SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING

(Deemed to be University)

MDSC-206

Image Segmentation & Classification

Akula. Venkata Satya Sai Gopinadh

22228

Table of Contents

Sri Sathya Sai Institute of Higher Learning

(Deemed to be University)

CERTIFICATE

This is to certify that this Mini-Project titled **Image Segmentation & Classification** was submitted by **Akula. Venkata Satya Sai Gopinadh**, **22228**, Department of Mathematics and Computer Science, Muddenahalli Campus is a bonafide record of the original work done under my supervision as a Course requirement for the Degree of **Master of Science in Data Science & Computing**.

.....

Sri V Bhaskaran,
Project Supervisor.

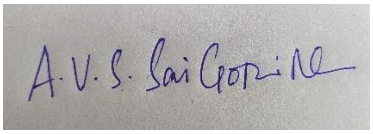
Place: Muddenahalli
Date:01 May 2023.

Sri Sathya Sai Institute of Higher Learning

(Deemed to be University)

DECLARATION

The Mini-Project titled **Image Segmentation & Classification** was carried out by me under the supervision of **Sri V Bhaskaran sir**, Department of Mathematics and Computer Science, Muddenahalli Campus as a course requirement for the Degree of **Master of Science in Data Science & Computing,** and has not formed the basis for the award of any degree, diploma or any other such title by this or any other University.

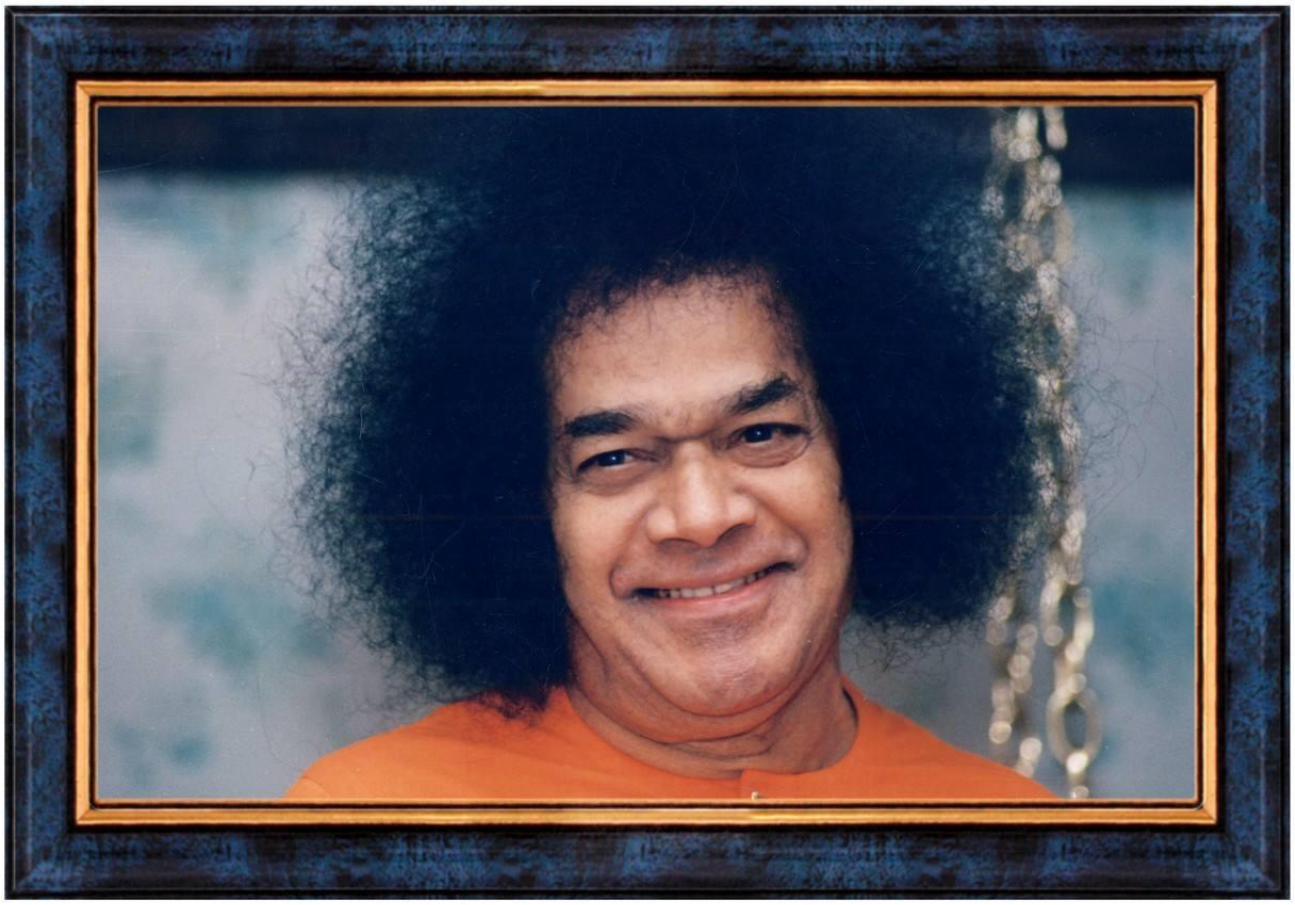


……..……

Akula Venkata Satya Sai Gopinadh

22228

Place: Muddenahalli
Date: 01 May 2023



Dedicated to Bhagawan Sri Sathya SaiBaba

ACKNOWLEDGEMENT

I express my gratitude from the bottom of my heart to **Bhagawan Sri Sathya Sai Baba** Varu who blessed and guided me all along with endeavor.

My heartfelt gratitude to my parents for supporting me throughout this journey.

I greatly thank my project supervisor **Sri V. Bhaskaran sir** for providing and supporting me with requirements, ideas, and thoughtsthat led to the success of this project. If not for him the project wouldn't have been completed.

His encouragement helped me gain deeper knowledge and completethis mini-project successfully.

Furthermore, I am grateful to him, for guiding me throughout the mini-project with a full amount of patience and care. Without that, nothing of this sort could have occurred. I wholeheartedly thank everyone who helped me in every step of this mini-project.

I sincerely thank all the teachers for supporting me in developing this mini-project.

I thank all my seniors for supporting and encouraging me in completing this mini-project answering my queries and providing valuable input.

ABSTRACT

The goal of the project is to extract and classify handwritten digits(marks) from an image of an evaluated answer sheet by image segmentation and classification. The segmenting of the marks uses image processing techniques to extract the handwritten digits. After that, a classification model can be trained on the extracted digits to classify them into their respective numbers. This can be implemented using a combination of computer vision and deep learning techniques, including image segmentation, preprocessing, and classification using a neural network model. The ultimate goal of this project is to accurately identify and classify handwritten digits, which could have many real-world applications, such as automatically grading tests or digitizing written records.

Chapter – 1: Introduction

Implemented using a combination of computer vision and deep learning techniques, including image segmentation, preprocessing, and classification using a neural network model.

To extract the handwritten digits, several image processing techniques are employed. Firstly, Gaussian blur is used to reduce the noise in the image, which helps to improve the accuracy of the subsequent processing steps.

Next, threshold and contour detection is used to extract the circle part of the image, which is then cropped.

Once the circle image is isolated, further threshold techniques like Binary and Otsu threshold techniques are used to separate the handwritten digits from the rest of the image. These digits are then displayed separately and are passed through a neural network for classification.

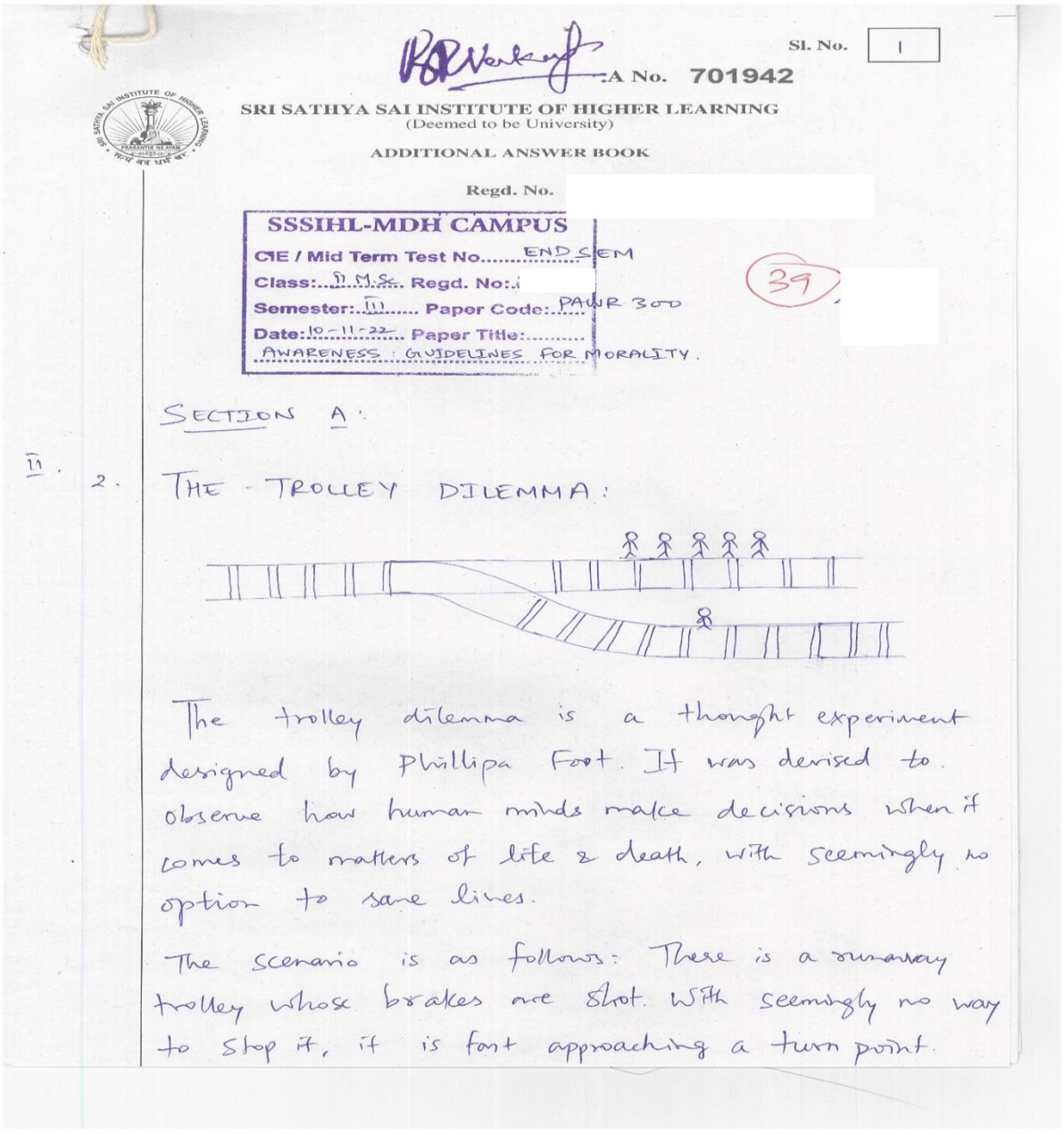
The dataset consists of cropped circle images of single digits. After loading the images, resize them to 28*28 pixels, apply normalization, and then train a neural network using a feed-forward architecture with two hidden layers. And I also used Stochastic Gradient Descent (SGD) optimizer to optimize the neural network weights. And then I trained the images batches-wise using a Data loader, the loss function used in this is Cross Entropy Loss.

Chapter – 2: Dataset Description

Data used for this project is the evaluated answer scripts of the students. Images contain scanned copies of the answer scripts, including handwriting or typed text, diagrams, graphs, or other types of content that might be included in answer scripts.

- **Source**: SSSIHL Office Muddenahalli Campus.
- **Format of Data**: Images.
- **Dataset Size**: 1000

Sample image:



Sl. No. 1

A No. 701942

SRI SATHYA SAI INSTITUTE OF HIGHER LEARNING
(Deemed to be University)

ADDITIONAL ANSWER BOOK

Regd. No.

SSSIHL-MDH CAMPUS
CIE / Mid Term Test No. END SEM
Class: II M.Sc. Regd. No:
Semester: III Paper Code: PAUR 300
Date: 10-11-22 Paper Title:
AWARENESS : GUIDELINES FOR MORALITY.

39

SECTION A:

II. 2. THE TROLLEY DILEMMA:

The trolley dilemma is a thought experiment designed by Phillipa Foot. It was devised to observe how human minds make decisions when it comes to matters of life & death, with seemingly no option to save lives.

The scenario is as follows: There is a runaway trolley whose brakes are shot. With seemingly no way to stop it, it is fast approaching a turn point.

Fig.1: Dataset Image

Chapter – 3: Methodology

3.1 Segmentation:

Some of the image processing operations are performed on a given image to extract the red pixels. The overall goal is to create a binary image of the original image where only the red pixels are white and the rest are black.

The image is first loaded using the PIL library and converted to a tensor using the NumPy and torch libraries. The image is then converted into a grayscale and a red filter is applied to extract red pixels. The resulting image is thresholded to binary using a threshold value of 50.

The noise of the image is removed from the binary image using a max pool with a kernel size of 3. And the final step is to extract the red pixels from the grayscale image using the binary image as a mask.
The main purpose of this part is to extract the relevant information from the input image which will be further processed in the pipeline to extract the digits.

```
import torch
import numpy as np
from PIL import Image
import cv2
from google.colab.patches import cv2_imshow

# Load the image
image = Image.open("/content/drive/MyDrive/Mini-
Project/Train data/Sam_20221201120159_page-0017.jpg")
# Convert to tensor
image = torch.Tensor(np.array(image))
# Convert to grayscale
grayscale_image = image.mean(dim=2, keepdim=True)
# Apply red filter
red_filtered_image = image[:, :, 0] - (image[:, :, 1] + image[:, :, 2]) / 2
# Threshold the image
threshold = 50
binary_image = red_filtered_image.gt(threshold).float()
# Remove noise
kernel_size = 3
binary_image = torch.nn.functional.max_pool2d(binary_image.unsqueeze(0), ke
rnel_size, stride=1, padding=kernel_size//2).squeeze(0)
# Extract red pixels
red_pixels = binary_image * grayscale_image.squeeze(-1)
# Save the result
result = Image.fromarray(red_pixels.numpy().astype(np.uint8)
display(result)
print(f"Shape of the image after extracting the red pixels only : {result.s
ize}")
```

The above code gives the output:

Fig.2: Red pixels Extracted Image

Now we will crop the image. Because some faculty will round the marks for the questions also, to avoid that confusion, we will crop the image.

```
result_array = np.array(result)
cropped_image = result_array[20:1700 , 20:3000]
print(cropped_image.shape)
print(cropped_image.shape[-1])
cv2_imshow(cropped_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
# Cropped_image = result_array[x1:x2 , y1:y2]
```

The resulting image obtained will be cropped by slicing the array. The cropped image array is created by selecting a specific range of rows and columns from the result array. The range starts from row x1 and ends at row x2 and starts from column y1 and ends at y2.

Fig.3: Cropped Red Pixels Extracted Image

Now we will extract digits from the given image of the answer sheet containing the handwritten digits. The image is first preprocessed by applying a Gaussian blur to remove any noise and then a binary threshold is applied to convert it into a binary image.

Gaussian Blur: It is a smoothing technique used in image processing to reduce image noise and details. It is based on the Gaussian function (normal function), which generates a kernel matrix that is used to convolve with the image. The kernel matrix applies a weighted average to the pixels in the image, with the highest weight being given to the central pixel and decreasing as the distance from the center increases. This results in a blurred image with reduced noise and details making it useful in various applications such as edge detection and image segmentation.

Contour Detection: It is a process of detecting and extracting the boundaries of objects in an image. It is commonly used in computer vision applications, such as object recognition and tracking. The contour of an object is represented as a curve connecting all the continuous points along its boundary. These are various techniques for contour detection, including edge detection and thresholding.

Otsu's Thresholding: It automatically determines the optimal threshold value for image segmentation. It assumes that the image contains only two classes of pixels i.e., Front and background class. Its goal is to maximize the distance class of the two classes of pixels.

Then, the contours of the image are detected using the cv2.findContours function. The largest contour is assumed to be the circle surrounding the digits, and its bounding rectangle coordinates are extracted. This bounding rectangle is then cropped from the original image containing the handwritten digits. And we will invert the image to match the MNIST dataset for the further usage of those digits for classification.

Fig.4: Cropped Marks Region Image & Inverted Image.

Next, the individual digits are extracted from the circle image using the contours detected earlier. For each digit, the bounding rectangle coordinates are extracted, and the digit image is cropped from the binary image.

Fig.5: Segmented Digits (separated)

With this, the preprocessing for the image is done.

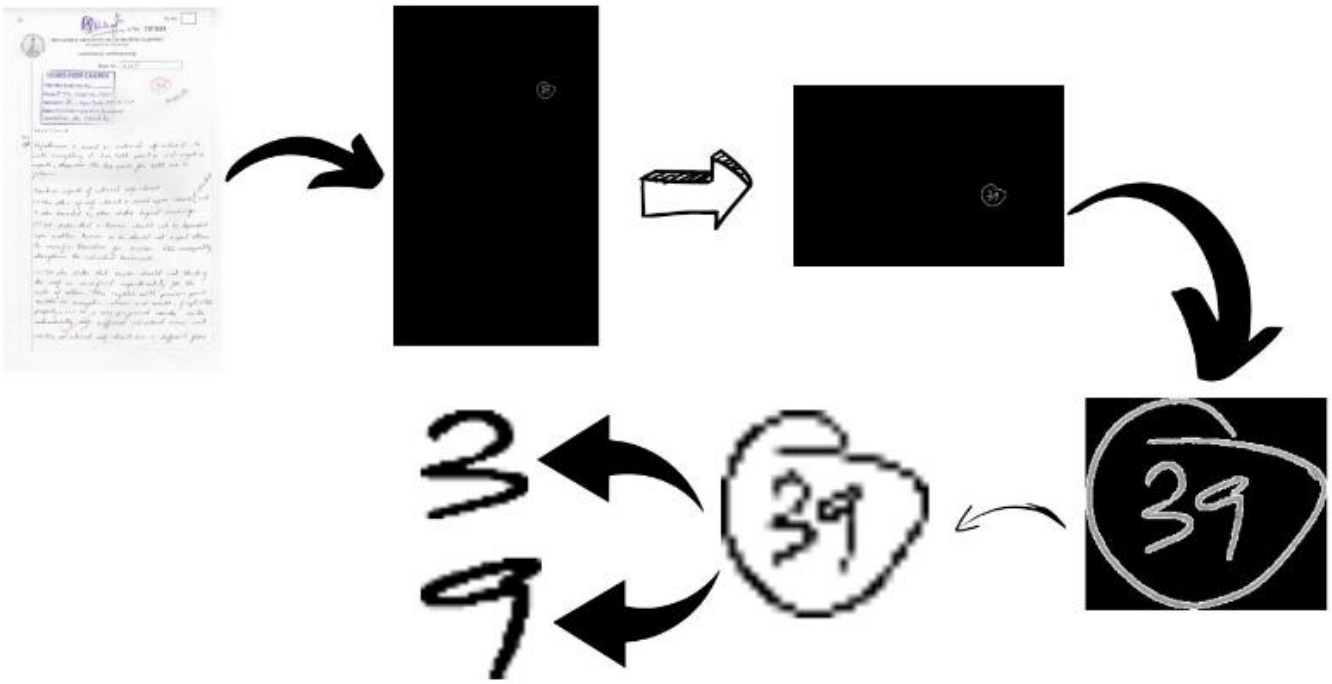


Fig.6: Flow Chart of Project.

3.2 Classification:

We performed some image transformations to preprocess the image data before feeding it into the pipeline.

It was done using the PyTorch package, ***torchvision.transforms***. These transformations can be considered as custom edits performed on the images to make them uniform in size and properties. Some of the common transformations include resizing, cropping, flipping, rotating, normalization, and conversion to tensors.

These transformations help to make the images consistent and compatible with the deep learning models that expect standardized inputs.

By using ***torchvision.transforms***, we can easily apply the required transformations to the dataset and create a preprocessed version of the data that can be used for training and testing the deep learning models.

```
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,)),
                                ])
```

1. transforms.ToTensor() – changes over the picture into numbers, that are reasonable by the framework. It isolates the picture into three variety channels (separate pictures): red, green, and blue. Then, at that point, it switches the pixels of each picture over completely to the brightness of the image in the range of o and 255. These qualities are then downsized to a reach somewhere in the range of 0 and 1. The image is now a Torch Tensor.
2. transforms.Normalize() – Normalizes the tensor with a mean and standard deviation which goes as the two parameters respectively.

Neural Network Used:

```
from torch import nn

# Layer details for the neural network
input_size = 784
hidden_sizes = [128, 64]
output_size = 10

# Build a feed-forward network
model = nn.Sequential(nn.Linear(input_size, hidden_sizes[0]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[0], hidden_sizes[1]),
                      nn.ReLU(),
                      nn.Linear(hidden_sizes[1], output_size),
                      nn.LogSoftmax(dim=1))
print(model)
```

Using PyTorch **nn** module **feed-forward** neural network is used for this problem.

And this neural network has three layers, an input layer with 784 neurons (28*28) and two hidden layers with 128 and 64 neurons respectively, and an output layer with 10 neurons.

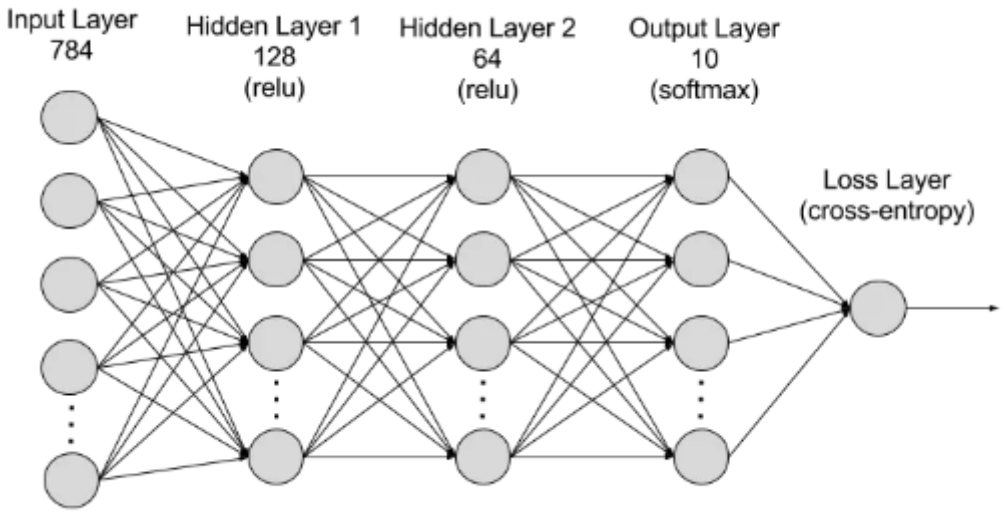


Fig.7: Neural Network

The ***nn. Sequential*** class is used to create a sequence of layers. The ***nn. Linear*** function creates a fully connected layer that applies a linear transformation to the input data. The ***ReLU*** function is used as an activation function between the layers. The ***nn. LogSoftmax*** function is used as the output activation function which applies the log of the softmax function on the output layer.

The training dataset used in this mini-project is the MNIST handwritten digits dataset,

which contains 70,000 images of which 60,000 images are for training and 10,000 images are for testing. All the images are in grayscale and 28*28 pixels in size.

Sample images:

Fig.8: Training Data Samples

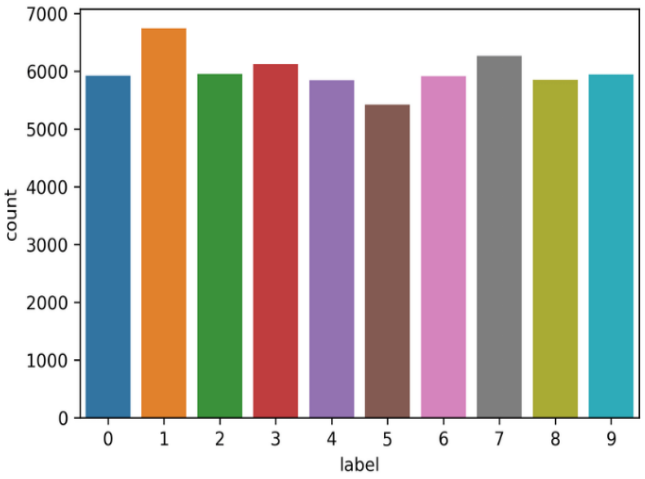
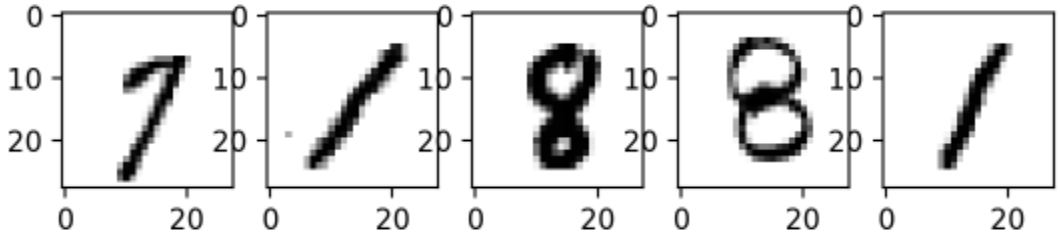


Fig.9: Distribution of digits in MNIST



And for testing, we used the output digits from the segmented image from the answer sheet.

Sample images:

Fig.10: Test Images

***Loss Function*:** The loss function used is the Cross-Entropy Loss. It is especially used for classification problems. It measures the dissimilarity between the predicted probability distribution and the true probability distribution.

The goal is to minimize this dissimilarity to improve the accuracy of the model's predictions. It is often used in combination with a softmax activation function to produce the predicted probability distribution.

The optimizer used in this mini-project is Stochastic Gradient Descent (SGD).

Stochastic Gradient Descent: It is an optimization algorithm used to minimize the loss function. It updates the model parameters in small batches of random samples instead of the entire dataset, which makes it faster and computationally efficient. It uses a learning rate that controls the step size of the gradient descent and a momentum term that helps in accelerating the optimization. It is widely used in deep learning for training neural networks.

```
# Take an update step and few the new weights
optimizer. Step ()
print ('Updated weights - ', model [0]. weight)
```

The *optimizer.step()* function takes a step in the direction of the negative gradient with a step size determined by the learning rate and the momentum and updates the model parameters. The model[0].weight attribute contains the weights of the first linear layer in the model, and we are printing its value to see the updated weights after performing the optimization step. This process is repeated iteratively until the loss function converges to a minimum.

```
optimizer = optim.SGD(model.parameters(), lr=0.003, momentum=0.9)
time0 = time()
epochs = int(input("Enter the no of EPOCHS : "))
for e in range(epochs):
    running_loss = 0
    for images, labels in trainloader:
        # Flatten MNIST images into a 784 long vector
        images = images.view(images.shape[0], -1)

        # Training pass
        optimizer.zero_grad()

        output = model(images)
        loss = criterion(output, labels)

        #This is where the model learns by backpropagating
        loss.backward()

        #And optimizes its weights here
        optimizer.step()

        running_loss += loss.item()
    else:
        print("Epoch {} - Training loss: {}".format(e, running_loss/len(tra
inloader)))
print("\nTraining Time (in minutes) =",(time()-time0)/60)
```

The model computes the loss, backpropagates, and optimizes its weights. The running loss variable keeps track of the average loss over all batches in an epoch.

```
 # Flatten MNIST images into a 784-long vector
        images = images.view(images.shape[0], -1)
```

This is used to reshape the input images before they are passed to the neural network model. The MNIST images are initially 28*28 pixels, then it is flattened into a 784-dimensional vector. The first dimension of the tensor is preserved (which represents the batch size), and the second dimension is flattened to a single vector. This transformation is necessary because most deep learning models expect the input to be a vector rather than a multi-dimensional array.

Chapter – 4: Learning Process

- Referred to courses on Coursera and Simplilearn and completed the required parts of the courses.
- Learnt basics of PyTorch.
- Covered around 15 blogs to understand image segmentation and classification.
- Experimented with image classification using Dog & Cat dataset.
- Explored different articles to clarify doubts.
- Went through all the basics of PyTorch and implementation is done for a few for understanding.
- Explored image segmentation using some publicly available datasets.
- Covered some neural network concepts for understanding and usage.
- Did a guided project on Coursera titled Deep Learning with PyTorch: Image Segmentation for understanding Image Segmentation.
- 

Chapter – 5: Results

The algorithm used is a feed-forward neural network with two hidden layers. The input layer has a size of 784 which is reduced to 128 for the first hidden layer and then to 64 for the next hidden layer and the output layer to 10 neurons. Since there are 10 different classes (i.e., 0 to 9 digits).

The ***Accuracy*** of the Algorithm is ***either 0 or 1;*** since we are using only ***one test image***. If the test image is predicted correctly the accuracy is 1, else 0.

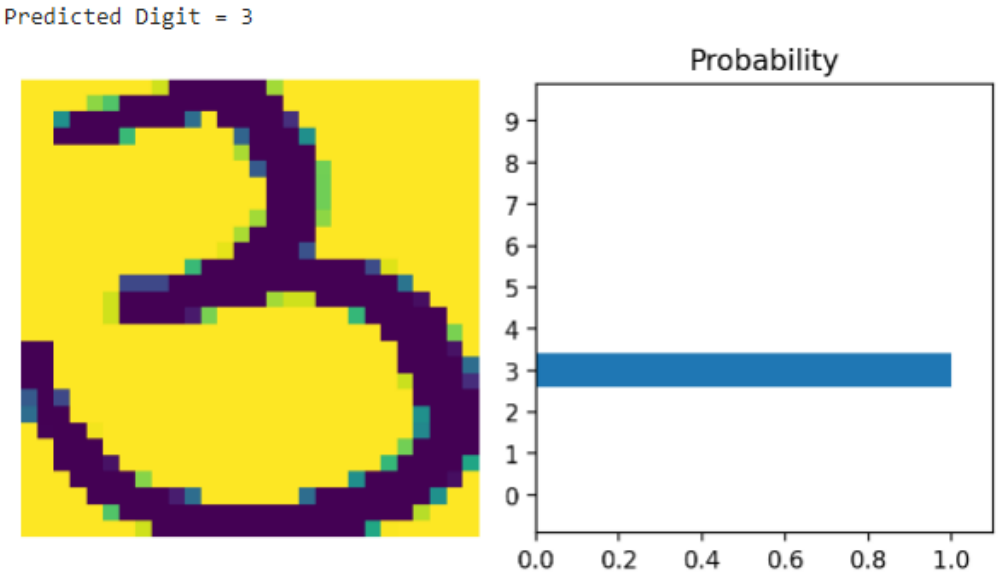


Fig.7: Output

Chapter – 6: Learning Outcomes

Learning outcomes after completing this project are as follows:

- Understood the working of Neural networks and Activation functions.
- Power of PyTorch in image segmentation and classification.
- Had an understanding of OpenCV (Computer Vision). Like how to convert images into grayscale and RGB format, extracting red pixels separately and working on that.
- And had a thorough understanding of how to create a dataset for both train and test images and from that how to create a train loader and test loader using Dataloader in PyTorch.
- Insights and implementation of how to convert an image into a tensor before feeding it to the network.

Chapter – 7: Future Work

There is a lot that can be done in continuation with this:

- Now the segmenting of the marks from the evaluated answer sheet is done and the classification on just a one-digit image from the digits segmented is done.
- Further, the segmented digits all can be classified and display the marks concatenated (combining both digits output and displaying them).
- Similarly, the registration number of the student from the answer sheet can be segmented, classified, and displayed.
- In the same way, the course code can also get segmented, and classification can be done using MNIST handwritten characters dataset.
- It can be done as an application where the image of the answer sheet is taken as the input, the marks are segmented, and classified, the registration number is segmented and classified and the course code can be segmented and classified and save all the results in an excel file which can save a lot of time to the faculty or the office staff in entering the marks for all the students.

Chapter – 8: Bibliography

1. Coursera course:

   https://www.coursera.org/learn/deep-neural-networks-with-pytorch/

2. Simplilearn course:

   https://www.simplilearn.com/introduction-to-deep-learning-free-course-skillup?tag=deep%20learning%20for%20beginners

3.PyTorch Fundamentals:

   https://learn.microsoft.com/en-us/training/paths/pytorch-fundamentals/

   https://learn.microsoft.com/en-us/training/modules/intro-computer-vision-pytorch/

4.Medium Articles:

   https://towardsdatascience.com/handwritten-digit-mnist-pytorch-977b5338e627

   https://medium.com/analytics-vidhya/training-mnist-handwritten-digit-data-using-pytorch-5513bf4614fb

   https://machinelearningmastery.com/handwritten-digit-recognition-with-lenet5-model-in-pytorch/

   https://yash-kukreja-98.medium.com/recognizing-handwritten-digits-in-real-life-images-using-cnn-3b48a9ae5e3

5.Kaggle:

   https://www.kaggle.com/code/franklemuchahary/mnist-digit-recognition-using-pytorch