Udacity Machine Learning Engineer Nanodegree

Capstone Project

Handwritten Digits Recognition Web App

Abdelrahman Fekry Ali

31 August 2021

Table Of Contents

Definition

Project Overview

in nowadays the need for Document digitization is in demand. due to many benefits such as the ease of accessing the data in the digital form from anywhere and cost reduction and data security through defining accessibility privileges and data storage and recovery.
in computer vision field the process of converting these documents into digital form is called Optical Character Recognition (OCR).
the process of Optical Character Recognition (OCR) includes many steps, in this project we are going to focus on the base step which is Handwritten digit recognition (HDR).

Handwritten digit recognition (HDR) is considered a base step for many applications such as

- Signature Verification,
- Postal-Address Interpretation,
- Bank-Check Processing,
- Writer Recognition.

for this project we used the MNIST (Modified National Institute of Standards and Technology) dataset.
based on the paper[1] The set of images in the MNIST database was created in 1998 as a combination of two of NIST's databases Special Database 1 (digits written by high school students) and Special Database 3 (digits written by employees) of the United States Census Bureau.
There are 70,000 images and each image has 784 features. This is because each image is 28 x 28 pixels, and each feature represents a pixel's intensity, from 0 to 255.

Sample images from MNIST test dataset

Problem Statement

the main goal of this project is to implement web app that as asks the user to sketch a digit and then send the pixels information to backend where the data is processed then return the predicted value to the frontend. to solve this problem we used simple HTML and CSS style sheet for the frontend, and we used Flask for the backend where the pre-trained model is loaded. The machine learning algorithm that used to train the model is going to be a modified version of the famous LeNet-5 CNN.

Metrics

as dealing with a multiclass classification problem that equally cares about Type I Error and Type II Error we are going to select accuracy as the evaluation metric. the accuracy

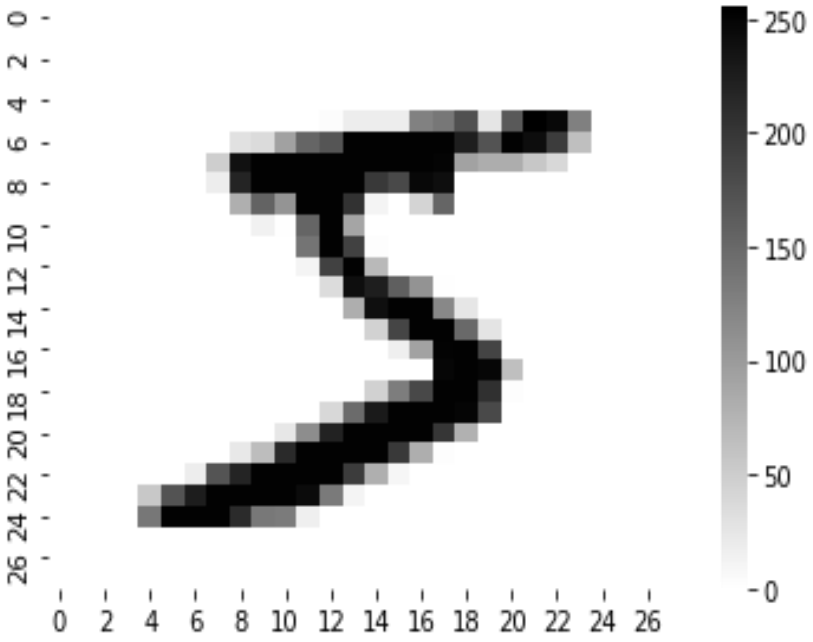
$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Accuracy is the percentage of correctly classified samples in a dataset

Analysis

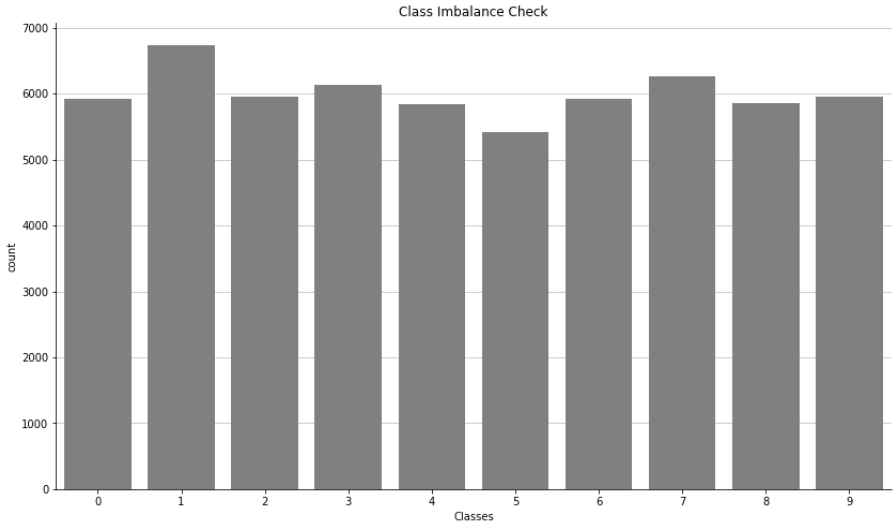
Data Exploration

There are 70,000 images and each image has 784 features. This is because each image is 28 x 28 pixels, and each feature represents a pixel's intensity, from 0 to 255.



Sample image

the classes are represented equally in the MNIST dataset



Methodology

Data Preprocessing

1. **Missing Values Check**
   First we checked for missing values

```
np.isnan(x_train).any()
np.isnan(y_train).any()
np.isnan(x_test).any()
np.isnan(y_test).any()
```

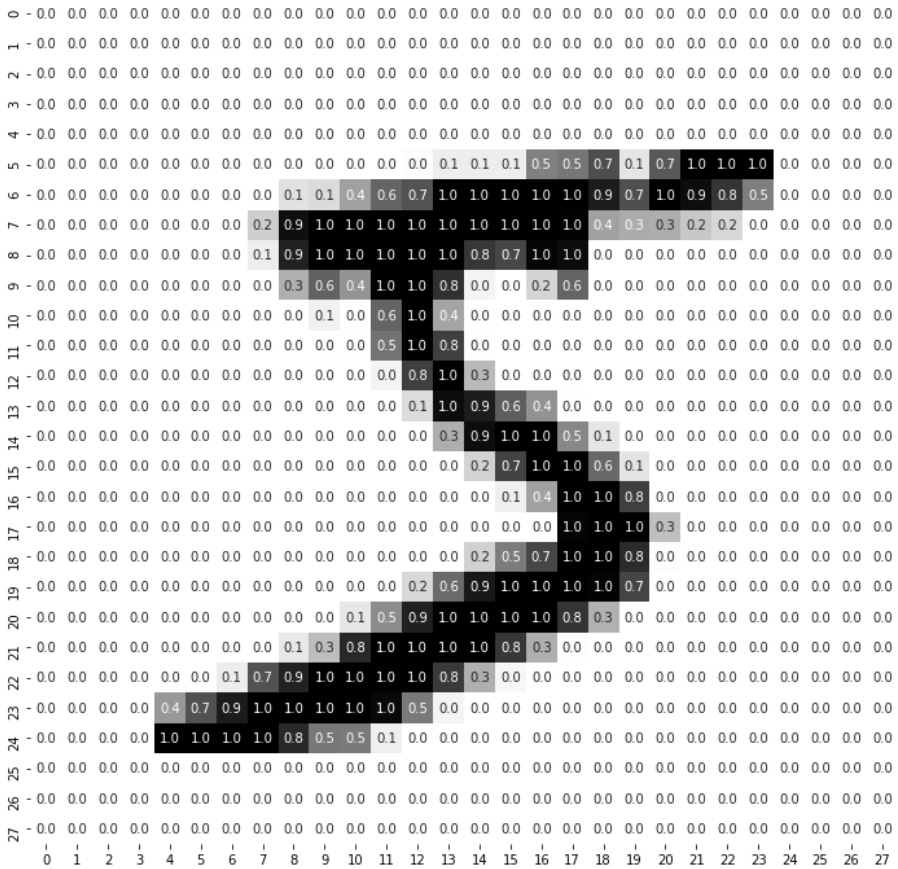
   as expected the returned value was False.

2. **MaxAbs Scaling**
   The maximum pixels intensity is 255 as we checked

```
x_train[0].max()
```

   using scikit `preprocessing.maxabs_scale()` function to rescale pixels intensity range from 0 to 1

```
x_train_scaled = np.array([maxabs_scale(x) for x in x_train])
x_test_scaled = np.array([maxabs_scale(x) for x in x_test])
```



MaxAbs Scaled Sample

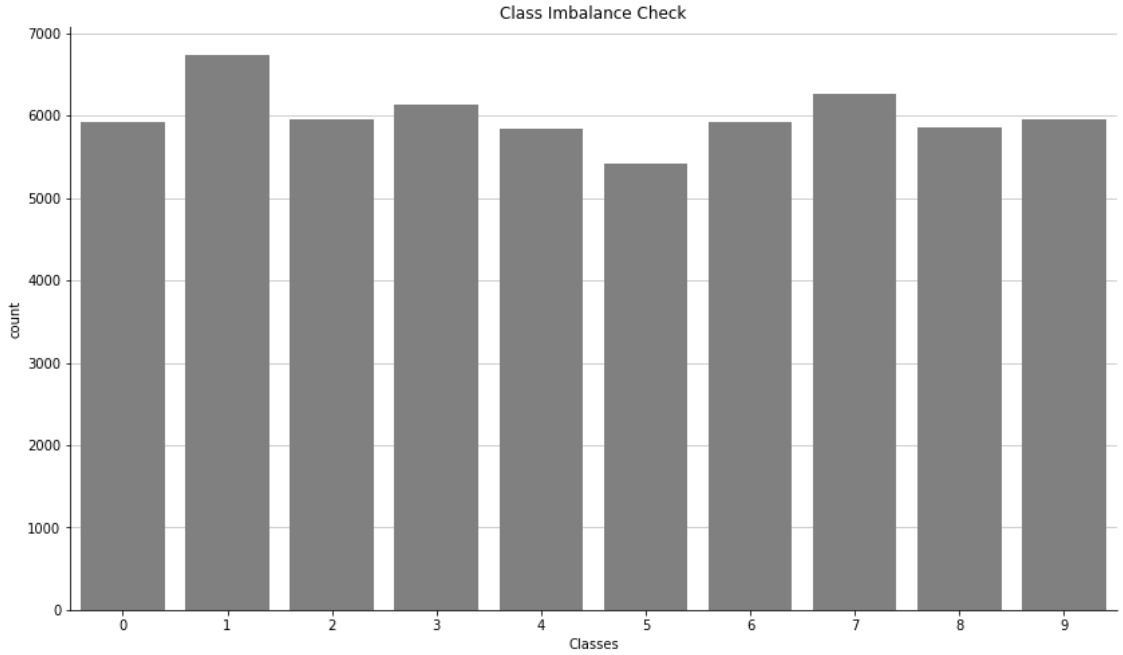
3. **Adjusting Shape And Dimensions**
The input data of the model have the shape (32,32,1) so implemented two steps to approach the targeted shape.
First we expanded the most inner dimension

```
x_train_scaled = np.expand_dims(x_train_scaled,axis=-1)
x_test_scaled = np.expand_dims(x_test_scaled,axis=-1)
```

Second we padded the input data by 2 constant values “0” each side on the axis 1 and 2.

```
x_train_scaled = np.pad(x_train_scaled, ((0,0),(2,2),(2,2),(0,0)), 'constant')
x_test_scaled = np.pad(x_test_scaled, ((0,0),(2,2),(2,2),(0,0)), 'constant')
```

4. **Class Imbalance Check**
the classes are represented equally in the MNIST dataset



5. **OneHot Encoding**
The output data of the model have the shape (10,) which is a softmax value/probability of each digit from 0-9 so we OneHot encoded the data labels to approach the targeted shape.

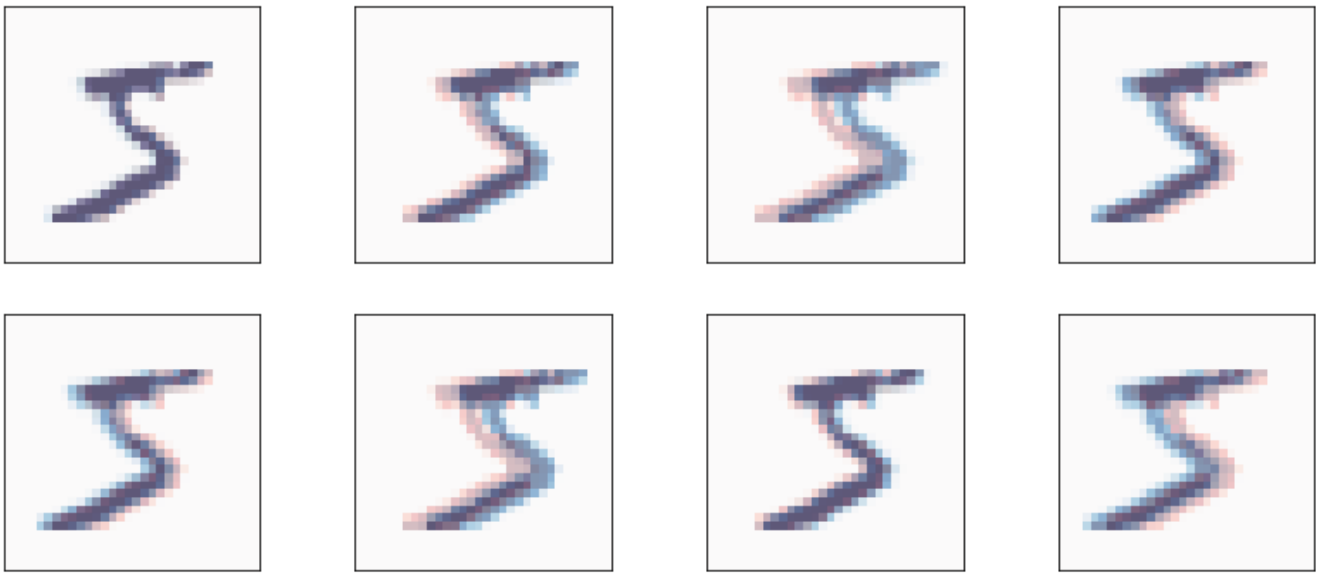
```
y_train_OneHot = to_categorical(y_train, num_classes = 10)
y_test_OneHot = to_categorical(y_test, num_classes = 10)
```

6. **Data augmentation**

Applying different transformations to input data by Shifting and scaling the input data by 10%, and rotation in range -10 to 10 degrees. this reduces the variance and allow model to generalize better to unseen data.

```
datagen = ImageDataGenerator(rotation_range=10,zoom_range = 0.1,width_shift_range=0.1,
height_shift_range=0.1)
datagen.fit(x_train_scaled)
```

we can see in the following figure how the generated output (Blue) is transformed from the original (Red)



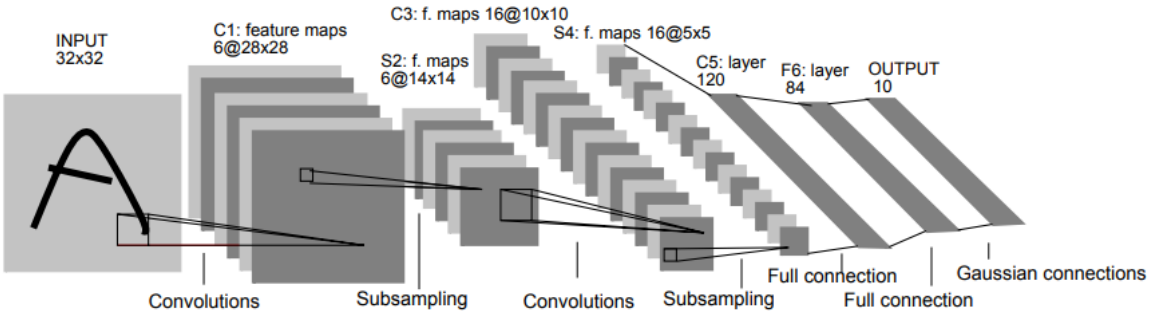
Transformed Training Samples

Implementation

LeNet-5 CNN

based on the paper[3] LeNet-5 CNN is considered as ideal for digit/character recognition so we are going to design our model base on it, with slight modifications.

1. **Structuring**



Architecture of LeNet-5

- **Input Layer**

consists of 32x32 pixel image the reason for padding that the digit can be more centered to be detected by the receptive field. the input pixels are grayscale with a value 0 for a white pixel and 1 for a black pixel

- **Convolution Layer**

It consists of 6 feature maps with Size 28x28 .the learnable kernels size is 5x5. the convolution layer is responsible for feature extraction from the input data.

- **ReLU activation function**

is used at the end of each convolution layer as well as a fully connected layer to enhance the performance of the model.

- **pooling layer**

It reduces the output information from the convolution layer and reduces the number of parameters and computational complexity of the model

- **Flatten layer**

is used after the pooling layer which converts the 2D featured map matrix to a 1D feature vector and allows the output to get handled by the fully connected
layers

- **fully connected layer**

Is another hidden layer also known as the dense layer. It is similar to the hidden layer of Artificial Neural Networks (ANNs) but here it is fully connected and connects every neuron from the previous layer to the next layer

- **output layer**

Consists of ten neurons and determines the digits numbered from 0 to 9. Since the output layer uses an activation function such as softmax, which is used to enhance the performance of the model

We made a slight modifications to the LeNet-5 model summarized on the following

- **Adding dropout regularization layers:**
  by randomly switching off 25% of neurons during training we make more robust CNN. this reduces overfitting and increase the generalization due to not relying too much on any particular neurons to produce an output.

- **Adding Batch Normalization layers:**
  Standard scaling the layers output makes the learning algorithm converge faster.

$$z_{norm} = \frac{z - \bar{z}}{\sqrt{{s_z}^2 + \epsilon}}$$
$$z = \gamma z_{norm} + \beta$$

- **Adding Kernel l2 regularization:**
  Increasing the loss causes the weights to be relatively smaller, which reduce overfitting we can approach that by penalizing the sum of the square of the weights (weight²) to the loss.

$$loss + \lambda \sum_{i=0}^{n} W_i^2$$

- **Adding more Dense layer:**
  Reduces the bias by allowing the model to learn more complex features of the input data.

Final Model Structure

```
model = Sequential()

model.add(layers.Conv2D(filters = 6, kernel_size = (5,5),padding='valid',
input_shape = (32,32,1),kernel_regularizer=l2(0.0005)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D(pool_size = (2,2), strides = (2,2)))

model.add(layers.Conv2D(filters = 16, kernel_size = (5,5),padding='valid',
kernel_regularizer=l2(0.0005)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D(pool_size = (2,2), strides = (2,2)))
model.add(layers.Flatten())

model.add(layers.Dense(units = 250))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.Dropout(0.25))

model.add(layers.Dense(units = 120))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.Dropout(0.25))

model.add(layers.Dense(units = 84))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))


model.add(layers.Dense(units = 10, activation = 'softmax'))
```

If we didn't pad the data manually earlier we could let the model pad the input with the shape (28,28,1) giving the same result using `padding='Same'` in the first layer

```
Model.add(layers.Conv2D(filters = 6, kernel_size = (5,5),padding = 'Same',
activation ='relu',input_shape = (28,28,1)))
```

2. Training

We used `ReduceLROnPlateau` callback function to reduce learning rate by `factor` when the validation loss metric has stopped improving for a `patience` number of epochs

$$\text{new\_lr} = \text{lr} * \text{factor}$$

dynamically reducing the learning rate helps the algorithm to converge faster and reach closer to the global minima.

```
learning_rate_reduction = callbacks.ReduceLROnPlateau(monitor='val_loss',patience=2,
factor=0.1,min_lr=0.00001)
```

we used adam optimizer as it combine the benefits of RMSProp and AdaGrad optimizers

- RMSProp - adapting learning rates based on the average of the first moment
- AdaGrad - adapting learning rates based on the average of the second moments

beta1 and beta2 control the decay rates of these moving averages.

beta1: The exponential decay rate for the first moment estimates (the mean) as in RMSProp
beta2:The exponential decay rate for the second-moment estimates (the non-centered variance) as in AdaGrad
Epsilon: Is a very small number to prevent any division by zero in the implementation

```
optimizer = optimizers.Adam(learning_rate=0.001,beta_1=0.9,beta_2=0.999,epsilon=1e-7)
```

as we dealing with multi-class classification problem with Softmax activation function output we used Categorical CrossEntropy

$$loss = -\sum_{i=1}^{output\ size} y_i . \log \hat{y}_i$$

- y_i OneHot encoded true labels
- $\hat{y}_i$ Softmax activation function output

```
loss = losses.CategoricalCrossentropy()
```

the targeted metric is accuracy as mentioned earlier

```
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
```

we trained the model for 30 epochs with batch size of 60,000/100 = 600 samples per epoch.

```
history = model.fit(datagen.flow(x_train_scaled,y_train_OneHot,batch_size=100),
                    epochs = 30,
                    validation_data = (x_test_scaled,y_test_OneHot),
                    callbacks=[learning_rate_reduction],
                    verbose = 1)
```

Results

Model Evaluation and Validation

We calculated the predicted labels by taking the argmax of Softmax function output.

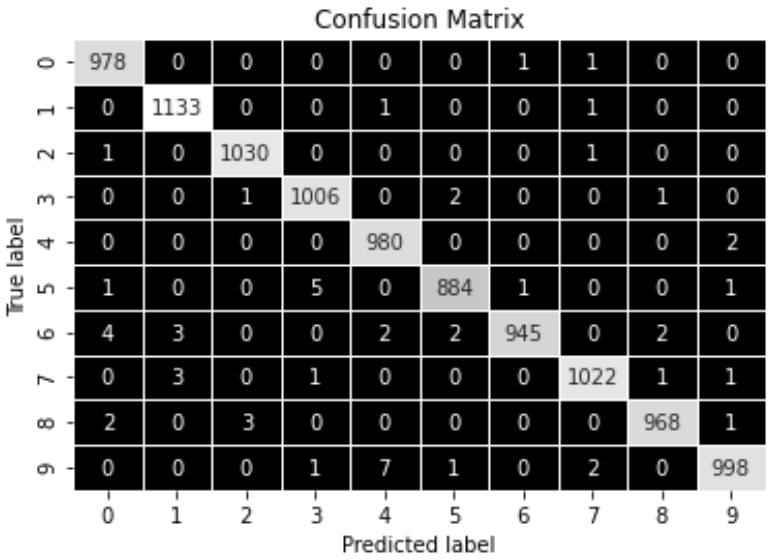
```
y_pred = np.argmax(model.predict(x_test_scaled), axis=-1)
```

the accuracy of the model is excellent as well as macro precision and macro recall

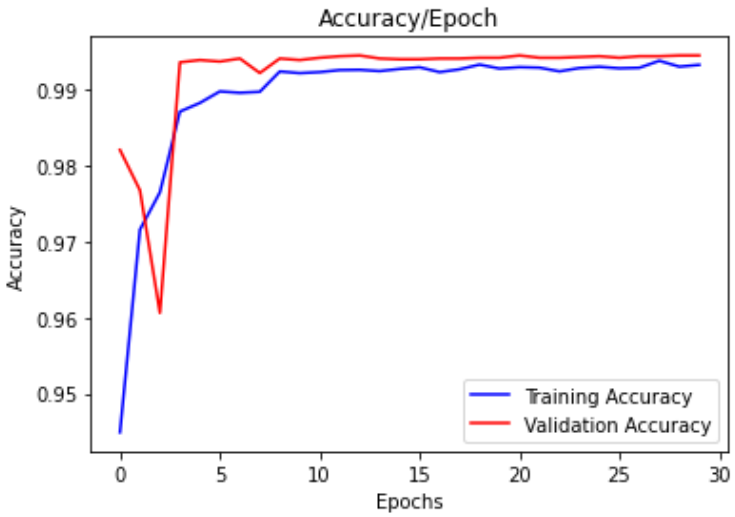
```
print(f'{"accuracy":10} {(y_pred == y_test).mean():.4f}')
print(f'{"precision":10} {precision_score(y_test,y_pred,average="macro"):.4f}')
print(f'{"recall":10} {recall_score(y_test,y_pred,average="macro"):.4f}')
```

- accuracy 0.9944
- precision 0.9944
- recall 0.9943

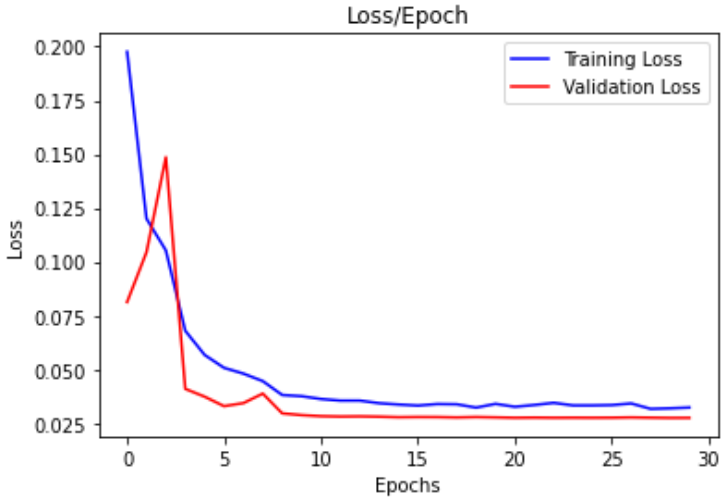
from the following confusion matrix the most miss predicted digit is 9 labeled as 4 due to the similarity between the two digits



The accuracy of the validation set had a drop on the second epoch then it has recovered in the following epoch



The loss of the validation set had a increased on the second epoch then it has recovered in the following epoch



Pre-trained Model Saving

```
data_dir = 'trained_models'
if not os.path.exists(data_dir):
    os.makedirs(data_dir)
model_1.save(f'{data_dir}/LeNet_5.h5')
```

References

- [1] "THE MNIST DATABASE of handwritten digits" Yann LeCun, Corinna Cortes, Christopher J.C. Burges http://yann.lecun.com/exdb/mnist/
- [2] "ARDIS: a Swedish historical handwritten digit dataset" Huseyin Kusetogullari, Amir Yavariabdi, Abbas Cheddad March 2019 https://link.springer.com/article/10.1007/s00521-019-04163-3
- [3] "GradientBased Learning Applied to Document Recognition" Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner ,November 1998 http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf