B. TECH (CSE(AI)) (2021)

AMRITA VISHWA VIDYAPEETHAM, COIMBATORE

19AIE205 PYTHON FOR MACHINE LEARNING

PROJECT REPORT

GROUP-06 TEAM MEMBERS:

- M.Visweswaran [CB.EN. U4AIE20075]
- Vishnu Radhakrishnan [CB.EN. U4AIE20074]
- Thushit Kumar R [CB.EN. U4AIE20072]
- Menta Sai Akshay [CB.EN. U4AIE20040]
- Krishnan K M [CB.EN. U4AIE20031]

PROJECT TOPIC: Hand written Digit recognition using KNN, Random Forest, SVM, Neural Networks (NN) and Convoluted Neural Networks (CNN)

PROJECT REPORT SUBMITTED FOR THE END SEMESTER EXAMINATION OF 19AIE205

EXTERNAL EXAMINER INTERNAL EXAMINER

ACKNOWLEDGEMENT

We would like to thank all those who have helped us in completing this project of “HANDWRITTEN DIGIT RECOGNITION” under the subject “MATHEMATICS FOR INTELLIGENT SYSTEMS-3”.

We would like to show our sincere gratitude to our professor NEETHU MOHAN without whom the project would not have been initiated, who taught us the basics to start and visualize the project and enlightened us with the ideas regarding the project, and helped us by clarifying all the doubts whenever being asked.

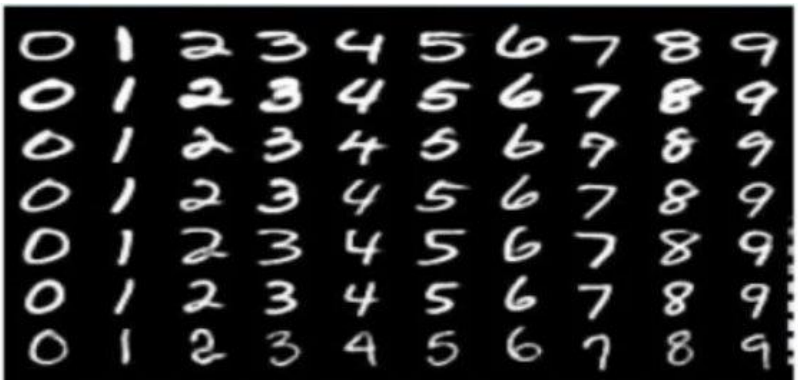
We would like to thank ourselves. We helped each other and taught each other about various concepts regarding the project which helped in increasing our knowledge.

TABLE OF CONTENTS:

1. **Abstract**
2. **KNN**
3. **Random Forest**
4. **NN**
5. **SVM**
6. **CNN**
7. **GUI**

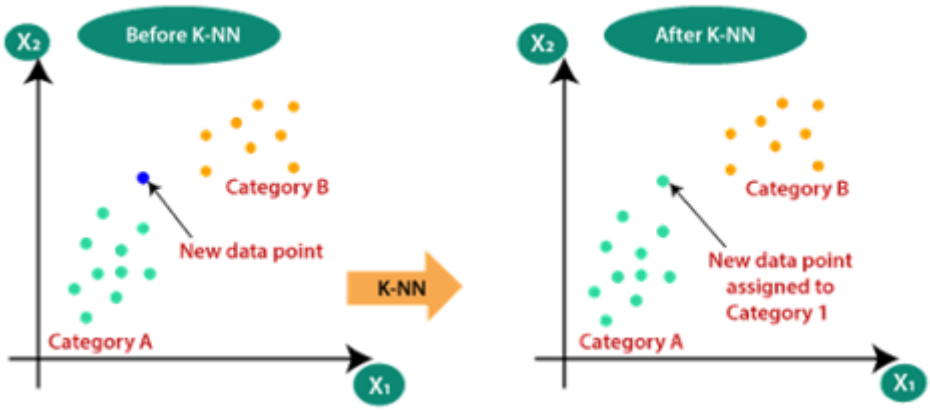
Abstract:

Handwritten Digit Recognition is used to recognize the Digits which are written by hand.A handwritten digit recognition system is used to visualize artificial neural networks.It is already widely used in the automatic processing of bank cheques, postal addresses, in mobile phones etc. Training of the network is done by a dataset named MNIST dataset.MNIST dataset has a training set of 60,000 examples. All the images in the dataset are of 28 x 28 pixels.



KNN :

KNN is a machine learning algorithm where the test data's distance in the space from each training data is calculated and the first k nearest neighbors are chosen from this training data which are closest to the test data , among these k nearest neighbors the class having highest occurrence is considered as the class of the test data.



CODE :

```
import cv2                              # importing necessary packages
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.datasets as ds
import math
```

```
r = 2000                                # no. of train data
data = pd.read_csv("train_data.csv")    # reading train data

Data = data.values[0:r]                 # necessary set
Pix = Data[:,1:]                        # getting the pixel array of train dtaset
Lab = Data[:,0]                         # and Labels correspnding them
```

```
# function to calculate the distance between d1 and d2 points
def euc_dist(d1,d2):
    dis = 0                             # distance ( initially 0 )
    for i in range(len(d1)):            # traversing each feature of data

        dis += (d1[i]-d2[i])**2         # calculating square euclidean distance
    dis = math.sqrt(dis)                # euclidean distance
    return dis
```

```
def knnf(train_x,train_y,test,k):       # Knn clasifier
    train_y = np.array(train_y)         # traning data's labels
    dist = []                           # list storing distance between the test data and other train datas

    for i in range(len(train_x)):       # calculating the euclidean distances between the test data and other train datas

        dist.append(euc_dist(train_x[i],test))

    args = np.argsort(dist)             # sorting ( ascending ) the distances
    train_y = train_y[args]             # arranging the labels according to the sorted array containing distances
    train_y = train_y[:k]               # taking all the labels of first k nearest neighbours

    # return the class having higher number of occurences in the k nearest neighbours
    # since it is considered as the class of the test data

    return np.bincount(train_y).argmax()
```

TESTING :

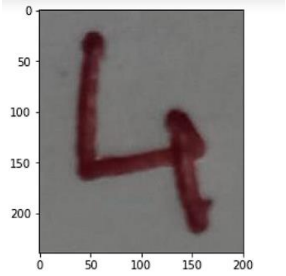
```
Test = pd.read_csv("test_data.csv")     # reading the testdata
T = Test.values[:100]                   # taking the first 100 values
y_pred = []                             # list containg the classes of the pictures from KNN

# labels of first 100 data
k=[2, 0, 9, 0, 3, 7, 0, 3, 0, 3, 5, 7, 4, 0, 4, 3, 3, 1, 9, 0, 9, 1, 1, 5, 7, 4, 2, 7, 4, 7, 7, 5, 4, 2, 6, 2, 5, 5, 1, 6, 7, 7,

# for every test data appending the label from KNN and printing the output
for i in range(len(T)):
    label = knnf(Pix,Lab,T[i],5)
    img = np.reshape(T[i],(28,28))
    plt.imshow(img)
    plt.show()
    y_pred.append(label)
    print("Number is: ",label)

# calculating acuracy
names = [ int(kk) for kk in k]
lisy = [int(y_pred[i]==names[i]) for i in range(len(names))]
# since if the real label and KNN label are same it will be 1 else 0 accuracy = lisy's sum / len(lisy)
accuracy = np.array(lisy).sum()/len(names)
print(f"Acuuarcy is {accuracy*100}%")
```

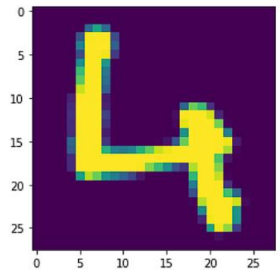
```
Acuracy is 89.0%
```

For any Arbitrary Image :

```
k = cv2.imread('pics/i3.jpeg')
plt.imshow(k)
plt.show()

# preprocesssing the image
k = cv2.cvtColor(k,cv2.COLOR_BGR2GRAY)
k = cv2.bitwise_not(cv2.threshold(k, 100, 255, cv2.THRESH_BINARY)[1])

resized = cv2.resize(k,[28,28], interpolation = cv2.INTER_AREA)
plt.imshow(resized)
plt.show()
print('Resized Dimensions : ',resized.shape)

img = np.asarray(resized).reshape(-1)
num = knnf(Pix,Lab,img,5)
print("The number is:",num)
```

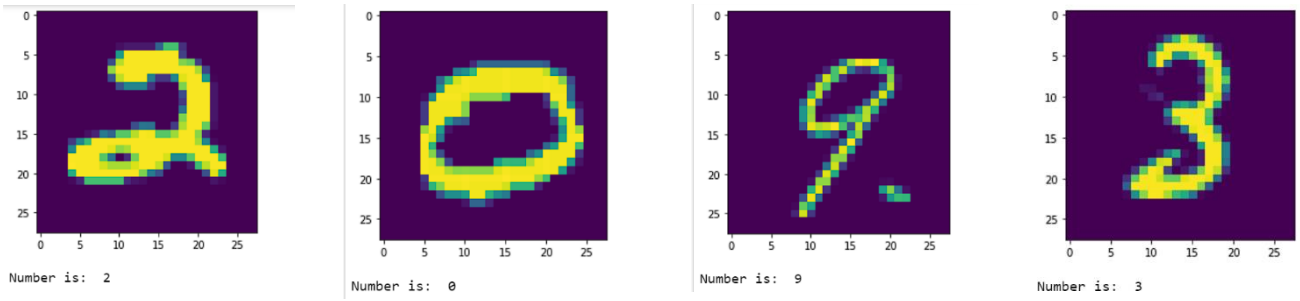
For an arbitrary input (number 4) :





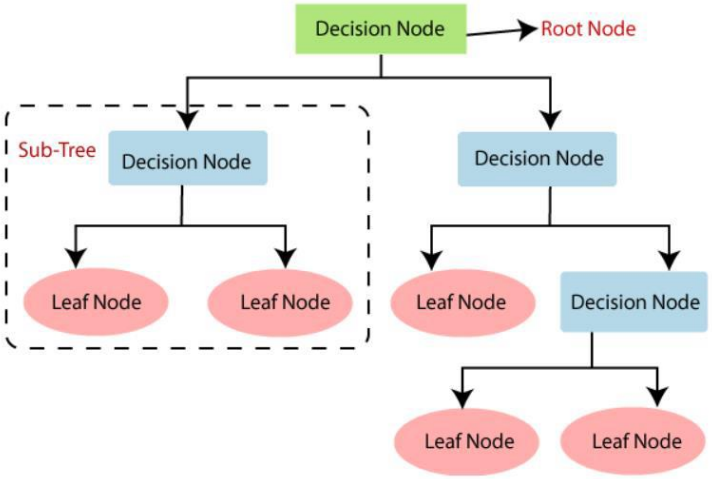

```
Resized Dimensions :  (28, 28)
The number is: 4
```

SAMPLE OUTPUT :



Decision Trees:

A random forest algorithm's building components are decision trees. A decision tree is a decision-making technique with a tree-like structure. It is a supervised learning technique that may be used to solve classification and regression problems.



A decision tree is made up of three parts: decision nodes, leaf nodes, and a root node. A decision tree method separates a training dataset into branches, which are then subdivided into sub-branches. This sequence is repeated until a leaf node is reached. The leaf node cannot be further separated. The decision tree's nodes indicate attributes that are used to forecast the outcome.

It is a graphical representation of all possible solutions to a problem/decision given certain parameters. It is named a decision tree because, like a tree, it begins with the root node and then develops on subsequent branches to form a tree-like structure. In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and,

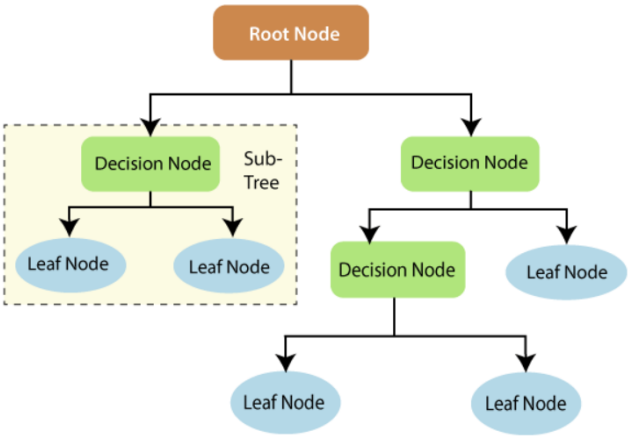
based on the comparison, follows the branch and jumps to the next node. For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree.

Random Forest:

The Random Forest algorithm is a supervised learning algorithm which builds multiple decision trees and merges them together to get a more accurate and stable prediction. The "forest" it creates is an ensemble of decision trees, which are often trained using the "bagging" method. The bagging method is based on the idea that combining learning models improves the final output.

Features of a Random Forest Algorithm:

- It outperforms the decision tree algorithm in terms of accuracy.
- It gives an efficient method of dealing with missing data.
- It is capable of producing a reasonable prediction without the use of hyper-parameter tuning.
- It eliminates the problem of overfitting in decision trees.
- At the node's splitting point in every random forest tree, a subset of features is chosen at random.



The algorithm establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome.

Bagging:

Bagging (bootstrapping + aggregating) is the usage of an ensemble of models in which each model uses a bootstrapped data set and the predictions of the models are aggregated.

Bootstrapping:

Each tree in a random forest learns from a random selection of data points during training. The samples are drawn utilising replacement, which implies that certain samples will be used several times in a single tree. The theory is that by training each tree on diverse samples, even if each tree has a high variation with regard to a specific set of training data, the overall variance of the forest will be smaller, but not at the expense of increasing the bias.

Aggregation:

An overall forecast or estimate is created using random forest methods by aggregating predictions given by individual decision trees.

Forecasts are made at test time by averaging the predictions of each decision tree. Bagging, short for bootstrap aggregating, is the process of training each individual learner on distinct bootstrapped subsets of the data and then averaging the predictions.

CODE:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.datasets as ds
import math
```

```
#reading train and test files
train_file = pd.read_csv('train_data.csv',nrows=1000)
test_file = pd.read_csv('test_data.csv',nrows=1000)
len(test_file)
```

```
1000
```

```
#list of all digits that are going to be predicted
np.sort(train_file.label.unique())
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int64)
```

```
#defining the number of samples for training set and for validation set
num_train,num_validation = int(len(train_file)*0.8),int(len(train_file)*0.2)
```

```
#generating training data from train_file
x_train,y_train=train_file.iloc[:num_train,1:].values,train_file.iloc[:num_train,0].values
```

```
#validating the data set
x_validation,y_validation=train_file.iloc[num_train:,1:].values,train_file.iloc[num_train:,0].values
```

```
X_test=test_file
```

Decision Tree

In [2]:
```
from collections import Counter

import numpy as np

# Entropy is the measure of randomness or unpredictability In the dataset
def entropy(y):
    hist = np.bincount(y)
    ps = hist / len(y)
    return -np.sum([p * np.log2(p) for p in ps if p > 0])


class Node:
    def __init__(
        self, feature=None, threshold=None, left=None, right=None, *, value=None
    ):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

    def is_leaf_node(self):
        return self.value is not None       # Leaf Node carries the classification or the decision
```

```
class DecisionTree:
    def __init__(self, min_samples_split=2, max_depth=100, n_feats=None):
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.n_feats = n_feats
        self.root = None

    def fit(self, X, y):
        self.n_feats = X.shape[1] if not self.n_feats else min(self.n_feats, X.shape[1])
        self.root = self._grow_tree(X, y)

    def predict(self, X):
        l = list()
        for i in range(len(X)):
            l.append(self._traverse_tree(X.iloc[i],self.root))
        return np.array(l)

    def _grow_tree(self, X, y, depth=0):
        n_samples, n_features = X.shape
        n_labels = len(np.unique(y))

        # stopping criteria
        if (
            depth >= self.max_depth
            or n_labels == 1
            or n_samples < self.min_samples_split
        ):
            leaf_value = self._most_common_label(y)
            return Node(value=leaf_value)

        feat_idxs = np.random.choice(n_features, self.n_feats, replace=False)

        # greedily select the best split according to information gain
        best_feat, best_thresh = self._best_criteria(X, y, feat_idxs)
```

```
        # grow the children that result from the split
        left_idxs, right_idxs = self._split(X[:, best_feat], best_thresh)
        left = self._grow_tree(X[left_idxs, :], y[left_idxs], depth + 1)
        right = self._grow_tree(X[right_idxs, :], y[right_idxs], depth + 1)
        return Node(best_feat, best_thresh, left, right)

    def _best_criteria(self, X, y, feat_idxs):
        best_gain = -1
        split_idx, split_thresh = None, None
# starting for loop in criteria,len(feat_idxs)

        for feat_idx in feat_idxs:
            X_column = X[:, feat_idx]
            thresholds = np.unique(X_column)
            print(len(thresholds),"Length")
            i=0
            for threshold in thresholds:
                gain = self._information_gain(y, X_column, threshold)
                print("Done with",i)
                i = i+1
                if gain > best_gain:
                    best_gain = gain
                    split_idx = feat_idx
                    split_thresh = threshold

        return split_idx, split_thresh
# It is the measure of decrease in entropy after the dataset is split
    def _information_gain(self, y, X_column, split_thresh):
        # parent loss
        parent_entropy = entropy(y)

        # generate split
        left_idxs, right_idxs = self._split(X_column, split_thresh)
```

```
        # generate split
        left_idxs, right_idxs = self._split(X_column, split_thresh)

        if len(left_idxs) == 0 or len(right_idxs) == 0:
            return 0

        # compute the weighted avg. of the loss for the children
        n = len(y)
        n_l, n_r = len(left_idxs), len(right_idxs)
        e_l, e_r = entropy(y[left_idxs]), entropy(y[right_idxs])
        child_entropy = (n_l / n) * e_l + (n_r / n) * e_r

        # information gain is difference in loss before vs. after split
        ig = parent_entropy - child_entropy
        return ig

    def _split(self, X_column, split_thresh):
        left_idxs = np.argwhere(X_column <= split_thresh).flatten()
        right_idxs = np.argwhere(X_column > split_thresh).flatten()
        return left_idxs, right_idxs

    def _traverse_tree(self, x, node):
        if node.is_leaf_node():
            return node.value
        if x[node.feature] <= node.threshold:
            return self._traverse_tree(x, node.left)
        return self._traverse_tree(x, node.right)

    def _most_common_label(self, y):
        counter = Counter(y)
        most_common = counter.most_common(1)[0][0]
        return most_common
```

```
            print("Tree done is :",i)
            X_samp, y_samp = bootstrap_sample(X, y)
            tree.fit(X_samp, y_samp)
            self.trees.append(tree)

    def predict(self, X):
        tree_preds = np.array([tree.predict(X) for tree in self.trees])
        tree_preds = np.swapaxes(tree_preds, 0, 1)
        y_pred = [most_common_label(tree_pred) for tree_pred in tree_preds]
        return np.array(y_pred)

    def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy
```

Training The model

```
# Instantiating model with 125 decision trees
clf = RandomForest(n_trees=125, max_depth=100,min_samples_split=20)
# Training the model on training data
clf.fit(x_train, y_train)
```

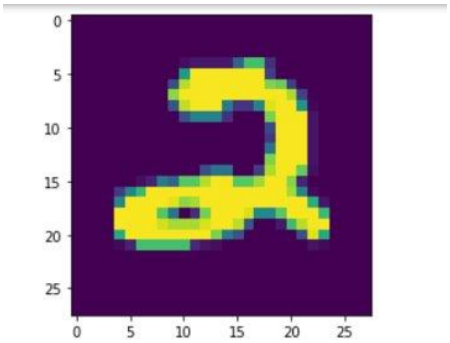
Testing and Accuracy

```
Test = pd.read_csv("test_data.csv")

T = Test.values[0:100]
y_pred = clf.predict(Test)
names=list()
k=[2, 0, 9, 0, 3, 7, 0, 3, 0, 3, 5, 7, 4, 0, 4, 3, 3, 1, 9, 0, 9, 1, 1, 5, 7, 4, 2, 7, 4, 7, 7, 5
k = k[0:20]
for i in range(len(T)):
    img = np.reshape(T[i],(28,28))
    plt.imshow(img)
    plt.show()
    print("The number according to Rf is", y_pred[i])
```

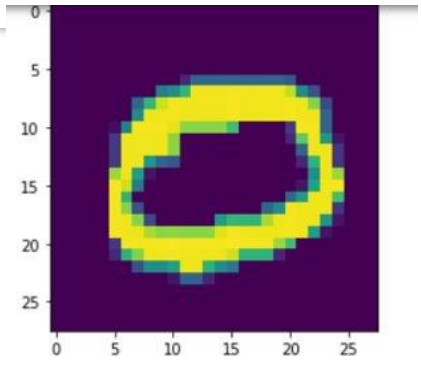
In [18]:
```
names = [ int(kk) for kk in k]
lisy = [int(y_pred[i]==names[i]) for i in range(len(names))]
accuracy = np.array(lisy).sum()/len(names)
print(f"Acuuarcy is {accuracy*100}%")
```

```
Acuuarcy is 90.0%
```

OUTPUT:



The number according to Rf is 2



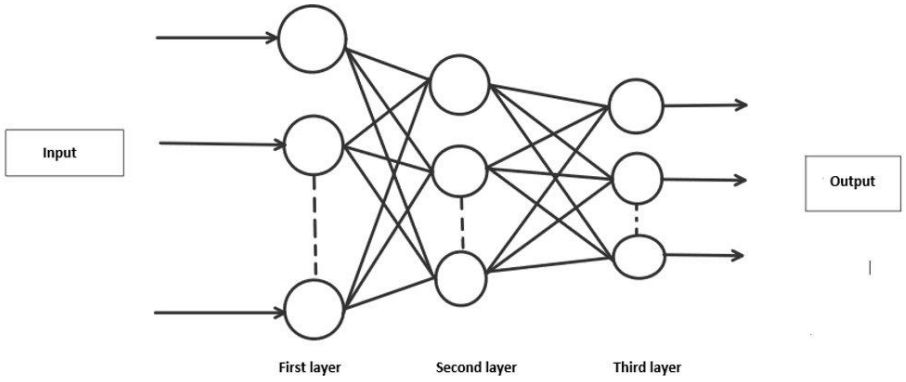
The number according to Rf is 0

Neural Network

A neural network is a model inspired by how the brain works. It consists of multiple layers having many activations, this activation resembles neurons of our brain. A neural network tries to learn a set of parameters in a set of data which could help to recognize the underlying relationships. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria.

Algorithm:

We have implemented a Neural Network with 1 hidden layer having *100* activation units (excluding bias units). The data is loaded from a *.mat* file, features(X) and labels(y) were extracted. Then features are divided by *255* to rescale them into a range of *[0,1]* to avoid overflow during computation. Data is split up into *60,000* training and *10,000* testing examples. Feedforward is performed with the training set for calculating the hypothesis and then backpropagation is done in order to reduce the error between the layers. The regularization parameter lambda is set to 0.1 to address the problem of overfitting. Optimizer is run for 70 iterations to find the best fit model.



Elements of Neural Network

Elements of a Neural Network :-
Input Layer :- This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.

Hidden Layer :- Nodes of this layer are not exposed to the outer world, they are the part of the abstraction provided by any neural network. Hidden layer performs all sort of computation on the features entered through the input layer and transfer the result to the output layer.
Output Layer :- This layer bring up the information learned by the network to the outer world.

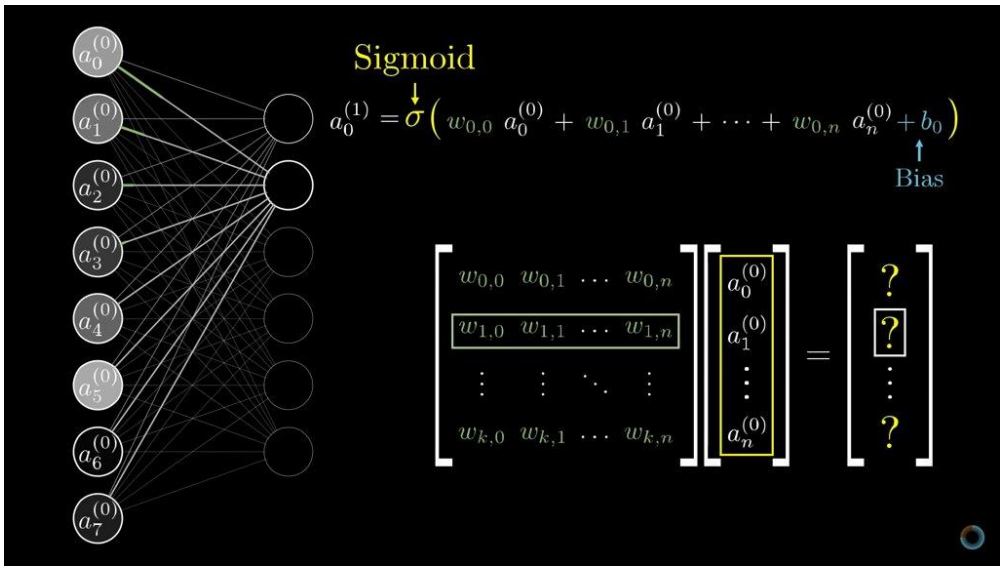
Activation function

Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

neural network has neurons that work in correspondence of *weight, bias* and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as *back-propagation*. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

In our case we have used the Sigmoid function



Code:

In [5]:
```
import numpy as np


def neural_network(nn_params, input_layer_size, hidden_layer_size, num_labels, X, y, lamb):
    # Weights are split back to Theta1, Theta2
    Theta1 = np.reshape(nn_params[:hidden_layer_size * (input_layer_size + 1)],
                        (hidden_layer_size, input_layer_size + 1))
    Theta2 = np.reshape(nn_params[hidden_layer_size * (input_layer_size + 1):],
                        (num_labels, hidden_layer_size + 1))

    # Forward propagation
    m = X.shape[0]
    one_matrix = np.ones((m, 1))
    X = np.append(one_matrix, X, axis=1) # Adding bias unit to first layer
    a1 = X
    z2 = np.dot(X, Theta1.transpose())
    a2 = 1 / (1 + np.exp(-z2)) # Activation for second layer
    one_matrix = np.ones((m, 1))
    a2 = np.append(one_matrix, a2, axis=1) # Adding bias unit to hidden layer
    z3 = np.dot(a2, Theta2.transpose())
    a3 = 1 / (1 + np.exp(-z3)) # Activation for third layer

    # Changing the y labels into vectors of boolean values.
    # For each label between 0 and 9, there will be a vector of length 10
    # where the ith element will be 1 if the label equals i
    y_vect = np.zeros((m, 10))
    for i in range(m):
        y_vect[i, int(y[i])] = 1

    # Calculating cost function
    J = (1 / m) * (np.sum(np.sum(-y_vect * np.log(a3) - (1 - y_vect) * np.log(1 - a3)))) + (lamb / (2 * m)) * (
                sum(sum(pow(Theta1[:, 1:], 2))) + sum(sum(pow(Theta2[:, 1:], 2))))
```

```
    # backprop
    Delta3 = a3 - y_vect
    Delta2 = np.dot(Delta3, Theta2) * a2 * (1 - a2)
    Delta2 = Delta2[:, 1:]

    #gradient
    Theta1[:, 0] = 0
    Theta1_grad = (1 / m) * np.dot(Delta2.transpose(), a1) + (lamb / m) * Theta1
    Theta2[:, 0] = 0
    Theta2_grad = (1 / m) * np.dot(Delta3.transpose(), a2) + (lamb / m) * Theta2
    grad = np.concatenate((Theta1_grad.flatten(), Theta2_grad.flatten()))

    return J, grad
```

In [6]:
```
import numpy as np
  # Randomly initialises values of thetas between [-epsilon, +epsilon]
def initialise(a, b):
    epsilon = 0.15
    c = np.random.rand(a, b + 1) * (2 * epsilon) - epsilon
    return c
```

In [7]:
```
import numpy as np


def predict(Theta1, Theta2, X):
    m = X.shape[0]
    one_matrix = np.ones((m, 1))
    X = np.append(one_matrix, X, axis=1) # Adding bias unit to first layer
    z2 = np.dot(X, Theta1.transpose())
    a2 = 1 / (1 + np.exp(-z2)) # Activation for second layer
    one_matrix = np.ones((m, 1))
    a2 = np.append(one_matrix, a2, axis=1) # Adding bias unit to hidden layer
    z3 = np.dot(a2, Theta2.transpose())
    a3 = 1 / (1 + np.exp(-z3)) # Activation for third layer
    p = (np.argmax(a3, axis=1)) # Predicting the class on the basis of max value of hypothesis
    return p
```

In [8]:
```
from scipy.io import loadmat
import numpy as np
from scipy.optimize import minimize


# Loading mat file
data = loadmat('mnist-original.mat')

# Extracting features from mat file
X = data['data']
X = X.transpose()

# Normalizing the data
X = X / 255

# Extracting labels from mat file
y = data['label']
y = y.flatten()

# Splitting data into training set with 60,000 examples
X_train = X[:60000, :]
y_train = y[:60000]

# Splitting data into testing set with 10,000 examples
X_test = X[60000:, :]
y_test = y[60000:]

m = X.shape[0]
input_layer_size = 784 # Images are of (28 X 28) px so there will be 784 features
hidden_layer_size = 100
num_labels = 10 # There are 10 classes [0, 9]
```

```
# Randomly initialising Thetas
initial_Theta1 = initialise(hidden_layer_size, input_layer_size)
initial_Theta2 = initialise(num_labels, hidden_layer_size)

# Unrolling parameters into a single column vector
initial_nn_params = np.concatenate((initial_Theta1.flatten(), initial_Theta2.flatten()))
maxiter = 100
lambda_reg = 0.1 # To avoid overfitting
myargs = (input_layer_size, hidden_layer_size, num_labels, X_train, y_train, lambda_reg)

# Calling minimize function to minimize cost function and to train weights
results = minimize(neural_network, x0=initial_nn_params, args=myargs,
        options={'disp': True, 'maxiter': maxiter}, method="L-BFGS-B", jac=True)

nn_params = results["x"] # Trained Theta is extracted

# Weights are split back to Theta1, Theta2
Theta1 = np.reshape(nn_params[:hidden_layer_size * (input_layer_size + 1)], (
                        hidden_layer_size, input_layer_size + 1)) # shape = (100, 785)
Theta2 = np.reshape(nn_params[hidden_layer_size * (input_layer_size + 1):],
                    (num_labels, hidden_layer_size + 1)) # shape = (10, 101)

# Checking test set accuracy of our model
pred = predict(Theta1, Theta2, X_test)
print('Test Set Accuracy: {:f}'.format((np.mean(pred == y_test) * 100)))

# Checking train set accuracy of our model
pred = predict(Theta1, Theta2, X_train)
print('Training Set Accuracy: {:f}'.format((np.mean(pred == y_train) * 100)))

# Evaluating precision of our model
true_positive = 0
for i in range(len(pred)):
    if pred[i] == y_train[i]:
        true_positive += 1
false_positive = len(y_train) - true_positive
print('Precision =', true_positive/(true_positive + false_positive))

# Saving Thetas in .txt file
np.savetxt('Theta1.txt', Theta1, delimiter=' ')
np.savetxt('Theta2.txt', Theta2, delimiter=' ')
```

```
Test Set Accuracy: 97.420000
Training Set Accuracy: 99.405000
Precision = 0.99405
```

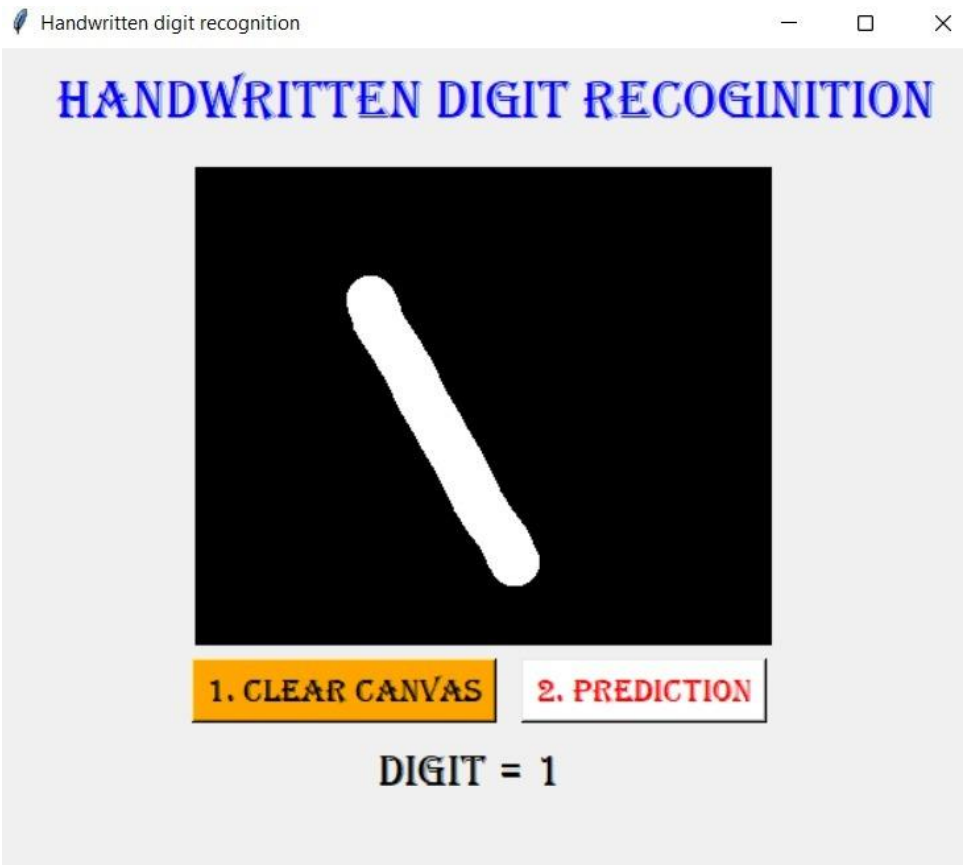
GUI:

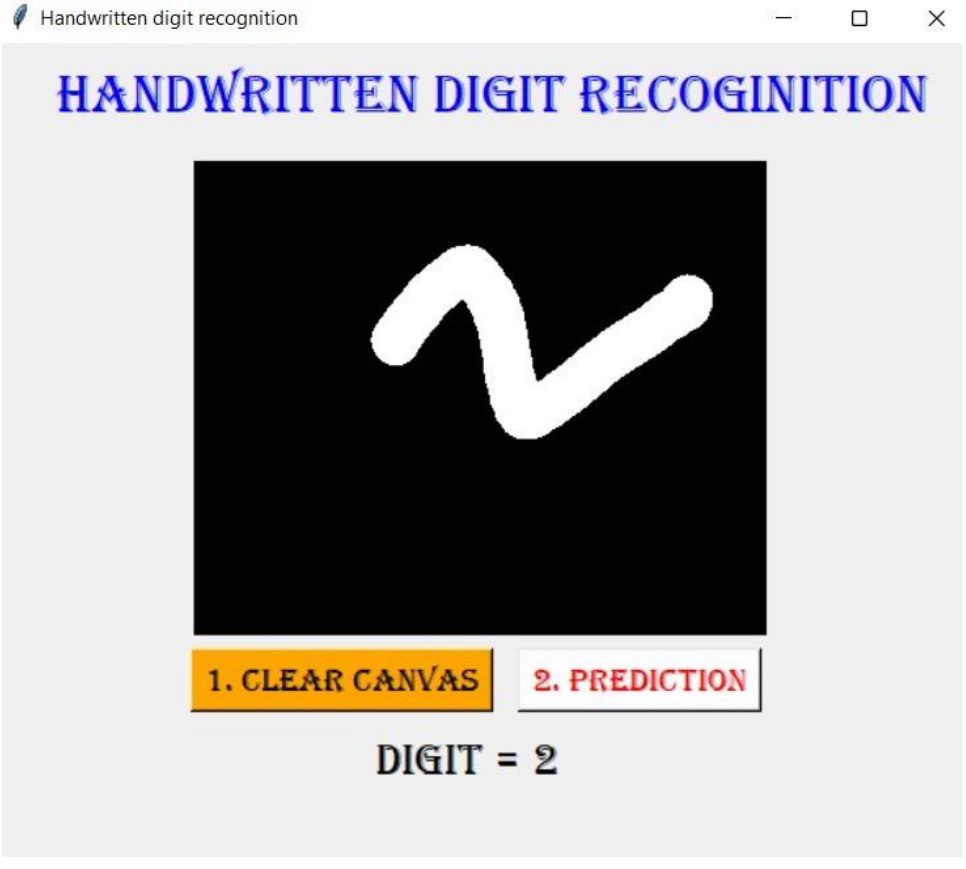
In [23]:

```
from tkinter import *
import numpy as np
from PIL import ImageGrab
from tkinter import filedialog
from tkinter.filedialog import askopenfile
from PIL import Image, ImageTk
window = Tk()
window.title("Handwritten digit recognition")
l1 = Label()


def MyProject():
    global l1

    widget = cv
    # Setting co-ordinates of canvas
    x = window.winfo_rootx() + widget.winfo_x()
    y = window.winfo_rooty() + widget.winfo_y()
    x1 = x + widget.winfo_width()
    y1 = y + widget.winfo_height()

    # Image is captured from canvas and is resized to (28 X 28) px
    img = ImageGrab.grab().crop((x, y, x1, y1)).resize((28, 28))

    # Converting rgb to grayscale image
    img = img.convert('L')

    # Extracting pixel matrix of image and converting it to a vector of (1, 784)
    x = np.asarray(img)
    vec = np.zeros((1, 784))
    k = 0
    for i in range(28):
        for j in range(28):
            vec[0][k] = x[i][j]
            k += 1
```

```
    # Loading thetas
    Theta1 = np.loadtxt('Theta1.txt')
    Theta2 = np.loadtxt('Theta2.txt')
    # Calling function for prediction
    pred = predict(Theta1, Theta2, vec / 255)
    # Displaying the result
    l1 = Label(window, text="Digit = " + str(pred[0]), font=('Algerian', 20))
    l1.place(x=230, y=420)
lastx, lasty = None, None
# Clears the canvas
def clear_widget():
    global cv, l1
    cv.delete("all")
    l1.destroy()
# Activate canvas
def event_activation(event):
    global lastx, lasty
    cv.bind('<B1-Motion>', draw_lines)
    lastx, lasty = event.x, event.y
# To draw on canvas
def draw_lines(event):
    global lastx, lasty
    x, y = event.x, event.y
    cv.create_line((lastx, lasty, x, y), width=30, fill='white', capstyle=ROUND, smooth=TRUE, splinesteps=12)
    lastx, lasty = x, y
# Label
L1 = Label(window, text="Handwritten Digit Recoginition", font=('Algerian', 25), fg="blue")
L1.place(x=35, y=10)
# Button to clear canvas
b1 = Button(window, text="1. Clear Canvas", font=('Algerian', 15), bg="orange", fg="black", command=clear_widget)
b1.place(x=120, y=370)
# Button to predict digit drawn on canvas
b2 = Button(window, text="2. Prediction", font=('Algerian', 15), bg="white", fg="red", command=MyProject)
b2.place(x=320, y=370)S
# Setting properties of canvas
cv = Canvas(window, width=350, height=290, bg='black')
cv.place(x=120, y=70)
cv.bind('<Button-1>', event_activation)
window.geometry("600x500")
window.mainloop()
```

OUTPUT:


Handwritten digit recognition
HANDWRITTEN DIGIT RECOGINITION
1. CLEAR CANVAS
2. PREDICTION
DIGIT = 1


Handwritten digit recognition
HANDWRITTEN DIGIT RECOGINITION
1. CLEAR CANVAS
2. PREDICTION
DIGIT = 2

Support Vector Machine

Here the problem is in the field of pattern recognition and we have to do is ***Handwritten digit recognition***.

Suppose that you have images of handwritten digits ranging from 0-9 written by various people in boxes of a specific size - similar to the application forms in banks and universities.

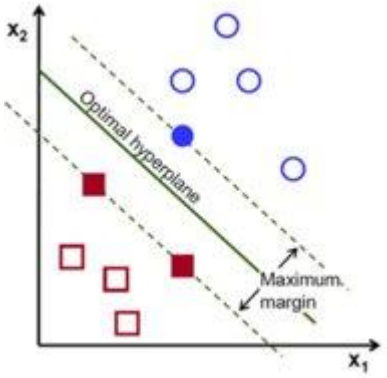
The goal is to develop a model that can correctly identify the digit (between 0-9) written in an image.

So here we should develop a model using Support Vector Machine which should correctly classify the handwritten digits from 0-9 based on the pixel values given as features.

For this problem, we use a large database of handwritten digits. The 'pixel values' of each digit (image) comprise the features, and the actual number between 0-9 is the label.

Since each image is of 28 x 28 pixels, and each pixel forms a feature, there are 784 features. Before the popularity of neural networks, though, models such as SVMs and boosted trees were the state-of-the-art in such problems.

THEORY:



A Support Vector Machine(SVM) is a discriminative classifier which intakes training data and the algorithm outputs an optimal hyperplane which categorizes the data.

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

Here for classification,

for $i = 1, \ldots, M$.

Least squares version to the SVM classifier by formulating the classification problem as

$$\min_{\omega,b,e} J_3(\omega, b, e) = \frac{1}{2}\omega^T\omega + \gamma\frac{1}{2}\sum_{k=1}^{N} e_k^2$$

Subject to the equality constraints.

$$y_k[w^T\phi(x_k) + b] = 1 - e_k, k = 1, \ldots, N.$$

Lagrangian is defined as

$$L_3(\omega, b, e, \alpha) = J_3(\omega, b, e) - \sum_{k=1}^{N} \alpha_k \{y_k[\omega^T \phi(x_k) + b] - 1 + e_k\}$$

Conditions for optimality

$$\begin{cases} \frac{\partial \mathcal{L}_3}{\partial w} = 0 \rightarrow w = \sum_{k=1}^{N} \alpha_k y_k \varphi(x_k) \\ \frac{\partial \mathcal{L}_3}{\partial b} = 0 \rightarrow \sum_{k=1}^{N} \alpha_k y_k = 0 \\ \frac{\partial \mathcal{L}_3}{\partial e_k} = 0 \rightarrow \alpha_k = \gamma e_k, \qquad k = 1, \ldots, N \\ \frac{\partial \mathcal{L}_3}{\partial \alpha_k} = 0 \rightarrow y_k[w^T \varphi(x_k) + b] - 1 + e_k = 0, \quad k = 1, \ldots, N \end{cases}$$

The above equations can be written as the solution for the following set of Linear Equations

$$\left[\begin{array}{ccc|c} I & 0 & 0 & -Z^T \\ 0 & 0 & 0 & -Y^T \\ 0 & 0 & \gamma I & -I \\ \hline Z & Y & I & 0 \end{array}\right] \left[\begin{array}{c} w \\ b \\ e \\ \hline \alpha \end{array}\right] = \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ \hline \vec{1} \end{array}\right]$$

$$\begin{aligned} Z &= [\phi(x_1)^T y_1; \ldots \phi(x_N)^T y_N] \\ Y &= [y_1; \ldots; y_N] \\ \vec{1} &= [1; \ldots 1] \\ e &= [e_1, \ldots e_N], \\ \alpha &= [\alpha_1, \ldots \alpha_N] \end{aligned}$$

The solution is given by,

$$\left[\begin{array}{c|c} 0 & -Y^T \\ \hline Y & ZZ^T + \gamma^{-1} I \end{array}\right] \left[\begin{array}{c} b \\ \hline \alpha \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline \vec{1} \end{array}\right].$$

After applying Mercer's Solution to the matrix Ω = ZZ^T

$$\begin{aligned} \Omega_{kl} &= y_k y_l \, \varphi(x_k)^T \varphi(x_l) \\ &= y_k y_l \, \Psi(x_k, x_l). \end{aligned}$$

Hence the classifier is found by solving the linear set of equations instead of quadratic programming and The support values αk are proportional to the errors at the data points

CODE :

```
import pandas as pd
import numpy as np
import SVM as Model
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

```
def KernelFunction(X1,X2,sigma):
    # np.exp((-1)*(np.linalg.norm(X1-X2))/(2*(sigma**2)))  ## RBF Kernel Function
    # np.dot(X1,X2)                                        ## Liner Kernel function
    return np.exp((-1)*(np.linalg.norm(X1-X2))/(2*(sigma**2)))

def KernelMatrix(Data,Y,NoOfDataSets,si):
    K = np.ones((NoOfDataSets,NoOfDataSets))
    for i in range(0,NoOfDataSets):
        for j in range(0,NoOfDataSets):
            K[i,j] = Y[i,0]*Y[j,0]*KernelFunction(Data[i,0:],Data[j,0:],si)
    return K

def SVMPara(Data,Y,C,NoOfDataSets,sigma=0):
    UpperMatrix = np.concatenate((np.array([[0]]),(-1)*Y.transpose()),axis = 1)
    LowerMatrix = np.concatenate((Y,KernelMatrix(Data,Y,NoOfDataSets,sigma) + ((1/C)*np.eye(NoOfDataSets))),axis = 1)
    A = np.concatenate((UpperMatrix,LowerMatrix),axis = 0)
    B = (np.concatenate((np.array([[0]]),np.array([np.ones(NoOfDataSets)])),axis = 1)).transpose()
    return np.linalg.inv(A)@B
```

```
def Weight(w,X_train,Y_train):
    return np.concatenate((np.array([[w[0,0]]]),
                           np.transpose(np.array([np.sum(np.multiply(np.multiply(Y_train,w[1:]),X_train),axis=0)]))),axis=0)
```

For Digit 0:

```
Dataset_0 = (pd.read_excel('Data.xlsx', sheet_name=0).to_numpy())[:100,:]
Dataset_1 = (pd.read_excel('Data.xlsx', sheet_name=1).to_numpy())[:10,:]
Dataset_2 = (pd.read_excel('Data.xlsx', sheet_name=2).to_numpy())[:10,:]
Dataset_3 = (pd.read_excel('Data.xlsx', sheet_name=3).to_numpy())[:10,:]
Dataset_4 = (pd.read_excel('Data.xlsx', sheet_name=4).to_numpy())[:10,:]
Dataset_5 = (pd.read_excel('Data.xlsx', sheet_name=5).to_numpy())[:10,:]
Dataset_6 = (pd.read_excel('Data.xlsx', sheet_name=6).to_numpy())[:10,:]
Dataset_7 = (pd.read_excel('Data.xlsx', sheet_name=7).to_numpy())[:10,:]
Dataset_8 = (pd.read_excel('Data.xlsx', sheet_name=8).to_numpy())[:10,:]
Dataset_9 = (pd.read_excel('Data.xlsx', sheet_name=9).to_numpy())[:10,:]

Dataset = np.concatenate((Dataset_0,np.concatenate((Dataset_1,np.concatenate((Dataset_2,np.concatenate((Dataset_3,np.concatenat
```

```
Y_0 = np.array([np.concatenate(((1)*np.ones(Dataset_0.shape[0]),np.concatenate(((-1)*np.ones(Dataset_1.shape[0]),np.concatenate(

C = [0.01,0.1,1,10,100]
Sigma = [0.001,0.01,0.1,1,10]

for c in C:
    for s in Sigma:
        X_train, X_test, Y_train, Y_test = train_test_split(Dataset[:,1:],Y_0,test_size=0.3, random_state=364)

        Alpha = SVMPara(X_train,Y_train,c,X_train.shape[0],s)
        w = Weight(Alpha,X_train,Y_train)

        X_test = np.concatenate((np.ones((X_test.shape[0],1)),X_test), axis=1)
        y = np.sign(X_test@w)
        Acc = np.trace(confusion_matrix(Y_test,y))/len(Y_test)
        print(f"C: {c}, Sigma: {s}, Accuracy: {Acc}")
```

(For Digit 0 the above code will be used and for rest of the digits same code will be used (so not including here) and the change will be in the "Y_(digit)" array so accordingly output is shown for every digit as you can see below)

For Digit 0:

```
C: 0.01, Sigma: 0.001, Accuracy: 0.6491228070175439
C: 0.01, Sigma: 0.01, Accuracy: 0.6491228070175439
C: 0.01, Sigma: 0.1, Accuracy: 0.6491228070175439
C: 0.01, Sigma: 1, Accuracy: 0.6491228070175439
C: 0.01, Sigma: 10, Accuracy: 0.6491228070175439
C: 0.1, Sigma: 0.001, Accuracy: 0.6491228070175439
C: 0.1, Sigma: 0.01, Accuracy: 0.6491228070175439
C: 0.1, Sigma: 0.1, Accuracy: 0.6491228070175439
C: 0.1, Sigma: 1, Accuracy: 0.6491228070175439
C: 0.1, Sigma: 10, Accuracy: 0.6491228070175439
C: 1, Sigma: 0.001, Accuracy: 0.6491228070175439
C: 1, Sigma: 0.01, Accuracy: 0.6491228070175439
C: 1, Sigma: 0.1, Accuracy: 0.6491228070175439
C: 1, Sigma: 1, Accuracy: 0.6491228070175439
C: 1, Sigma: 10, Accuracy: 0.6491228070175439
C: 10, Sigma: 0.001, Accuracy: 0.6491228070175439
C: 10, Sigma: 0.01, Accuracy: 0.6491228070175439
C: 10, Sigma: 0.1, Accuracy: 0.6491228070175439
C: 10, Sigma: 1, Accuracy: 0.6491228070175439
C: 10, Sigma: 10, Accuracy: 0.6491228070175439
C: 100, Sigma: 0.001, Accuracy: 0.6491228070175439
C: 100, Sigma: 0.01, Accuracy: 0.6491228070175439
C: 100, Sigma: 0.1, Accuracy: 0.6491228070175439
C: 100, Sigma: 1, Accuracy: 0.6491228070175439
C: 100, Sigma: 10, Accuracy: 0.6491228070175439
```

For Digit 1:

```
C: 0.01, Sigma: 0.001, Accuracy: 0.8596491228070176
C: 0.01, Sigma: 0.01, Accuracy: 0.8596491228070176
C: 0.01, Sigma: 0.1, Accuracy: 0.8596491228070176
C: 0.01, Sigma: 1, Accuracy: 0.8596491228070176
C: 0.01, Sigma: 10, Accuracy: 0.8596491228070176
C: 0.1, Sigma: 0.001, Accuracy: 0.8596491228070176
C: 0.1, Sigma: 0.01, Accuracy: 0.8596491228070176
C: 0.1, Sigma: 0.1, Accuracy: 0.8596491228070176
C: 0.1, Sigma: 1, Accuracy: 0.8596491228070176
C: 0.1, Sigma: 10, Accuracy: 0.8596491228070176
C: 1, Sigma: 0.001, Accuracy: 0.8596491228070176
C: 1, Sigma: 0.01, Accuracy: 0.8596491228070176
C: 1, Sigma: 0.1, Accuracy: 0.8596491228070176
C: 1, Sigma: 1, Accuracy: 0.8596491228070176
C: 1, Sigma: 10, Accuracy: 0.8421052631578947
C: 10, Sigma: 0.001, Accuracy: 0.8596491228070176
C: 10, Sigma: 0.01, Accuracy: 0.8596491228070176
C: 10, Sigma: 0.1, Accuracy: 0.8596491228070176
C: 10, Sigma: 1, Accuracy: 0.8596491228070176
C: 10, Sigma: 10, Accuracy: 0.8421052631578947
C: 100, Sigma: 0.001, Accuracy: 0.8596491228070176
C: 100, Sigma: 0.01, Accuracy: 0.8596491228070176
C: 100, Sigma: 0.1, Accuracy: 0.8596491228070176
C: 100, Sigma: 1, Accuracy: 0.8596491228070176
C: 100, Sigma: 10, Accuracy: 0.8245614035087719
```

For Digit 2:

```
C: 0.01, Sigma: 0.001, Accuracy: 0.7719298245614035
C: 0.01, Sigma: 0.01, Accuracy: 0.7719298245614035
C: 0.01, Sigma: 0.1, Accuracy: 0.7719298245614035
C: 0.01, Sigma: 1, Accuracy: 0.7719298245614035
C: 0.01, Sigma: 10, Accuracy: 0.7719298245614035
C: 0.1, Sigma: 0.001, Accuracy: 0.7719298245614035
C: 0.1, Sigma: 0.01, Accuracy: 0.7719298245614035
C: 0.1, Sigma: 0.1, Accuracy: 0.7719298245614035
C: 0.1, Sigma: 1, Accuracy: 0.7719298245614035
C: 0.1, Sigma: 10, Accuracy: 0.7719298245614035
C: 1, Sigma: 0.001, Accuracy: 0.7719298245614035
C: 1, Sigma: 0.01, Accuracy: 0.7719298245614035
C: 1, Sigma: 0.1, Accuracy: 0.7719298245614035
C: 1, Sigma: 1, Accuracy: 0.7719298245614035
C: 1, Sigma: 10, Accuracy: 0.7719298245614035
C: 10, Sigma: 0.001, Accuracy: 0.7719298245614035
C: 10, Sigma: 0.01, Accuracy: 0.7719298245614035
C: 10, Sigma: 0.1, Accuracy: 0.7719298245614035
C: 10, Sigma: 1, Accuracy: 0.7719298245614035
C: 10, Sigma: 10, Accuracy: 0.7719298245614035
C: 100, Sigma: 0.001, Accuracy: 0.7719298245614035
C: 100, Sigma: 0.01, Accuracy: 0.7719298245614035
C: 100, Sigma: 0.1, Accuracy: 0.7719298245614035
C: 100, Sigma: 1, Accuracy: 0.7719298245614035
C: 100, Sigma: 10, Accuracy: 0.7719298245614035
```

For Digit 3:

```
C: 0.01, Sigma: 0.001, Accuracy: 0.7719298245614035
C: 0.01, Sigma: 0.01, Accuracy: 0.7719298245614035
C: 0.01, Sigma: 0.1, Accuracy: 0.7719298245614035
C: 0.01, Sigma: 1, Accuracy: 0.7719298245614035
C: 0.01, Sigma: 10, Accuracy: 0.7719298245614035
C: 0.1, Sigma: 0.001, Accuracy: 0.7719298245614035
C: 0.1, Sigma: 0.01, Accuracy: 0.7719298245614035
C: 0.1, Sigma: 0.1, Accuracy: 0.7719298245614035
C: 0.1, Sigma: 1, Accuracy: 0.7719298245614035
C: 0.1, Sigma: 10, Accuracy: 0.7719298245614035
C: 1, Sigma: 0.001, Accuracy: 0.7719298245614035
C: 1, Sigma: 0.01, Accuracy: 0.7719298245614035
C: 1, Sigma: 0.1, Accuracy: 0.7719298245614035
C: 1, Sigma: 1, Accuracy: 0.7719298245614035
C: 1, Sigma: 10, Accuracy: 0.7719298245614035
C: 10, Sigma: 0.001, Accuracy: 0.7719298245614035
C: 10, Sigma: 0.01, Accuracy: 0.7719298245614035
C: 10, Sigma: 0.1, Accuracy: 0.7719298245614035
C: 10, Sigma: 1, Accuracy: 0.7719298245614035
C: 10, Sigma: 10, Accuracy: 0.7719298245614035
C: 100, Sigma: 0.001, Accuracy: 0.7719298245614035
C: 100, Sigma: 0.01, Accuracy: 0.7719298245614035
C: 100, Sigma: 0.1, Accuracy: 0.7719298245614035
C: 100, Sigma: 1, Accuracy: 0.7719298245614035
C: 100, Sigma: 10, Accuracy: 0.7719298245614035
```

For Digit 4:

```
C: 0.01, Sigma: 0.001, Accuracy: 0.5263157894736842
C: 0.01, Sigma: 0.01, Accuracy: 0.5263157894736842
C: 0.01, Sigma: 0.1, Accuracy: 0.5263157894736842
C: 0.01, Sigma: 1, Accuracy: 0.5263157894736842
C: 0.01, Sigma: 10, Accuracy: 0.5263157894736842
C: 0.1, Sigma: 0.001, Accuracy: 0.5263157894736842
C: 0.1, Sigma: 0.01, Accuracy: 0.5263157894736842
C: 0.1, Sigma: 0.1, Accuracy: 0.5263157894736842
C: 0.1, Sigma: 1, Accuracy: 0.5263157894736842
C: 0.1, Sigma: 10, Accuracy: 0.5263157894736842
C: 1, Sigma: 0.001, Accuracy: 0.5263157894736842
C: 1, Sigma: 0.01, Accuracy: 0.5263157894736842
C: 1, Sigma: 0.1, Accuracy: 0.5263157894736842
C: 1, Sigma: 1, Accuracy: 0.5263157894736842
C: 1, Sigma: 10, Accuracy: 0.5263157894736842
C: 10, Sigma: 0.001, Accuracy: 0.5263157894736842
C: 10, Sigma: 0.01, Accuracy: 0.5263157894736842
C: 10, Sigma: 0.1, Accuracy: 0.5263157894736842
C: 10, Sigma: 1, Accuracy: 0.5263157894736842
C: 10, Sigma: 10, Accuracy: 0.5263157894736842
C: 100, Sigma: 0.001, Accuracy: 0.5263157894736842
C: 100, Sigma: 0.01, Accuracy: 0.5263157894736842
C: 100, Sigma: 0.1, Accuracy: 0.5263157894736842
C: 100, Sigma: 1, Accuracy: 0.5263157894736842
C: 100, Sigma: 10, Accuracy: 0.5263157894736842
```

For Digit 5:

```
C: 0.01, Sigma: 0.001, Accuracy: 0.631578947368421
C: 0.01, Sigma: 0.01, Accuracy: 0.631578947368421
C: 0.01, Sigma: 0.1, Accuracy: 0.631578947368421
C: 0.01, Sigma: 1, Accuracy: 0.631578947368421
C: 0.01, Sigma: 10, Accuracy: 0.631578947368421
C: 0.1, Sigma: 0.001, Accuracy: 0.631578947368421
C: 0.1, Sigma: 0.01, Accuracy: 0.631578947368421
C: 0.1, Sigma: 0.1, Accuracy: 0.631578947368421
C: 0.1, Sigma: 1, Accuracy: 0.631578947368421
C: 0.1, Sigma: 10, Accuracy: 0.631578947368421
C: 1, Sigma: 0.001, Accuracy: 0.631578947368421
C: 1, Sigma: 0.01, Accuracy: 0.631578947368421
C: 1, Sigma: 0.1, Accuracy: 0.631578947368421
C: 1, Sigma: 1, Accuracy: 0.631578947368421
C: 1, Sigma: 10, Accuracy: 0.631578947368421
C: 10, Sigma: 0.001, Accuracy: 0.631578947368421
C: 10, Sigma: 0.01, Accuracy: 0.631578947368421
C: 10, Sigma: 0.1, Accuracy: 0.631578947368421
C: 10, Sigma: 1, Accuracy: 0.631578947368421
C: 10, Sigma: 10, Accuracy: 0.6140350877192983
C: 100, Sigma: 0.001, Accuracy: 0.631578947368421
C: 100, Sigma: 0.01, Accuracy: 0.631578947368421
C: 100, Sigma: 0.1, Accuracy: 0.631578947368421
C: 100, Sigma: 1, Accuracy: 0.631578947368421
C: 100, Sigma: 10, Accuracy: 0.6140350877192983
```

For Digit 6:

```
C: 0.01, Sigma: 0.001, Accuracy: 0.49122807017543857
C: 0.01, Sigma: 0.01, Accuracy: 0.49122807017543857
C: 0.01, Sigma: 0.1, Accuracy: 0.49122807017543857
C: 0.01, Sigma: 1, Accuracy: 0.49122807017543857
C: 0.01, Sigma: 10, Accuracy: 0.49122807017543857
C: 0.1, Sigma: 0.001, Accuracy: 0.49122807017543857
C: 0.1, Sigma: 0.01, Accuracy: 0.49122807017543857
C: 0.1, Sigma: 0.1, Accuracy: 0.49122807017543857
C: 0.1, Sigma: 1, Accuracy: 0.49122807017543857
C: 0.1, Sigma: 10, Accuracy: 0.49122807017543857
C: 1, Sigma: 0.001, Accuracy: 0.49122807017543857
C: 1, Sigma: 0.01, Accuracy: 0.49122807017543857
C: 1, Sigma: 0.1, Accuracy: 0.49122807017543857
C: 1, Sigma: 1, Accuracy: 0.49122807017543857
C: 1, Sigma: 10, Accuracy: 0.49122807017543857
C: 10, Sigma: 0.001, Accuracy: 0.49122807017543857
C: 10, Sigma: 0.01, Accuracy: 0.49122807017543857
C: 10, Sigma: 0.1, Accuracy: 0.49122807017543857
C: 10, Sigma: 1, Accuracy: 0.49122807017543857
C: 10, Sigma: 10, Accuracy: 0.49122807017543857
C: 100, Sigma: 0.001, Accuracy: 0.49122807017543857
C: 100, Sigma: 0.01, Accuracy: 0.49122807017543857
C: 100, Sigma: 0.1, Accuracy: 0.49122807017543857
C: 100, Sigma: 1, Accuracy: 0.49122807017543857
C: 100, Sigma: 10, Accuracy: 0.49122807017543857
```

For Digit 7:

```
C: 0.01, Sigma: 0.001, Accuracy: 0.6491228070175439
C: 0.01, Sigma: 0.01, Accuracy: 0.6491228070175439
C: 0.01, Sigma: 0.1, Accuracy: 0.6491228070175439
C: 0.01, Sigma: 1, Accuracy: 0.6491228070175439
C: 0.01, Sigma: 10, Accuracy: 0.6491228070175439
C: 0.1, Sigma: 0.001, Accuracy: 0.6491228070175439
C: 0.1, Sigma: 0.01, Accuracy: 0.6491228070175439
C: 0.1, Sigma: 0.1, Accuracy: 0.6491228070175439
C: 0.1, Sigma: 1, Accuracy: 0.6491228070175439
C: 0.1, Sigma: 10, Accuracy: 0.6491228070175439
C: 1, Sigma: 0.001, Accuracy: 0.6491228070175439
C: 1, Sigma: 0.01, Accuracy: 0.6491228070175439
C: 1, Sigma: 0.1, Accuracy: 0.6491228070175439
C: 1, Sigma: 1, Accuracy: 0.6491228070175439
C: 1, Sigma: 10, Accuracy: 0.6491228070175439
C: 10, Sigma: 0.001, Accuracy: 0.6491228070175439
C: 10, Sigma: 0.01, Accuracy: 0.6491228070175439
C: 10, Sigma: 0.1, Accuracy: 0.6491228070175439
C: 10, Sigma: 1, Accuracy: 0.6491228070175439
C: 10, Sigma: 10, Accuracy: 0.6491228070175439
C: 100, Sigma: 0.001, Accuracy: 0.6491228070175439
C: 100, Sigma: 0.01, Accuracy: 0.6491228070175439
C: 100, Sigma: 0.1, Accuracy: 0.6491228070175439
C: 100, Sigma: 1, Accuracy: 0.6491228070175439
C: 100, Sigma: 10, Accuracy: 0.6491228070175439
```

For Digit 8:

```
C: 0.01, Sigma: 0.001, Accuracy: 0.45614035087719296
C: 0.01, Sigma: 0.01, Accuracy: 0.45614035087719296
C: 0.01, Sigma: 0.1, Accuracy: 0.45614035087719296
C: 0.01, Sigma: 1, Accuracy: 0.45614035087719296
C: 0.01, Sigma: 10, Accuracy: 0.45614035087719296
C: 0.1, Sigma: 0.001, Accuracy: 0.45614035087719296
C: 0.1, Sigma: 0.01, Accuracy: 0.45614035087719296
C: 0.1, Sigma: 0.1, Accuracy: 0.45614035087719296
C: 0.1, Sigma: 1, Accuracy: 0.45614035087719296
C: 0.1, Sigma: 10, Accuracy: 0.45614035087719296
C: 1, Sigma: 0.001, Accuracy: 0.45614035087719296
C: 1, Sigma: 0.01, Accuracy: 0.45614035087719296
C: 1, Sigma: 0.1, Accuracy: 0.45614035087719296
C: 1, Sigma: 1, Accuracy: 0.45614035087719296
C: 1, Sigma: 10, Accuracy: 0.45614035087719296
C: 10, Sigma: 0.001, Accuracy: 0.45614035087719296
C: 10, Sigma: 0.01, Accuracy: 0.45614035087719296
C: 10, Sigma: 0.1, Accuracy: 0.45614035087719296
C: 10, Sigma: 1, Accuracy: 0.45614035087719296
C: 10, Sigma: 10, Accuracy: 0.45614035087719296
C: 100, Sigma: 0.001, Accuracy: 0.45614035087719296
C: 100, Sigma: 0.01, Accuracy: 0.45614035087719296
C: 100, Sigma: 0.1, Accuracy: 0.45614035087719296
C: 100, Sigma: 1, Accuracy: 0.45614035087719296
C: 100, Sigma: 10, Accuracy: 0.45614035087719296
```

For Digit 9:

```
C: 0.01, Sigma: 0.001, Accuracy: 0.6140350877192983
C: 0.01, Sigma: 0.01, Accuracy: 0.6140350877192983
C: 0.01, Sigma: 0.1, Accuracy: 0.6140350877192983
C: 0.01, Sigma: 1, Accuracy: 0.6140350877192983
C: 0.01, Sigma: 10, Accuracy: 0.6140350877192983
C: 0.1, Sigma: 0.001, Accuracy: 0.6140350877192983
C: 0.1, Sigma: 0.01, Accuracy: 0.6140350877192983
C: 0.1, Sigma: 0.1, Accuracy: 0.6140350877192983
C: 0.1, Sigma: 1, Accuracy: 0.6140350877192983
C: 0.1, Sigma: 10, Accuracy: 0.6140350877192983
C: 1, Sigma: 0.001, Accuracy: 0.6140350877192983
C: 1, Sigma: 0.01, Accuracy: 0.6140350877192983
C: 1, Sigma: 0.1, Accuracy: 0.6140350877192983
C: 1, Sigma: 1, Accuracy: 0.6140350877192983
C: 1, Sigma: 10, Accuracy: 0.6140350877192983
C: 10, Sigma: 0.001, Accuracy: 0.6140350877192983
C: 10, Sigma: 0.01, Accuracy: 0.6140350877192983
C: 10, Sigma: 0.1, Accuracy: 0.6140350877192983
C: 10, Sigma: 1, Accuracy: 0.6140350877192983
C: 10, Sigma: 10, Accuracy: 0.6140350877192983
C: 100, Sigma: 0.001, Accuracy: 0.6140350877192983
C: 100, Sigma: 0.01, Accuracy: 0.6140350877192983
C: 100, Sigma: 0.1, Accuracy: 0.6140350877192983
C: 100, Sigma: 1, Accuracy: 0.6140350877192983
C: 100, Sigma: 10, Accuracy: 0.6140350877192983
```

Convolution Neural Network:

Convolutional Neural Network (CNN) is a Deep learning algorithm which can learn important features in the image which helps in identifying and differentiating between images. The architecture of the CNN resembles the connectivity patterns of neurons in human brain.

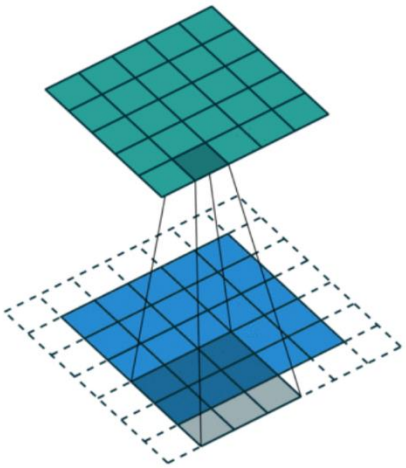
The CNN consists of two important process which are as follows,

- Features Learning
- Classification

Features Learning:

Convolution Layer:

The operation of convolution is used to extract high level features and low level features.



Pooling Layer:

This is to decrease the computational power required to process the data through dimensionality reduction.

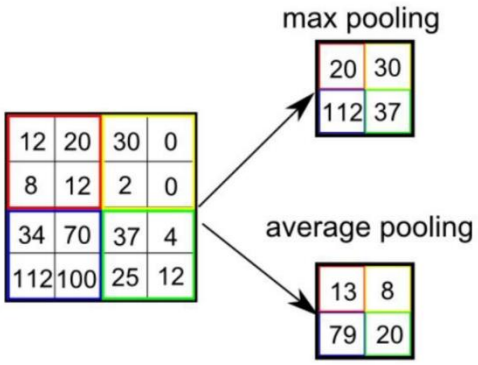
There are two types of Pooling:

Max Pooling:

Max Pooling returns the maximum value from the portion of the image covered by the Kernel.

Average Pooling:

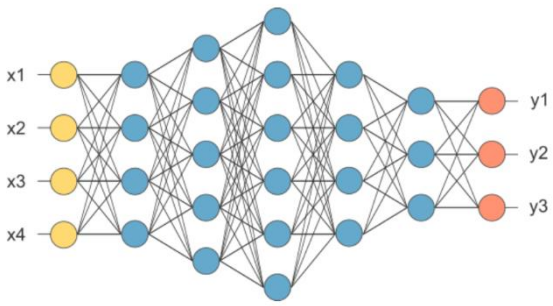
Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.



CLASSIFICATION:

Fully Connected Layer:

The matrix is flattened into vector and is feed into fully connected layer called neural network.

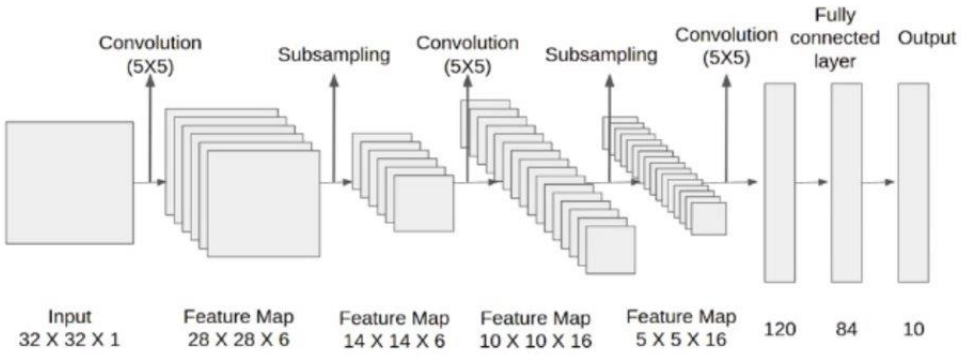


LENET – 5

Lenet-5 is one of the earliest pre-trained models proposed by Yann LeCun and others in the year 1998, in the research paper Gradient-Based Learning Applied

to Document Recognition. They used this architecture for recognizing the handwritten and machine-printed characters.

Architecture of the Lenet-5 model



Layer	# filters / neurons	Filter size	Stride	Size of feature map
Input	-	-	-	32 X 32 X 1
Conv 1	6	5 * 5	1	28 X 28 X 6
Avg. pooling 1		2 * 2	2	14 X 14 X 6
Conv 2	16	5 * 5	1	10 X 10 X 16
Avg. pooling 2		2 * 2	2	5 X 5 X 16
Conv 3	120	5 * 5	1	120
Fully Connected 1	-	-	-	84
Fully Connected 2	-	-	-	10

CODE:

LENET-5 MODEL

```
model = tf.keras.Sequential()
```

[5] ✓ 0.4s

```
model.add(Conv2D(6, kernel_size=(5,5), input_shape = (28,28,1), activation='relu', padding='same', name="C1"))
model.add(Conv2D(6, (2,2), strides=2, name="S2", activation='relu'))
# the dropout rate is a hyper parameter which can be tuned.
model.add(Dropout(0.4))
model.add(Conv2D(16, (5,5), activation='relu', name='C3'))
model.add(Conv2D(16, (2,2), strides=2, activation='relu',name="S4"))
model.add(Conv2D(120, (5,5), activation='relu', name="C5"))
model.add(Flatten())
model.add(Dense(84, activation='relu', name='F6'))
model.add(Dense(10, activation='softmax', name='OUTPUT'))
```

[6] ✓ 0.1s

+ Code + Markdown

Layer (type)	Output Shape	Param #
C1 (Conv2D)	(None, 28, 28, 6)	156
S2 (Conv2D)	(None, 14, 14, 6)	150
dropout (Dropout)	(None, 14, 14, 6)	0
C3 (Conv2D)	(None, 10, 10, 16)	2416
S4 (Conv2D)	(None, 5, 5, 16)	1040
C5 (Conv2D)	(None, 1, 1, 120)	48120
flatten (Flatten)	(None, 120)	0
F6 (Dense)	(None, 84)	10164
OUTPUT (Dense)	(None, 10)	850

Total params: 62,896
Trainable params: 62,896
Non-trainable params: 0

TRAINING THE MODEL AND TESTING

```
history = model.fit(trainX_rescale, trainY_onehot, epochs=5, validation_data=(testX_rescale, testY_onehot), verbose=1,
```

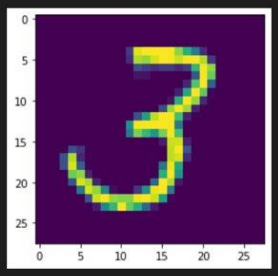
[9] ✓ 1m 3.5s

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 14s 232us/sample - loss: 0.3474 - accuracy: 0.8897 - val_loss: 0.0955 -
val_accuracy: 0.9703
Epoch 2/5
60000/60000 [==============================] - 14s 225us/sample - loss: 0.1196 - accuracy: 0.9631 - val_loss: 0.0651 -
val_accuracy: 0.9787
Epoch 3/5
60000/60000 [==============================] - 12s 200us/sample - loss: 0.0865 - accuracy: 0.9734 - val_loss: 0.0628 -
val_accuracy: 0.9795
Epoch 4/5
60000/60000 [==============================] - 12s 204us/sample - loss: 0.0607 - accuracy: 0.9813 - val_loss: 0.0468 -
val_accuracy: 0.9848
Epoch 5/5
60000/60000 [==============================] - 12s 196us/sample - loss: 0.0534 - accuracy: 0.9836 - val_loss: 0.0401 -
val_accuracy: 0.9863
```

PREDICTION

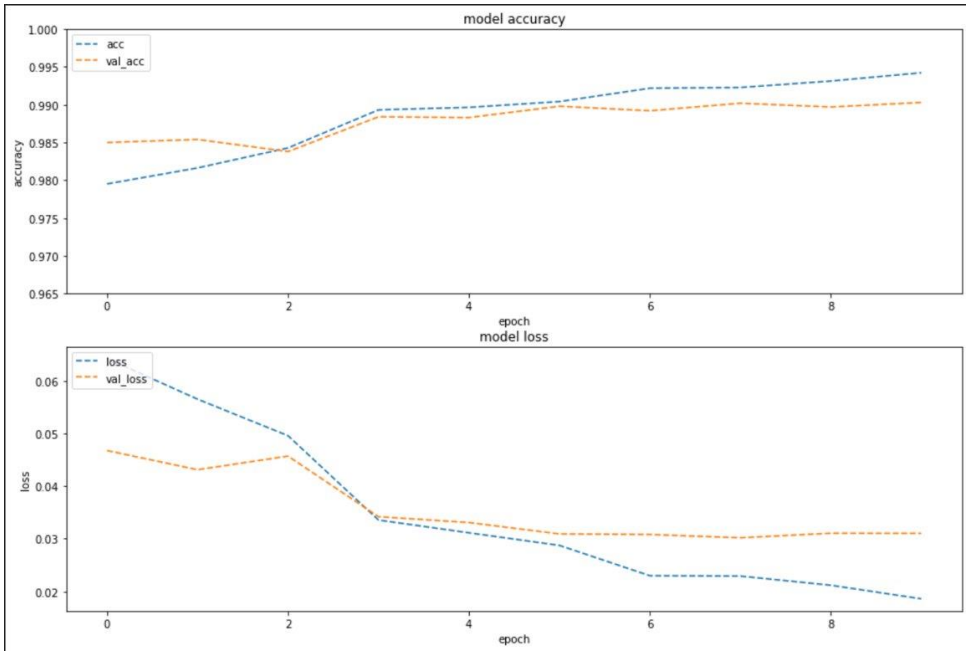
```
Data = testX_rescale[np.random.randint(10000, size=1),:,:,:]
reshaped_image = Data.reshape((28, 28))
plt.imshow(reshaped_image)
plt.show()

p = model.predict(Data)
p_class = np.argmax(p,axis=1)
print(f"Prediction: {p_class}")
```

[19] ✓ 0.1s



Prediction: [3]

GRAPH:



The final CNN Accuracy: 98.6 (for 10 epoch cycles run)

Conclusion :

The Accuracy of the algorithms were:

- KNN – 89%
- Random Forest – 90%

- Neural Networks – 97.42 %
- CNN – 98 %
- SVM – 63.9 % (average)

Since the accuracy of CNN is maximum it is the most suitable algorithm among KNN ,Random forest , Neural networks , SVM which can be used for digit recognition.

GUI

A GUI was made in python using the tkinter package. It allows the user to select the image from the system or to draw in a canvas to identify the number .

It also allows the user to specify the algorithm for identification.

Code :

```
from tkinter import *                                   # importing necessary packages
from tkinter import ttk, filedialog
from PIL import ImageTk, Image , ImageGrab
from sklearn import svm
import pandas as pd
import dig_KNN
import cv2
import numpy as np
from sklearn import preprocessing
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
last_x , last_y = None,None                             # used to save x and y co-ordinates of mouse in canvas
img_no = 0                                              # number of image to be saved ( used in name )

def show():                                             # function to show the image
    global img,panel
    img = Image.open(path)                              # open the image using it's path
    # img = Image.open("pc5.png")
    img = img.resize((int(img.width/6),int(img.height/6)),Image.ANTIALIAS)  # resize and print it as a label
    img = ImageTk.PhotoImage(img)
    panel = Label(root, image = img)
    panel.place(x=startx+465,y=starty+50)

def choose():                                           # function to select the image
    global path
    path = filedialog.askopenfilename(title="Choose The Image") # open the directory interface and allow the user to choose
                                                        # the image and get the path
```

```python
def preprocess(img):  # preprocess the image to match our algorithm
    k = img
    k = cv2.cvtColor(k, cv2.COLOR_BGR2GRAY)  # coloured to grayscale image
    k = cv2.bitwise_not(cv2.threshold(k, 100, 255, cv2.THRESH_BINARY)[1])  # modifying the image to match our dataset
    resized = cv2.resize(k, [28, 28], interpolation=cv2.INTER_AREA)  # resize to 28x28 size image
    img = np.asarray(resized).reshape(-1)  # returning the pixel array
    return img


def compute():  # function to compute the number
    img = cv2.imread(path)  # read the image
    global Knn, Rf, NN, svm_c
    chosen = List.get()  # get the selected algorithm
    if Knn != None:  # refresh the screen ( remove old recognitions )
        Knn.place_forget()
        Rf.place_forget()
        NN.place_forget()
        svm_c.place_forget()

    colour = "Orange"
    back_colour = "Green"
```

```python
    # Label for all the Algorithm's output
    Knn = Label(root, text="KNN says the number is :", bg="black", fg=colour, font='sans 9 bold')
    Rf = Label(root, text="Rf says the number is :", bg="black", fg=colour, font='sans 9 bold')
    NN = Label(root, text="NN says the number is :", bg="black", fg=colour, font='sans 9 bold')
    svm_c = Label(root, text="Svm says the number is :", bg="black", fg=colour, font='sans 9 bold')

    if chosen == "KNN":  # Call the chosen algorithm's function
        identified = dig_KNN.classifier(img)
        Knn.config(text=Knn["text"] + "\n\t\t\t" + str(identified))
        Knn.place(x=startx, y=starty + 120)

    if chosen == "Random Forest":
        img = preprocess(img)
        rf = RandomForestClassifier(n_estimators=125, max_depth=100, min_samples_split=20)
        rf.fit(X, Y)
        identified = rf.predict(img.reshape(1, -1))
        Rf.config(text=Rf["text"] + "\n\t\t\t" + str(identified))
        Rf.place(x=startx, y=starty + 120)


    if chosen == "SVM":
        img = preprocess(img)
        Svm = svm.SVC()
        Svm.fit(X, Y)
        identified = Svm.predict(img.reshape(1, -1))
        svm_c.config(text=svm_c["text"] + "\n\t\t\t" + str(identified))
        svm_c.place(x=startx, y=starty + 120)

    if chosen == "NN":
        img = preprocess(img)
        Nn = MLPClassifier(solver='lbfgs', alpha=0.1, hidden_layer_sizes=(20,), random_state=2, max_iter=30000)
        Nn.fit(X, Y)
        nn = Nn.predict(img.reshape(1, -1))
        NN.config(text=NN["text"] + "\n\t\t\t" + str(nn))
        NN.place(x=startx, y=starty + 120)
```

```
def resize_background(event):  # function to resize the background image when
    # the screen's size is changed
    im = cop_im.resize((event.width, event.height))  # resize the background image
    bg_img = ImageTk.PhotoImage(im)  # read the resized image and make it as the background image
    mL.config(image=bg_img)
    mL.image = bg_img


def clear_all():  # function to clear the canvas
    global can
    can.delete("all")


def activate(event):  # function to activate the canvas for drawing
    global last_x, last_y
    can.bind('<B1-Motion>', draw)
    last_x, last_y = event.x, event.y



def draw(event):  # function to draw the lines when the cursor is used to draw on the canvas
    global last_x, last_y
    x, y = event.x, event.y  # get the current x,y co-ordinates of the cursor

    # draw the line from previous recorded x,y co-ordinates (last_x,last_y) to the current x,y co-ordinates (x,y)
    can.create_line((last_x, last_y, x, y), width=8, fill='red', smooth=TRUE, capstyle=ROUND, splinesteps=12)

    # change the last x,y co-ordinates
    last_x, last_y = x, y



def Recognize():  # function to get the image in the canvas
    global img_no
    predict = []
    file = f'image_{img_no}.png'  # name for the image to be saved
    widget = can  # select the canvas as widget

    # get the x,y co-ordinates of the image to be captured ( position of the canavs in the screen )
    x = new.winfo_x() + widget.winfo_x() + root.winfo_x()
    y = new.winfo_y() + widget.winfo_y() + root.winfo_y() + 20
    x1 = x + widget.winfo_width() + 60
    y1 = y + widget.winfo_height() + 20

    ImageGrab.grab().crop((x, y, x1, y1)).save(file)  # get the screen shot of the screen
    image = cv2.imread(f'image_{img_no}.png')  # save the image ( optional )

    # preprocess the captured image for identification
    g = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret, th_im = cv2.threshold(g, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

    # find the contours in the image ( corresponding to the numbers )
    contours = cv2.findContours(th_im, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]

    # for every number in the image
```

```
    # for every number in the image
    for cnt in contours:
        # get the image of the number alone
        x, y, w, h = cv2.boundingRect(cnt)
        cv2.rectangle(image, (x - 2, y - 2), (x + w + 2, y + h + 2), (255, 0, 0), 1)
        top = int(0.05 * th_im.shape[0])
        bottom = top
        left = int(0.05 * th_im.shape[1])
        right = left

        # crop the image of the number
        th_up = cv2.copyMakeBorder(th_im, top, bottom, left, right, cv2.BORDER_REPLICATE)
        roi = th_up[y - top:y + h + bottom, x - left:x + w + right]
        imga = cv2.resize(roi, (28, 28), interpolation=cv2.INTER_AREA)
        imga = np.asarray(imga).reshape(-1)

        # send the processed image into the classifier
        Svm = svm.SVC()
        Svm.fit(X, Y)
        identified = Svm.predict(imga.reshape(1, -1))
        pr = Label(new, text="The number is : " + str(identified))
        pr.place(x=new.winfo_width() / 2, y=new.winfo_height() / 2)
        # get the number

            pred = identified

            # print the number over the image ( optional )
            text = str(pred)
            font = cv2.FONT_HERSHEY_SIMPLEX
            fontscale = 0.5
            colour = (20, 25, 40)
            thickness = 4
            cv2.putText(image, text, (x, y - 5), font, fontscale, colour, thickness)



def open_new():  # function to open the canvas
    global new, can
    new = Toplevel(root)  # create a new level of screen for canavs
    new.title("Hello !")
    can = Canvas(new, width=200, height=200, bg='white')  # creating the canvas
    can.pack()

    can.bind('<Button-1>', activate)  # button for capturing the drawing and recognizing
    save = Button(new, text="Save and Identify", command=Recognize)
    save.pack()

    clear = Button(new, text="Clear Screen", command=clear_all)  # button for clearing the screen
    clear.pack()
```

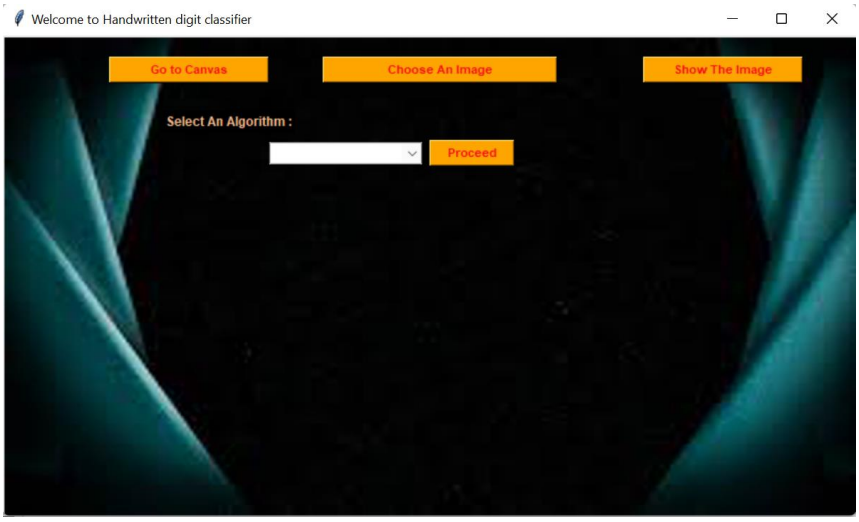
```
D = pd.read_csv("train_data.csv")  # read the training data and get the required values
D = D.values[:10000]
X = D[:, 1:]
Y = D[:, 0]

# create the screen for GUI
root = Tk()
root.geometry("800x450")
root.title("Welcome to Handwritten digit classifier")

# set the background image and option for it to resize automatically
im = Image.open("backgrounds/bg8.jfif")
cop_im = im.copy()
bg_img = ImageTk.PhotoImage(im)
mL = ttk.Label(root, image=bg_img)
mL.bind('<Configure>', resize_background)
mL.pack(fill=BOTH, expand=YES)
```

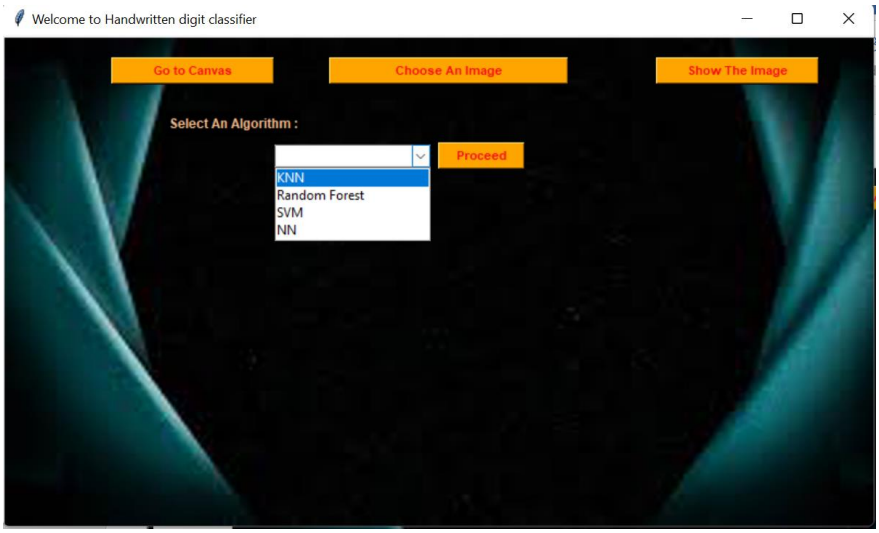
```
# algorithm options
options = ["KNN", "Random Forest", "SVM", "NN"]

# variables used to place widgets in the screen
starty = 20
startx = 150

# instruction label to select the algorithm
Instruct = Label(root, text="Select An Algorithm :", bg="black", fg="#F8B88B", font='sans 9 bold')
Instruct.place(x=startx, y=starty + 50)

# used to see if Knn label is declared atleast once
Knn = None

# combobox for selecting the algorithm
List = ttk.Combobox(root, value=options)
List.place(x=100 + startx, y=starty + 80)

# Button to choose the image
Choose = Button(root, text="Choose An Image", command=lambda: choose(), bg="orange", fg="#F70D1A", font='sans 8 bold',
                width=30)
Choose.place(x=150 + startx, y=starty)
```

```
# Button to proceed to identification of the number
Proceed = Button(root, text="Proceed", command=lambda: compute(), bg="orange", fg="#F70D1A", font='sans 8 bold',
                 width=10)
Proceed.place(x=250 + startx, y=starty + 78)

# Button to show the image
Show = Button(root, text="Show The Image", command=lambda: show(), bg="orange", fg="#F70D1A", font='sans 8 bold',
              width=20)
Show.place(x=startx + 450, y=starty)

# Button to open the canvas
Canv = Button(root, text="Go to Canvas", command=lambda: open_new(), bg="orange", fg="#F70D1A", font='sans 8 bold',
              width=20)
Canv.place(x=startx - 50, y=starty)

root.mainloop()
```

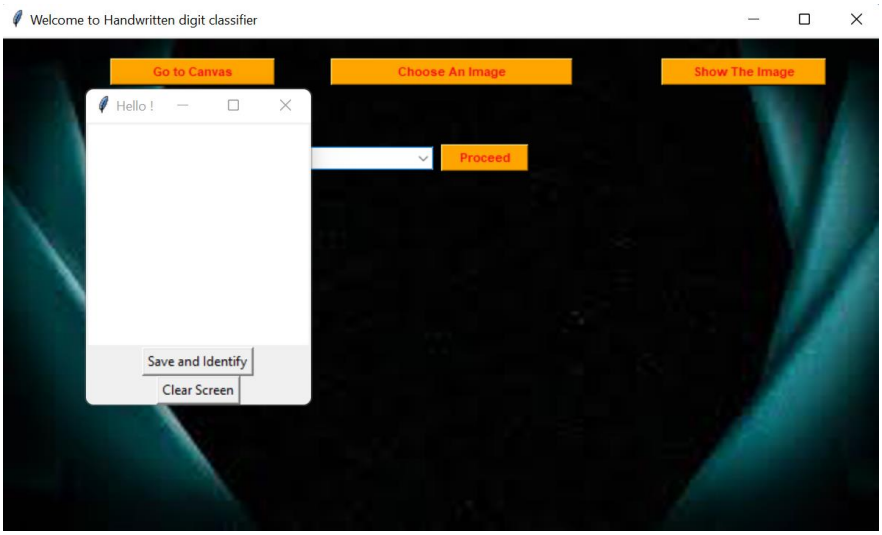
GUI :

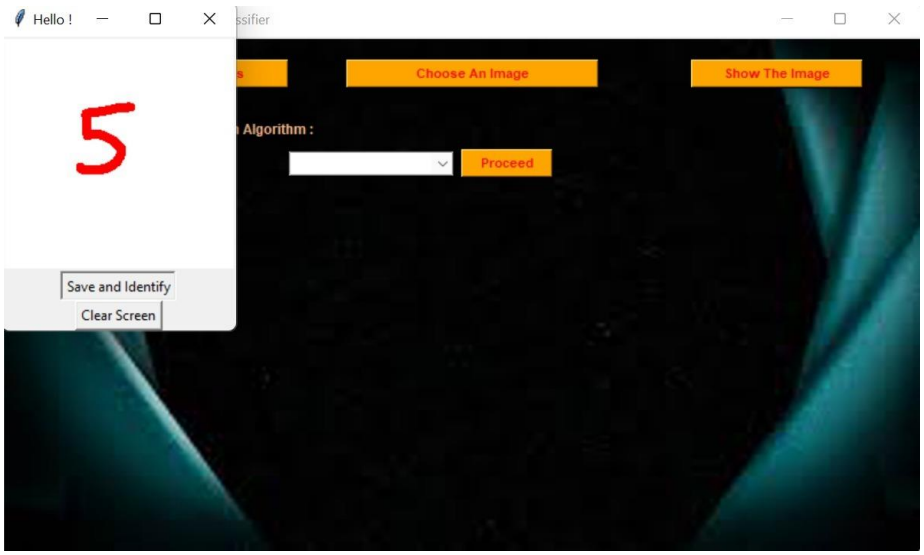
Screen

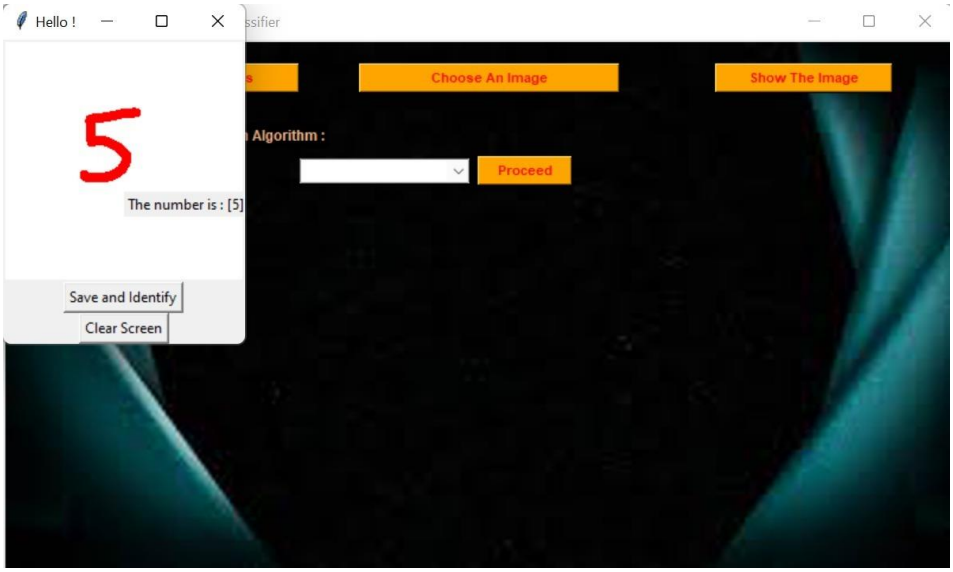

Welcome to Handwritten digit classifier
Go to Canvas
Choose An Image
Show The Image
Select An Algorithm :
Proceed


Welcome to Handwritten digit classifier
Go to Canvas
Choose An Image
Show The Image
Select An Algorithm :
Proceed
KNN
Random Forest
SVM
NN

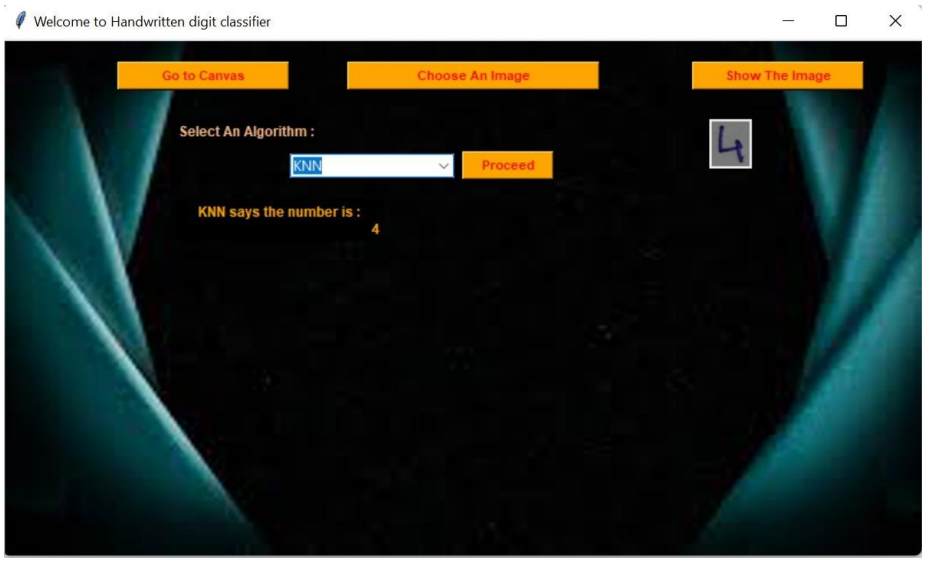
Canvas:

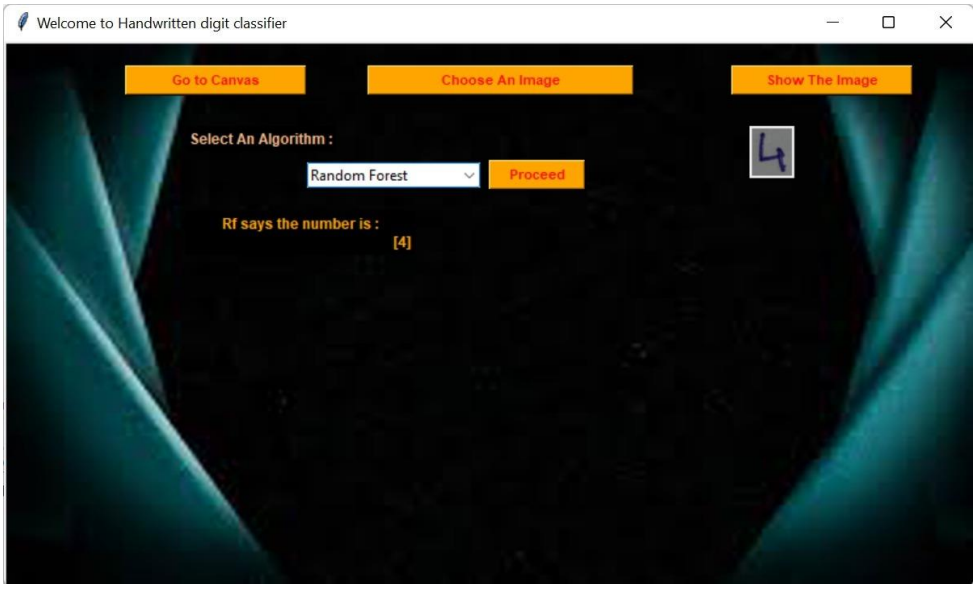

Welcome to Handwritten digit classifier
Go to Canvas
Choose An Image
Show The Image
Hello !
Proceed
Save and Identify
Clear Screen

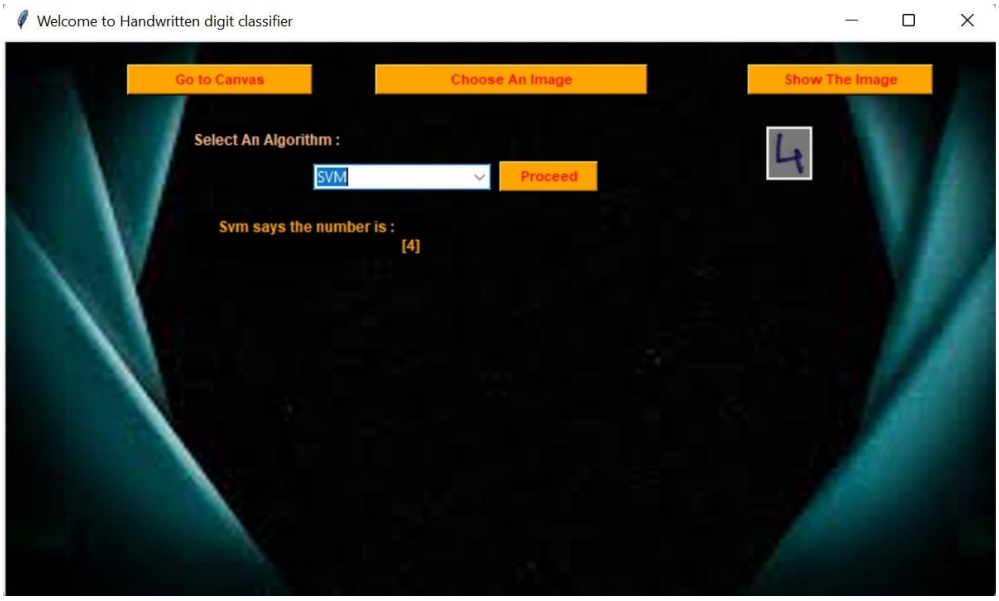




Identifying from an predefined image:







Implementation :

Digit recognition can be used in identifying digits in forms or cheques.

A program for getting the images of digits from cheques was coded in python.

This program first identifies the biggest contour (the image of the cheque) in the image of a cheque (with background), after identifying the biggest contour the image is cropped according to the biggest contour and then is resized to the necessary dimension.

Once the scaling and resizing have been done the position of dates and amount of money is the same for all cheque images (since the position of these sections within the cheque is the same). The position was identified and was divided into eight equal parts (each for date and amount of money) since all the digits are given equal space.

Code:

```
import cv2                                                  # importing necessary packages
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

widthImg=800                                                # width and height of image to be resized
heightImg =400

r = 2000 # no. of train data
D = pd.read_csv("train_data.csv")                           # reading the train data and getting the necessary datas
D = D.values[:10000]
X = D[:,1:]
Y = D[:,0]
```

```
def preProcessing(img):  # preprocess the image to find the biggest contour
    img_Gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # coloured to gray image
    img_Blur = cv2.GaussianBlur(img_Gray, (5, 5), 1)  # blurring thr image
    img_Canny = cv2.Canny(img_Blur, 200, 200)  # noise reduction , dialation
    imgDial = cv2.dilate(img_Canny, np.ones((5, 5)), iterations=2)
    imgThres = cv2.erode(imgDial, np.ones((5, 5)), iterations=1)
    return imgThres
```

```
def getContours(img):
    biggest_contour = np.array([])  # list to store the position of the contour
    maxArea = 0  # used to store of areas of contours
    contours, _ = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)  # get the contours in the image

    for cnt in contours:  # traverse through all contour values
        area = cv2.contourArea(cnt)  # find its area
        if area > 5000:
            # if its area > 5000 and is also the greatest contour ( in terms of area ) replace biggest with the
            # current contour positions
            perimeter = cv2.arcLength(cnt, True)
            cnt_current = cv2.approxPolyDP(cnt, 0.02 * perimeter, True)
            if area > maxArea and len(cnt_current) == 4:
                biggest_contour = cnt_current
                maxArea = area
    cv2.drawContours(imgContour, biggest_contour, -1, (255, 0, 0), 20)

    # return the biggest the contour
    return biggest_contour


def reorder(myPoints):
    # reorder the position of the contour such that the image is cropped correctly
    # since the first point has the least sum and the last point in the set of co-ordinates has the maximum sum
    # and in the second set x co-ordinate - y co-ordinate will be lowest compared to that of third set
    # rearranging according to the condition
    if type(myPoints) != list:
        myPoints = myPoints.reshape((4, 2))
    myPointsNew = np.zeros((4, 1, 2), np.int32)
    add = myPoints.sum(1)
    myPointsNew[0] = myPoints[np.argmin(add)]
    myPointsNew[3] = myPoints[np.argmax(add)]
    diff = np.diff(myPoints, axis=1)
    myPointsNew[1] = myPoints[np.argmin(diff)]
    myPointsNew[2] = myPoints[np.argmax(diff)]
    return myPointsNew
```

```python
def getWarp(img, biggest_contour):  # crop around the biggest contour ( representing the cheque )
    biggest_contour = reorder(biggest_contour)  # reorder the set of co-ordinates
    pts1 = np.float32(biggest_contour)  # convert the list values to float type
    pts2 = np.float32([[0, 0], [widthImg, 0], [0, heightImg], [widthImg, heightImg]])  # co-ordinates of the total image
    matrix = cv2.getPerspectiveTransform(pts1, pts2)  # change the perspective
    imgOutput = cv2.warpPerspective(img, matrix, (widthImg, heightImg))  # warping
    imgCropped = imgOutput[20:imgOutput.shape[0] - 20, 20:imgOutput.shape[1] - 20]  # get the cropped image
    imgCropped = cv2.resize(imgCropped, (widthImg, heightImg))  # resize to required dimensions

    # return the cropped image
    return imgCropped


def preprocess(img):  # process the image ( to detect the number )
    k = img  # processing as explained previously
    k = cv2.cvtColor(k, cv2.COLOR_BGR2GRAY)
    k = cv2.bitwise_not(cv2.threshold(k, 100, 255, cv2.THRESH_BINARY)[1])
    resized = cv2.resize(k, [28, 28], interpolation=cv2.INTER_AREA)
    img = np.asarray(resized).reshape(-1)
    return img


def get_info(img, info):  # function to get the date and money int he cheque

    rf = RandomForestClassifier(n_estimators=125, max_depth=100, min_samples_split=20)
    rf.fit(X, Y)
    # get the shape of the image
    h, w, c = img.shape
    nums = list()

    # if dat is being read
    if info == "date":
        for i in range(8):  # get all the images corresponding each number in the date

            # since all numbers are treated equally divide the date section to 8 column and get the image of all digits
            img_new = getWarp(img, np.array(
                [[2 + i * w / 8 + 2, h], [2 + i * w / 8 + 2, 2], [(i + 1) * w / 8 - 2, 2], [(i + 1) * w / 8 - 2, h]]))

            cv2.imwrite(f"dat{i}.jpg", img_new)  # write the image ( optional )
            img_new = preprocess(img_new)  # process the image and send it into the classifier
            num = rf.predict(img_new.reshape(1, -1))
            nums.append(num)  # add to the number to the date
        date = ""

        for j in range(8):
            if j == 2 or j == 4:  # change the numbers obtained to date using "," when needed
                date = date + "," + str(nums[j])
            else:
                date = date + str(nums[j])
        print("Dated for : ", date)

    elif info == "money":
        for i in range(8):  # get all the images corresponding each number in the amount of money
            if i != 7:
                # since all numbers are treated equally divide the date section to 8 column and get the image of all digits
                img_new = getWarp(img, np.array(
                    [[i * w / 3.4 * 0.4 + 2, h], [i * w / 3.4 * 0.4 + 2, 0], [(i + 1) * w / 3.4 * 0.4 - 4, 0],
                     [(i + 1) * w / 3.4 * 0.4 - 4, h]]))
            else:
                # for the last digit read till the end
                img_new = getWarp(img, np.array(
                    [[i * w / 3.4 * 0.4 + 2, h], [i * w / 3.4 * 0.4 + 2, 0], [w - 2, 0], [w - 2, h]]))
            cv2.imwrite(f"mon{i}.jpg", img_new)  # write the image ( optional )
            # process the image and send it into the classifier
            img_new = preprocess(img_new)
            num = rf.predict(img_new.reshape(1, -1))
```

```
            nums.append(num)  # add to the number to cost
        money = ""
        for j in range(7):
            if j == 2 or j == 4:
                money = money + "," + str(nums[j])  # change the numbers obtained to money using "," when needed
            else:
                money = money + str(nums[j])
        print("Money is : ", money)
```

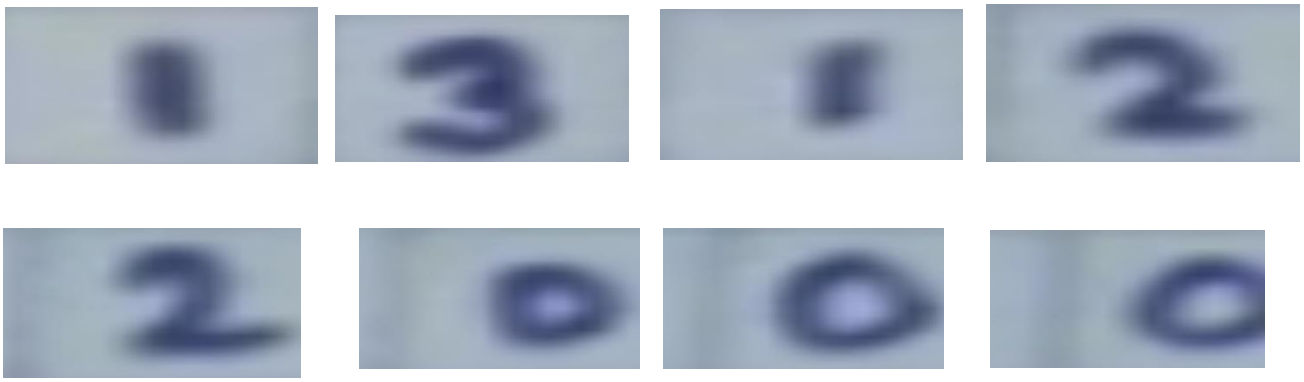
```
img = cv2.imread("c11.jpg")  # read the image of the cheque
img = cv2.resize(img, (widthImg, heightImg))  # resize the image
imgContour = img.copy()

imgThres = preProcessing(img)  # preprocess the image

# get the biggest contour in the image ( the points corresponds to the posititon of
# cheque in the screen since it will be the largest object in the screen )
biggest_contour = getContours(imgThres)

# crop the image to the cheque
imgWarped = getWarp(img, biggest_contour)
```

```
# if the image of the cheque is scaled to the specified size
# then the posititon of the date and money section is constant
# so the images can be cropped again and sent into the get_infos function to get the information
date_pic = getWarp(imgWarped, np.array([[615, 15], [615, 40], [777, 15], [777, 40]]))
cv2.imwrite("dates.png", date_pic)
# get_info(date_pic,"date")

money_pic = getWarp(imgWarped, np.array([[617, 171], [617, 138], [779, 139], [778, 174]]))
cv2.imwrite("mon.jpg", money_pic)
get_info(money_pic, "money")
money_pic = preprocess(money_pic)
```

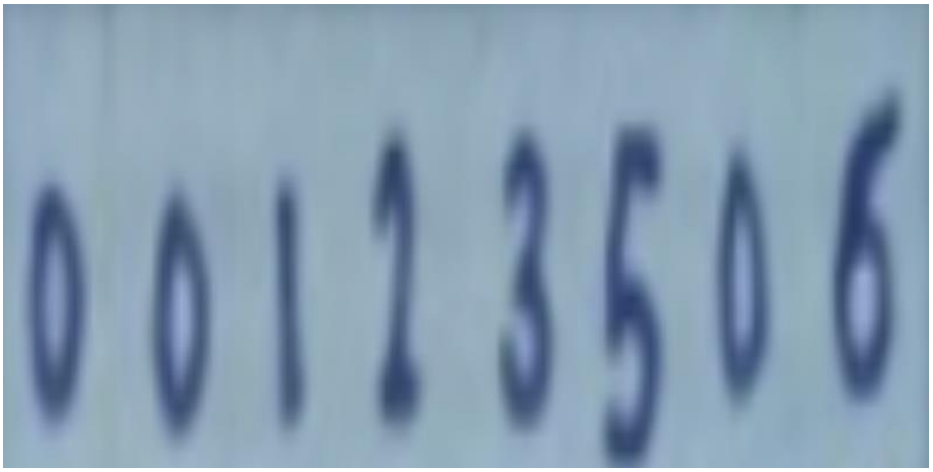
Output:

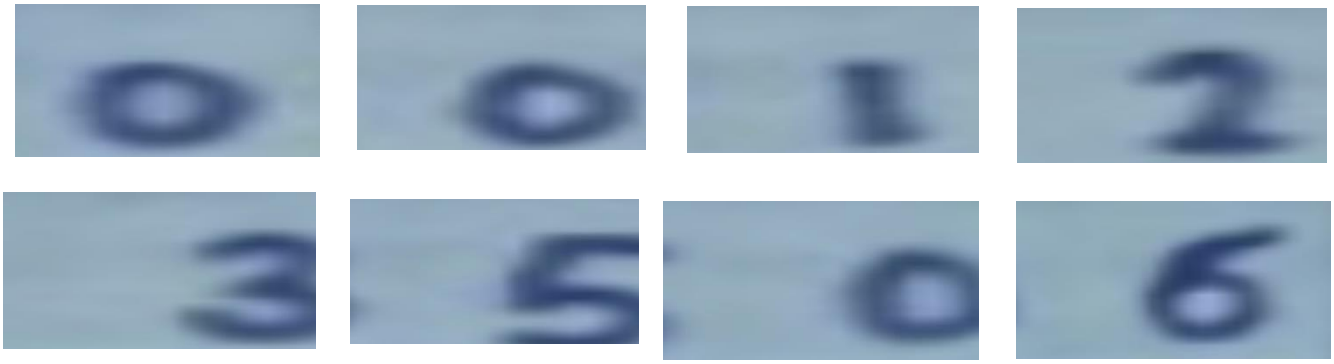
Cheque:

Saved Dates section:



Saved amount image:





These images can then be again sent into the classifiers to get the final images.

Thank You