502 Shubham Lad

MLDL Mini Project: Handwritten Digit Classifier Using Logistic Regression.

In this mini project we're going to implement from scratch a one-vs-all logistic regression classifier for the MNIST digits dataset with a neural network mindset. The neural network aspect of this implementation is the use of a forward and backward propagation to claculate the value of the cost function and the partial derivatives of the cost function with respect to weights and the bias.

When it comes to the training phase, the forward propagation uses a loss function to determine the cost, and the backward propagation uses the chain rule to calculate the partial derivates of the cost function with regard to the weights and bias. This method of implementation imitates the forward and backward propagation used in neural network training.

Loss and cost functions:

- Loss function:

$$\mathcal{L}(\hat{y}, y) = -\left(y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right)$$

- Cost function:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Partial derivatives of the cost functions:

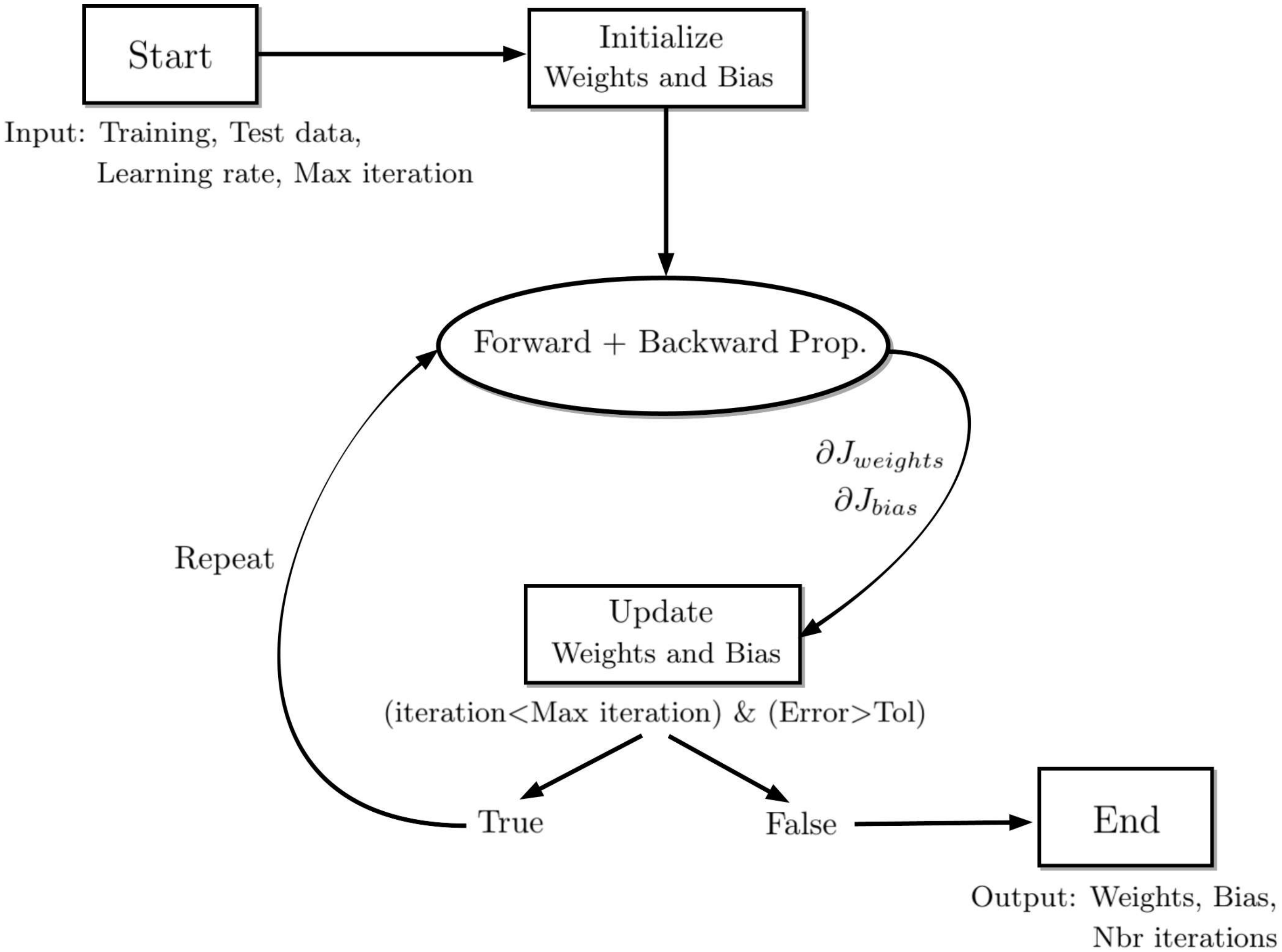
The vectorized form of the partial derivatives of the cost function J with respect to the weights and the bias are given under a vectorized form by:

$$\frac{\partial J}{\partial w} = \frac{1}{m}(X^T(\hat{Y} - Y)), \quad \frac{\partial J}{\partial b} = \frac{1}{m}Y^T\hat{Y}$$

Where,

$$X = \begin{pmatrix} \cdots Image_1 \cdots \\ \cdots Image_2 \cdots \\ . \\ . \\ . \\ \cdots Image_{m-1} \cdots \\ \cdots Image_m \cdots \end{pmatrix}_{(m\times n)} \quad , \quad Y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ . \\ . \\ . \\ y^{(m-1)} \\ y^{(m)} \end{pmatrix}_{(m\times 1)} \quad , \quad \hat{Y} = \begin{pmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ . \\ . \\ . \\ \hat{y}^{(m-1)} \\ \hat{y}^{(m)} \end{pmatrix}_{(m\times 1)}$$

In order to classify 10 digits (i.e., from 0 to 9), using logistic regression. This strategy is referred to as the One-vs-all classification method and requires that we build 10 models, one for each digit. The model with the highest probability is used to categorize the provided image once each model's likelihood of a particular image has been determined. The steps of applying gradient descent to minimize the cost function are shown in the following diagram.



Download the MNIST dataset

```
# @title
!pip -qq install --upgrade --no-cache-dir gdown
```

```
!pip -qq  install imageio
# https://drive.google.com/file/d/1lyP8UkVxEFm6cAhjYXwRUP3k3n0ddTgD/view?usp=sharing
!gdown 1lyP8UkVxEFm6cAhjYXwRUP3k3n0ddTgD
!unzip -qq mnist-original.mat.zip
```

```
Downloading...
From: https://drive.google.com/uc?id=1lyP8UkVxEFm6cAhjYXwRUP3k3n0ddTgD
To: /content/mnist-original.mat.zip
100% 11.4M/11.4M [00:00<00:00, 46.8MB/s]
```

Importing and preprocessing data

Show code

Show code

Show code

```
(70000, 784)
```

Show code

```
✅ [Info] The images size is ( 28 x 28 )
```

Show code

Show code

✅ [Info] The number in the image below is: 7.0

Show code

Show code

Show code

✅ [Info] The shape of the training set feature matrix is: (56000, 784)
✅ [Info] The shape of the training label vector is: (56000, 1)
✅ [Info] The shape of the test set feature matrix is: (14000, 784)
✅ [Infp] The shape of the test label vector is: (14000, 1)

Show code

Creating model functions

Show code

Show code

Show code

Show code

Show code

Show code

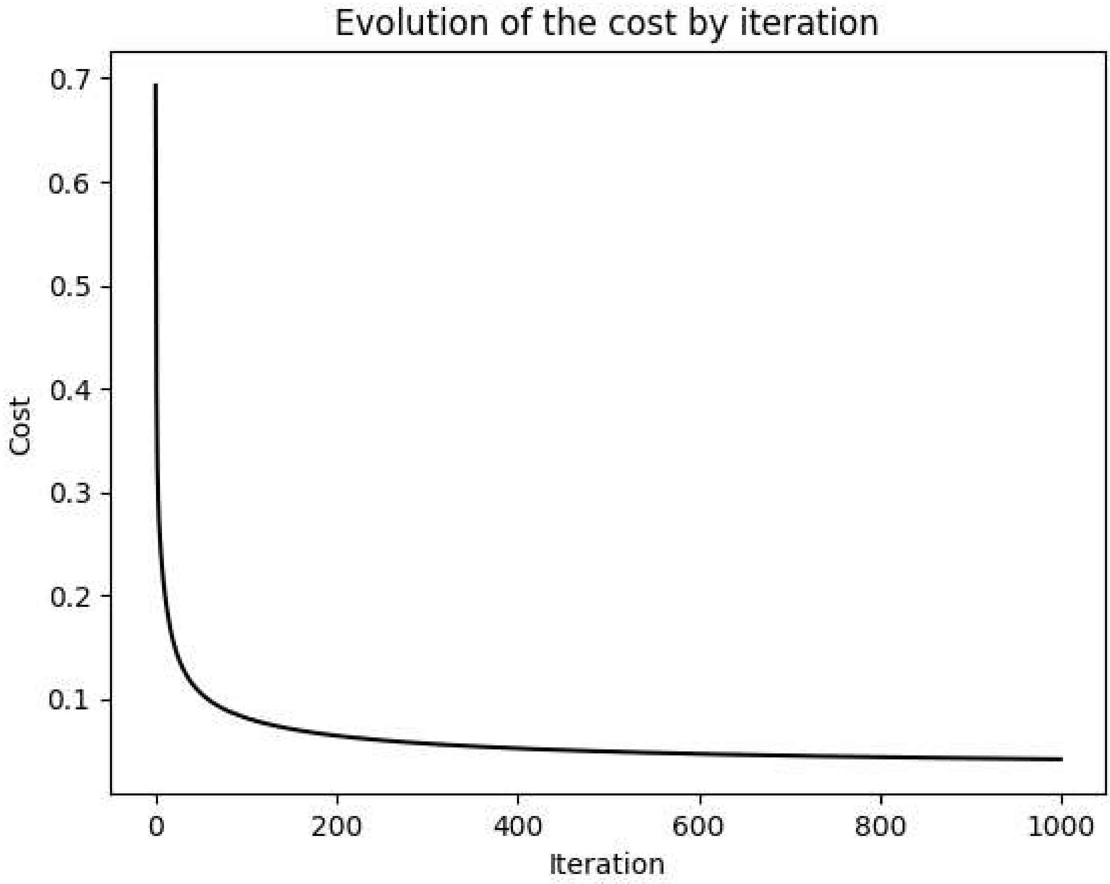
Training a test model for the digit "0"

Show code

```
[Info] Progress bar: 1 step each 50 iteration ✅
[Info ]Training Progress: ✅
====================]
[Info] Training completed! ✅
```

Show code

```
Text(0, 0.5, 'Cost')
```



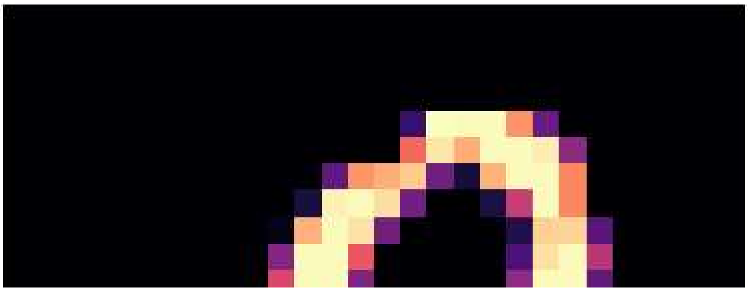
Show code

```
✅ [Info] The training accuracy of the model 0.9882321428571429
✅ [Info] The test accuracy of the model 0.9875
```

Show code

Show code

```
[Info] The number in the image below is: 0  and predicted as: 0
```



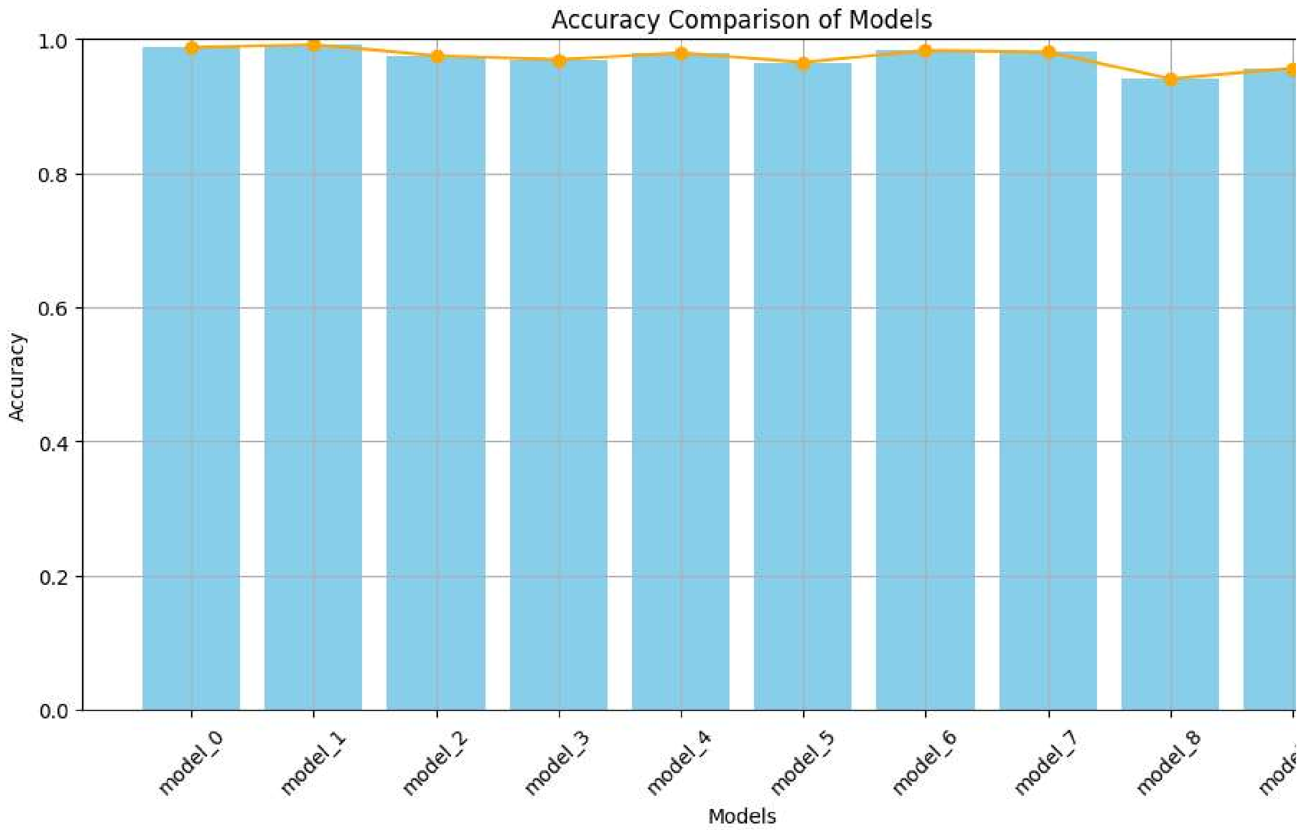
Training a model for each digit

Show code

```
[Info] Training of a classifier for each digit:
[Info] Training model: model_0, to recognize digit: 0 ✅
[Info] Training progress bar: 1 step each 50 iterations ✅
[Info ]Training Progress: ✅
====================]
[Info] Training completed! ✅
[Info] Accuracy: 0.9875
▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬
[Info] Training model: model_1, to recognize digit: 1 ✅
[Info] Training progress bar: 1 step each 50 iterations ✅
[Info ]Training Progress: ✅
====================]
[Info] Training completed! ✅
[Info] Accuracy: 0.9917857142857143
▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬
[Info] Training model: model_2, to recognize digit: 2 ✅
[Info] Training progress bar: 1 step each 50 iterations ✅
[Info ]Training Progress: ✅
====================]
[Info] Training completed! ✅
[Info] Accuracy: 0.9753571428571428
▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬
[Info] Training model: model_3, to recognize digit: 3 ✅
[Info] Training progress bar: 1 step each 50 iterations ✅
[Info ]Training Progress: ✅
====================]
[Info] Training completed! ✅
[Info] Accuracy: 0.9692142857142857
▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬
[Info] Training model: model_4, to recognize digit: 4 ✅
[Info] Training progress bar: 1 step each 50 iterations ✅
[Info ]Training Progress: ✅
====================]
[Info] Training completed! ✅
[Info] Accuracy: 0.9796428571428571
▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬
[Info] Training model: model_5, to recognize digit: 5 ✅
[Info] Training progress bar: 1 step each 50 iterations ✅
[Info ]Training Progress: ✅
====================]
[Info] Training completed! ✅
[Info] Accuracy: 0.9651428571428572
```

```
▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬
[Info] Training model: model_6, to recognize digit: 6 ✅
[Info] Training progress bar: 1 step each 50 iterations ✅
[Info ]Training Progress: ✅
====================]
[Info] Training completed! ✅
[Info] Accuracy: 0.9831428571428571
▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬
[Info] Training model: model_7, to recognize digit: 7 ✅
[Info] Training progress bar: 1 step each 50 iterations ✅
[Info ]Training Progress: ✅
====================]
[Info] Training completed! ✅
[Info] Accuracy: 0.9806428571428571
```

Show code



Show code

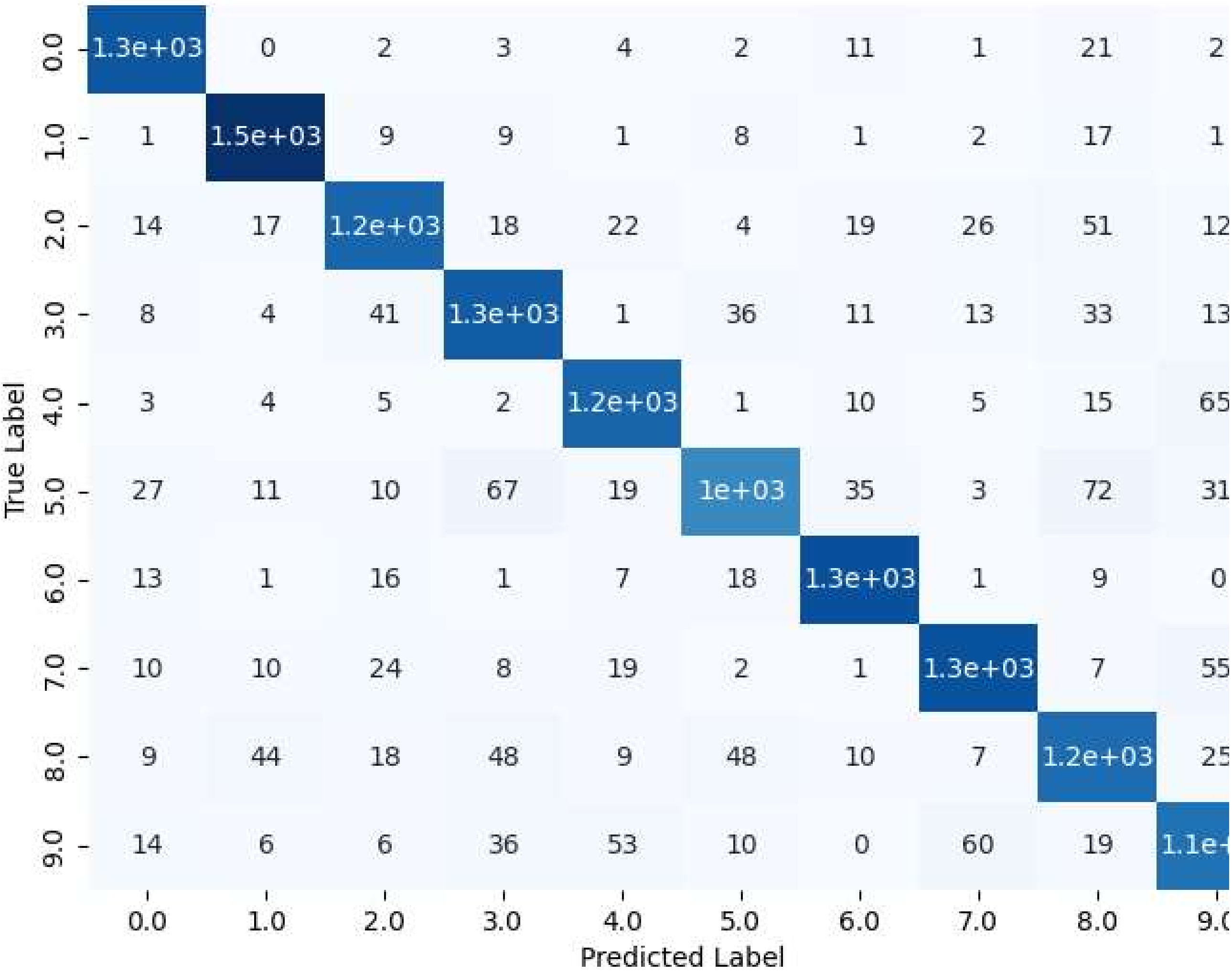
```
✅ [Info] The accuracy of the One-Vs-All model is: 0.9729928571428571
```

Final model for digit classification

Show code

Show code

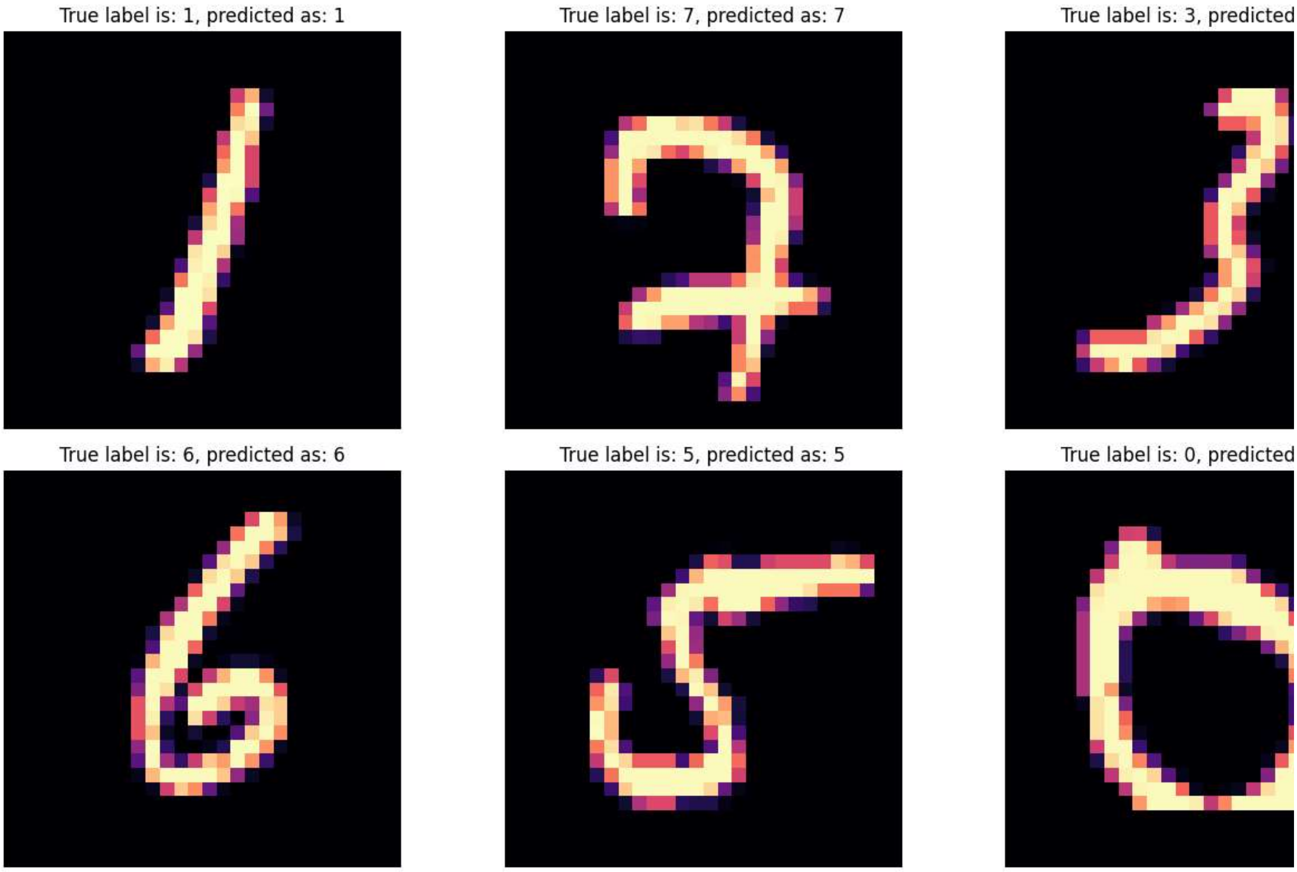
Show code



Show code

Results

Show code



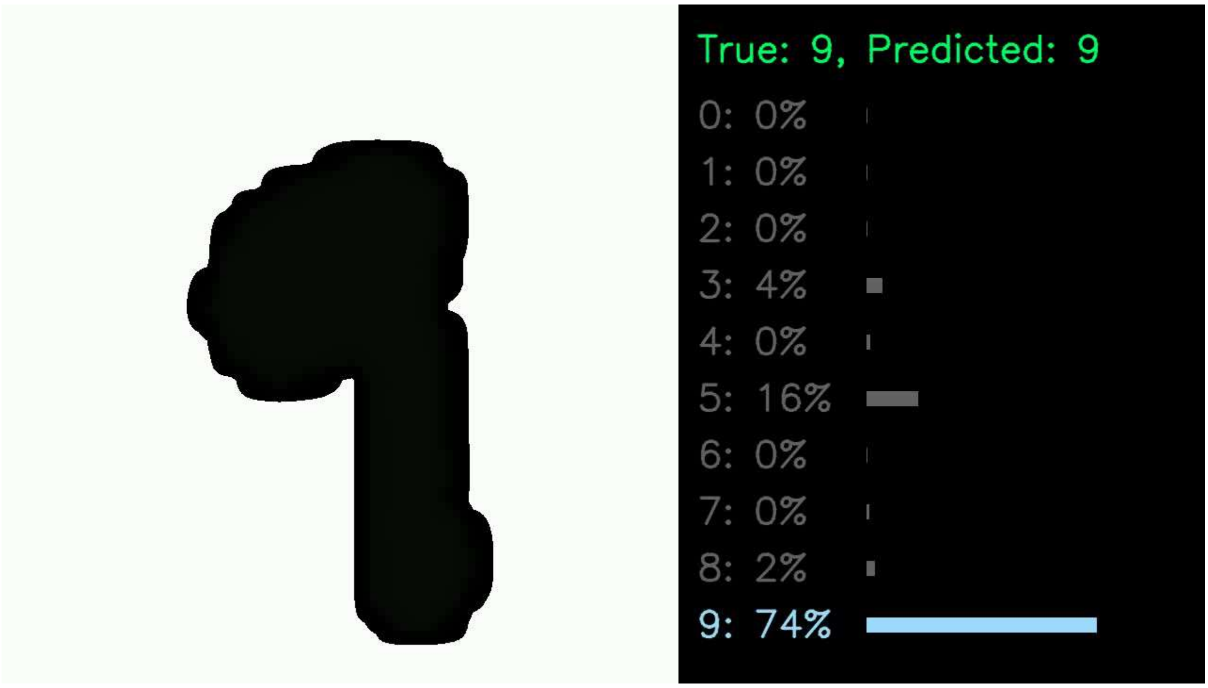
Show code

```
# @title
def render_and_save_examples(example_data, true_labels, predicted_labels, predicted_
    examples_number = example_data.shape[0]
    video_output_path = 'output_video.mp4'
    codec = cv2.VideoWriter_fourcc(*'mp4v')
    vid_width_height = 1280, 720
    vw = cv2.VideoWriter(video_output_path, codec, 30, vid_width_height)

    font_face = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 1.3
    thickness = 2
```

```

    for i in range(examples_number):
        image = example_data[i].reshape(image_size, image_size)
        image_disp = cv2.resize(image*5, (720, 720))

        # Check if prediction is correct or not
        is_correct = true_labels[i] == predicted_labels[i]

        title = f"True: {true_labels[i]}, Predicted: {predicted_labels[i]}"

        preds = predicted_score[i]*100  # Updated to use specific example's probabil

        img = np.zeros((720, 1280, 3), dtype=np.uint8)
        img[:720, :720, 0] = image_disp
        img[:720, :720, 1] = image_disp
        img[:720, :720, 2] = image_disp

        x, y = 740, 60
        txt_color = (100, 255, 0) if is_correct else (0, 0, 255)
        cv2.putText(img, text=title, org=(x, y), fontScale=font_scale, fontFace=font
                    thickness=thickness, color=txt_color, lineType=cv2.LINE_AA)

        bar_x, bar_y = 740, 130
        for j, p in enumerate(preds):
          if j < 10:
            rect_width = int(p * 3.3)
            rect_start = 180
            color = (255, 218, 158) if j == predicted_labels[i] else (100, 100, 100)
            cv2.rectangle(img, (bar_x + rect_start, bar_y - 5), (bar_x + rect_start
                          color, -1)
            text = f'{j}: {int(p)}%'
            cv2.putText(img, text=text, org=(bar_x, bar_y), fontScale=font_scale, fo
                        thickness=thickness, color=color, lineType=cv2.LINE_AA)
            bar_y += 60

        vw.write(img)

    vw.release()


examples_number = 60
index_random_sample = np.random.randint(70000, size=(1, examples_number))
example = mnist_data_normalized[index_random_sample].reshape(examples_number, 784)
true_labels = mnist_label[index_random_sample].flatten().astype(int)
predicted_labels, predicted_score = one_vs_all_score(example, models)

# Render and save examples to video
render_and_save_examples(example, true_labels, predicted_labels, predicted_score, im

```

Show code

Show code



Conclusion

In this mini-project, we successfully developed a handwritten digit classifier using logistic regression. The goal was to train a separate model for each digit from 0 to 9, enabling accurate recognition of individual digits. The training process involved multiple steps, and the results achieved were impressive.

Here's a summary of our achievements:

- For each digit, from 0 to 9, a dedicated model was trained and tested.
- The training process was successful for all models, achieving impressive accuracy rates.
- Model accuracies varied for different digits, ranging from approximately 94% to 99%.
- The training progress was visually represented with progress bars, adding clarity and insight into the process.

- Accuracy scores were displayed after training each model, providing a clear overview of their performance.

Overall, this project demonstrated the power of logistic regression in classifying handwritten digits. The achieved accuracies showcase the effectiveness of this approach in recognizing a wide range of digits. Through this mini-project, we gained practical experience in model training, testing, and accuracy evaluation, highlighting the potential of machine learning techniques in solving real-world challenges.