Assignment - 2
Course: Introduction to Machine Learning

Code: ELL784
Professor: Brejesh Lall

Submitted By:

Ashutosh Pandey (2021EET2114)
Pratik Wadhwa (2021EET2378)
Nishant Chauhan (2021EET2377)

Problem 1

a) Use 10K images from the training set as validation data. The two hidden layers have a dimension of 500 each. Using a suitable loss function, train the network for 250 epochs, and report the classification accuracy on test data.

Creating the model

```
h=500 #var for hidden layer
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)), # input layer
    tf.keras.layers.Dense(h, activation='relu'), # 1st hidden layer
    tf.keras.layers.Dense(h, activation='relu'), # 2nd hidden layer
    tf.keras.layers.Dense(10, activation='softmax') # output layer
])
```

loss function

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Model Training

```
%%time
his=model.fit(X_train_batch, epochs=250,
validation_data=(validation_inputs, validation_targets), verbose =2)
```

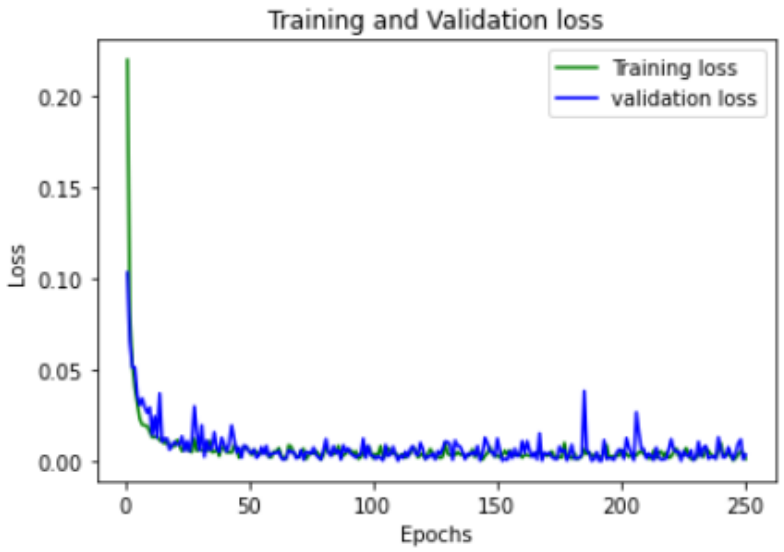
```
540/540 - 5s - loss: 0.0033 - accuracy: 0.9996 - val_loss: 0.0017 - val_accuracy: 0.9997
Epoch 242/250
540/540 - 5s - loss: 0.0033 - accuracy: 0.9995 - val_loss: 0.0023 - val_accuracy: 0.9998
Epoch 243/250
540/540 - 5s - loss: 0.0037 - accuracy: 0.9996 - val_loss: 0.0074 - val_accuracy: 0.9993
Epoch 244/250
540/540 - 5s - loss: 0.0025 - accuracy: 0.9997 - val_loss: 0.0038 - val_accuracy: 0.9995
Epoch 245/250
540/540 - 5s - loss: 2.2567e-04 - accuracy: 0.9999 - val_loss: 6.5676e-04 - val_accuracy: 0.9998
Epoch 246/250
540/540 - 5s - loss: 0.0014 - accuracy: 0.9998 - val_loss: 0.0043 - val_accuracy: 0.9993
Epoch 247/250
540/540 - 5s - loss: 0.0032 - accuracy: 0.9996 - val_loss: 0.0095 - val_accuracy: 0.9990
Epoch 248/250
540/540 - 5s - loss: 0.0053 - accuracy: 0.9995 - val_loss: 0.0117 - val_accuracy: 0.9992
Epoch 249/250
540/540 - 5s - loss: 0.0041 - accuracy: 0.9995 - val_loss: 8.0940e-04 - val_accuracy: 0.9998
Epoch 250/250
540/540 - 5s - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0033 - val_accuracy: 0.9995
Wall time: 24min 32s
```

Test

```
test_loss, test_accuracy = model.evaluate(test_data)
print('Test loss: {0:.2f}. Test accuracy: {1:.2f}%'.format(test_loss,
                                                    test_accuracy*100.))
```

```
1/1 [==============================] - 0s 436ms/step - loss: 0.4447 - accuracy: 0.9840
Test loss: 0.44. Test accuracy: 98.40%
```

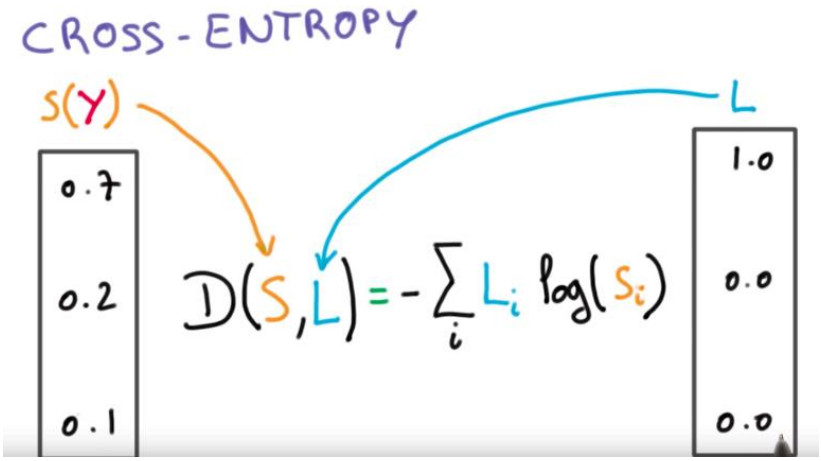
Plot the error and classification accuracy on both training and validation data over the epochs. Justify the loss used to train the network. Also, normalize the data appropriately.



Justify the loss used to train the network.

Cross-entropy builds upon the idea of entropy from information theory and calculates the number of bits required to represent or transmit an average event from one distribution compared to another distribution.

The aim is to minimize the loss, i.e, the smaller the loss the better the model. We use sparse categorical cross entropy since our classes are mutually exclusive.
The intuition for this definition comes if we consider a target or underlying probability distribution P and an approximation of the target distribution Q, then the cross-entropy of Q from P is the number of additional bits to represent an event using Q instead of P.



scaling

```
def scale(image, label):
    image = tf.cast(image, tf.float32)
    image /= 255.

    return image, label
```

Here we have normalized or rescaled the images from 0-255 to 0-1.

b) L2 - Regularization:

L2 regularization

```
from keras import regularizers
h=500
model_l2 = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)), # input layer
    tf.keras.layers.Dense(h, activation='relu',kernel_regularizer=regularizers.l2(0.001)), # 1st
    tf.keras.layers.Dense(h, activation='relu',kernel_regularizer=regularizers.l2(0.001)), # 2nd
    tf.keras.layers.Dense(10, activation='softmax') # output layer
])
```

```
model_l2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

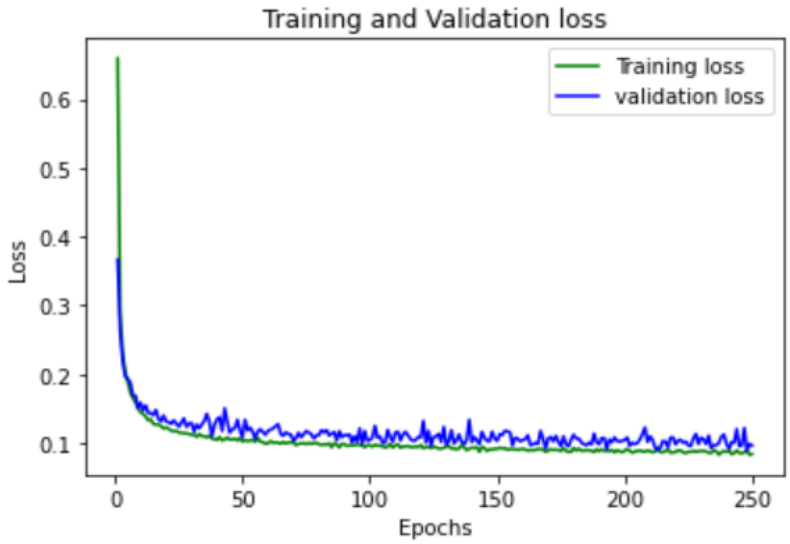
loss function

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Model Training

```
%%time
his=model.fit(X_train_batch, epochs=250,
validation_data=(validation_inputs, validation_targets), verbose =2)
```

```
Epoch 242/250
540/540 - 13s - loss: 0.0837 - accuracy: 0.9896 - val_loss: 0.1014 - val_accuracy: 0.9832
Epoch 243/250
540/540 - 14s - loss: 0.0851 - accuracy: 0.9883 - val_loss: 0.1005 - val_accuracy: 0.9855
Epoch 244/250
540/540 - 13s - loss: 0.0889 - accuracy: 0.9877 - val_loss: 0.1202 - val_accuracy: 0.9792
Epoch 245/250
540/540 - 14s - loss: 0.0855 - accuracy: 0.9887 - val_loss: 0.0950 - val_accuracy: 0.9855
Epoch 246/250
540/540 - 15s - loss: 0.0849 - accuracy: 0.9889 - val_loss: 0.0966 - val_accuracy: 0.9847
Epoch 247/250
540/540 - 14s - loss: 0.0863 - accuracy: 0.9882 - val_loss: 0.1217 - val_accuracy: 0.9777
Epoch 248/250
540/540 - 14s - loss: 0.0895 - accuracy: 0.9871 - val_loss: 0.0884 - val_accuracy: 0.9887
Epoch 249/250
540/540 - 13s - loss: 0.0826 - accuracy: 0.9895 - val_loss: 0.0987 - val_accuracy: 0.9845
Epoch 250/250
540/540 - 13s - loss: 0.0839 - accuracy: 0.9897 - val_loss: 0.0963 - val_accuracy: 0.9845
Wall time: 1h 2min 3s
```



```
test_loss, test_accuracy = model_l2.evaluate(test_data)
print('Test loss: {0:.2f}. Test accuracy: {1:.2f}%'.format(test_loss, test_accuracy*100.))
```

```
1/1 [==============================] - 2s 2s/step - loss: 0.1223 - accuracy: 0.9786
Test loss: 0.12. Test accuracy: 97.86%
```

L2 regularization can add a penalty to the cost depending upon the model complexity, so at the place of computing the cost by using a loss function, there will be an auxiliary component, known as regularization terms.

By adding Regularization term, the value of weights matrices reduces by assuming that a neural network having less weights makes simpler models. And hence, it reduces the overfitting to a certain level.

b) Dropout :

```
h=500 #var for hidden layer
model_Dropout = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)), # input layer
    tf.keras.layers.Dense(h, activation='relu'),Dropout(0.25), # 1st hidden layer
    tf.keras.layers.Dense(h, activation='relu'),Dropout(0.25), # 2nd hidden layer
    tf.keras.layers.Dense(10, activation='softmax') # output layer
])
```

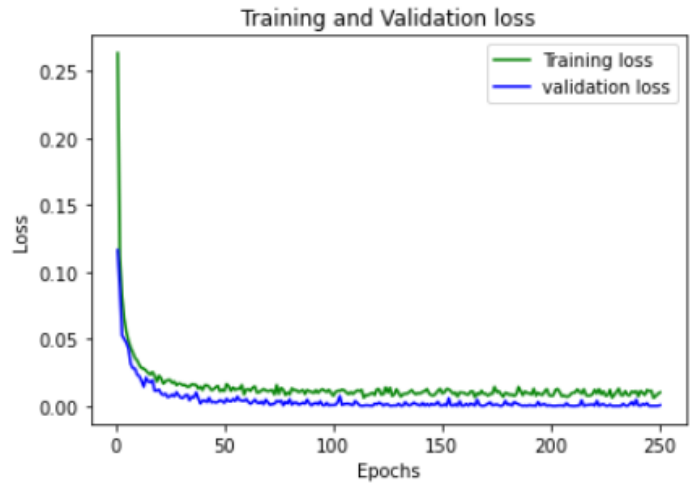
loss function

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Model Training

```
%%time
his=model.fit(X_train_batch, epochs=250,
validation_data=(validation_inputs, validation_targets), verbose =2)
```

```
540/540 - 7s - loss: 0.0113 - accuracy: 0.9984 - val_loss: 2.0709e-04 - val_accuracy: 0.9998
Epoch 242/250
540/540 - 9s - loss: 0.0099 - accuracy: 0.9987 - val_loss: 0.0015 - val_accuracy: 0.9997
Epoch 243/250
540/540 - 7s - loss: 0.0083 - accuracy: 0.9987 - val_loss: 8.1281e-04 - val_accuracy: 0.9998
Epoch 244/250
540/540 - 7s - loss: 0.0120 - accuracy: 0.9984 - val_loss: 0.0021 - val_accuracy: 0.9995
Epoch 245/250
540/540 - 7s - loss: 0.0109 - accuracy: 0.9982 - val_loss: 6.3353e-05 - val_accuracy: 1.0000
Epoch 246/250
540/540 - 7s - loss: 0.0114 - accuracy: 0.9982 - val_loss: 7.3304e-05 - val_accuracy: 1.0000
Epoch 247/250
540/540 - 7s - loss: 0.0060 - accuracy: 0.9990 - val_loss: 1.0673e-04 - val_accuracy: 1.0000
Epoch 248/250
540/540 - 7s - loss: 0.0076 - accuracy: 0.9988 - val_loss: 1.5897e-04 - val_accuracy: 0.9998
Epoch 249/250
540/540 - 7s - loss: 0.0090 - accuracy: 0.9984 - val_loss: 1.3538e-04 - val_accuracy: 1.0000
Epoch 250/250
540/540 - 7s - loss: 0.0102 - accuracy: 0.9987 - val_loss: 6.4460e-04 - val_accuracy: 0.9997
```




```
test_loss, test_accuracy = model_Dropout.evaluate(test_data)
print('Test loss: {0:.2f}. Test accuracy: {1:.2f}%'.format(test_loss, test_accuracy*100.))
```

```
1/1 [==============================] - 0s 310ms/step - loss: 0.2340 - accuracy: 0.9857
Test loss: 0.23. Test accuracy: 98.57%
```

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections. We had a slight increase in accuracy by applying Dropout.

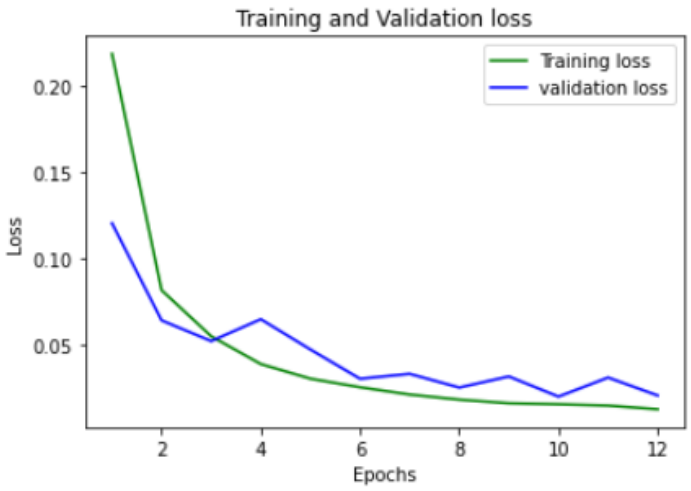
b) Early Stopping :

```
h=500 #var for hidden layer
model_ES = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)), # input layer
    tf.keras.layers.Dense(h, activation='relu'), # 1st hidden layer
    tf.keras.layers.Dense(h, activation='relu'), # 2nd hidden layer
    tf.keras.layers.Dense(10, activation='softmax') # output layer
])
```

```
%%time
history_ES=model_ES.fit(X_train_batch, epochs=250,
validation_data=(validation_inputs, validation_targets), verbose =2,
callbacks = [EarlyStopping(monitor='val_loss', patience=5)])
```

Epoch 1/250
540/540 - 19s - loss: 0.2213 - accuracy: 0.9336 - val_loss: 0.1088 - val_accuracy: 0.9665
Epoch 2/250
540/540 - 13s - loss: 0.0835 - accuracy: 0.9742 - val_loss: 0.0895 - val_accuracy: 0.9737
Epoch 3/250
540/540 - 14s - loss: 0.0532 - accuracy: 0.9834 - val_loss: 0.0650 - val_accuracy: 0.9813
Epoch 4/250
540/540 - 12s - loss: 0.0400 - accuracy: 0.9873 - val_loss: 0.0534 - val_accuracy: 0.9817
Epoch 5/250
540/540 - 14s - loss: 0.0304 - accuracy: 0.9906 - val_loss: 0.0364 - val_accuracy: 0.9888
Epoch 6/250
540/540 - 13s - loss: 0.0264 - accuracy: 0.9917 - val_loss: 0.0362 - val_accuracy: 0.9892
Epoch 7/250
540/540 - 14s - loss: 0.0248 - accuracy: 0.9919 - val_loss: 0.0385 - val_accuracy: 0.9873
Epoch 8/250
540/540 - 14s - loss: 0.0198 - accuracy: 0.9935 - val_loss: 0.0315 - val_accuracy: 0.9913
Epoch 9/250
540/540 - 14s - loss: 0.0154 - accuracy: 0.9949 - val_loss: 0.0232 - val_accuracy: 0.9930
Epoch 10/250
540/540 - 14s - loss: 0.0171 - accuracy: 0.9942 - val_loss: 0.0299 - val_accuracy: 0.9900
Epoch 11/250
540/540 - 14s - loss: 0.0183 - accuracy: 0.9942 - val_loss: 0.0214 - val_accuracy: 0.9935

Epoch 12/250
540/540 - 13s - loss: 0.0110 - accuracy: 0.9964 - val_loss: 0.0112 - val_accuracy: 0.9965
Epoch 13/250
540/540 - 13s - loss: 0.0099 - accuracy: 0.9966 - val_loss: 0.0125 - val_accuracy: 0.9960
Epoch 14/250
540/540 - 13s - loss: 0.0115 - accuracy: 0.9964 - val_loss: 0.0229 - val_accuracy: 0.9925
Epoch 15/250
540/540 - 13s - loss: 0.0123 - accuracy: 0.9961 - val_loss: 0.0215 - val_accuracy: 0.9937
Epoch 16/250
540/540 - 12s - loss: 0.0083 - accuracy: 0.9973 - val_loss: 0.0115 - val_accuracy: 0.9955
Epoch 17/250
540/540 - 13s - loss: 0.0135 - accuracy: 0.9957 - val_loss: 0.0194 - val_accuracy: 0.9940
Wall time: 4min 29s




```
test_loss, test_accuracy = model_ES.evaluate(test_data)
print('Test loss: {0:.2f}. Test accuracy: {1:.2f}%'.format(test_loss, test_accuracy*100.))
```

```
1/1 [==============================] - 0s 195ms/step - loss: 0.0873 - accuracy: 0.9825
Test loss: 0.09. Test accuracy: 98.25%
```

Too little training will mean that the model will underfit the train and the test sets. Too much training will mean that the model will overfit the training dataset and have poor performance on the test set. A compromise is to train on the training dataset but to stop training at the point when performance on a validation dataset starts to degrade. This

simple, effective, and widely used approach to training neural networks is called early stopping.

Here we have set the patience=5, now when our validation loss increases for 5 times then we will stop to avoid overfitting.