```
#importing the dataset and libraries

import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras import regularizers
import plotly.graph_objects as go


(X_train, y_train), (X_test, y_test) = mnist.load_data()    #Loading the dataset

#Dataset split to validation and train and normalizing the greyscale intensities
X_train = X_train.reshape(-1, 784) / 255.0
X_test = X_test.reshape(-1, 784) / 255.0

X_validation, X_train = X_train[:5000], X_train[5000:]
y_validation, y_train = y_train[:5000], y_train[5000:]
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
11490434/11490434 [==============================] - 0s 0us/step
```

```
#Printing the shapes
X_train.shape, X_validation.shape, X_test.shape, y_train.shape, y_validation.shape,
```

```
((55000, 784), (5000, 784), (10000, 784), (55000,), (5000,), (10000,))
```

```
#Deep Neural Network with 4 layers, relu as activation for hidden layers and regula
#l2 regularization with C = 0.01, softmax as activation for the output layer

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(784,)),
    tf.keras.layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l
    tf.keras.layers.Dense(64, activation='relu', kernel_regularizer=regularizers.l2
    tf.keras.layers.Dense(10, activation='softmax')
])


#Compiling the model with SGD Optimizer with sparse categorical cross entropy as th
model.compile(optimizer='SGD', loss='sparse_categorical_crossentropy', metrics=['ac
```

```
#Fitting the model and saving the values to history_regularized_model for plotting
history_regularized_model = model.fit(X_train, y_train, validation_data=(X_validati
```

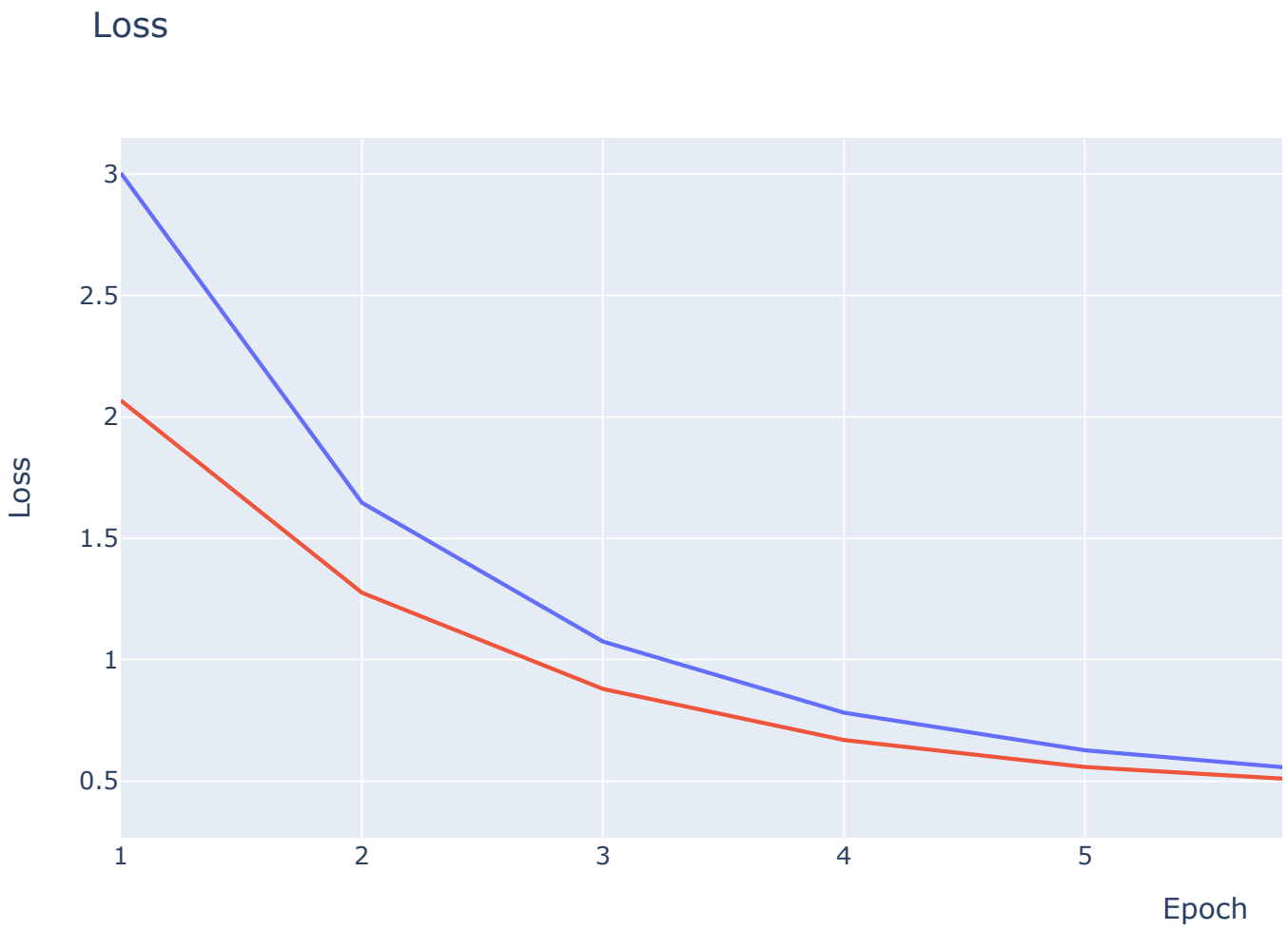
```
Epoch 1/10
1719/1719 [==============================] - 7s 4ms/step - loss: 3.0041 - accu
Epoch 2/10
1719/1719 [==============================] - 4s 3ms/step - loss: 1.6469 - accu
Epoch 3/10
1719/1719 [==============================] - 3s 2ms/step - loss: 1.0749 - accu
Epoch 4/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.7820 - accu
Epoch 5/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.6272 - accu
Epoch 6/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.5423 - accu
Epoch 7/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.4937 - accu
Epoch 8/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.4636 - accu
Epoch 9/10
1719/1719 [==============================] - 4s 2ms/step - loss: 0.4429 - accu
Epoch 10/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.4284 - accu
```

```
#Accessing the different metrics and saving it to plot the graph
loss = history_regularized_model.history['loss']
val_loss = history_regularized_model.history['val_loss']
accuracy = history_regularized_model.history['accuracy']
val_accuracy = history_regularized_model.history['val_accuracy']
```

```
#Loss over Epochs

def loss_accuracy_figure(input_1, input_2, kind):
    fig_loss = go.Figure()
    fig_loss.add_trace(go.Scatter(x=list(range(1, len(input_1)+1)), y=input_1, mode
    fig_loss.add_trace(go.Scatter(x=list(range(1, len(input_2)+1)), y=input_2, mode
    fig_loss.update_layout(title=f'{kind}',
                      xaxis_title='Epoch',
                      yaxis_title=f'{kind}')
    fig_loss.show()
```

```
loss_accuracy_figure(loss, val_loss, 'Loss')
```



```
#Accuracy over Epochs
loss_accuracy_figure(accuracy, val_accuracy, 'Accuracy')
```