```
pip install h5py
```

```
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (2.8.0)
Requirement already satisfied: numpy>=1.7 in /usr/local/lib/python3.6/dist-packages (fro
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from h5py)
```

```
import tensorflow as tf
import h5py
import numpy as np
from sklearn.metrics import accuracy_score,classification_report
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
%matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=9473189

Enter your authorization code:
··········
Mounted at /content/drive
```

```
f = h5py.File('/content/drive/My Drive/AIML PG/Introduction_to_Neural_Networks_and_Deep_Learn
```

```
list(f.keys())
```

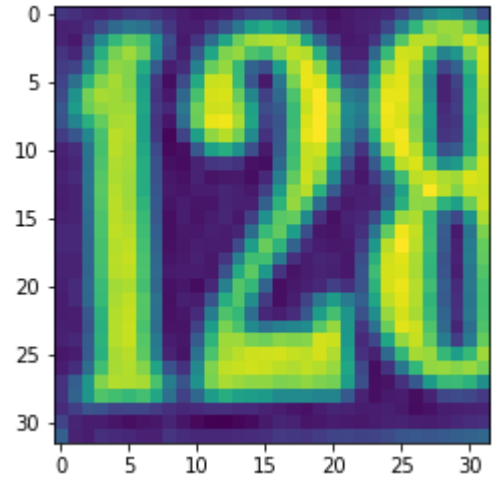
```
['X_test', 'X_train', 'X_val', 'y_test', 'y_train', 'y_val']
```

```
print(f['X_train'].shape)
print(f['X_val'].shape)
print(f['X_test'].shape)
print(f['y_train'].shape)
print(f['y_test'].shape)
print(f['y_val'].shape)
```

```
(42000, 32, 32)
(60000, 32, 32)
(18000, 32, 32)
(42000,)
(18000,)
(60000,)
```

```
plt.imshow(f['X_train'][0])
f['y_train'][0]
```

2



```
X_train = np.array(f['X_train'])
X_test = np.array(f['X_test'])
X_val = np.array(f['X_val'])
y_val = tf.keras.utils.to_categorical(y=f['y_val'],num_classes=10)
y_train = tf.keras.utils.to_categorical(y=f['y_train'],num_classes=10)

X_train = X_train/255
X_test = X_test/255
X_val = X_val/255


X_train_knn = X_train.reshape((X_train.shape[0],(X_train.shape[1]*X_train.shape[2])))
X_test_knn = X_test.reshape((X_test.shape[0],(X_test.shape[1]*X_test.shape[2])))
print(X_train_knn.shape)
print(X_test_knn.shape)
```

```
(42000, 1024)
(18000, 1024)
```

```
#Implementing KNN model
knn_model = KNeighborsClassifier()
knn_model.fit(X_train_knn,f['y_train'])
knn_model.score(X_test_knn,f['y_test'])
```

```
0.49016666666666664
```

K nearest neighbor gives the precision score of 49%

```
print(classification_report(y_true=f['y_test'],y_pred=knn_model.predict(X_test_knn)))
```

```
                   precision    recall  f1-score   support

                0       0.38      0.66      0.48      1814
                1       0.44      0.70      0.54      1828
                2       0.55      0.52      0.53      1803
                3       0.42      0.39      0.41      1719
                4       0.67      0.61      0.64      1812
                5       0.48      0.36      0.41      1768
                6       0.47      0.38      0.42      1832
                7       0.72      0.58      0.64      1808
                8       0.42      0.33      0.37      1812
                9       0.52      0.36      0.43      1804

         accuracy                           0.49     18000
        macro avg       0.51      0.49      0.49     18000
     weighted avg       0.51      0.49      0.49     18000
```

```
tf.keras.backend.clear_session()
```

```
#implementing Deep Neural Network
model = tf.keras.Sequential()
model.add(tf.keras.layers.Reshape(target_shape=(1024,),input_shape=(32,32)))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(units=500,activation = 'relu'))
model.add(tf.keras.layers.Dropout(rate=0.2))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(units=50,activation='relu'))
model.add(tf.keras.layers.Dropout(rate=0.2))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(units=10,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='SGD',metrics=['accuracy'])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/op
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape (Reshape)            (None, 1024)              0
_________________________________________________________________
batch_normalization (BatchNo (None, 1024)              4096
_________________________________________________________________
dense (Dense)                (None, 500)               512500
_________________________________________________________________
dropout (Dropout)            (None, 500)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 500)               2000
_________________________________________________________________
dense_1 (Dense)              (None, 50)                25050
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
batch_normalization_2 (Batch (None, 50)                200
_________________________________________________________________
dense_2 (Dense)              (None, 10)                510
=================================================================
Total params: 544,356
Trainable params: 541,208
Non-trainable params: 3,148
_________________________________________________________________
```

```
model.fit(x=X_train,y=y_train,epochs=25,validation_data=(X_val,y_val),batch_size=32)
```

```
Train on 42000 samples, validate on 60000 samples
Epoch 1/25
42000/42000 [==============================] - 16s 375us/sample - loss: 1.9866 - acc: 0.
Epoch 2/25
42000/42000 [==============================] - 15s 351us/sample - loss: 1.4642 - acc: 0.
Epoch 3/25
42000/42000 [==============================] - 15s 353us/sample - loss: 1.2983 - acc: 0.
Epoch 4/25
42000/42000 [==============================] - 15s 352us/sample - loss: 1.1991 - acc: 0.
Epoch 5/25
42000/42000 [==============================] - 15s 356us/sample - loss: 1.1329 - acc: 0.
Epoch 6/25
42000/42000 [==============================] - 16s 376us/sample - loss: 1.0783 - acc: 0.
Epoch 7/25
42000/42000 [==============================] - 15s 355us/sample - loss: 1.0325 - acc: 0.
Epoch 8/25
42000/42000 [==============================] - 15s 347us/sample - loss: 1.0057 - acc: 0.
Epoch 9/25
42000/42000 [==============================] - 14s 344us/sample - loss: 0.9679 - acc: 0.
Epoch 10/25
42000/42000 [==============================] - 15s 352us/sample - loss: 0.9363 - acc: 0.
Epoch 11/25
42000/42000 [==============================] - 14s 343us/sample - loss: 0.9153 - acc: 0.
Epoch 12/25
42000/42000 [==============================] - 14s 344us/sample - loss: 0.8868 - acc: 0.
Epoch 13/25
42000/42000 [==============================] - 15s 348us/sample - loss: 0.8717 - acc: 0.
Epoch 14/25
42000/42000 [==============================] - 14s 340us/sample - loss: 0.8589 - acc: 0.
Epoch 15/25
42000/42000 [==============================] - 15s 363us/sample - loss: 0.8478 - acc: 0.
Epoch 16/25
42000/42000 [==============================] - 14s 336us/sample - loss: 0.8287 - acc: 0.
Epoch 17/25
42000/42000 [==============================] - 14s 339us/sample - loss: 0.8164 - acc: 0.
Epoch 18/25
42000/42000 [==============================] - 15s 348us/sample - loss: 0.8061 - acc: 0.
Epoch 19/25
42000/42000 [==============================] - 14s 339us/sample - loss: 0.7953 - acc: 0.
Epoch 20/25
42000/42000 [==============================] - 14s 338us/sample - loss: 0.7862 - acc: 0.
Epoch 21/25
42000/42000 [==============================] - 14s 333us/sample - loss: 0.7712 - acc: 0.
Epoch 22/25
42000/42000 [==============================] - 14s 342us/sample - loss: 0.7643 - acc: 0.
Epoch 23/25
42000/42000 [==============================] - 14s 340us/sample - loss: 0.7570 - acc: 0.
Epoch 24/25
42000/42000 [==============================] - 14s 330us/sample - loss: 0.7501 - acc: 0.
Epoch 25/25
42000/42000 [==============================] - 15s 345us/sample - loss: 0.7485 - acc: 0.
<tensorflow.python.keras.callbacks.History at 0x7f7f5f17ecc0>
```

```
accuracy_score(y_true=f['y_test'],y_pred=model.predict_classes(X_test))
```

0.8343888888888888

```
print(classification_report(y_true=f['y_test'],y_pred=model.predict_classes(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85      1814
           1       0.85      0.85      0.85      1828
           2       0.87      0.86      0.87      1803
           3       0.72      0.76      0.74      1719
           4       0.88      0.88      0.88      1812
           5       0.82      0.82      0.82      1768
           6       0.82      0.82      0.82      1832
           7       0.87      0.88      0.87      1808
           8       0.87      0.75      0.81      1812
           9       0.83      0.83      0.83      1804

    accuracy                           0.83     18000
   macro avg       0.84      0.83      0.83     18000
weighted avg       0.84      0.83      0.83     18000
```

```
#Let's build another model with 'adam' optimiser
model2 = tf.keras.Sequential()
model2.add(tf.keras.layers.Reshape(target_shape=(1024,),input_shape=(32,32)))
model2.add(tf.keras.layers.BatchNormalization())
model2.add(tf.keras.layers.Dense(units=500,activation = 'relu'))
model2.add(tf.keras.layers.Dropout(rate=0.2))
model2.add(tf.keras.layers.BatchNormalization())
model2.add(tf.keras.layers.Dense(units=50,activation='relu'))
model2.add(tf.keras.layers.Dropout(rate=0.2))
model2.add(tf.keras.layers.BatchNormalization())
model2.add(tf.keras.layers.Dense(units=10,activation='softmax'))
model2.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

model2.fit(x=X_train,y=y_train,epochs=25,validation_data=(X_val,y_val),batch_size=32)
```

```
Train on 42000 samples, validate on 60000 samples
Epoch 1/25
42000/42000 [==============================] - 18s 437us/sample - loss: 1.7134 - acc: 0.
Epoch 2/25
42000/42000 [==============================] - 17s 406us/sample - loss: 1.2885 - acc: 0.
Epoch 3/25
42000/42000 [==============================] - 17s 397us/sample - loss: 1.1592 - acc: 0.
Epoch 4/25
42000/42000 [==============================] - 17s 397us/sample - loss: 1.0791 - acc: 0.
Epoch 5/25
42000/42000 [==============================] - 17s 399us/sample - loss: 1.0137 - acc: 0.
Epoch 6/25
42000/42000 [==============================] - 17s 405us/sample - loss: 0.9633 - acc: 0.
Epoch 7/25
42000/42000 [==============================] - 17s 406us/sample - loss: 0.9207 - acc: 0.
Epoch 8/25
42000/42000 [==============================] - 17s 395us/sample - loss: 0.9030 - acc: 0.
Epoch 9/25
42000/42000 [==============================] - 16s 393us/sample - loss: 0.8756 - acc: 0.
Epoch 10/25
42000/42000 [==============================] - 16s 391us/sample - loss: 0.8569 - acc: 0.
Epoch 11/25
42000/42000 [==============================] - 16s 391us/sample - loss: 0.8292 - acc: 0.
Epoch 12/25
42000/42000 [==============================] - 16s 390us/sample - loss: 0.8108 - acc: 0.
Epoch 13/25
42000/42000 [==============================] - 17s 394us/sample - loss: 0.8066 - acc: 0.
Epoch 14/25
42000/42000 [==============================] - 17s 394us/sample - loss: 0.7952 - acc: 0.
Epoch 15/25
42000/42000 [==============================] - 17s 410us/sample - loss: 0.7745 - acc: 0.
Epoch 16/25
42000/42000 [==============================] - 17s 400us/sample - loss: 0.7646 - acc: 0.
Epoch 17/25
42000/42000 [==============================] - 17s 398us/sample - loss: 0.7604 - acc: 0.
Epoch 18/25
42000/42000 [==============================] - 17s 395us/sample - loss: 0.7504 - acc: 0.
Epoch 19/25
42000/42000 [==============================] - 17s 408us/sample - loss: 0.7416 - acc: 0.
Epoch 20/25
42000/42000 [==============================] - 17s 407us/sample - loss: 0.7346 - acc: 0.
Epoch 21/25
42000/42000 [==============================] - 17s 405us/sample - loss: 0.7185 - acc: 0.
Epoch 22/25
42000/42000 [==============================] - 17s 400us/sample - loss: 0.7173 - acc: 0.
Epoch 23/25
42000/42000 [==============================] - 17s 394us/sample - loss: 0.7080 - acc: 0.
Epoch 24/25
42000/42000 [==============================] - 16s 388us/sample - loss: 0.6967 - acc: 0.
Epoch 25/25
42000/42000 [==============================] - 16s 388us/sample - loss: 0.7082 - acc: 0.
<tensorflow.python.keras.callbacks.History at 0x7f7f5f11d748>
```

```
accuracy_score(y_true=f['y_test'],y_pred=model2.predict_classes(X_test))
```

0.8543888888888889

```
print(classification_report(y_true=f['y_test'],y_pred=model2.predict_classes(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1814
           1       0.83      0.87      0.85      1828
           2       0.89      0.85      0.87      1803
           3       0.83      0.80      0.82      1719
           4       0.89      0.87      0.88      1812
           5       0.85      0.85      0.85      1768
           6       0.89      0.83      0.86      1832
           7       0.84      0.91      0.87      1808
           8       0.90      0.78      0.83      1812
           9       0.82      0.86      0.84      1804

    accuracy                           0.85     18000
   macro avg       0.86      0.85      0.85     18000
weighted avg       0.86      0.85      0.85     18000
```

## Trade off between KNN and Deep neural networks

1. KNN models are simple and easy to construct as compared to Deep Neural networks.
2. KNN models however produce very poor precesion as compared to Neural networks.
3. KNN models are very slow when applied on image data, Neural networks are quite faster models on image da
4. The added complexity in neural netwoks results in good accuracy on complex data like image.