPClassifier

Figure 13: setting of MLP

8.3.2 Result:

This is not giving us a proper learning through our validation data. This gives us only 14 percent of total leaning. This time 4 percent has been increased from the previous learning. And I changed hidden layers and perceptrons many times but the learning percentage was same. One thing was different that when we change the hidden layers or learning rate, it gives us different character but it does not effect on the learning.

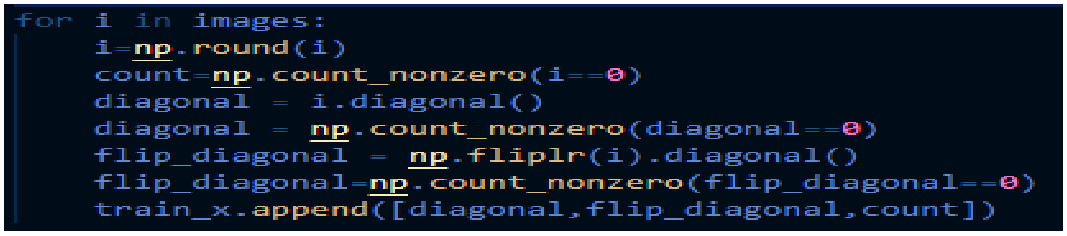


```
[ 1  4  4  1  1  1  1  1  1  1  1  1  4  1  4  1  4  1  4  1  1  1  1  1
  4  1  9  4  1  1  4  1  1  1  4  9  1  1  1  1  4  1  1  1  1  1  1  1
  1  4  1  1  4  1  1  1  1  1  1  1  1  1  1  4  1  1  1  4  1  1  1  9
  1  1  1  1  1  1  1  1  1  1  9  1  1  4  1  1  9  9  9  1  1  1 10  1
  1  1  4  4  4  9  1  1  4  4  1  9  9  1  9  1  4  4  1  1  9  9  9  9
  9  1  9  9  9  4  4  4  1  1  1  4  4  4  1  4  9  1  4  1  1  1  1  4
  1  1  1  4  1  1  1  1  1  4  1  1  1  1  1  1  1  1  1  4  9  9  4  1
  1  4  4  1  9  1  4  4  1  4  4  9  4  1  1  9  4  4  9  4  9  9  4  1
  9  4  1  1  9  4  1  9  1  1  4  1  1  4  4  1  4  9  4  1  4  4  4  9
  4  4  1  9  9  4  1  4  4  1  4  1  4  1  4  4  9  1  4  4  4  4  4  1
  1  9  9  4  1  9  4  4  4  1  4  9  4  1  1  4  1  4  9  4  4  1  4  4
  1  4  4  9  1  4  4  9  1  1  9  1  1  1  4  1  1  1  1  1  4  9  4  4
  1  9  1  4  1  9  4  1  1  4  1  4  1  4  9  4  4  4  4  1  1  1  1  1
  4  4  9  1  9  1  1  4  4  9  4  4  1  1  9  4  9  1  1  1  1  4  4  1
  4  1  9  4  1  9  1  1  4  1  1  1  1  1  4  1  4  4  9  9  4  9  9  4
  1  1  9  4  1  4  4  1  1  1  1  1  1  4  1  9  1  1  1  1  1  1  4  4
  1  1  4  9  4  1  1  4  4  1  1  1  1  9  1  1  9  4  9  1  1 10  1  4
  4  1  1  1  4  9  1  1  9  1  1  1  4  9  4  3  1  4  1  1  1  1  4  1
  1  1  1  9  1  4  9  1  1  4  4  1  4  9  1  4  1  1  4  4  4  1  1  1
  1  1  4  4  1  1  1  1  1  1  1  9  4  9  1  1  1  4  1  4 10  1  9  1
  1  1  4  9  9  9  9  4  1  1  4  4  4  9  4  4  9  1  1  4  4  9  1  9
  4  4  9  4  1  1  4  9  1  1  9  1  9  4  4  4  4  9  4  1  1  4  4  9
  1  1  9  4  9  4  4  4  4  4  9  9  1  1  4  4  4  1  1  1  4  4  4  1
  4  9  4  9  1  4  4  1  9  1  9  1  1  4  1  9  4  1  4  4  4  4  1  1
  4  9  4  1  4  1  4  4  4  4  4  1  1  4  9  4  4  4  4  9  4  1  9  9
  4  9  1  9  4  1  9  1  1  1  4  1  1  1  4  4  9  4  9  4  4  9  4  9
  9  9  9  1  4  1  9 10  4  9  9  4  1  4  1  1  9  9  4  4  4  4  4  9
  4  4  4  9  4  9  4  9  9  9  9  9  1  1  4  1  1  4  9  4  1  9  4  1
  4  4  1  9  4  1  4  4  4  9  4  4  4  4  4  4  1  4  4  4  1  4  1  1
  1  1  1  1  1  4  1  1  1  4  4  9  1  1  4  4  4  4  1 10  4  1  4  4
  4  1  1  4  4  4  4  4  1  1  4  1  9  4  1  9  4  4  1  1  9  9  4  4
  4  4  1  1  4  9  4  9  1  1  1  1  9  1  1  1  1  9  1  1  1  9  4  4
  1  4  4  4  1  4  1  1  4  4  1  9  4  1  9  1  1  4  9  4  1  1  9  4
  1  1  4  9  9  1  1  1  1  9  1  4  9  1  9  1  1  4  4  1  1]
13.653136531365314
```

Figure 14: Total Learning

8.4 Extracting all above three features in single one:

In fourth experiment,We extracted three features and and stored them in an array and counted no of zero's in each feature and stored them in an array and that is array is passed to MLP Classifier.



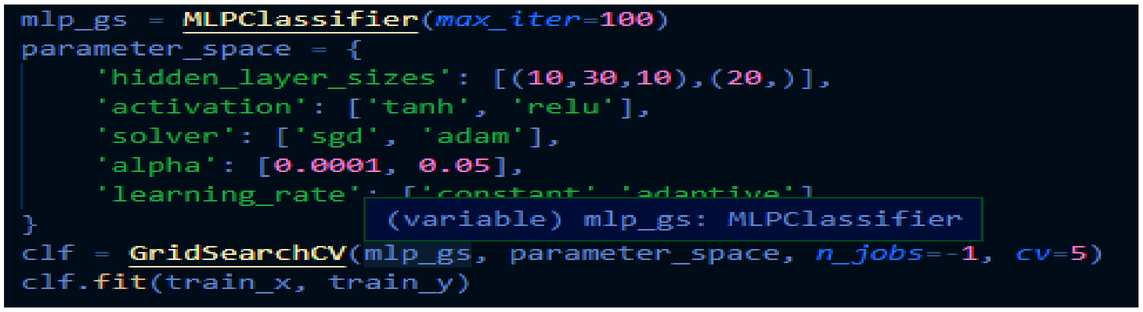
```
for i in images:
    i=np.round(i)
    count=np.count_nonzero(i==0)
    diagonal = i.diagonal()
    diagonal = np.count_nonzero(diagonal==0)
    flip_diagonal = np.fliplr(i).diagonal()
    flip_diagonal=np.count_nonzero(flip_diagonal==0)
    train_x.append([diagonal,flip_diagonal,count])
```

Figure 15: Extracting all three features at the same time:

8.4.1 Setting MLP Classifier:

This time I used previous version of MLP classifier which is using GridSearchCV. We set this classifier as given below;
Hidden-Layer-Sizes = 4
Learning Rate = 0.0001

max-iteration = 100
activation = tanh, relu

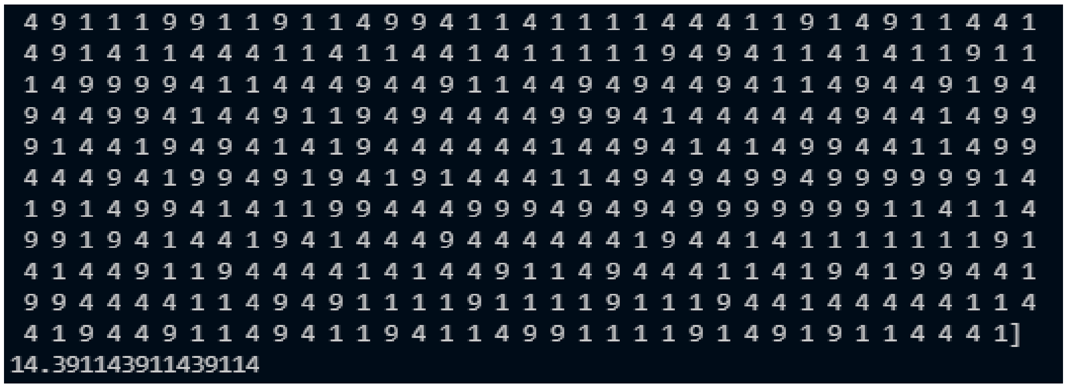


```
mlp_gs = MLPClassifier(max_iter=100)
parameter_space = {
    'hidden_layer_sizes': [(10,30,10),(20,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive']
}                      (variable) mlp_gs: MLPClassifier
clf = GridSearchCV(mlp_gs, parameter_space, n_jobs=-1, cv=5)
clf.fit(train_x, train_y)
```

Figure 16: setting of MLP

8.4.2 Result:

This is not giving us a proper learning through our validation data. This gives us only 14 percent of total leaning. This time 4 percent has been increased from the previous learning. And we changed hidden layers and perceptrons many times but the learning percentage was same. One thing was different that when we change the hidden layers or learning rate, it gives us different character but it does not effect on the learning.

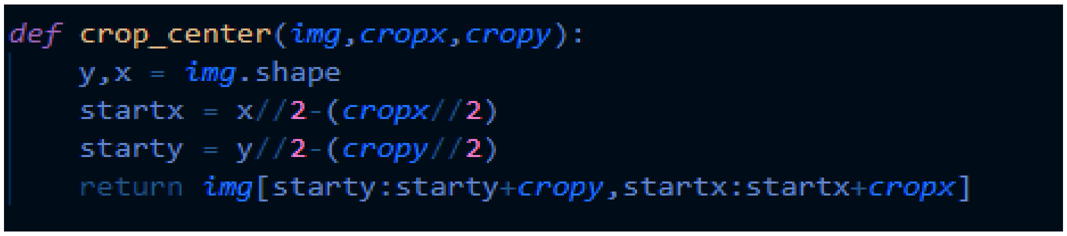


```
 4 9 1 1 1 9 9 1 1 9 1 1 4 9 9 4 1 1 4 1 1 1 1 4 4 4 1 1 9 1 4 9 1 1 4 4 1
 4 9 1 4 1 1 4 4 4 1 1 4 1 1 4 4 1 4 1 1 1 1 1 9 4 9 4 1 1 4 1 4 1 1 9 1 1
 1 4 9 9 9 9 4 1 1 4 4 4 9 4 4 9 1 1 4 4 9 4 9 4 4 9 4 1 1 4 9 4 4 9 1 9 4
 9 4 4 9 9 4 1 4 4 9 1 1 9 4 9 4 4 4 4 9 9 9 4 1 4 4 4 4 4 4 9 4 4 1 4 9 9
 9 1 4 4 1 9 4 9 4 1 4 1 9 4 4 4 4 4 4 1 4 4 9 4 1 4 1 4 9 9 4 4 1 1 4 9 9
 4 4 4 9 4 1 9 9 4 9 1 9 4 1 9 1 4 4 4 1 1 4 9 4 9 4 9 9 4 9 9 9 9 9 9 1 4
 1 9 1 4 9 9 4 1 4 1 1 9 9 4 4 4 9 9 9 4 9 4 9 4 9 9 9 9 9 9 9 1 1 4 1 1 4
 9 9 1 9 4 1 4 4 1 9 4 1 4 4 4 9 4 4 4 4 4 4 1 9 4 4 1 4 1 1 1 1 1 1 1 9 1
 4 1 4 4 9 1 1 9 4 4 4 4 1 4 1 4 4 9 1 1 4 9 4 4 4 1 1 4 1 9 4 1 9 9 4 4 1
 9 9 4 4 4 4 1 1 4 9 4 9 1 1 1 1 9 1 1 1 1 9 1 1 1 9 4 4 1 4 4 4 4 4 1 1 4
 4 1 9 4 4 9 1 1 4 9 4 1 1 9 4 1 1 4 9 9 1 1 1 1 9 1 4 9 1 9 1 1 4 4 4 1]
14.391143911439114
```

Figure 17: Total Learning

8.5 Image Cropping:

In fifth experiment,We resized each image up to 300x300 and then croped it from the center and store each image in array and that is array is passed to MLP Classifier.

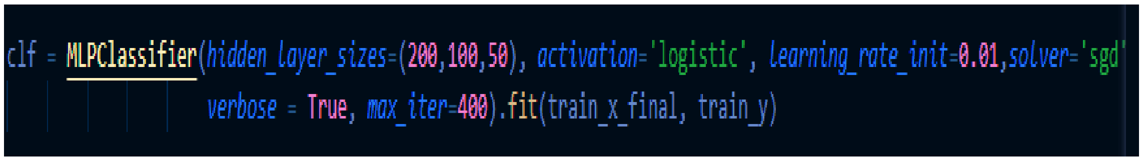


```python
def crop_center(img,cropx,cropy):
    y,x = img.shape
    startx = x//2-(cropx//2)
    starty = y//2-(cropy//2)
    return img[starty:starty+cropy,startx:startx+cropx]
```

Figure 18: Image croping:

8.5.1 Setting MLP Classifier:

This time I used previous version of MLP classifier. We set this classifier as given below;
Hidden-Layer-Sizes = 3
Learning Rate = 0.01
max-iteration = 400
activation = logistic

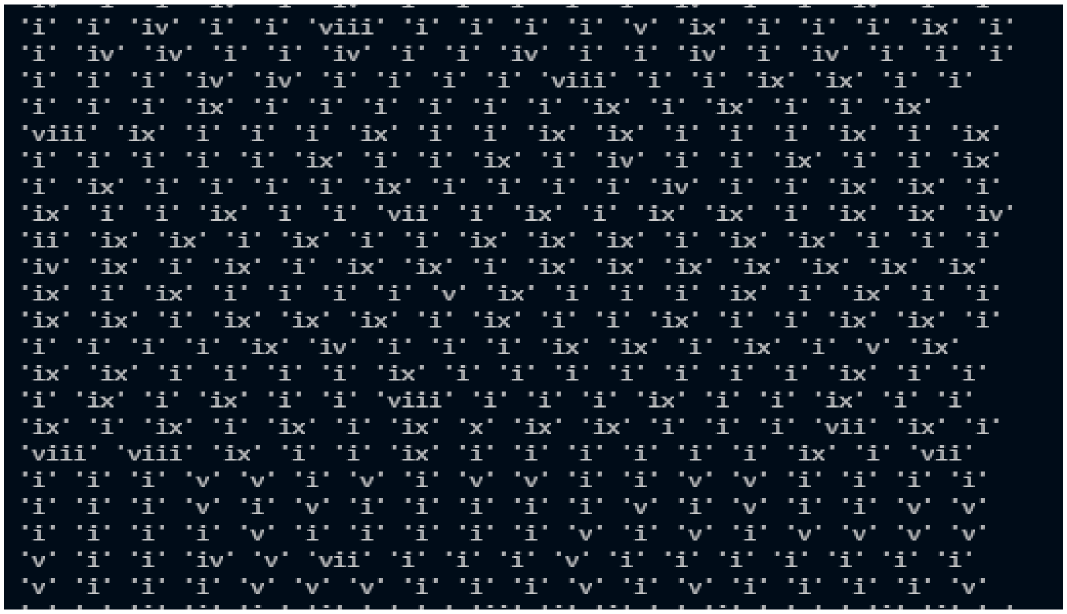


```python
clf = MLPClassifier(hidden_layer_sizes=(200,100,50), activation='logistic', learning_rate_init=0.01,solver='sgd'
                    verbose = True, max_iter=400).fit(train_x_final, train_y)
```

Figure 19: setting of MLP

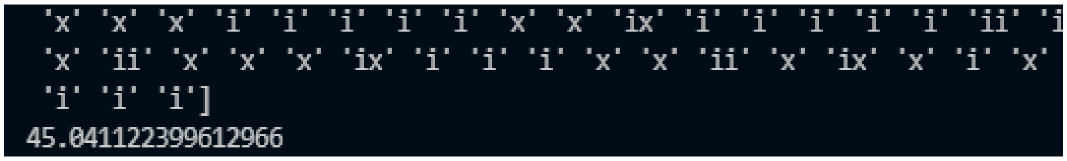
8.5.2 Result:

This time it gives us only 45 percent of total leaning. which was unexpected. We again ran the code but it did not give us again 45 rather it gives us 15 to 25. We changed hidden layers and perceptrons many times but the learning percentage was same. One thing was different that when we change the hidden layers or learning rate, it gives us different character but it does not effect on the learning.



```
'i' 'i' 'iv' 'i' 'i' 'viii' 'i' 'i' 'i' 'i' 'v' 'ix' 'i' 'i' 'i' 'ix' 'i'
'i' 'iv' 'iv' 'i' 'i' 'iv' 'i' 'i' 'iv' 'i' 'i' 'iv' 'i' 'iv' 'i' 'i' 'i'
'i' 'i' 'i' 'iv' 'iv' 'i' 'i' 'i' 'i' 'viii' 'i' 'i' 'ix' 'ix' 'i' 'i'
'i' 'i' 'i' 'ix' 'i' 'i' 'i' 'i' 'i' 'i' 'ix' 'i' 'ix' 'i' 'i' 'ix'
'viii' 'ix' 'i' 'i' 'i' 'ix' 'i' 'i' 'ix' 'ix' 'i' 'i' 'i' 'ix' 'i' 'ix'
'i' 'i' 'i' 'i' 'i' 'ix' 'i' 'i' 'ix' 'i' 'iv' 'i' 'i' 'ix' 'i' 'i' 'ix'
'i' 'ix' 'i' 'i' 'i' 'i' 'ix' 'i' 'i' 'i' 'i' 'iv' 'i' 'i' 'ix' 'ix' 'i'
'ix' 'i' 'i' 'ix' 'i' 'i' 'vii' 'i' 'ix' 'i' 'ix' 'ix' 'i' 'ix' 'ix' 'iv'
'ii' 'ix' 'ix' 'i' 'ix' 'i' 'i' 'ix' 'ix' 'ix' 'i' 'ix' 'ix' 'i' 'i' 'i'
'iv' 'ix' 'i' 'ix' 'i' 'ix' 'ix' 'i' 'ix' 'ix' 'ix' 'ix' 'ix' 'ix' 'ix'
'ix' 'i' 'ix' 'i' 'i' 'i' 'i' 'v' 'ix' 'i' 'i' 'i' 'ix' 'i' 'ix' 'i' 'i'
'ix' 'ix' 'i' 'ix' 'ix' 'ix' 'i' 'ix' 'i' 'i' 'ix' 'i' 'i' 'ix' 'ix' 'i'
'i' 'i' 'i' 'i' 'ix' 'iv' 'i' 'i' 'i' 'ix' 'ix' 'i' 'ix' 'i' 'v' 'ix'
'ix' 'ix' 'i' 'i' 'i' 'i' 'ix' 'i' 'i' 'i' 'i' 'i' 'i' 'i' 'ix' 'i' 'i'
'i' 'ix' 'i' 'ix' 'i' 'i' 'viii' 'i' 'i' 'i' 'ix' 'i' 'i' 'ix' 'i' 'i'
'ix' 'i' 'ix' 'i' 'ix' 'i' 'ix' 'x' 'ix' 'ix' 'i' 'i' 'i' 'vii' 'ix' 'i'
'viii' 'viii' 'ix' 'i' 'i' 'ix' 'i' 'i' 'i' 'i' 'i' 'i' 'ix' 'i' 'vii'
'i' 'i' 'i' 'v' 'v' 'i' 'v' 'i' 'v' 'v' 'i' 'i' 'v' 'v' 'i' 'i' 'i' 'i'
'i' 'i' 'i' 'v' 'i' 'v' 'i' 'i' 'i' 'i' 'i' 'v' 'i' 'v' 'i' 'i' 'v' 'v'
'i' 'i' 'i' 'i' 'v' 'i' 'i' 'i' 'i' 'i' 'v' 'i' 'v' 'i' 'v' 'v' 'v' 'v'
'v' 'i' 'i' 'iv' 'v' 'vii' 'i' 'i' 'i' 'v' 'i' 'i' 'i' 'i' 'i' 'i' 'i'
'v' 'i' 'i' 'i' 'v' 'v' 'v' 'i' 'i' 'i' 'v' 'i' 'v' 'i' 'i' 'i' 'i' 'v'
```

Figure 20: Total Learning



```
 'x' 'x' 'x' 'i' 'i' 'i' 'i' 'i' 'x' 'x' 'ix' 'i' 'i' 'i' 'i' 'i' 'ii' 'i
 'x' 'ii' 'x' 'x' 'x' 'ix' 'i' 'i' 'i' 'x' 'x' 'ii' 'x' 'ix' 'x' 'i' 'x'
 'i' 'i' 'i']
45.041122399612966
```

Figure 21: Total Learning

9 Conclusion

In this report, we proposed an image classification algorithm from the sci-kit library of python. We were given two type of data one was training data and other was validation data. Both data comprised of ten different kind of images in each directory. We had to train our machine with training data and then we had to check the learning through our validation data. In this project, we performed different types of experiments but the result was not different. Only once it gave us better result that was 45 percent learning. When we trained the machine with training data and learn through training then it gave us 50 to 80 percent learning. But it was not giving learning with validation data.
We tried our best to achieve the best learning percentage but could not achieve it. If the time was more, may be we could learn more.