Soft Computing (INT246)

Project Report on

Hand written digit recognition

Submitted To:

Ishan Kumar

Submitted By:

Shivam Kumar

11906335

RKM024A17

Contents

Introduction

Human can see and visually sense the world around them by using their eyes and brains. Computer visions work on enabling computers to see and process images on same way that human vision does. Hand written Digit Recognition is a popular problem of Artificial Neural Network in which the model has ability to recognize human written digit, we try to extract pattern from image and make a logic so that computer predict a digit value. Generally, we take a grey scale image of size 28 by 28 pixels. Size might vary but as size will increase complexity will also be increase because each pixel of image represents a characteristic. For example, 28 by 28-pixel image it has Seven Hundred and Eighty-Four features. In this project I have used Python pytorch module to build and train model which is based upon Conventional Neural Network. For training and testing of model I have used image of MNIST Database (Modified National Institutes of Standard Technology Database). The MNIST Database have Sixty thousand image for training and ten thousand images for testing. In this model I have divided Sixty thousand training data into two parts, first parts contain fifty thousand images for training and second parts contain ten thousand images for cross validation purpose.

Recently Convolutional Neural Networks (CNN) becomes one of the most appealing approaches and has been an ultimate factor in a variety of recent success and challenging machine learning applications such as challenge ImageNet object detection image segmentation and face recognition. Therefore, we choose CNN for our challenging tasks of image classification. We can use it for handwriting digits recognition which is one of high academic and business transactions. There are many applications of handwriting digit recognition in our real-life purposes. Precisely, we can use it in banks for reading checks, post offices for sorting letter, and many other related works. Convolutional neural networks are deep artificial neural networks. We can use it to classify images (e.g., name what they see), cluster them by similarity (photo search) and perform object recognition within scenes. It can be used to identify faces, individuals, street signs, tumors, platypuses and many other aspects of visual data. The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernals) which have a small receptive field but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product, and producing a 2-Dimensional activation mat of that filter. As a result, the network learns when they see some specific type of feature at some spatial position in the input. Then the activation maps are fed into a down sampling layer, and like convolutions, this method is applied one patch at a time. CNN has also fully connected layer that classifies output with one label per node.

The MNIST database (Modified National Institute of Standards and Technology database) is a handwritten digit's dataset. We can use it for training various image processing system. The database is also widely used for training and testing in the field of machine learning. It has 60,000 training and 10,000 testing examples. Each image has fixed size. The images are of size 28*28 pixels. It is a database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on pre-processing and formatting.

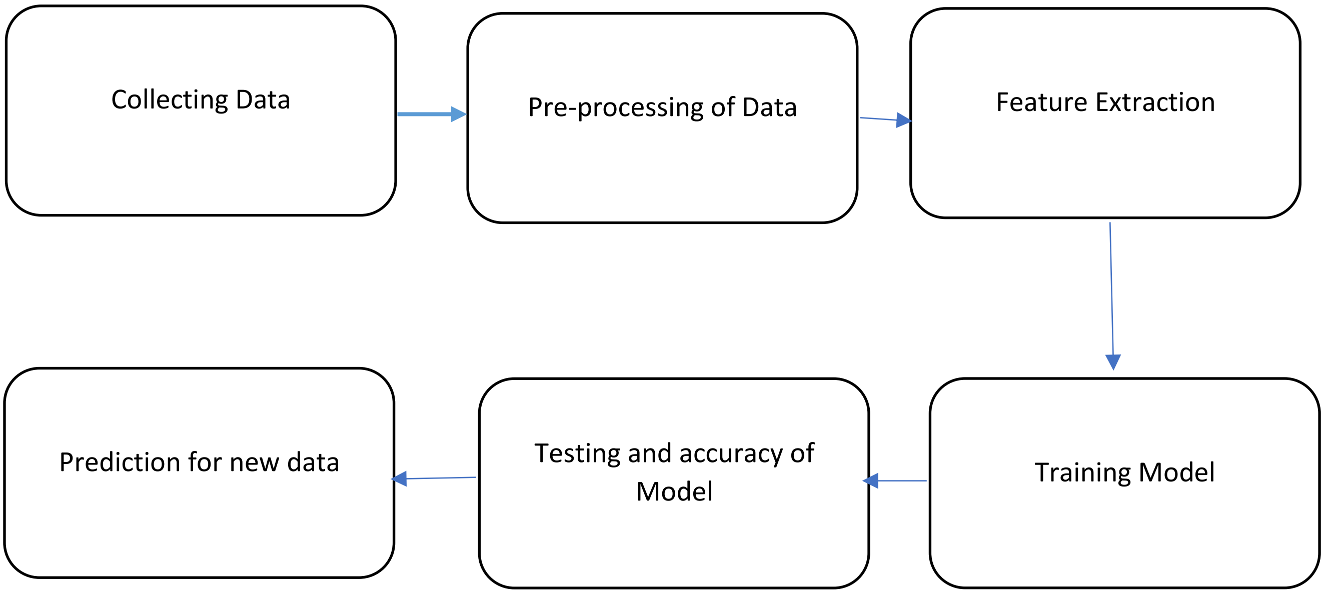
Student Work

I am alone working on this project. Following are steps taken by me during this project:

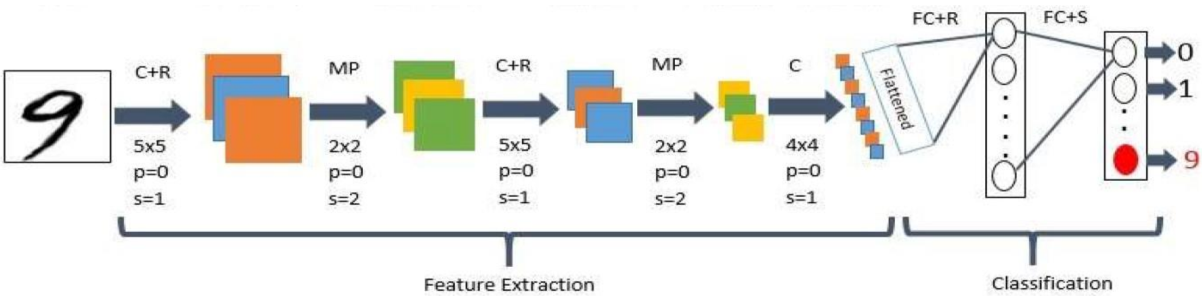
- Finding the topic of Project work
- Searching algorithms to be suitable for this project
- Understanding CNN
- Collecting Training and Testing Data
- Coding/Building Model on Jupyter NoteBook
- Preparing Project Report
- Preparing Power Point Presentation

Methodology

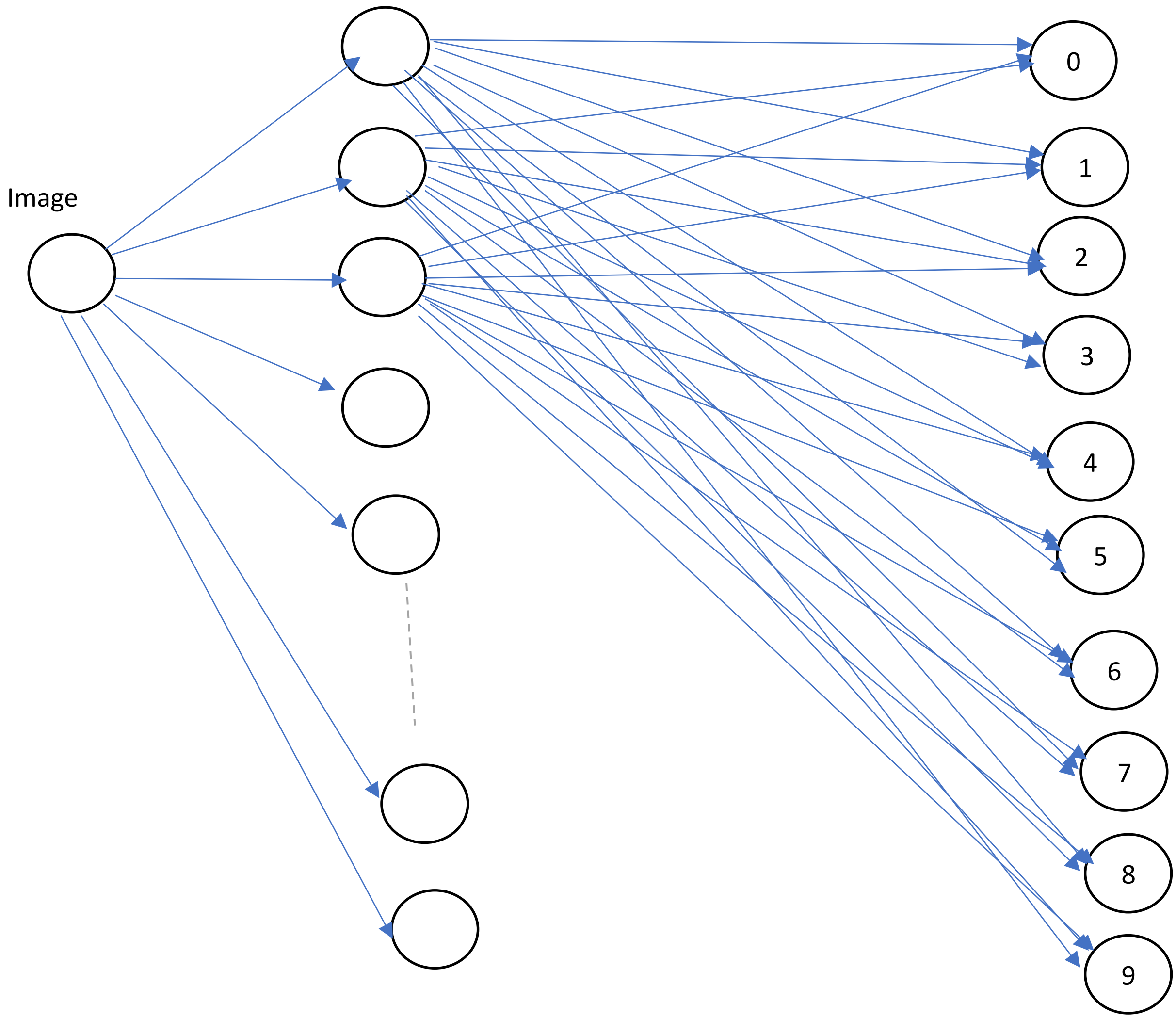
Flow Diagram



Architecture



Algorithm



1) img=image of 28*28 pixel
2) t= one dimensional array of size 10 corresponding to every image set probability equas to 1 for which that image belongs and 0 for all other cluster.
3) x= one dimensional array of img which is of size 784 it will work as x1,x2,……x784
4) w= two dimensional randomize array of shape (784,10) it represent weight of every x corresponding to every cluster there are 10 cluster 0 to 10.
5) b= one dimensional array of size 10. It represents bias value corresponding to every cluster.
6) temp=x*w+b
7) output=which cluster have max probability in temp
8) find error
9) update weight and bias
10) retrain

Result

Successfully model has been built with accuracy score 0.82. It is quite good working with MNIST image but not with other images.

Hand Written Digit Recognition Project

Table of Content

1. Importing Library

In [1]:

```
import torch
import torchvision
from torchvision.datasets import MNIST
import torchvision.transforms as transforms
from torch.utils.data import random_split
from torch.utils.data import DataLoader
import torch.nn.functional as F
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as im
%matplotlib inline
```

2.Loading Data and Preprocessing

In [2]:

```
d=MNIST(root='data/',download=True)
```

In [3]:

```
img,label=d[0]
```

In [4]:

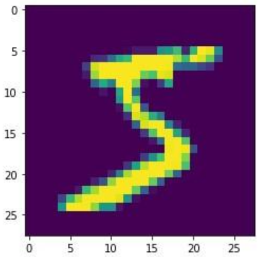
```
d[0]
```

Out[4]:

```
(<PIL.Image.Image image mode=L size=28x28 at 0x1F17D9E8400>, 5)
```

In [5]:
```
img
```
Out[5]:

In [6]:
```
plt.imshow(img)
```
Out[6]:
```
<matplotlib.image.AxesImage at 0x1f17dab1780>
```



In [7]:
```
dataset=MNIST(root='data/',train=True,transform=transforms.ToTensor())
```
In [8]:
```
test_dataset=MNIST(root='data/',train=False,transform=transforms.ToTensor())
```
In [9]:
```
train_ds,val_ds=random_split(dataset,[50000,10000])
len(train_ds),len(val_ds)
```
Out[9]:
```
(50000, 10000)
```
In [101]:
```
len(test_dataset)
```
Out[101]:
```
10000
```
In [10]:
```
batch_size=128
train_loader=DataLoader(train_ds,batch_size,shuffle=True)
val_loader=DataLoader(val_ds,batch_size)
```
In [11]:
```
input_size=28*28
num_classes=10
```

Traning

In [12]:
```
def accuracy(outputs,labels):
    _,preds=torch.max(outputs,dim=1)
    return torch.tensor(torch.sum(preds==labels).item()/len(preds))
```
In [13]:
```
class MnistModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear=nn.Linear(input_size,num_classes)

    def forward(self,xb):
        xb=xb.reshape(-1,784)
        out=self.linear(xb)
        return out

    def training_step(self,batch):
        images,labels=batch
        out=self(images)
        loss=F.cross_entropy(out,labels)
        return loss

    def validation_step(self,batch):
        images,labels=batch
        out=self(images)
        loss=F.cross_entropy(out,labels)
        acc=accuracy(out,labels)
        return {'val_loss':loss,'val_acc':acc}

    def validation_epoch_end(self,outputs):
        batch_losses=[x['val_loss'] for x in outputs]
        epoch_loss=torch.stack(batch_losses).mean()
        batch_accs=[x['val_acc'] for x in outputs]
        epoch_acc=torch.stack(batch_accs).mean()
        return {'val_loss':epoch_loss.item(),'val_acc':epoch_acc.item()}

    def epoch_end(self,epoch,result):
        print("Epoch [{}], val_loss: {:.4f}, val_acc: {:.4f}".format(epoch,result['val_loss'],result['val_acc']))

model=MnistModel()
```
In [ ]:
```

```

In [14]:
```
def fit(epochs,lr,model,train_loader,val_loader,opt_func=torch.optim.SGD):
    optimizer=opt_func(model.parameters(),lr)
    history=[]
    for epoch in range(epochs):
        for batch in train_loader:
            loss=model.training_step(batch)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

        result=evaluate(model,val_loader)
        model.epoch_end(epoch,result)
        history.append(result)
    return history
```

In [15]:
```
def evaluate(model,val_loader):
    outputs=[model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)
```

In [ ]:

In [16]:
```
result0=evaluate(model,val_loader)
result0
```

Out[16]:
```
{'val_loss': 2.333756923675537, 'val_acc': 0.08405854552984238}
```

In [17]:
```
history1=fit(5,0.001,model,train_loader,val_loader)
```

```
Epoch [0], val_loss: 1.9608, val_acc: 0.6368
Epoch [1], val_loss: 1.6868, val_acc: 0.7392
Epoch [2], val_loss: 1.4829, val_acc: 0.7697
Epoch [3], val_loss: 1.3295, val_acc: 0.7866
Epoch [4], val_loss: 1.2116, val_acc: 0.8001
```

In [18]:
```
history2=fit(5,0.001,model,train_loader,val_loader)
```

```
Epoch [0], val_loss: 1.1191, val_acc: 0.8084
Epoch [1], val_loss: 1.0451, val_acc: 0.8164
Epoch [2], val_loss: 0.9844, val_acc: 0.8223
Epoch [3], val_loss: 0.9339, val_acc: 0.8277
Epoch [4], val_loss: 0.8913, val_acc: 0.8314
```

In [19]:
```
history3=fit(5,0.001,model,train_loader,val_loader)
```

```
Epoch [0], val_loss: 0.8547, val_acc: 0.8340
Epoch [1], val_loss: 0.8231, val_acc: 0.8376
Epoch [2], val_loss: 0.7953, val_acc: 0.8409
Epoch [3], val_loss: 0.7709, val_acc: 0.8436
Epoch [4], val_loss: 0.7491, val_acc: 0.8459
```

In [20]:
```
history4=fit(5,0.001,model,train_loader,val_loader)
```

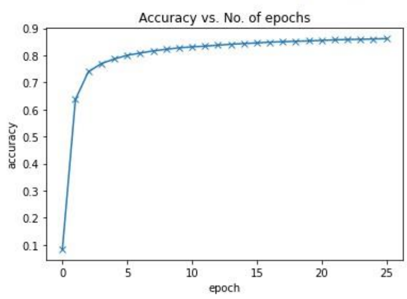
```
Epoch [0], val_loss: 0.7296, val_acc: 0.8486
Epoch [1], val_loss: 0.7119, val_acc: 0.8503
Epoch [2], val_loss: 0.6959, val_acc: 0.8520
Epoch [3], val_loss: 0.6814, val_acc: 0.8535
Epoch [4], val_loss: 0.6680, val_acc: 0.8549
```

In [21]:
```
history5=fit(5,0.001,model,train_loader,val_loader)
```

```
Epoch [0], val_loss: 0.6557, val_acc: 0.8573
Epoch [1], val_loss: 0.6443, val_acc: 0.8585
Epoch [2], val_loss: 0.6337, val_acc: 0.8593
Epoch [3], val_loss: 0.6240, val_acc: 0.8605
Epoch [4], val_loss: 0.6149, val_acc: 0.8618
```

In [22]:
```
history=[result0]+history1+history2+history3+history4+history5
accuricies=[result['val_acc'] for result in history]
plt.plot(accuricies,'-x')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.title('Accuracy vs. No. of epochs')
```

Out[22]:
```
Text(0.5, 1.0, 'Accuracy vs. No. of epochs')
```

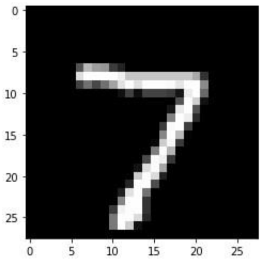


Testing

In [23]:
```
def predicted_image(img,model):
    xb=img.unsqueeze(0)
    yb=model(xb)
    _,preds=torch.max(yb,dim=1)
    return preds[0].item()
```

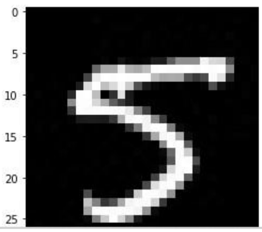
In [24]:
```
img,label=test_dataset[0]
plt.imshow(img[0],cmap='gray')
print('Label: ',label,'\nPredicted: ',predicted_image(img,model))
```

```
Label:  7
Predicted:  7
```



In [60]:
```
img=plt.imread('C:\\Users\\hp\\Downloads\\Picture\\t102.jpg')
plt.imshow(img)
img=np.array(img,dtype='float32')
t=torch.from_numpy(img)
print(predicted_image(t,model))
```

```
5
```



```
In [26]: test_loader=DataLoader(test_dataset,batch_size=256)
         result=evaluate(model,test_loader)
         result
```

```
Out[26]: {'val_loss': 0.5886543989181519, 'val_acc': 0.8677734136581421}
```

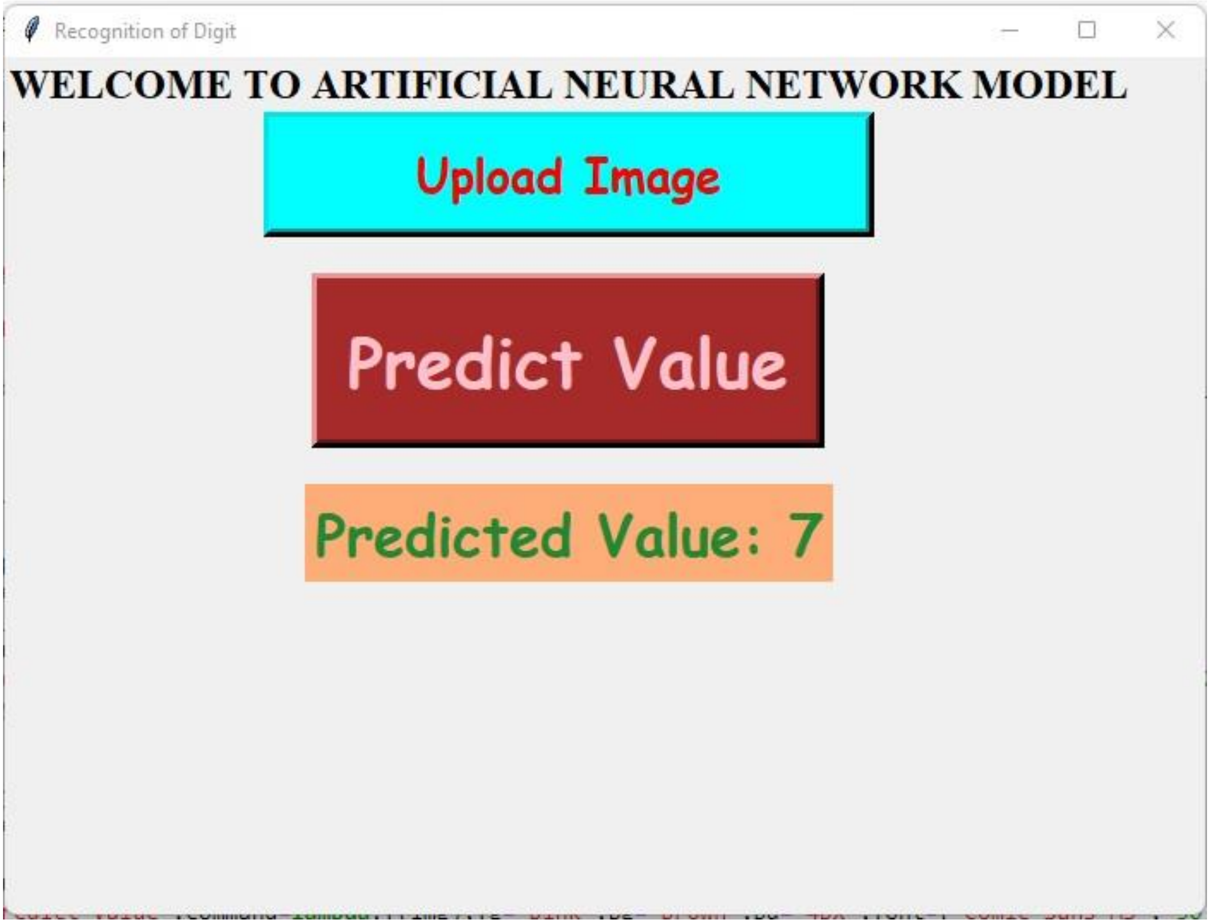
```
In [100]: mport tkinter as tk
          rom tkinter import PhotoImage
          rom tkinter import filedialog
          rom tkinter.filedialog import askopenfile
          oot=tk.Tk()
          oot.geometry("700x500")
          oot.title("Recognition of Digit")
          ont=('times',18,'bold')
          l=tk.Label(root,text='WELCOME TO ARTIFICIAL NEURAL NETWORK MODEL',font=font)
          l.grid(row=1,column=1)
          l=tk.Button(root,text='Upload Image',width=20,command=lambda:upload_file(),fg='red',bg='cyan',font=("Comic Sans MS", 20, "bold"),b
          l.grid(row=2,column=1)

          ef f(img):
             temp=np.array(img,dtype='float32')/255
             plt.imshow(img)
             temp=torch.from_numpy(temp)
             prediction=predicted_image(temp,model)
             print('Predicted Value: ',prediction)
             tk.Label(root,text=' ').grid(row=6,column=1)
             l2=tk.Label(root,text="Predicted Value: "+str(prediction),bg='#feac78',fg='#238832',bd='4px',font=("Comic Sans MS", 25, "bold")
             #plt.imshow(img,cmap='gray')

          ef upload_file():
             f_types=[('Jpg Files','*.jpg')]
             filename=filedialog.askopenfilename(filetypes=f_types)
             img=im.imread(filename)
             tk.Label(root,text=" ").grid(row=4,column=1)
             b2=tk.Button(root,text="Predict Value",command=lambda:f(img),fg='pink',bg='brown',bd='4px',font=("Comic Sans MS", 30, "bold"))
             b2.grid(row=5,column=1)


          oot.mainloop()
```

```
Predicted Value:  9
```



GitHub Link:

https://github.com/shivam-2002/Hand-Written-Digit-Recognition