Data Science Intern at Data Glacier

Week 4: Deployment on Flask

Name: Yash Jadwani

Batch Code: LISUM19

Date: 24 March 2023

Submitted to: Data Glacier

Table of contents

1) Introduction

The MNIST dataset is a classic benchmark dataset in machine learning that consists of 70,000 handwritten digits. The goal of this project is to deploy a neural network model that can recognize handwritten digits on a web app using Flask. The tools and technologies used in this project are Python, Flask, and TensorFlow. The MNIST dataset consists of 60,000 training images and 10,000 testing images. Each image is a 28x28 pixel grayscale image, which means it has only one colour channel.

2) Data pre-processing

Before training the neural network model, we need to pre-process the data in the following ways:

Scaling: We scale the pixel values to be between 0 and 1, which helps the model converge faster during training.
Normalization: We normalize the pixel values to have zero mean and unit variance, which makes the model more robust to changes in lighting conditions and reduces the effects of differences in image backgrounds.
One-hot encoding: we have done one-hot encode the target labels, which means we convert each label (which is a digit from 0 to 9) into a vector of length 10, where the index of the digit is set to 1 and all other indices are set to 0.

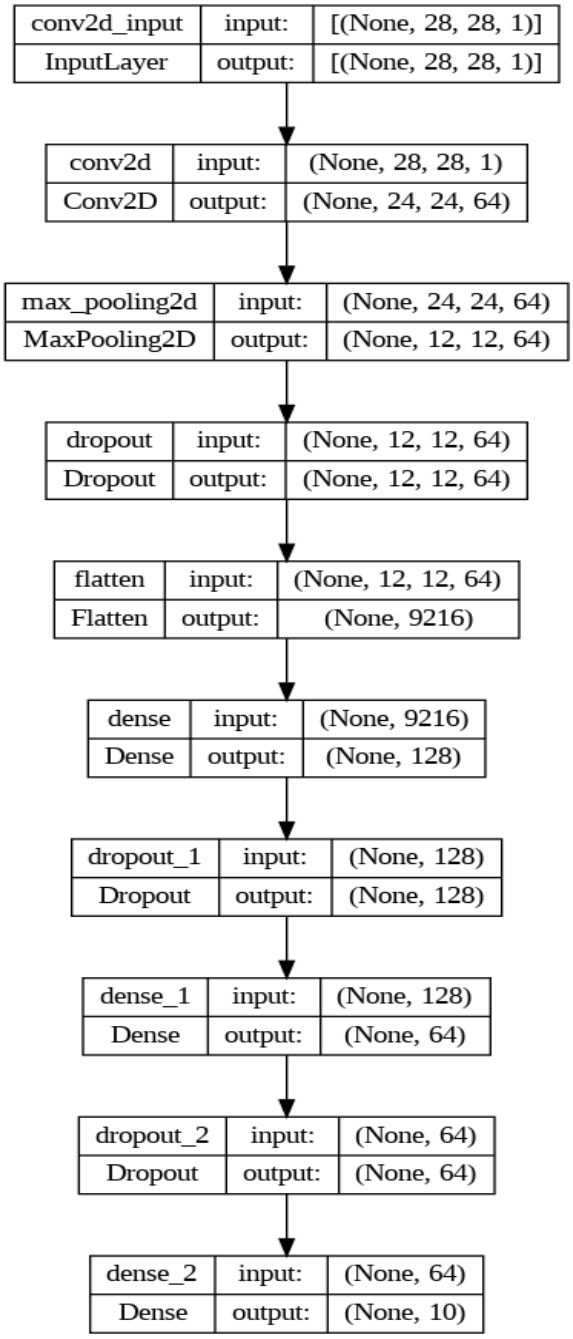
3) Neural Network

```
[13] network = Sequential()
```

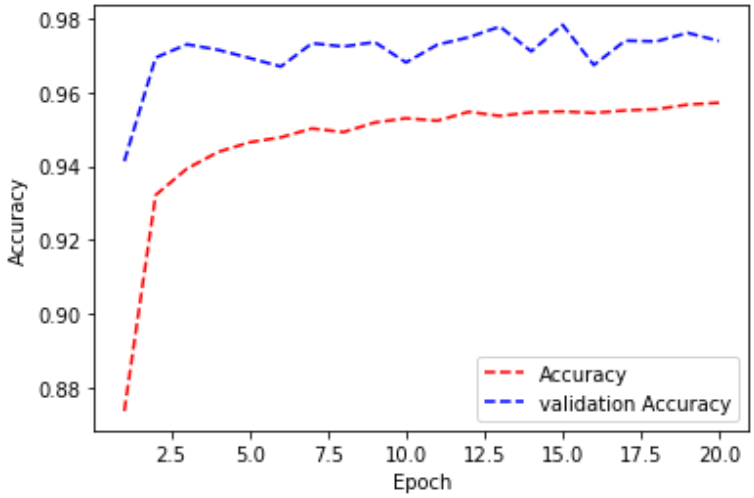
```
## layer1
network.add(Conv2D(filters=64,kernel_size=(5,5),input_shape=(width,height,channels), activation ='relu'))
network.add(MaxPooling2D(pool_size=(2,2)))
network.add(Dropout(0.5))

network.add(Flatten())

##layer2
network.add(Dense(128, activation="relu",kernel_regularizer=regularizers.l2(0.01)))
network.add(Dropout(0.3))

##layer3
network.add(Dense(64, activation="relu",kernel_regularizer=regularizers.l2(0.01)))
network.add(Dropout(0.3))

## output layer
network.add(Dense(number_of_classes, activation="softmax"))
```

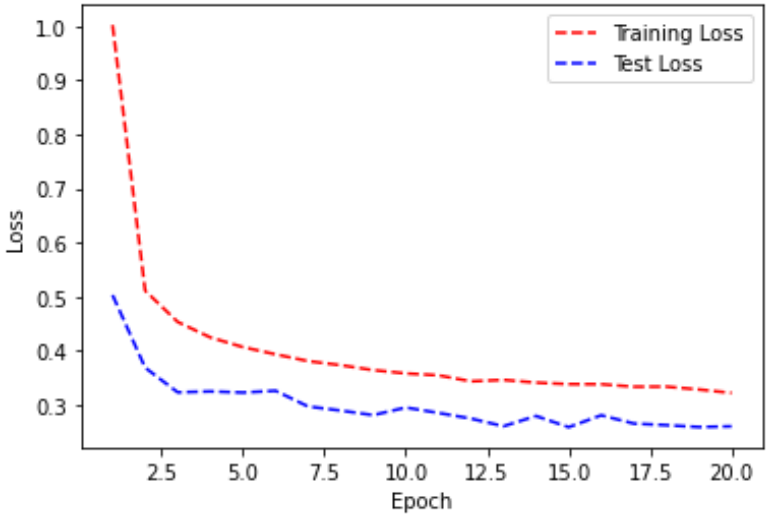
```
[15] ## opt = tf.keras.optimizers.RMSprop(learning_rate=0.5)
     network.compile(loss="categorical_crossentropy",
                     optimizer='rmsprop',
                     metrics = ["accuracy"])
```

```
[16] history = network.fit(feature_train,
                           target_train,
                     epochs =20,
                     verbose=1,
                     batch_size=100,
                     validation_data= (feature_test, target_test))
```



We have developed a sequential neural network model with three layers, using the NHWC channel format. The model was trained using the categorical cross-entropy loss function and the RMSprop optimizer, with the ReLU activation function. We also implemented regularization techniques and dropout layers to prevent overfitting.

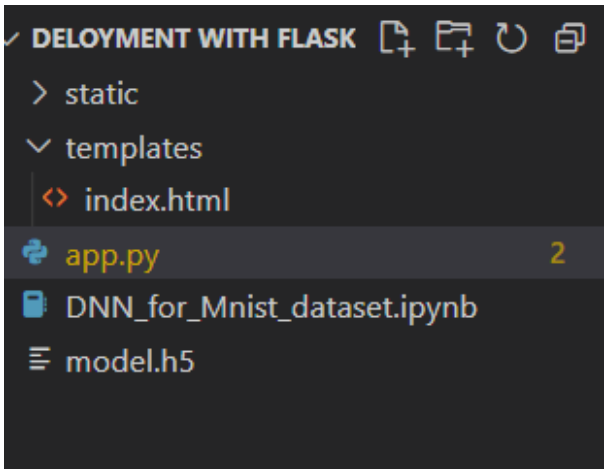




4) Deployment on Flask

After making model we have need to create a Flask Web app for that we need to save the model. I have used Keras HDF5 format to save the file over pickle because of CNN was used in the model.

```
[29] # Saving the model for Future Inferences

     network.save("model.h5")
```



This is Flolder containing all the Files required to deploy the MNIST model using Flask.

app.py

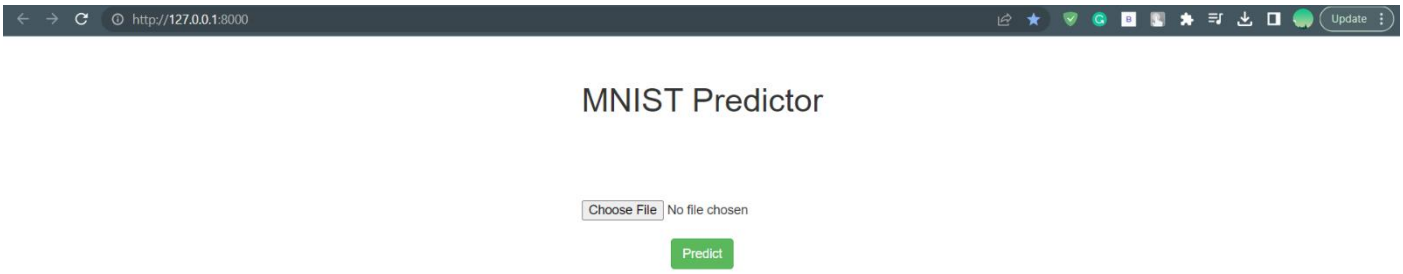
```
app = Flask(__name__)
model = load_model('model.h5',compile=False)
model.compile(optimizer='rmsprop',
    loss='categorical_crossentropy',
    metrics=None,
    loss_weights=None,
    weighted_metrics=None,
    run_eagerly=None,
    steps_per_execution=None,
    jit_compile=None)

target_img = os.path.join(os.getcwd() , 'static/images')
@app.route('/')
def index_view():
    return render_template('index.html')
```

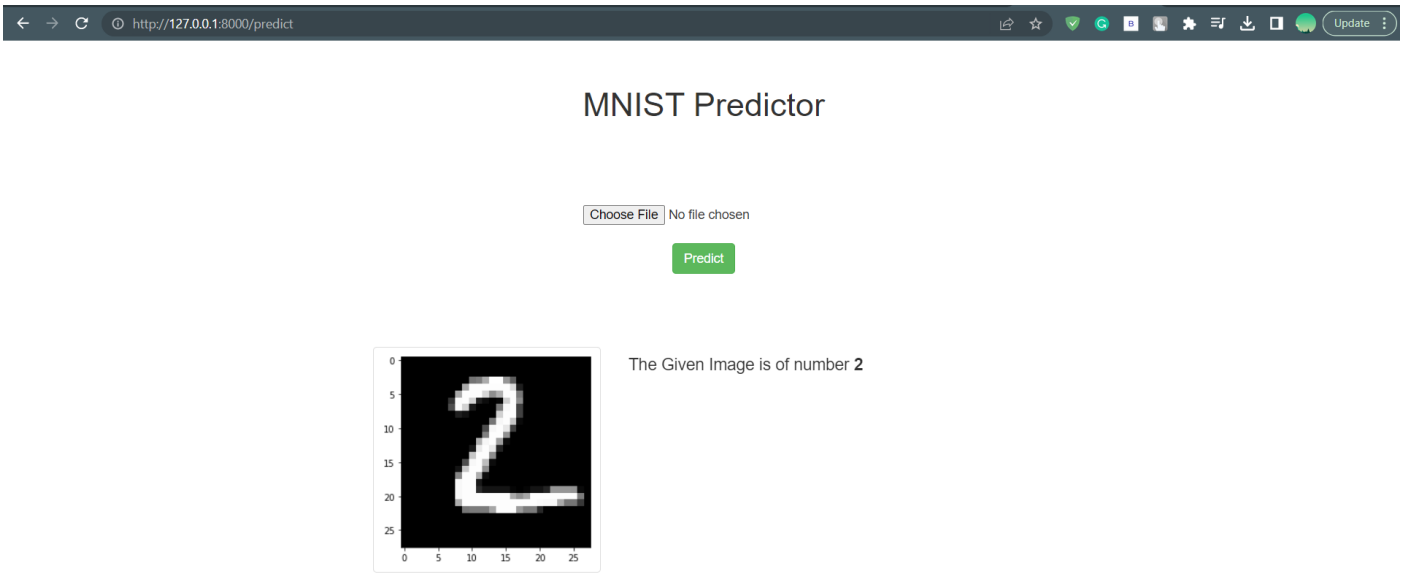
app.py is the centrepiece of the deployment, as it connects the trained model with the rendering of HTML templates for displaying the web application's user interface. One important method in app.py is **predict(),** which takes a user's input image and passes it through the **MNIST** neural network model to predict the digit label. However, before passing the image through the model, **predict()** applies file type checks and image pre-processing to ensure that the user's image is in the same format as the model's input data. This is necessary to obtain accurate prediction results. Once the digit label is predicted, it is displayed in the **web interface** for the user to see.

```python
ALLOWED_EXT = set(['jpg' , 'jpeg' , 'png'])
def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1] in ALLOWED_EXT

# Function to load and prepare the image in right shape
@app.route('/predict',methods=['GET','POST'])
def predict():
    if request.method == 'POST':
        file = request.files['file']
        if file and allowed_file(file.filename): #Checking file format
            filename = file.filename
            file_path = os.path.join('static/images', filename)
            file.save(file_path)
            img = cv2.imread(file_path)
            img = cv2.resize(img,(28,28))
            img = np.resize(img, (1, 28, 28, 1))
            img = img/255.0
            img = 1 - img

            predict_prob=model.predict(img)
            prediction=np.argmax(predict_prob,axis=1)
            return render_template('index.html', prediction = prediction[0], user_image = file_path)
        else:
            return "Unable to read the file. Please check file extension"
if __name__ == '__main__':
    app.run(debug=True,use_reloader=False, port=8000)
```

5) Webpage

Below are screentshots of simple webpage which is created by basic HTML and CSS which lets user to upload the image and the get the predicted results with the image they have uploaded





6) Conclusion:

In conclusion, we have successfully deployed a neural network model for recognizing handwritten digits on a Flask web app. The model was trained on the MNIST dataset using TensorFlow, and the web app was created using Flask. We deployed the app to a cloud server using a virtual machine and evaluated its performance based on accuracy and response time. Overall, the app provides a convenient way for users to input handwritten digits and receive a prediction of the digit's value. Future work could involve improving the accuracy of the model or adding additional features to the app.

Note: All the Files and whole folder along with the video of whole process can be found on GitHub link(https://github.com/yashjadwani/Deloyment-with-flask-Week4)