```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
Instructions for updating:
non-resource variables are not supported in the long term
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.io import loadmat
from skimage import color
from skimage import io
from sklearn.model_selection import train_test_split

import tensorflow as tf
%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=9473

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

```
import h5py

# reading the h1py file
data = h5py.File('/content/drive/My Drive/22dec/SVHN_single_grey1.h5', 'r')

 list(data.keys())
```

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

```
# respective values for each key in the h1py file
list(data.values())
```

```
[<HDF5 dataset "X_test": shape (18000, 32, 32), type "<f4">,
 <HDF5 dataset "X_train": shape (42000, 32, 32), type "<f4">,
 <HDF5 dataset "X_val": shape (60000, 32, 32), type "<f4">,
 <HDF5 dataset "y_test": shape (18000,), type "|u1">,
 <HDF5 dataset "y_train": shape (42000,), type "|u1">,
 <HDF5 dataset "y_val": shape (60000,), type "|u1">]
```

```
derive = list(data.keys())
```

```
# Data fetching and understand the train/val/test splits.


# Get the data
X_test = list(data[derive[0]])
X_train= list(data[derive[1]])
X_val=list(data[derive[2]])
y_test=list(data[derive[3]])
y_train=list(data[derive[4]])
y_val=list(data[derive[5]])


from numpy import array


x_train= array(X_train)
x_test=array(X_test)
y_train=array(y_train)
y_test=array(y_test)


x_train[0]
```

```
array([[ 33.0704,  30.2601,  26.852 , ...,  71.4471,  58.2204,  42.9939],
       [ 25.2283,  25.5533,  29.9765, ..., 113.0209, 103.3639,  84.2949],
       [ 26.2775,  22.6137,  40.4763, ..., 113.3028, 121.775 , 115.4228],
       ...,
       [ 28.5502,  36.212 ,  45.0801, ...,  24.1359,  25.0927,  26.0603],
       [ 38.4352,  26.4733,  23.2717, ...,  28.1094,  29.4683,  30.0661],
       [ 50.2984,  26.0773,  24.0389, ...,  49.6682,  50.853 ,  53.0377]],
      dtype=float32)
```

```
y_train
```

```
array([2, 6, 7, ..., 7, 0, 4], dtype=uint8)
```

```
#Converting train and test labels to one hot vectors


trainY = tf.keras.utils.to_categorical(y_train, num_classes=10)
testY = tf.keras.utils.to_categorical(y_test, num_classes=10)


x_train/= 255
x_test/= 255


x_train = x_train.reshape(x_train.shape[0], 1024)


x_test = x_test.reshape(x_test.shape[0], 1024)


#● Implement and apply an optimal k-Nearest Neighbor (kNN) classifier (7.5 points)


from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
import numpy as np
import pandas as pd
```

```
P2=np.arange(1,30,2)


accuracy_scores=[]
for i in P2:
# Instantiate the model .
  knn = KNeighborsClassifier(n_neighbors=i,algorithm='brute')
# Fit the model on the training data.
  knn=knn.fit(x_train, y_train)
# Predict the response for test dataset
  y_pred = knn.predict(x_test)
# Storing the results
  accuracy_scores.append(metrics.accuracy_score(y_test,y_pred))


knn_model = KNeighborsClassifier(n_neighbors=27,algorithm='brute')
knn_model = knn_model.fit(x_train, y_train)
y_pred_final = knn_model.predict(x_test)


# Getting the accuracy score for test data
knn_model.score(x_test, y_test)*100
```

```
53.227777777777774
```

```
print(metrics.classification_report(y_test, y_pred_final))
#Precision – Accuracy of positive predictions
#Recall - Fraction of positives That were correctly identified.
#F1 Score - harmonic mean of precision and recall
```

```
              precision    recall  f1-score   support

           0       0.46      0.71      0.56      1814
           1       0.46      0.73      0.56      1828
           2       0.64      0.54      0.58      1803
           3       0.47      0.43      0.45      1719
           4       0.64      0.65      0.65      1812
           5       0.53      0.40      0.45      1768
           6       0.51      0.41      0.46      1832
           7       0.71      0.62      0.66      1808
           8       0.47      0.37      0.41      1812
           9       0.55      0.44      0.49      1804

    accuracy                           0.53     18000
   macro avg       0.54      0.53      0.53     18000
weighted avg       0.54      0.53      0.53     18000
```

```
# Implement and apply a deep neural network classifier including (feedforward neural netwo


#Clear out tensorflow memory
tf.keras.backend.clear_session()

#Initialize Sequential model
model1 = tf.keras.models.Sequential()
```

```
model1 = tf.keras.models.Sequential()

#Normalize the data
model1.add(tf.keras.layers.BatchNormalization())

#Add 1st hidden layer
model1.add(tf.keras.layers.Dense(200, activation='relu'))

#Add 2nd hidden layer
model1.add(tf.keras.layers.Dense(100, activation='relu'))

#Add OUTPUT layer
model1.add(tf.keras.layers.Dense(10, activation='softmax'))

#Create optimizer with non-default learning rate
sgd_optimizer = tf.keras.optimizers.SGD(lr=0.03)

#Compile the model
model1.compile(optimizer=sgd_optimizer, loss='categorical_crossentropy', metrics=['accurac


model1.fit(x_train,trainY,
          validation_data=(x_test,testY),
          epochs=50,
          batch_size=24)
```

```
42000/42000 [==============================] - 7s 175us/sample - loss: 0.4887 - acc:
Epoch 23/50
42000/42000 [==============================] - 7s 176us/sample - loss: 0.4828 - acc:
Epoch 24/50
42000/42000 [==============================] - 7s 176us/sample - loss: 0.4760 - acc:
Epoch 25/50
42000/42000 [==============================] - 7s 174us/sample - loss: 0.4779 - acc:
Epoch 26/50
42000/42000 [==============================] - 7s 171us/sample - loss: 0.4623 - acc:
Epoch 27/50
42000/42000 [==============================] - 7s 171us/sample - loss: 0.4573 - acc:
Epoch 28/50
42000/42000 [==============================] - 7s 171us/sample - loss: 0.4564 - acc:
Epoch 29/50
42000/42000 [==============================] - 7s 171us/sample - loss: 0.4424 - acc:
Epoch 30/50
42000/42000 [==============================] - 7s 172us/sample - loss: 0.4514 - acc:
Epoch 31/50
42000/42000 [==============================] - 7s 171us/sample - loss: 0.4435 - acc:
Epoch 32/50
42000/42000 [==============================] - 7s 171us/sample - loss: 0.4352 - acc:
Epoch 33/50
42000/42000 [==============================] - 7s 170us/sample - loss: 0.4288 - acc:
Epoch 34/50
42000/42000 [==============================] - 7s 172us/sample - loss: 0.4299 - acc:
Epoch 35/50
42000/42000 [==============================] - 7s 171us/sample - loss: 0.4216 - acc:
Epoch 36/50
42000/42000 [==============================] - 7s 172us/sample - loss: 0.4166 - acc:
Epoch 37/50
42000/42000 [==============================] - 8s 180us/sample - loss: 0.4095 - acc:
Epoch 38/50
42000/42000 [==============================] - 8s 180us/sample - loss: 0.4144 - acc:
Epoch 39/50
42000/42000 [==============================] - 7s 173us/sample - loss: 0.4084 - acc:
Epoch 40/50
42000/42000 [==============================] - 7s 172us/sample - loss: 0.4010 - acc:
Epoch 41/50
42000/42000 [==============================] - 7s 171us/sample - loss: 0.3978 - acc:
Epoch 42/50
42000/42000 [==============================] - 7s 171us/sample - loss: 0.4015 - acc:
Epoch 43/50
42000/42000 [==============================] - 7s 174us/sample - loss: 0.3977 - acc:
Epoch 44/50
42000/42000 [==============================] - 7s 171us/sample - loss: 0.3884 - acc:
Epoch 45/50
42000/42000 [==============================] - 7s 172us/sample - loss: 0.3903 - acc:
Epoch 46/50
42000/42000 [==============================] - 7s 172us/sample - loss: 0.3873 - acc:
Epoch 47/50
42000/42000 [==============================] - 7s 172us/sample - loss: 0.3820 - acc:
Epoch 48/50
42000/42000 [==============================] - 7s 172us/sample - loss: 0.3830 - acc:
Epoch 49/50
42000/42000 [==============================] - 7s 172us/sample - loss: 0.3777 - acc:
Epoch 50/50
42000/42000 [==============================] - 7s 170us/sample - loss: 0.3759 - acc:
<tensorflow.python.keras.callbacks.History at 0x7f72765f1668>
```

```
predictionsNN = model1.predict(x_test)

ClassificationReportNN = metrics.classification_report(testY.argmax(axis=1), predictionsNN

print(ClassificationReportNN)
```