



**Data Glacier**

# **Data Science Intern at Data Glacier**

## **Week 5: Cloud and API Deployment**

**Name:** Yash Jadwani

**Batch Code:** LISUM19

**Date:** 02 April 2023

**Submitted to:** Data Glacier

## Table of contents

<b>1) Introduction.....</b>	<b>3</b>
<b>2) Data pre-processing .....</b>	<b>3</b>
<b>3) Neural Network .....</b>	<b>3</b>
<b>4) Deployment on Streamlit.....</b>	<b>4</b>
<b>5) Webpage.....</b>	<b>8</b>

## 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 Streamlit. The tools and technologies used in this project are Python, Streamlit, 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())

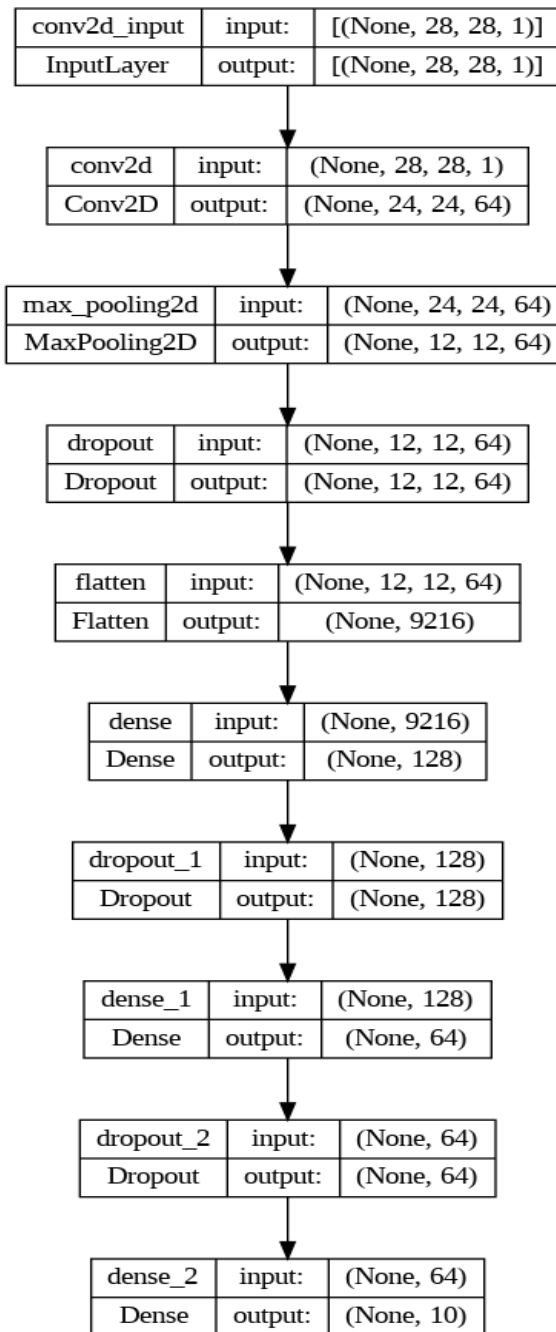
##layer2
network.add(Dense(128, activation="relu", kernel_regularizer=regularizers.l2(0.01)))
network.add(Dropout(0.3))

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

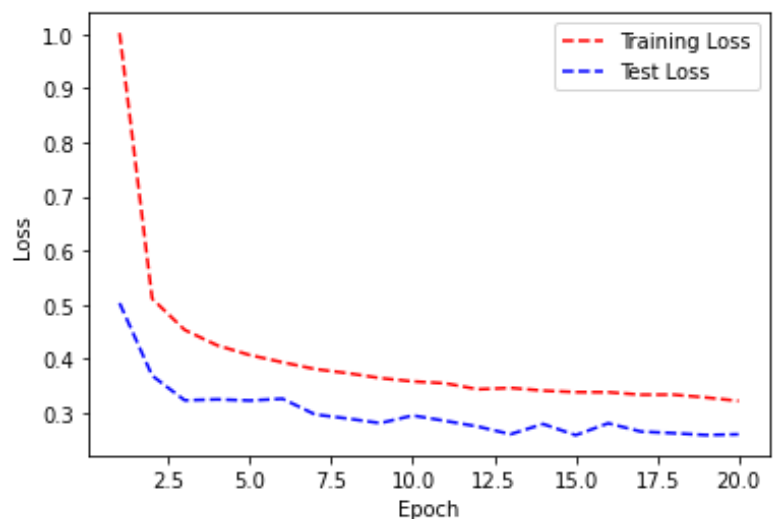
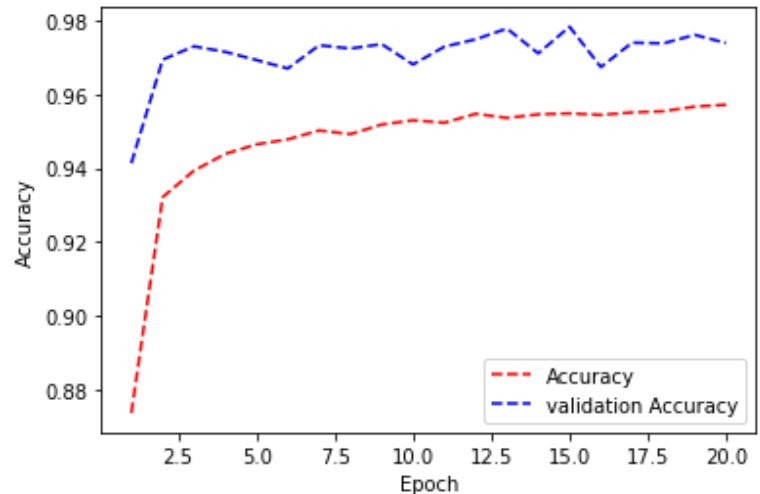
## output layer
network.add(Dense(number_of_classes, activation="softmax"))

[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 Streamlit

After making model we need to create a Streamlit 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
js
network.save("model.h5")

```

To deploy a web app to Streamlit, you'll first need to connect your GitHub repository with the Streamlit web app. Your repository should contain some necessary files, including a **Procfile**, **requirements.txt**, **setup.sh**, and your **main app.py** file.

The **Procfile** specifies the commands to run your app and tells Streamlit what command to use to start your app and what port to listen on.

```
Procfile
1 web: sh setup.sh && streamlit run app.py
```

The **requirements.txt** file lists the Python packages that your app depends on. Streamlit uses this file to create a virtual environment and install the required packages.

```
requirements.txt
1 tensorflow-cpu
2 streamlit ==1.7.0
3 numpy ==1.21.3
4 imblearn ==0.0
5 matplotlib ==3.4.3
6 opencv-python ==4.5.5.64
7
8
9
```

The **setup.sh** file is a script that Streamlit runs during deployment to set up your app environment. You can use this file to perform additional tasks, such as installing system dependencies or setting environment variables.

```
$ setup.sh
1 mkdir -p ~/.streamlit/
2 echo "\
3 [server]\n\
4 port = $PORT\n\
5 enableCORS = false\n\
6 headless = true\n\
7 \n\
8 " > ~/.streamlit/config.toml
```

Your **main app.py** file is where you write your Streamlit app code. This file should contain the code that sets up the app and defines the user interface, as well as any other necessary logic.

Once you have all of these files set up in your repository, you can connect it to the Streamlit web app and deploy your app. From there, you can make any necessary changes to your app code, update your repository, and deploy new versions of your app with ease. Streamlit also provides tools for managing app settings, monitoring app usage, and collaborating with other developers on your team.

## app.py

```

app.py > ...
1  import streamlit as st
2  from PIL import Image, ImageOps
3  import matplotlib.pyplot as plt
4
5  import tensorflow as tf
6  import numpy as np
7  from tensorflow import keras
8  from tensorflow.keras.models import load_model
9  from tensorflow.keras import preprocessing
10 import time
11 import warnings
12
13 warnings.filterwarnings("ignore", category=DeprecationWarning)
14
15 fig = plt.figure(figsize=(3,3))
16
17 st.set_page_config(layout="wide")
18
19 with open("custom.css") as f:
20     st.markdown(f"<style>{f.read()}</style>", unsafe_allow_html=True)
21
22 st.title('MNIST Predictor')
23
24 st.markdown("Upload a grayscale image of a handwritten digit (28x28 pixels)")
25

```

In an app that uses the MNIST model, `app.py` would typically include code for loading and preprocessing the MNIST dataset, as well as defining and training the machine learning model. This would involve using a deep learning library such as TensorFlow to define the model architecture, compile the model, and fit the model to the training data.

Once the model is trained, `app.py` would also include code for making predictions on new data. This might involve defining a `predict()` method that takes an input image, preprocesses it, and feeds it through the trained model to generate a prediction.

The `predict()` method would then return the predicted label, which could be displayed to the user along with the input image. This might involve using Streamlit's API to create a text or image widget that displays the predicted label, along with any other relevant information about the prediction.

```

app.py > ...
24 st.markdown("Upload a grayscale image of a handwritten digit (28x28 pixels)")
25
26
27 def main():
28     file_uploaded = st.file_uploader("Choose File", type=["png","jpg","jpeg"])
29     if file_uploaded is not None:
30         image = tf.keras.preprocessing.image.load_img(file_uploaded, target_size=(28, 28), color_mode='grayscale')
31         class_btn = st.button("Classify")
32
33         predictions = None
34
35     if class_btn:
36         if file_uploaded is None:
37             st.write("Invalid command, please upload an image")
38         else:
39             with st.spinner('Model working....'):
40
41                 predictions = predict(image)
42                 time.sleep(1)
43
44                 st.success(predictions)
45
46                 col1, col2 = st.columns(2)
47                 with col1:
48                     st.markdown('<h3 style="text-align:centre;" >Uploaded Image</h3>',unsafe_allow_html=True)
49                     st.image(file_uploaded, use_column_width= True)
50                 with col2:
51                     if predictions is not None:
52                         st.markdown('<h3 style="text-align:centre;" >HeatMap of the uploaded Image</h3>',unsafe_allow_html=True)
53                         plt.imshow(image)
54                         plt.axis("off")
55                         st.pyplot(fig)
56
57

```

```

app.py > ...
51
52     with col2:
53         if predictions is not None:
54             st.markdown('<h3 style="text-align:centre;" >HeatMap of the uploaded Image</h3>',unsafe_allow_html=True)
55             plt.imshow(image)
56             plt.axis("off")
57             st.pyplot(fig)
58
59 def predict(image):
60     classifier_model = "model.h5"
61
62     model = load_model(classifier_model,compile=False,)
63
64     image = np.asarray(image)
65     image = np.expand_dims(image, axis=0)
66     image = image.astype('float32') / 255.0
67
68     predict_prob = model.predict(image)
69     prediction=np.argmax(predict_prob,axis=1)
70
71     result = f"The Given Image is of number: {prediction[0]}"
72     return result
73
74
75
76 if __name__ == "__main__":
77     main()
78

```

Overall, the code in app.py for an MNIST model would involve a combination of data pre-processing, machine learning model definition and training, and prediction generation, all of which would be integrated into the Streamlit app to create an interactive user experience.

## 5) Webpage


Below are screenshots 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.

≡

### MNIST Predictor

Upload a grayscale image of a handwritten digit (28x28 pixels)

Choose File




Drag and drop file here  
Limit 200MB per file • PNG, JPG, JPEG

Browse files


Classify

The Given Image is of number: 2

Uploaded Image



HeatMap of the uploaded Image



This Webpage can be found at <https://yashjadwani-deployment-with-streamlit-week5-app-dwpiev.streamlit.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/Deployment-with-streamlit-week5>)