This is the HW3 of our group.

Group members are:

Kasra Mojallal 110124782

Kimia Tahayori 110124141

Siyam Sajnan Chowdhury 110124636

```
import tensorflow as tf                                    #Importing tensorflow
from tensorflow.keras.datasets import mnist                #Importing mnist
import plotly.graph_objects as go                          #Importing plotly
```

Shallow NN, Single Output, Identifying '1's from 'not-1's

```
(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:]
```

```
    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,), (5000,), (10000,))
```

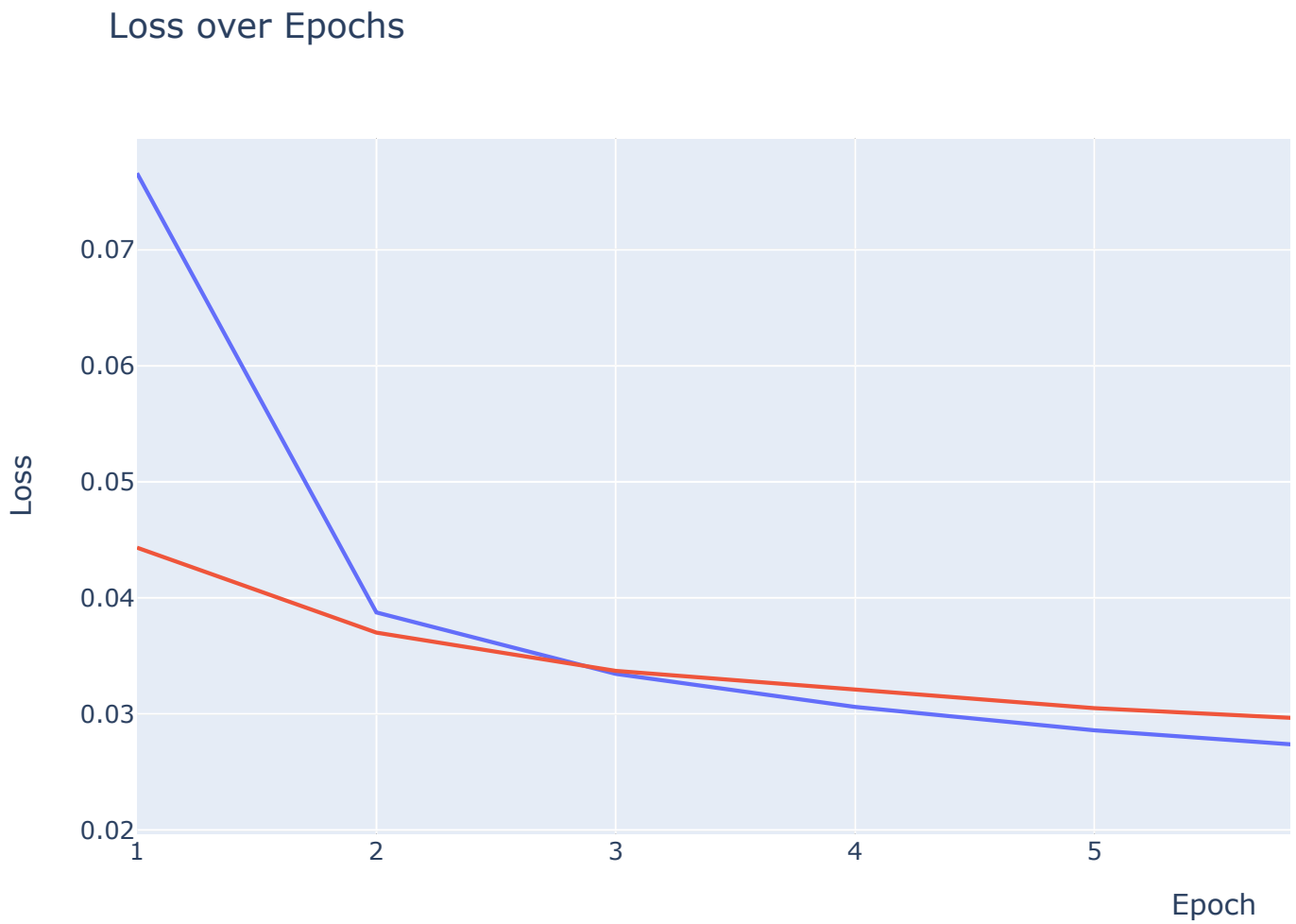
```
y_train_1 = (y_train == 1)                                 #List of boolean, true wher
y_validation_1 = (y_validation == 1)                       #List of boolean, true wher
y_test_1 = (y_test == 1)                                   #List of boolean, true wher
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(784,)),        #Setting the input shape
    tf.keras.layers.Dense(10, activation='relu'),       #Input layer with relu as a
    tf.keras.layers.Dense(1, activation='sigmoid')      #Output layer with sigmoid
])


model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy']) #C


history_shallowNN = model.fit(X_train, y_train_1, epochs=10, batch_size=32, validat
```

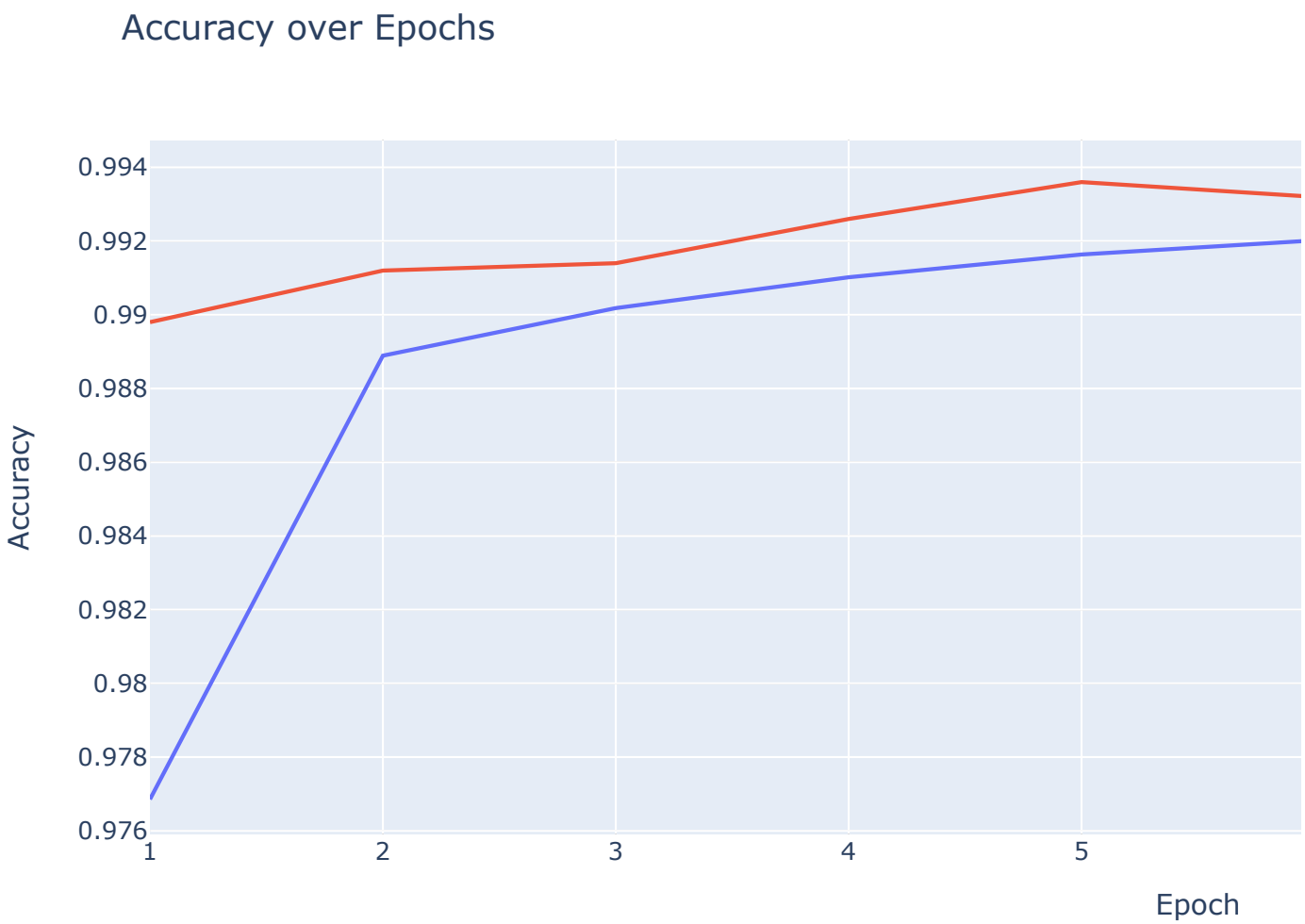
```
Epoch 1/10
1719/1719 [==============================] - 9s 5ms/step - loss: 0.0766 - accu
Epoch 2/10
1719/1719 [==============================] - 10s 6ms/step - loss: 0.0388 - acc
Epoch 3/10
1719/1719 [==============================] - 10s 6ms/step - loss: 0.0335 - acc
Epoch 4/10
1719/1719 [==============================] - 8s 5ms/step - loss: 0.0306 - accu
Epoch 5/10
1719/1719 [==============================] - 4s 3ms/step - loss: 0.0286 - accu
Epoch 6/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0271 - accu
Epoch 7/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0258 - accu
Epoch 8/10
1719/1719 [==============================] - 3s 2ms/step - loss: 0.0246 - accu
Epoch 9/10
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0237 - accu
Epoch 10/10
1719/1719 [==============================] - 5s 3ms/step - loss: 0.0227 - accu
```

```
#Accessing the different metrics to plot the graph
loss = history_shallowNN.history['loss']
val_loss = history_shallowNN.history['val_loss']
accuracy = history_shallowNN.history['accuracy']
val_accuracy = history_shallowNN.history['val_accuracy']
```

```
#Loss over Epochs

fig_loss = go.Figure()
fig_loss.add_trace(go.Scatter(x=list(range(1, len(loss)+1)), y=loss, mode='lines',
fig_loss.add_trace(go.Scatter(x=list(range(1, len(val_loss)+1)), y=val_loss, mode='
fig_loss.update_layout(title='Loss over Epochs',
                       xaxis_title='Epoch',
                       yaxis_title='Loss')
fig_loss.show()
```



```
#Improvement in Accuracy over epochs

fig_accuracy = go.Figure()
fig_accuracy.add_trace(go.Scatter(x=list(range(1, len(accuracy)+1)), y=accuracy, mo
fig_accuracy.add_trace(go.Scatter(x=list(range(1, len(val_accuracy)+1)), y=val_accu
fig_accuracy.update_layout(title='Accuracy over Epochs',
                           xaxis_title='Epoch',
                           yaxis_title='Accuracy')
fig_accuracy.show()
```




```
model.evaluate(X_test, y_test_1)      #evaluating the accuracy and loss
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0189 - accura
[0.018901420757174492, 0.9955999851226807]
```

Deep NN, Multiple Output, Minibatch SGD

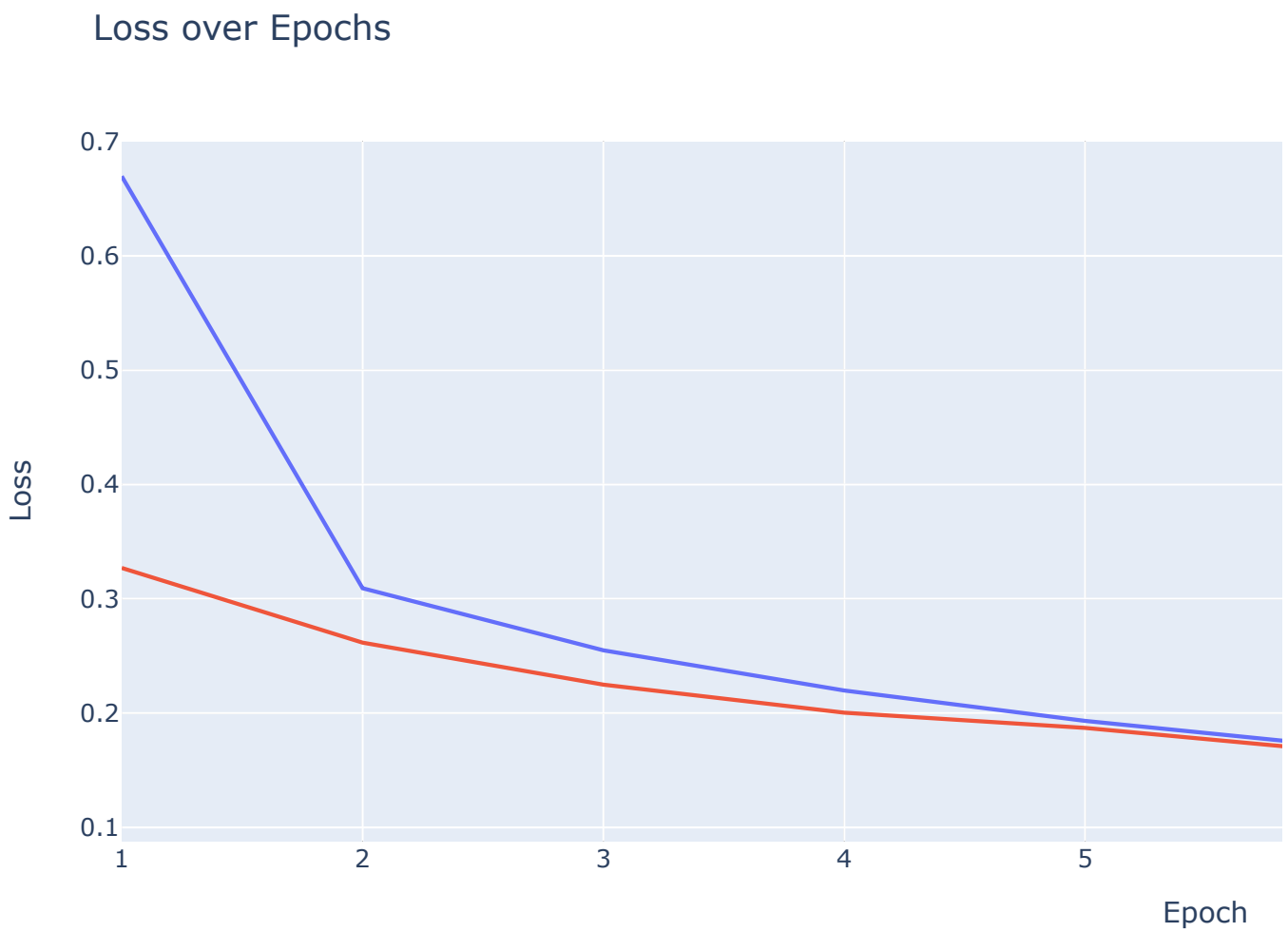
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()  #Loadir


#Dataset split to validation and train

split_index = int(0.8 * len(x_train))
x_val = x_train[split_index:]
y_val = y_train[split_index:]
x_train = x_train[:split_index]
y_train = y_train[:split_index]


#Normalizing the greyscale intensities between 0 and 1

x_train = x_train / 255.0
x_val = x_val / 255.0
x_test = x_test / 255.0


#Deep Neural Network with 4 layers, relu as activation for hidden layers, softmax a
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])


model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',   #Compiling the model with S
              metrics=['accuracy'])
```

```
history_deepNN = model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs
```

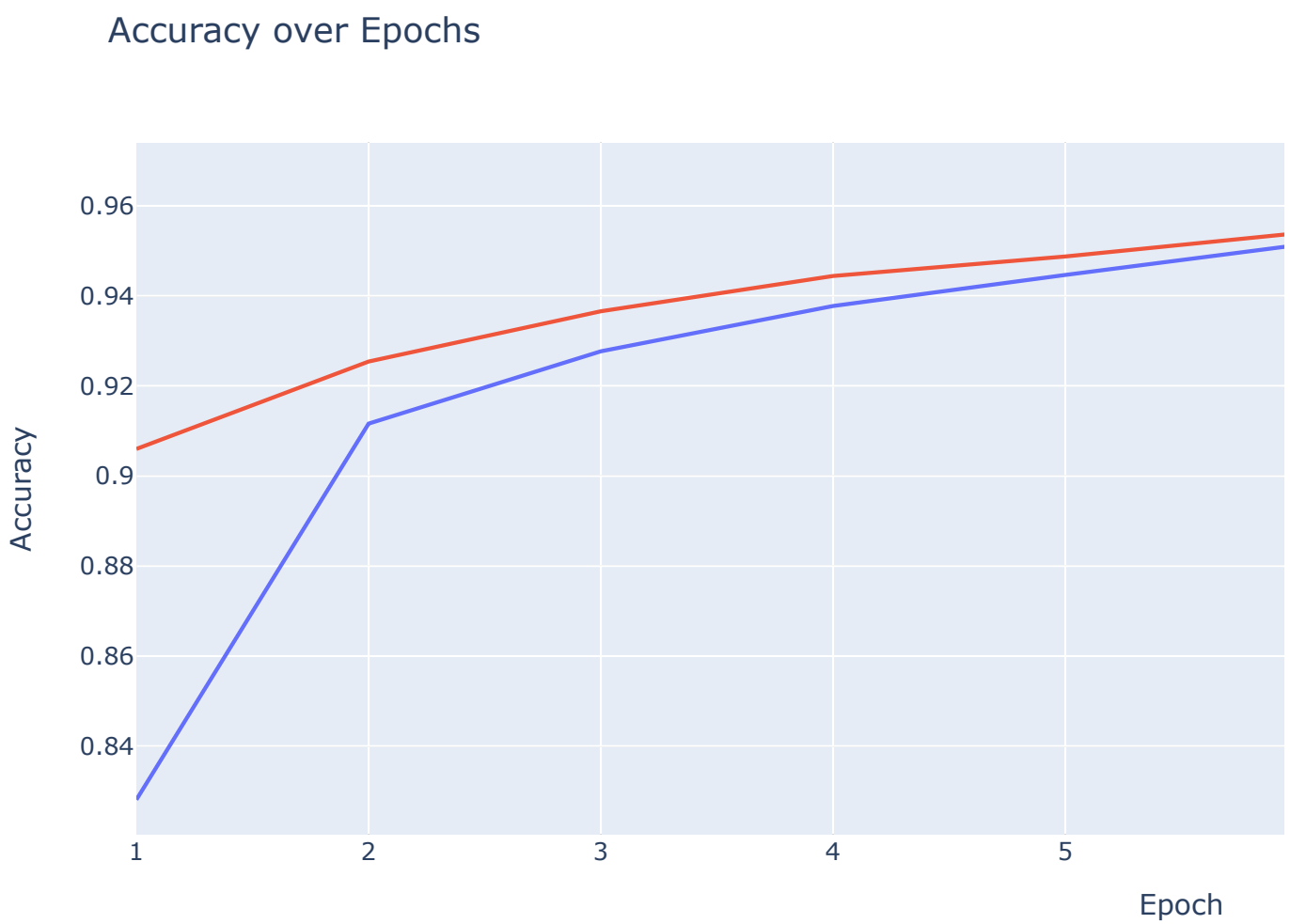
```
Epoch 1/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.6696 - accu
Epoch 2/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.3092 - accu
Epoch 3/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.2549 - accu
Epoch 4/10
1500/1500 [==============================] - 10s 6ms/step - loss: 0.2198 - acc
Epoch 5/10
1500/1500 [==============================] - 11s 7ms/step - loss: 0.1932 - acc
Epoch 6/10
1500/1500 [==============================] - 12s 8ms/step - loss: 0.1721 - acc
Epoch 7/10
1500/1500 [==============================] - 8s 6ms/step - loss: 0.1552 - accu
Epoch 8/10
1500/1500 [==============================] - 10s 7ms/step - loss: 0.1413 - acc
Epoch 9/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1289 - accu
Epoch 10/10
1500/1500 [==============================] - 7s 5ms/step - loss: 0.1186 - accu
```

```
#Accessing the different metrics to plot the graph

loss = history_deepNN.history['loss']
val_loss = history_deepNN.history['val_loss']
accuracy = history_deepNN.history['accuracy']
val_accuracy = history_deepNN.history['val_accuracy']
```

```
#Plotting Loss over Epochs

fig_loss = go.Figure()
fig_loss.add_trace(go.Scatter(x=list(range(1, len(loss)+1)), y=loss, mode='lines',
fig_loss.add_trace(go.Scatter(x=list(range(1, len(val_loss)+1)), y=val_loss, mode='
fig_loss.update_layout(title='Loss over Epochs',
                       xaxis_title='Epoch',
                       yaxis_title='Loss')
fig_loss.show()
```



```
#Plotting Acuracy over Epochs

fig_accuracy = go.Figure()
fig_accuracy.add_trace(go.Scatter(x=list(range(1, len(accuracy)+1)), y=accuracy, mo
fig_accuracy.add_trace(go.Scatter(x=list(range(1, len(val_accuracy)+1)), y=val_accu
fig_accuracy.update_layout(title='Accuracy over Epochs',
                           xaxis_title='Epoch',
                           yaxis_title='Accuracy')
fig_accuracy.show()
```




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

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1257 - accura
[0.1257256716489792, 0.9632999897003174]
```

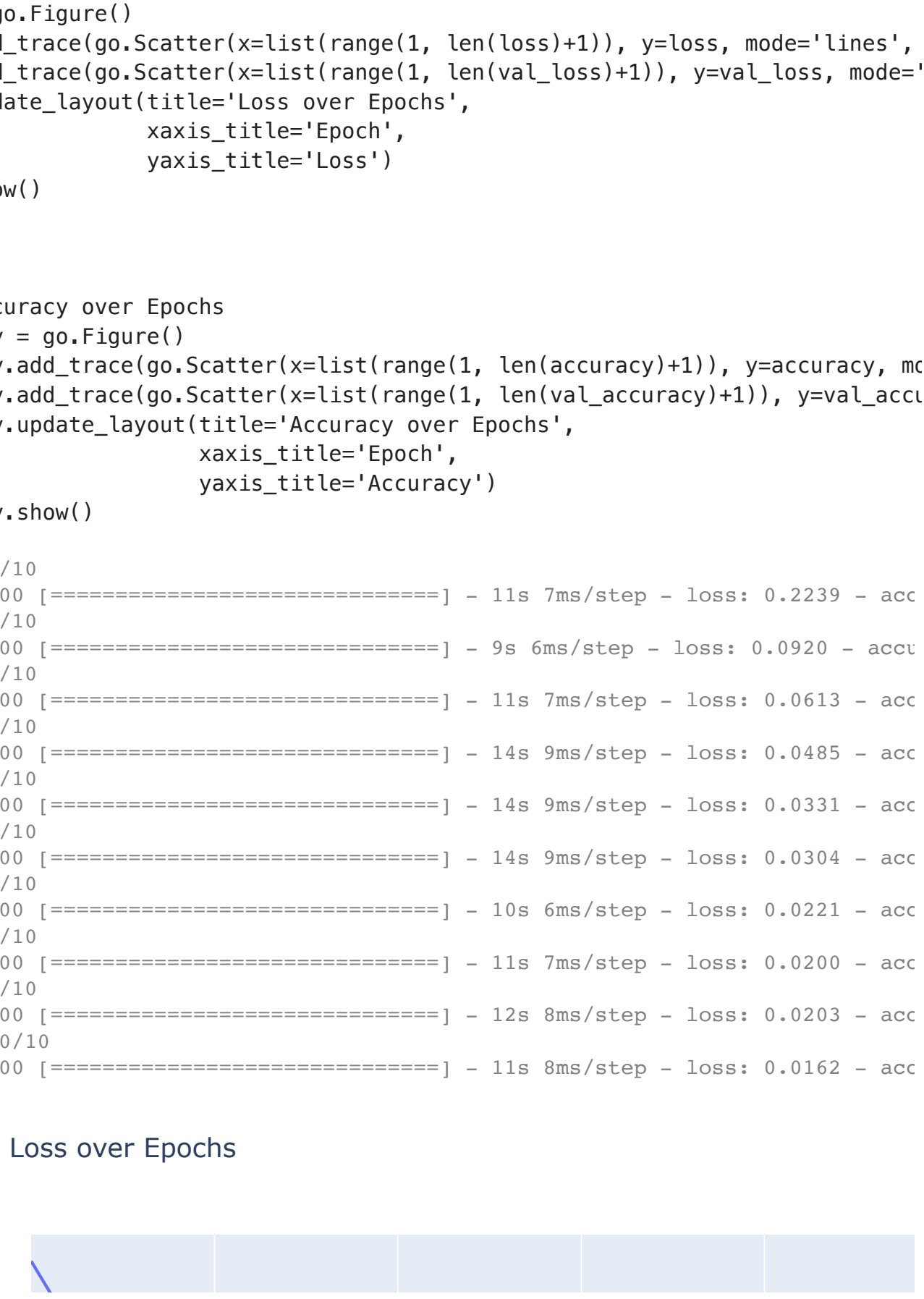
Deep NN, Multiple Output with Adam

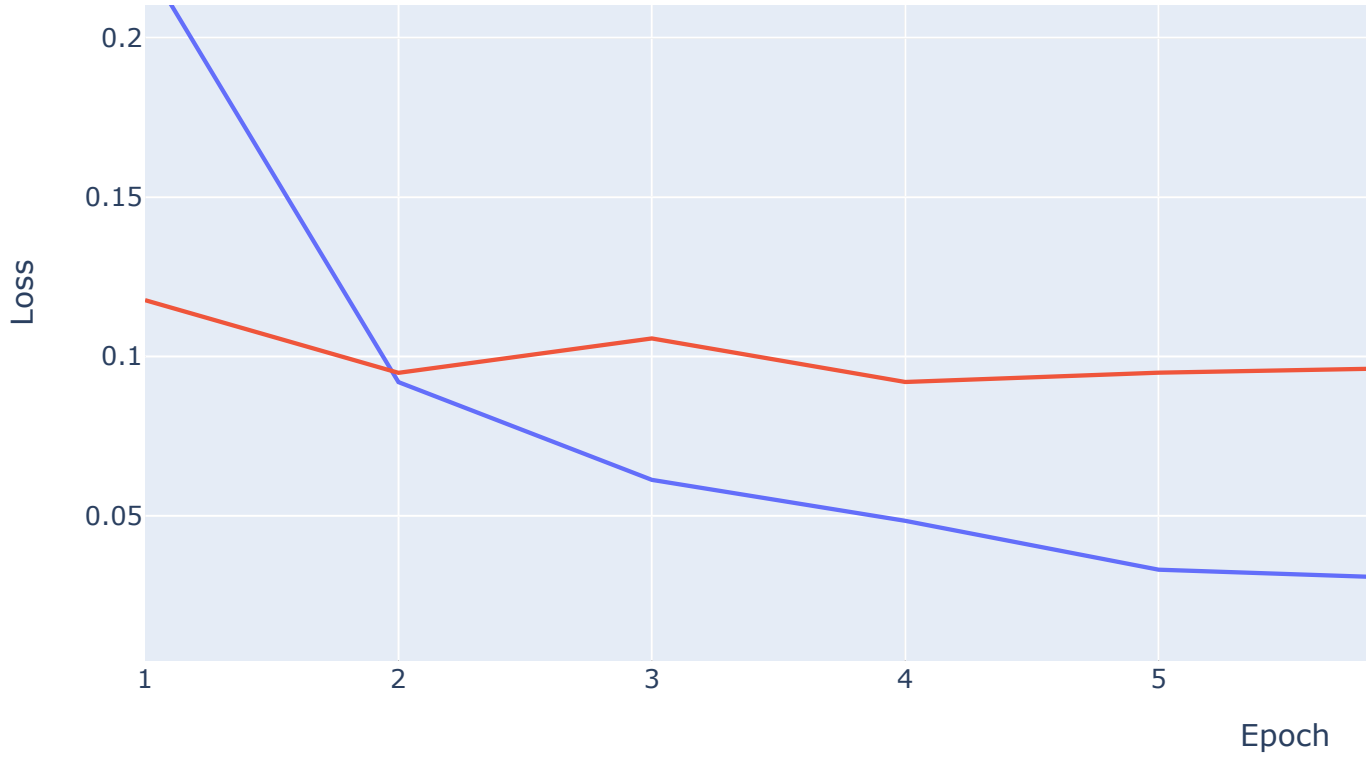
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()  #Loadir


#Dataset split to validation and train
split_index = int(0.8 * len(x_train))
x_val = x_train[split_index:]
y_val = y_train[split_index:]
x_train = x_train[:split_index]
y_train = y_train[:split_index]


#Normalizing the greyscale intensities between 0 and 1
x_train = x_train / 255.0
x_val = x_val / 255.0
x_test = x_test / 255.0

#Deep Neural Network with 4 layers, relu as activation for hidden layers, softmax a
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

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

#Fitting the model
history_deepNN_Adam = model.fit(x_train, y_train, validation_data=(x_val, y_val), e


#Accessing the different metrics to plot the graph
loss = history_deepNN_Adam.history['loss']
val_loss = history_deepNN_Adam.history['val_loss']
accuracy = history_deepNN_Adam.history['accuracy']
val_accuracy = history_deepNN_Adam.history['val_accuracy']
```

```
#Plotting Loss over Epochs
fig_loss = go.Figure()
fig_loss.add_trace(go.Scatter(x=list(range(1, len(loss)+1)), y=loss, mode='lines',
fig_loss.add_trace(go.Scatter(x=list(range(1, len(val_loss)+1)), y=val_loss, mode='
fig_loss.update_layout(title='Loss over Epochs',
                       xaxis_title='Epoch',
                       yaxis_title='Loss')
fig_loss.show()
```

```
#Plotting Acuracy over Epochs
fig_accuracy = go.Figure()
fig_accuracy.add_trace(go.Scatter(x=list(range(1, len(accuracy)+1)), y=accuracy, mo
fig_accuracy.add_trace(go.Scatter(x=list(range(1, len(val_accuracy)+1)), y=val_accu
fig_accuracy.update_layout(title='Accuracy over Epochs',
                           xaxis_title='Epoch',
                           yaxis_title='Accuracy')
fig_accuracy.show()
```

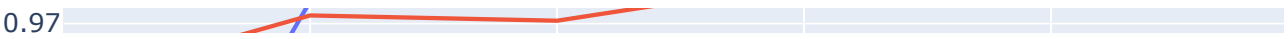
```
Epoch 1/10
1500/1500 [==============================] - 11s 7ms/step - loss: 0.2239 - acc
Epoch 2/10
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0920 - accu
Epoch 3/10
1500/1500 [==============================] - 11s 7ms/step - loss: 0.0613 - acc
Epoch 4/10
1500/1500 [==============================] - 14s 9ms/step - loss: 0.0485 - acc
Epoch 5/10
1500/1500 [==============================] - 14s 9ms/step - loss: 0.0331 - acc
Epoch 6/10
1500/1500 [==============================] - 14s 9ms/step - loss: 0.0304 - acc
Epoch 7/10
1500/1500 [==============================] - 10s 6ms/step - loss: 0.0221 - acc
Epoch 8/10
1500/1500 [==============================] - 11s 7ms/step - loss: 0.0200 - acc
Epoch 9/10
1500/1500 [==============================] - 12s 8ms/step - loss: 0.0203 - acc
Epoch 10/10
1500/1500 [==============================] - 11s 8ms/step - loss: 0.0162 - acc
```






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

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0903 - accura
[0.09034270793199539, 0.9782999753952026]
```



Grid Search CV

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import GridSearchCV
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
```

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

#Reshaping and normalizing the greyscale intensities
X_train = X_train.reshape(-1, 784) / 255.0
X_test = X_test.reshape(-1, 784) / 255.0


#Creating the wrapper for our model to fit to GridSearchCV2

def create_model(learning_rate=0.01):       #Setting learning rate to 0.01
    model = Sequential()
    model.add(Dense(256, activation='relu', input_shape=(784,)))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(10, activation='softmax'))

    optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate)    #Setting th

    model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer, metr
    return model


#Setting Model to Keras
model = KerasClassifier(build_fn=create_model, verbose=0)
```

```
<ipython-input-27-7b0f05354f25>:1: DeprecationWarning:

KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scil
```

```
#Initialising 2 learning rates and 3 batch sizes
param_grid = {
    'learning_rate': [0.01, 0.1],
    'batch_size': [16, 32, 64]
}


#Setting up GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, verbose=1)
```

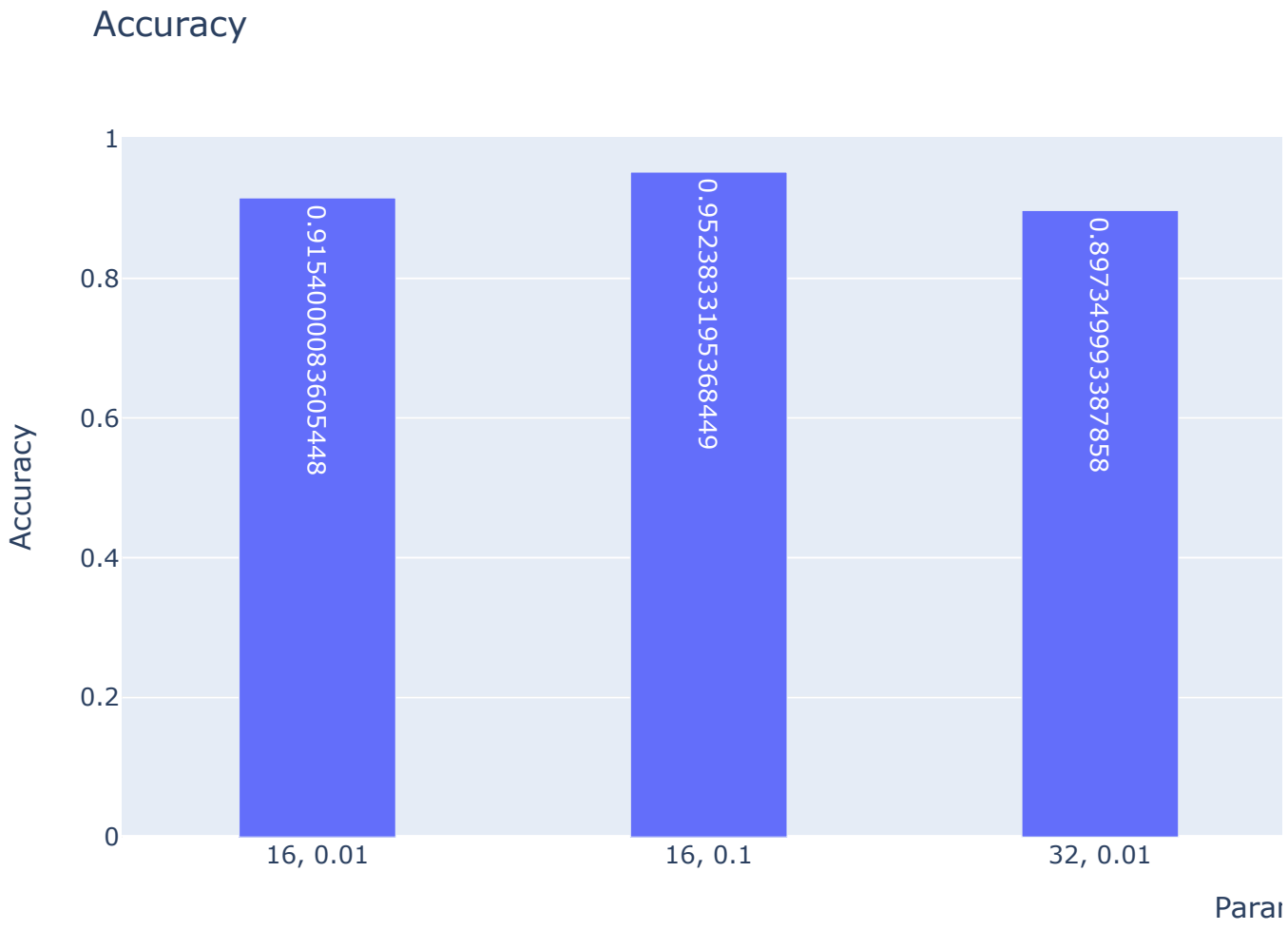
```
#Fitting the model
grid_result = grid_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
```

```
#Storing the results and saving it in params and mean_accuracy

results = grid_result.cv_results_
params = results['params']
mean_accuracy = results['mean_test_score']


for param, accuracy in zip(params, mean_accuracy):
    print("Parameters: ", param)
    print("Accuracy: ", accuracy)
    print("-----------------------------------")
```

```
Parameters:  {'batch_size': 16, 'learning_rate': 0.01}
Accuracy:  0.9154000083605448
-----------------------------------
Parameters:  {'batch_size': 16, 'learning_rate': 0.1}
Accuracy:  0.9523833195368449
-----------------------------------
Parameters:  {'batch_size': 32, 'learning_rate': 0.01}
Accuracy:  0.897349993387858
-----------------------------------
Parameters:  {'batch_size': 32, 'learning_rate': 0.1}
Accuracy:  0.9473166664441427
-----------------------------------
Parameters:  {'batch_size': 64, 'learning_rate': 0.01}
Accuracy:  0.8700666626294454
-----------------------------------
Parameters:  {'batch_size': 64, 'learning_rate': 0.1}
Accuracy:  0.9368000030517578
-----------------------------------
```

```
# Create traces for accuracy and loss
accuracy_trace = go.Bar(x=['16, 0.01', '16, 0.1', '32, 0.01', '32, 0.1', '64, 0.01'
                        y=mean_accuracy, name='Accuracy', width=0.4, text=mean_accu

# Create layout for accuracy plot
accuracy_layout = go.Layout(title='Accuracy',
                            xaxis=dict(title='Parameters'),
                            yaxis=dict(title='Accuracy'))

# Create figures and add traces
accuracy_fig = go.Figure(data=[accuracy_trace], layout=accuracy_layout)

# Show the figures
accuracy_fig.show()
```



```
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.952383 using {'batch_size': 16, 'learning_rate': 0.1}
```