```
#Importing the dataset and libraries
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
import plotly.graph_objects as go
from tensorflow.keras import regularizers
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping



#Checking for GPU availability

if tf.test.gpu_device_name():
    print('GPU device found: {}'.format(tf.test.gpu_device_name()))
    device = '/device:GPU:0'
else:
    print('No GPU device found. Using CPU.')
    device = '/device:CPU:0'
```

```
GPU device found: /device:GPU:0
```

```
(x_train, y_train),(x_test, y_test) = mnist.load_data() #Splitting data into test a

x_train = x_train.reshape(-1, 784) / 255.0              #Flattening the (28X28) dat
x_test = x_test.reshape(-1, 784) / 255.0                #and normalizing the greysc


x_validation, x_train = x_train[:5000], x_train[5000:]  #5000 data points for valic
y_validation, y_train = y_train[:5000], y_train[5000:]

y_train = tf.keras.utils.to_categorical(y_train)
y_validation = tf.keras.utils.to_categorical(y_validation)
y_test = tf.keras.utils.to_categorical(y_test)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
11490434/11490434 [==============================] - 1s 0us/step
```

```
x_train.shape, x_validation.shape, x_test.shape, y_train.shape, y_validation.shape,
```

```
((55000, 784), (5000, 784), (10000, 784), (55000, 10), (5000, 10), (10000,
10))
```

```
#Neural Network with 3 layers, relu as activation for hidden layers
#with softmax as activation for the output layer. Regularizer used: l2 and C: 0.001
#dropout set to 50%
model = Sequential()
model.add(Dense(500, activation='relu', kernel_regularizer=l2(0.001), input_shape=(
model.add(Dropout(0.5))
model.add(Dense(500, activation='relu', kernel_regularizer=l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

```
#Compiling the model with Adam Optimizer with categorical cross entropy as the loss
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy
```

```
#Setting batch size = 128 and epochs to 250
batch_size = 128
epochs = 250
```

```
with tf.device(device):

    loss_values = []
    error_values = []

    #each iteration of epochs
    for epoch in range(epochs):

        #each iteration of batch
        for batch_start in range(0, len(x_train), batch_size):
            batch_end = batch_start + batch_size
            x_batch = x_train[batch_start:batch_end]
            y_batch = y_train[batch_start:batch_end]

            model.train_on_batch(x_batch, y_batch)

        #calculating the loss and accuracy
        loss, accuracy = model.evaluate(x_validation, y_validation, batch_size=batc

        #calculating the error
```

```
    error = 1 - accuracy
    loss_values.append(loss)
    error_values.append(error)

    print(f"Epoch {epoch+1}/{epochs} - Loss: {loss:.4f} - Error: {error:.4f}")
```

```
Epoch 83/250 - Loss: 0.2430 - Error: 0.0278
Epoch 84/250 - Loss: 0.2448 - Error: 0.0298
Epoch 85/250 - Loss: 0.2501 - Error: 0.0298
Epoch 86/250 - Loss: 0.2345 - Error: 0.0248
Epoch 87/250 - Loss: 0.2357 - Error: 0.0270
Epoch 88/250 - Loss: 0.2447 - Error: 0.0286
Epoch 89/250 - Loss: 0.2382 - Error: 0.0280
Epoch 90/250 - Loss: 0.2480 - Error: 0.0306
Epoch 91/250 - Loss: 0.2374 - Error: 0.0276
Epoch 92/250 - Loss: 0.2324 - Error: 0.0262
Epoch 93/250 - Loss: 0.2425 - Error: 0.0266
Epoch 94/250 - Loss: 0.2341 - Error: 0.0276
Epoch 95/250 - Loss: 0.2347 - Error: 0.0256
Epoch 96/250 - Loss: 0.2358 - Error: 0.0262
Epoch 97/250 - Loss: 0.2387 - Error: 0.0256
Epoch 98/250 - Loss: 0.2443 - Error: 0.0280
Epoch 99/250 - Loss: 0.2377 - Error: 0.0276
Epoch 100/250 - Loss: 0.2400 - Error: 0.0278
Epoch 101/250 - Loss: 0.2320 - Error: 0.0254
Epoch 102/250 - Loss: 0.2350 - Error: 0.0270
Epoch 103/250 - Loss: 0.2438 - Error: 0.0284
Epoch 104/250 - Loss: 0.2356 - Error: 0.0260
Epoch 105/250 - Loss: 0.2446 - Error: 0.0296
Epoch 106/250 - Loss: 0.2322 - Error: 0.0252
Epoch 107/250 - Loss: 0.2293 - Error: 0.0256
Epoch 108/250 - Loss: 0.2386 - Error: 0.0280
Epoch 109/250 - Loss: 0.2451 - Error: 0.0326
Epoch 110/250 - Loss: 0.2336 - Error: 0.0258
Epoch 111/250 - Loss: 0.2314 - Error: 0.0256
Epoch 112/250 - Loss: 0.2324 - Error: 0.0258
Epoch 113/250 - Loss: 0.2347 - Error: 0.0274
Epoch 114/250 - Loss: 0.2362 - Error: 0.0300
Epoch 115/250 - Loss: 0.2351 - Error: 0.0280
Epoch 116/250 - Loss: 0.2347 - Error: 0.0286
Epoch 117/250 - Loss: 0.2299 - Error: 0.0244
Epoch 118/250 - Loss: 0.2388 - Error: 0.0272
Epoch 119/250 - Loss: 0.2366 - Error: 0.0314
Epoch 120/250 - Loss: 0.2409 - Error: 0.0296
Epoch 121/250 - Loss: 0.2290 - Error: 0.0266
Epoch 122/250 - Loss: 0.2312 - Error: 0.0258
Epoch 123/250 - Loss: 0.2327 - Error: 0.0284
Epoch 124/250 - Loss: 0.2366 - Error: 0.0276
Epoch 125/250 - Loss: 0.2360 - Error: 0.0284
Epoch 126/250 - Loss: 0.2343 - Error: 0.0258
```

```
Epoch 127/250 - Loss: 0.2274 - Error: 0.0254
Epoch 128/250 - Loss: 0.2365 - Error: 0.0270
Epoch 129/250 - Loss: 0.2338 - Error: 0.0280
Epoch 130/250 - Loss: 0.2464 - Error: 0.0318
Epoch 131/250 - Loss: 0.2377 - Error: 0.0294
Epoch 132/250 - Loss: 0.2290 - Error: 0.0280
Epoch 133/250 - Loss: 0.2331 - Error: 0.0266
Epoch 134/250 - Loss: 0.2253 - Error: 0.0226
Epoch 135/250 - Loss: 0.2348 - Error: 0.0276
Epoch 136/250 - Loss: 0.2292 - Error: 0.0252
Epoch 137/250 - Loss: 0.2315 - Error: 0.0270
Epoch 138/250 - Loss: 0.2484 - Error: 0.0340
Epoch 139/250 - Loss: 0.2353 - Error: 0.0278
Epoch 140/250 - Loss: 0.2320 - Error: 0.0246
Epoch 141/250 - Loss: 0.2336 - Error: 0.0272
Epoch 142/250 - Loss: 0.2311 - Error: 0.0262
```

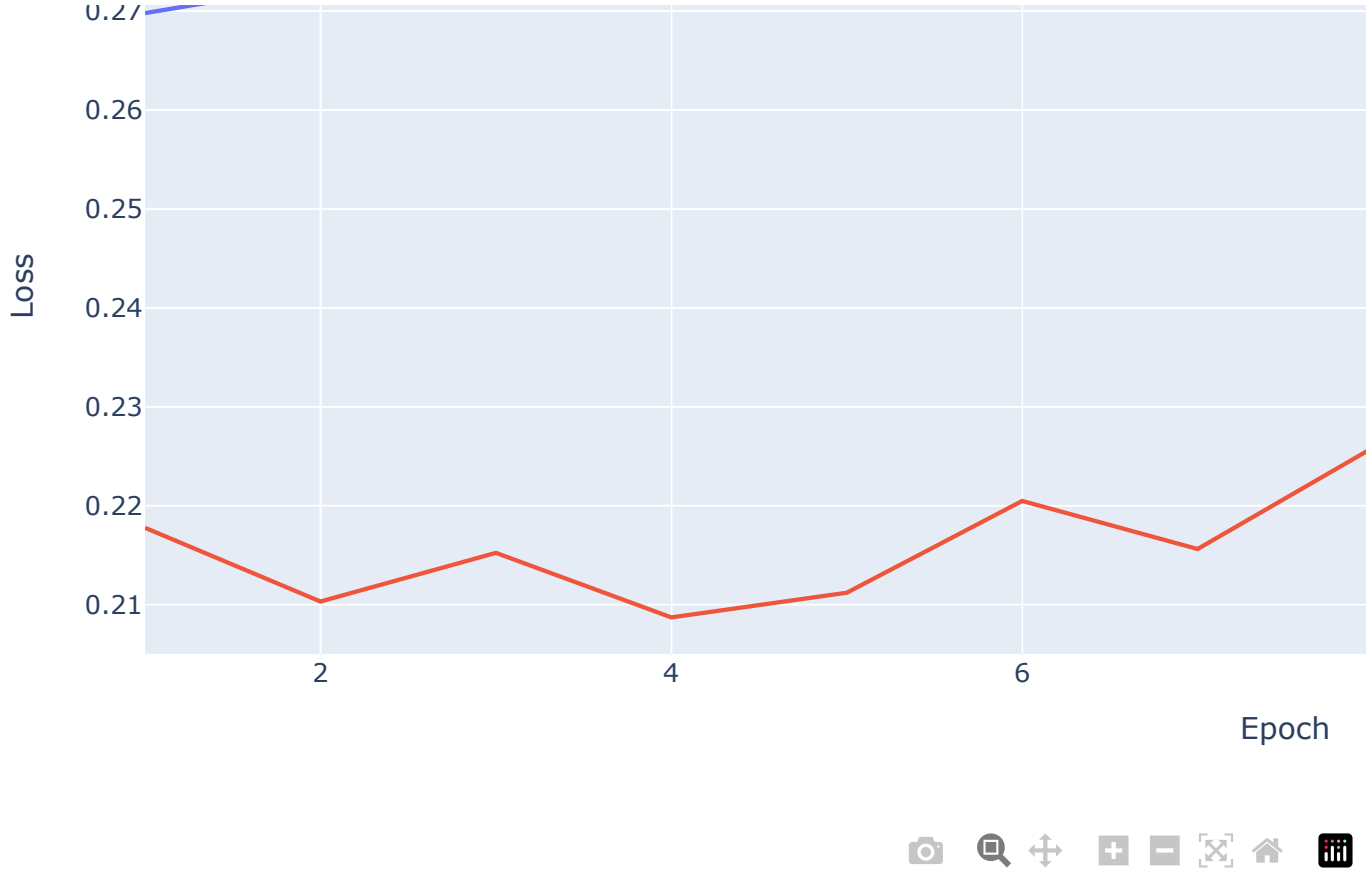
```
with tf.device(device):
    #Early Stopping
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1, rest

    history = model.fit(
        x_train,
        y_train,
        batch_size=batch_size,
        epochs=epochs,
        validation_data=(x_test, y_test),
        callbacks=[early_stopping]
    )
```

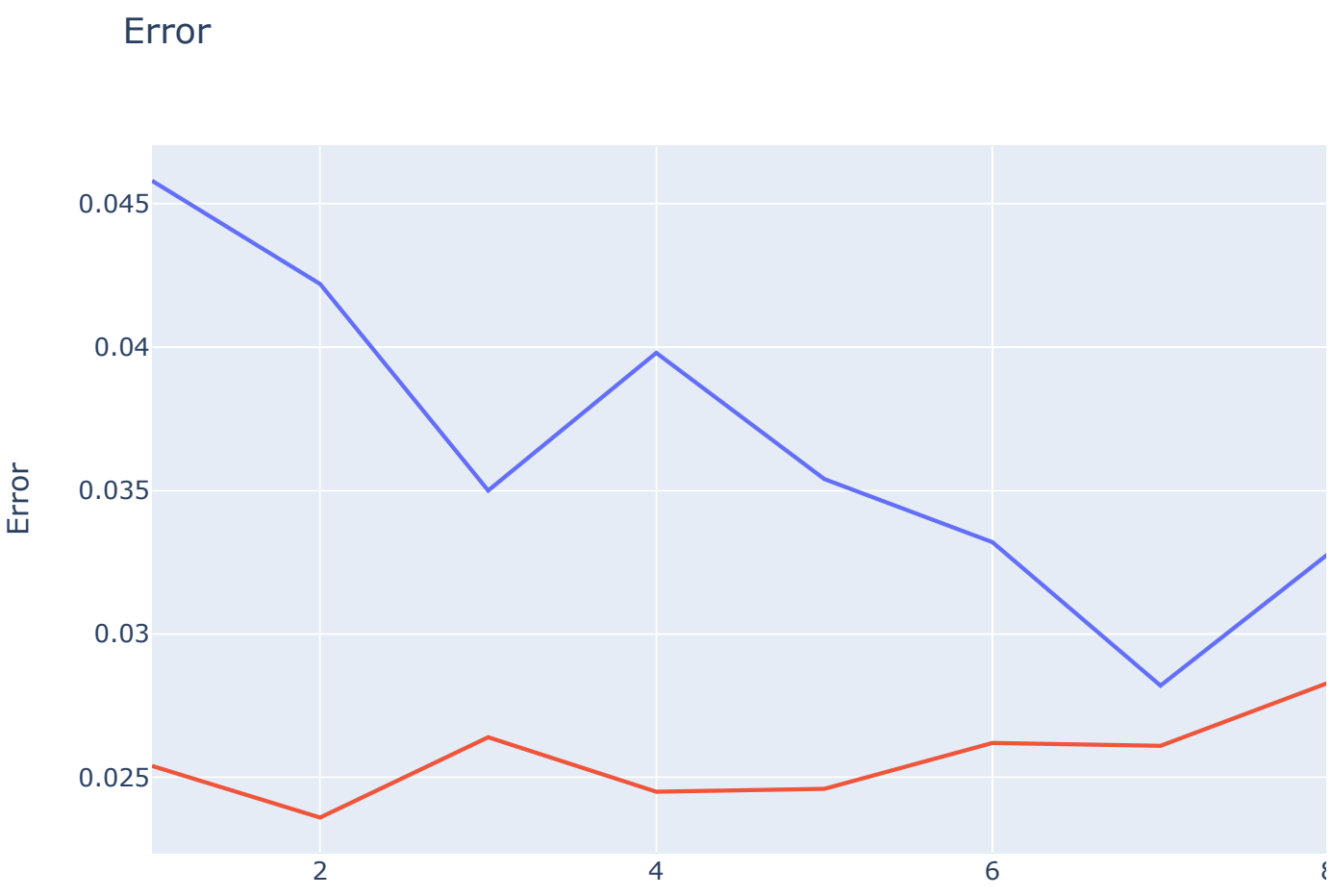
```
Epoch 1/250
430/430 [==============================] - 3s 6ms/step - loss: 0.2698 - accura
Epoch 2/250
430/430 [==============================] - 2s 5ms/step - loss: 0.2727 - accura
Epoch 3/250
430/430 [==============================] - 2s 5ms/step - loss: 0.2757 - accura
Epoch 4/250
430/430 [==============================] - 2s 5ms/step - loss: 0.2712 - accura
Epoch 5/250
430/430 [==============================] - 2s 5ms/step - loss: 0.2724 - accura
Epoch 6/250
430/430 [==============================] - 2s 4ms/step - loss: 0.2747 - accura
Epoch 7/250
430/430 [==============================] - 3s 6ms/step - loss: 0.2737 - accura
Epoch 8/250
430/430 [==============================] - 2s 5ms/step - loss: 0.2723 - accura
Epoch 9/250
430/430 [==============================] - 2s 5ms/step - loss: 0.2722 - accura
Epoch 10/250
430/430 [==============================] - 2s 5ms/step - loss: 0.2705 - accura
Epoch 11/250
430/430 [==============================] - 2s 5ms/step - loss: 0.2716 - accura
Epoch 12/250
430/430 [==============================] - 2s 4ms/step - loss: 0.2718 - accura
Epoch 13/250
430/430 [==============================] - 3s 6ms/step - loss: 0.2750 - accura
Epoch 14/250
427/430 [============================>.] - ETA: 0s - loss: 0.2734 - accuracy:
430/430 [==============================] - 2s 5ms/step - loss: 0.2733 - accura
Epoch 14: early stopping
```

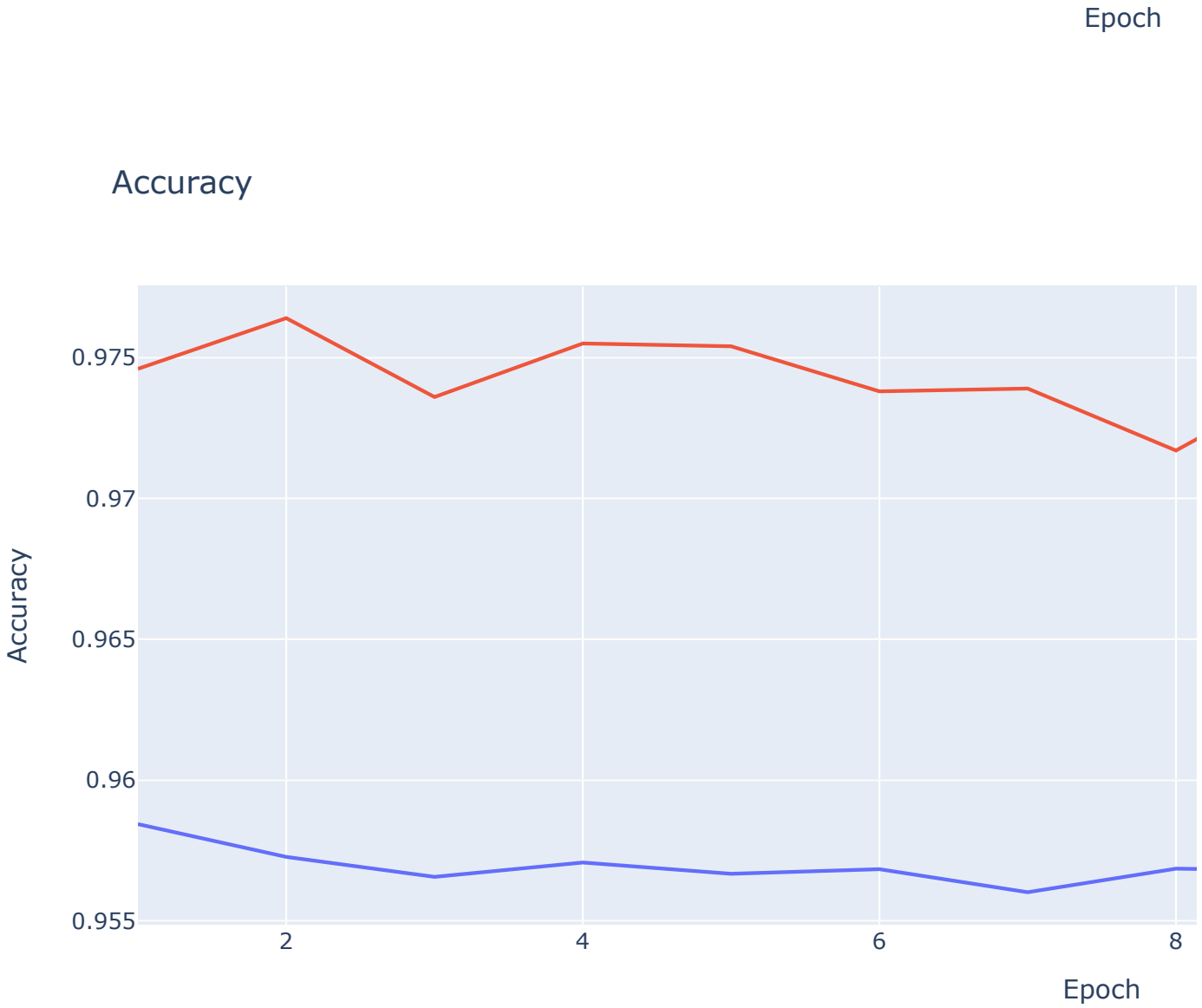
```
#Accessing the different metrics and saving it to plot the graph
loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
acc_values = history.history['accuracy']
val_acc_values = history.history['val_accuracy']


print(val_error_values)
```

```
    [0.025399982929229736, 0.023599982261657715, 0.02640002965927124, 0.024500012
```

```
epochs_range = range(1, len(loss_values) + 1)

# Create the loss plot
loss_plot = go.Scatter(x=list(epochs_range), y=loss_values, mode='lines', name='Tra
val_loss_plot = go.Scatter(x=list(epochs_range), y=val_loss_values, mode='lines', r
layout_loss = go.Layout(title='Cross-Entropy Loss', xaxis=dict(title='Epoch'), yaxi
fig_loss = go.Figure(data=[loss_plot, val_loss_plot], layout=layout_loss)
fig_loss.show()

#Creating the classification error plot
val_error_values = [1 - acc for acc in val_acc_values]
err_plot = go.Scatter(x=list(epochs_range), y=error_values, mode='lines', name='Tra
val_error_plot = go.Scatter(x=list(epochs_range), y=val_error_values, mode='lines',
layout_acc = go.Layout(title='Error', xaxis=dict(title='Epoch'), yaxis=dict(title='
fig_acc = go.Figure(data=[err_plot, val_error_plot], layout=layout_acc)
fig_acc.show()


# Create the accuracy plot
acc_plot = go.Scatter(x=list(epochs_range), y=acc_values, mode='lines', name='Trair
val_acc_plot = go.Scatter(x=list(epochs_range), y=val_acc_values, mode='lines', nam
layout_acc = go.Layout(title='Accuracy', xaxis=dict(title='Epoch'), yaxis=dict(titl
fig_acc = go.Figure(data=[acc_plot, val_acc_plot], layout=layout_acc)
fig_acc.show()
```


0.27
0.26
0.25
0.24
0.23
0.22
0.21
Loss
2
4
6
Epoch


Error
0.045
0.04
0.035
0.03
0.025
Error
2
4
6




```
print(model.evaluate(x_test, y_test))        #Evaluating the model's loss and accu
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.2087 - accura
[0.20873022079467773, 0.9754999876022339]
```