Name: **Janhavi Kolte**

Div: **BE09-R9**

Roll no: **43141**

Title: **Assignment 2: Implementing Feedforward neural networks with Keras and TensorFlow**

```
#installations
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.datasets import mnist
from tensorflow.keras import backend as K
import matplotlib.pyplot as plt
import numpy as np
```

```
#grabbing the mnist dataset
((X_train, Y_train), (X_test, Y_test)) = mnist.load_data()
X_train = X_train.reshape((X_train.shape[0], 28 * 28 * 1))
X_test = X_test.reshape((X_test.shape[0], 28 * 28 * 1))
X_train = X_train.astype("float32") / 255.0
X_test = X_test.astype("float32") / 255.0
```

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

```
type(X_train)
```

```
numpy.ndarray
```

```
lb = LabelBinarizer()
Y_train = lb.fit_transform(Y_train)
Y_test = lb.transform(Y_test)
```

```
lb.classes_
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

```
#building the model
model = Sequential()
model.add(Dense(128, input_shape=(784,), activation="sigmoid"))
model.add(Dense(64, activation="sigmoid"))
model.add(Dense(10, activation="softmax"))
```

```
sgd = SGD(0.01)
epochs=10
model.compile(loss="categorical_crossentropy", optimizer=sgd,metrics=["accuracy"])
```

```
H = model.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=epochs, batch_size
```

```
Epoch 1/10
469/469 [==============================] - 5s 9ms/step - loss: 2.2972 - accuracy: 0.1
Epoch 2/10
469/469 [==============================] - 5s 10ms/step - loss: 2.2260 - accuracy: 0
Epoch 3/10
469/469 [==============================] - 4s 9ms/step - loss: 2.1623 - accuracy: 0.5
Epoch 4/10
469/469 [==============================] - 4s 8ms/step - loss: 2.0732 - accuracy: 0.5
Epoch 5/10
469/469 [==============================] - 3s 6ms/step - loss: 1.9475 - accuracy: 0.6
Epoch 6/10
469/469 [==============================] - 2s 4ms/step - loss: 1.7817 - accuracy: 0.6
Epoch 7/10
469/469 [==============================] - 2s 4ms/step - loss: 1.5903 - accuracy: 0.6
Epoch 8/10
469/469 [==============================] - 3s 5ms/step - loss: 1.3989 - accuracy: 0.7
Epoch 9/10
469/469 [==============================] - 2s 5ms/step - loss: 1.2304 - accuracy: 0.7
Epoch 10/10
469/469 [==============================] - 2s 4ms/step - loss: 1.0939 - accuracy: 0.7
```

```
#making the predictions
predictions = model.predict(X_test, batch_size=128)
print(classification_report(Y_test.argmax(axis=1),predictions.argmax(axis=1),target_names=
```

```
79/79 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.81      0.97      0.88       980
           1       0.78      0.98      0.87      1135
           2       0.78      0.71      0.74      1032
           3       0.66      0.88      0.76      1010
           4       0.70      0.81      0.75       982
           5       0.86      0.37      0.51       892
           6       0.85      0.89      0.87       958
           7       0.74      0.88      0.80      1028
           8       0.84      0.56      0.67       974
           9       0.73      0.54      0.62      1009

    accuracy                           0.76     10000
   macro avg       0.77      0.76      0.75     10000
weighted avg       0.77      0.76      0.75     10000
```

```
#plotting the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, epochs), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, epochs), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, epochs), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, epochs), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
```

```
plt.ylabel("Loss/Accuracy")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa5322a4bd0>
```