ANN_Project

June 26, 2024

0.0.1 ARTIFICIAL NEURAL NETWORKS PROJECT

0.1 Background and Context

The ability to process visual information using machine learning algorithms can be very useful as demonstrated in various applications. The Street View House Numbers (SVHN) dataset is one of the most popular ones. It has been used in neural networks created by Google to read house numbers and match them to their geolocations. This is a great benchmark dataset to play with, learn and train models that accurately identify street numbers, and incorporate them into all sorts of projects.

Objective:

In this project, we will use a dataset with images centered around a single digit (many of the images do contain some distractors at the sides). Given the dataset, our aim is to build a model that can identify house numbers in an image.

Dataset:

The dataset has the following features: Number of classes: 10 Training data: 42000 images Testing data: 18000 images

Note: The dataset also contains validation data which will not be used. For purposes of the project I aim to use all data and not a sample of 2000 as suggested.

Import Libraries

[ ]:
```
# Ensure the necessary packages are installed
!pip install h5py tensorflow numpy pandas matplotlib seaborn scikit-learn

# Import necessary libraries
import h5py
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Activation, Dense, BatchNormalization, Dropout,␣
 ↪Flatten, Conv2D, MaxPooling2D
from keras.utils.np_utils import to_categorical
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, CSVLogger
from sklearn.model_selection import train_test_split
```

```
Requirement already satisfied: h5py in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (3.1.0)
Requirement already satisfied: tensorflow in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (2.7.0)
Requirement already satisfied: numpy in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (1.19.5)
Requirement already satisfied: pandas in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (1.2.4)
Requirement already satisfied: matplotlib in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (3.3.4)
Requirement already satisfied: seaborn in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (0.11.1)
Requirement already satisfied: scikit-learn in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (1.0)
Requirement already satisfied: pillow>=6.2.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from matplotlib)
(8.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from matplotlib)
(2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from matplotlib)
(1.3.1)
Requirement already satisfied: cycler>=0.10 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from matplotlib)
(0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from matplotlib)
(2.8.1)
Requirement already satisfied: six in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
cycler>=0.10->matplotlib) (1.15.0)
Requirement already satisfied: pytz>=2017.3 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from pandas)
(2021.1)
Requirement already satisfied: scipy>=1.1.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from scikit-
learn) (1.6.2)
Requirement already satisfied: joblib>=0.11 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from scikit-
learn) (1.0.1)
```

```
Requirement already satisfied: threadpoolctl>=2.0.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from scikit-
learn) (2.1.0)
Requirement already satisfied: wrapt>=1.11.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(1.12.1)
Requirement already satisfied: tensorflow-estimator<2.8,~=2.7.0rc0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(2.7.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(3.7.4.3)
Requirement already satisfied: absl-py>=0.4.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(0.15.0)
Requirement already satisfied: termcolor>=1.1.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(1.1.0)
Requirement already satisfied: tensorboard~=2.6 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(2.7.0)
Requirement already satisfied: gast<0.5.0,>=0.2.1 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(0.4.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.21.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(0.23.1)
Requirement already satisfied: astunparse>=1.6.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(1.6.3)
Requirement already satisfied: keras-preprocessing>=1.1.1 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(1.1.2)
Requirement already satisfied: opt-einsum>=2.3.2 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(3.3.0)
Requirement already satisfied: flatbuffers<3.0,>=1.12 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(1.12)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(1.41.0)
Requirement already satisfied: libclang>=9.0.1 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(12.0.0)
Requirement already satisfied: wheel<1.0,>=0.32.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(0.36.2)
```

```
Requirement already satisfied: keras<2.8,>=2.7.0rc0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(2.7.0)
Requirement already satisfied: google-pasta>=0.1.1 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(0.2.0)
Requirement already satisfied: protobuf>=3.9.2 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from tensorflow)
(3.19.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
tensorboard~=2.6->tensorflow) (2.3.0)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
tensorboard~=2.6->tensorflow) (0.6.1)
Requirement already satisfied: werkzeug>=0.11.15 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
tensorboard~=2.6->tensorflow) (1.0.1)
Requirement already satisfied: requests<3,>=2.21.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
tensorboard~=2.6->tensorflow) (2.25.1)
Requirement already satisfied: markdown>=2.6.8 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
tensorboard~=2.6->tensorflow) (3.3.4)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
tensorboard~=2.6->tensorflow) (0.4.6)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
tensorboard~=2.6->tensorflow) (1.8.0)
Requirement already satisfied: setuptools>=41.0.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
tensorboard~=2.6->tensorflow) (52.0.0.post20210125)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from google-
auth<3,>=1.6.3->tensorboard~=2.6->tensorflow) (4.2.4)
Requirement already satisfied: rsa<5,>=3.1.4 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from google-
auth<3,>=1.6.3->tensorboard~=2.6->tensorflow) (4.7.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from google-
auth<3,>=1.6.3->tensorboard~=2.6->tensorflow) (0.2.8)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from google-auth-
oauthlib<0.5,>=0.4.1->tensorboard~=2.6->tensorflow) (1.3.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard~=2.6->tensorflow)
```

```
(0.4.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (1.26.4)
Requirement already satisfied: certifi>=2017.4.17 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (2020.12.5)
Requirement already satisfied: idna<3,>=2.5 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from
requests<3,>=2.21.0->tensorboard~=2.6->tensorflow) (4.0.0)
Requirement already satisfied: oauthlib>=3.0.0 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from requests-
oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard~=2.6->tensorflow)
(3.1.1)
```

Load Data

[1]:
```
import os
os.path.abspath("SVHN_single_grey1.h5")
```

[1]:
```
'/Users/arturocasasa/Documents/Texas McCombs.Machine Learning and AI/10. Deep
Learning. Intro to Neural Networks/Additional Project/SVHN_single_grey1.h5'
```

[2]:
```
!pip install h5py #Ensuring h5 files can be read
```

```
Requirement already satisfied: h5py in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (3.1.0)
Requirement already satisfied: numpy>=1.17.5 in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from h5py)
(1.19.5)
```

[3]:
```
# Define the path to your .h5 file
file_path = os.path.expanduser('~/Documents/Texas McCombs.Machine Learning and␣
 ↪AI/10. Deep Learning. Intro to Neural Networks/Additional Project/
 ↪SVHN_single_grey1.h5')
```

[4]:
```
import h5py # Testing h5py

!pip install h5py

import h5py
import os
```

```
Requirement already satisfied: h5py in
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (3.1.0)
Requirement already satisfied: numpy>=1.17.5 in
```

```
/Users/arturocasasa/opt/anaconda3/lib/python3.8/site-packages (from h5py)
(1.19.5)
```

[5]:
```
h5f = h5py.File(file_path, 'r')
```

[6]:
```
# Load the datasets
X_train = h5f['X_train'][:]
y_train = h5f['y_train'][:]
X_test = h5f['X_test'][:]
y_test = h5f['y_test'][:]
```

[7]:
```
# Close the file
h5f.close()
```

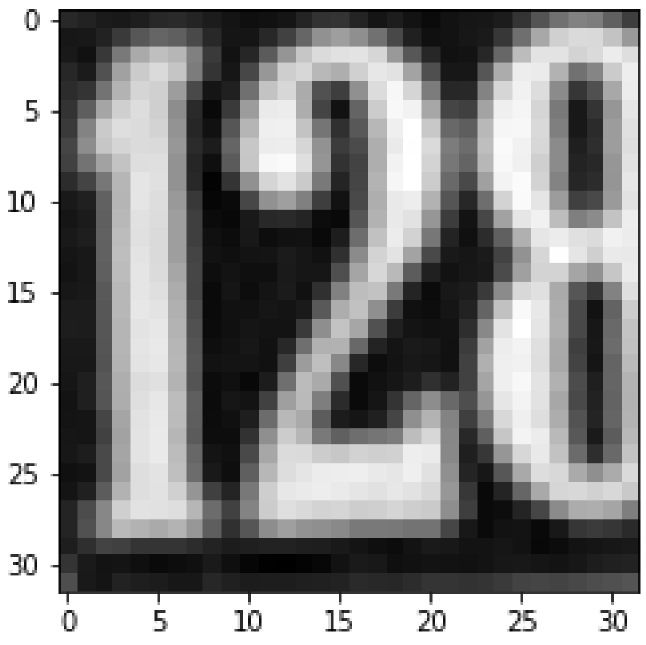
Visualizing Data

[9]:
```
# Print the shape of the datasets to verify
print("Training data X shape:", X_train.shape)
print("Training data y shape:", y_train.shape)
print("Testing data X shape:", X_test.shape)
print("Testing data y shape:", y_test.shape)
```

```
Training data X shape: (42000, 32, 32)
Training data y shape: (42000,)
Testing data X shape: (18000, 32, 32)
Testing data y shape: (18000,)
```

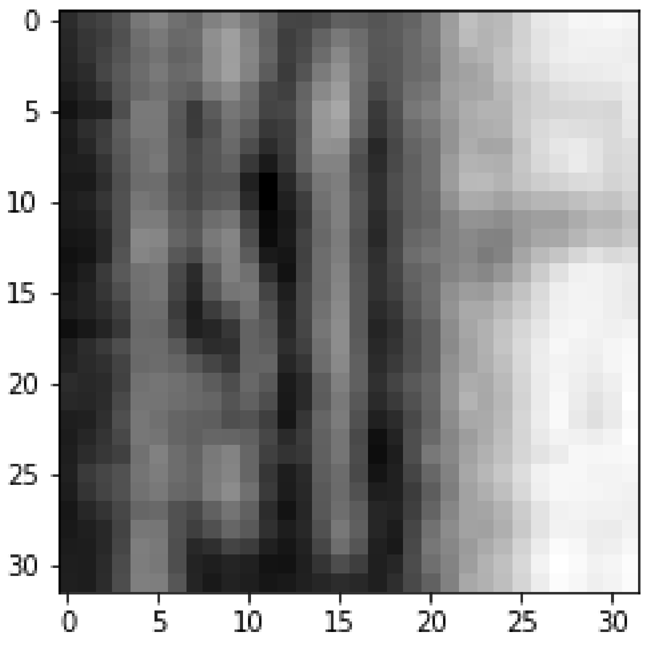
[13]:
```
# Visualizing some of the data
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8, 8))
columns = 10
rows = 10
for i in range(1, columns * rows + 1):
    img = X_test[i]
    fig.add_subplot(rows, columns, i)
    plt.imshow(img, cmap='gray')
plt.show()
```

```
[15]: # show the number in the dataset
      plt.imshow(X_train[0], cmap='gray')
      plt.show()
      print('Label: ', y_train[0])
```



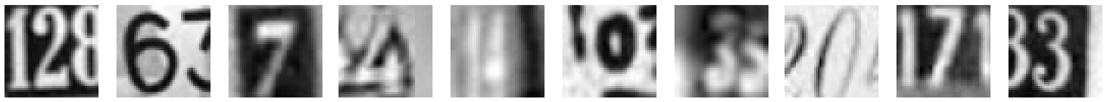
```
Label:  2
```

[16]:
```
# Show the first image in the testing dataset
plt.imshow(X_test[0], cmap='gray')
plt.show()
print('Label: ', y_test[0])
```



```
Label:  1
```

[17]:
```
# Visualizing the first 10 images in the training dataset and their labels
plt.figure(figsize=(10, 1))
for i in range(10):
    plt.subplot(1, 10, i+1)
    plt.imshow(X_train[i].reshape(32, 32), cmap='gray')
    plt.axis('off')
plt.show()
print('Label for each of the above image: %s' % (y_train[:10]))
```



```
Label for each of the above image: [2 6 7 4 4 0 3 0 7 3]
```

[18]:
```
# Close this file
#h5f.close()
```

Optimal K-Nearest Neighbour Classifier -No Neural Network-

[19]:
```
# Reshape data from 2D to 1D -> 32x32 to 1024
X_train_reshaped = X_train.reshape(42000, 1024)
X_test_reshaped = X_test.reshape(18000, 1024)
```

[24]:
```
# Initializing the value of k and finding the accuracies on validation data
k_test = range(1, 30, 2)
accuracies = []
```

[28]:
```
# Retraining the model using the best k value and predict the labels on test␣
 ↪data
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=k_test[i])
knn.fit(X_train, y_train)
predictions = knn.predict(X_test)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-28-0cd7716dc6e7> in <module>
      2 from sklearn.neighbors import KNeighborsClassifier
      3 knn = KNeighborsClassifier(n_neighbors=k_test[i])
----> 4 knn.fit(X_train, y_train)
```

```
      5 predictions = knn.predict(X_test)

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/neighbors/_classification.p␣
 ↪in fit(self, X, y)
    196         self.weights = _check_weights(self.weights)
    197
--> 198         return self._fit(X, y)
    199
    200     def predict(self, X):

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/neighbors/_base.py in␣
 ↪_fit(self, X, y)
    398         if self._get_tags()["requires_y"]:
    399             if not isinstance(X, (KDTree, BallTree, NeighborsBase)):
--> 400                 X, y = self._validate_data(X, y, accept_sparse="csr",␣
 ↪multi_output=True)
    401
    402             if is_classifier(self):

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/base.py in␣
 ↪_validate_data(self, X, y, reset, validate_separately, **check_params)
    570                 y = check_array(y, **check_y_params)
    571             else:
--> 572                 X, y = check_X_y(X, y, **check_params)
    573             out = X, y
    574

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py in␣
 ↪check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy,␣
 ↪force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples,␣
 ↪ensure_min_features, y_numeric, estimator)
    954         raise ValueError("y cannot be None")
    955
--> 956     X = check_array(

    957         X,
    958         accept_sparse=accept_sparse,

~/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py in␣
 ↪check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy,␣
 ↪force_all_finite, ensure_2d, allow_nd, ensure_min_samples,␣
 ↪ensure_min_features, estimator)
    784                 ) from e
    785         if not allow_nd and array.ndim >= 3:
--> 786             raise ValueError(

    787                 "Found array with dim %d. %s expected <= 2."
    788                 % (array.ndim, estimator_name)
```

```
ValueError: Found array with dim 3. Estimator expected <= 2.
```

```
[22]: for k in k_test:
          knn = KNeighborsClassifier(n_neighbors=k)
          knn.fit(X_train_reshaped, y_train)
          score = knn.score(X_test_reshaped, y_test)
          print("k value=%d, accuracy score=%.2f%%" % (k, score * 100))
          accuracies.append(score)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-db648ce6ee47> in <module>
      1 for k in k_test:
----> 2     knn = KNeighborsClassifier(n_neighbors=k)
      3     knn.fit(X_train_reshaped, y_train)
      4     score = knn.score(X_test_reshaped, y_test)
      5     print("k value=%d, accuracy score=%.2f%%" % (k, score * 100))

NameError: name 'KNeighborsClassifier' is not defined
```

Metrics Report

```
[ ]: # show a final classification report demonstrating the accuracy of the␣
      ↪classifier
     print("EVALUATION ON TESTING DATA")
     print(confusion_matrix(y_test1,predictions))
     print(classification_report(y_test1, predictions))
```

The accuracy of the model is at 0.53, same as tossing a coin

Predicting digits using previous KNN classifier

```
[ ]: plt.figure(figsize=(2,2))
     plt.imshow(X_test[59].reshape(32,32))
     plt.show()
     image = X_test[15]
     print(knn.predict(image.reshape(1, -1)))
```

The prediction is 6 while the number is 9

Immplementing Neural Networks

Converting output to 10 classes

```
[ ]: y_train1 = tf.keras.utils.to_categorical(y_train1, num_classes=10)
```

```
[ ]: y_test1 = tf.keras.utils.to_categorical(y_test1, num_classes=10)
[ ]: print(X_train.shape, X_test.shape, y_train1.shape, y_test1.shape)
```

```
[ ]: # fix random seed for reproducibility
     seed = 7
     np.random.seed(seed)
```

```
[ ]: # normalize inputs from 0-255 to 0-1; to be used with the NN
     X_train = X_train / 255.0
     X_test = X_test / 255.0
```

```
[ ]: print(X_train.shape, X_test.shape, y_train1.shape, y_test1.shape)
```

Initializing and Implementing Neural Network

```
[ ]: ##Initialize the Artificial Neural Network Classifier
     classifier_model = Sequential()
```

```
[ ]: # Input Layer
     #Adding Input layer and activation functions ReLU
     classifier_model.add(Dense(512, kernel_initializer='he_normal',input_shape =␣
      ↪(1024, )))
     #Adding Activation function
     classifier_model.add(Activation('relu'))

     #Hidden Layer 1
     #Adding first Hidden layer
     classifier_model.add(Dense(256, kernel_initializer='he_normal'))
     #Adding Activation function
     classifier_model.add(Activation('relu'))

     #Hidden Layer 2
     #Adding second Hidden layer
     classifier_model.add(Dense(128, kernel_initializer='he_normal'))
     #Adding Activation function
     classifier_model.add(Activation('relu'))

     #Hidden Layer 3
     #Adding third Hidden layer
     classifier_model.add(Dense(64, kernel_initializer='he_normal'))
     #Adding Activation function
     classifier_model.add(Activation('relu'))

     #Hidden Layer 4
     #Adding fourth Hidden layer
     classifier_model.add(Dense(32, kernel_initializer='he_normal'))
     #Adding Activation function
     classifier_model.add(Activation('relu'))
```

```
# Output Layer
#Adding output layer which is of 10 nodes (digits)
classifier_model.add(Dense(10))
#Adding Activation function
# Here, we are using softmax function because we have multiclass classsification
classifier_model.add(Activation('softmax'))
```

```
[ ]: classifier_model.summary()
```

```
[ ]: #Checkign data shape one more time
     print(X_train.shape, X_test.shape, y_train1.shape, y_test1.shape)
```

USING SGD IN CLASSIFIER

```
[ ]: # compiling the ANN classifier
     classifier_model.compile(optimizer = 'sgd', loss = 'categorical_crossentropy',␣
      ↪metrics = ['accuracy'])
```

```
[ ]: # Fitting the ANN to the Training data
     history = classifier_model.fit(X_train, y_train1,␣
      ↪validation_data=(X_test,y_test1), batch_size = 200, epochs = 50, verbose = 1)
```

```
[ ]: results = classifier_model.evaluate(X_test, y_test1)
     print('Test accuracy using sigmoid : ', results[1])
```

```
[ ]: #Store the accuracy results for each model for final comparison
     results_on_test = pd.DataFrame({'Method':['NN + SGD'], 'accuracy':␣
      ↪results[1]},index={'1'})
     results_on_test = results_on_test[['Method', 'accuracy']]
     results_on_test
```

```
[ ]: # Capturing learning history per epoch
     hist  = pd.DataFrame(history.history)
     hist['epoch'] = history.epoch

     # Plotting accuracy at different epochs
     plt.plot(hist['loss'])
     plt.plot(hist['val_loss'])
     plt.legend(("train" , "valid") , loc =0)

     #Printing results
     results = classifier_model.evaluate(X_test, y_test1)
```

USING ADAM CLASSIFIER

[ ]:
```
# compiling the ANN classifier
classifier_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy',␣
 ↪metrics = ['accuracy'])
```

[ ]:
```
# Fitting the ANN to the Training data
history = classifier_model.fit(X_train, y_train1,␣
 ↪validation_data=(X_test,y_test1),batch_size = 200, epochs = 50, verbose = 1)
```

[ ]:
```
results = classifier_model.evaluate(X_test, y_test1)
print('Test accuracy using simple Adam : ', results[1])
```

[ ]:
```
# Capturing learning history per epoch
hist  = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

# Plotting accuracy at different epochs
plt.plot(hist['loss'])
plt.plot(hist['val_loss'])
plt.legend(("train" , "valid") , loc =0)

#Printing results
results = classifier_model.evaluate(X_test, y_test1)
```

[ ]:
```
#Store the accuracy results for each model in a dataframe for final comparison
tempResultsDf = pd.DataFrame({'Method':['NN + Adam'], 'accuracy':␣
 ↪[results[1]]},index={'2'})
results_on_test= pd.concat([results_on_test, tempResultsDf])
results_on_test = results_on_test[['Method', 'accuracy']]
results_on_test
```

USING SGD CLASSIFIER with increased Epochs

[ ]:
```
# compiling the ANN classifier
classifier_model.compile(optimizer = 'sgd', loss = 'categorical_crossentropy',␣
 ↪metrics = ['accuracy'])
```

[ ]:
```
# Fitting the ANN to the Training data
history = classifier_model.fit(X_train, y_train1,␣
 ↪validation_data=(X_test,y_test1),batch_size = 200, epochs = 100, verbose = 1)
```

[ ]:
```
results = classifier_model.evaluate(X_test, y_test1)
print('Test accuracy using SGD + 100 Epochs: ', results[1])
```

[ ]:
```
# Capturing learning history per epoch
hist  = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
```

```
# Plotting accuracy at different epochs
plt.plot(hist['loss'])
plt.plot(hist['val_loss'])
plt.legend(("train" , "valid") , loc =0)

#Printing results
results = classifier_model.evaluate(X_test, y_test1)
```

[ ]:
```
#Store the accuracy results for each model in a dataframe for final comparison
tempResultsDf = pd.DataFrame({'Method':['NN + SGD + 100Epoch'], 'accuracy':␣
 ↪[results[1]]},index={'2'})
results_on_test= pd.concat([results_on_test, tempResultsDf])
results_on_test = results_on_test[['Method', 'accuracy']]
```

[ ]:
```
results_on_test
```

USING ADAM CLASSIFIER with increased Epochs

[ ]:
```
# compiling the ANN classifier
classifier_model.compile(optimizer = 'Adam', loss = 'categorical_crossentropy',␣
 ↪metrics = ['accuracy'])
```

[ ]:
```
# Fitting the ANN to the Training data
history = classifier_model.fit(X_train, y_train1,␣
 ↪validation_data=(X_test,y_test1),batch_size = 200, epochs = 100, verbose = 1)
```

[ ]:
```
results = classifier_model.evaluate(X_test, y_test1)
print('Test accuracy using Adam + 100 Epochs: ', results[1])
```

[ ]:
```
# Capturing learning history per epoch
hist  = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

# Plotting accuracy at different epochs
plt.plot(hist['loss'])
plt.plot(hist['val_loss'])
plt.legend(("train" , "valid") , loc =0)

#Printing results
results = classifier_model.evaluate(X_test, y_test1)
```

[ ]:
```
#Store the accuracy results for each model in a dataframe for final comparison
tempResultsDf = pd.DataFrame({'Method':['NN + ADAM + 100Epoch'], 'accuracy':␣
 ↪[results[1]]},index={'2'})
results_on_test= pd.concat([results_on_test, tempResultsDf])
results_on_test = results_on_test[['Method', 'accuracy']]
results_on_test
```

```
[ ]:
```

Implementing early stopping on SGD + 100 Epoch

```
[ ]: def create_model():
         #Initializing the neural network
         classifier_model = Sequential()#Adding Input layer and activation␣
     ↪functions ReLU
         classifier_model.add(Dense(512,␣
     ↪kernel_initializer='he_normal',input_shape = (1024, )))
         #Adding Activation function
         classifier_model.add(Activation('relu'))

         #Hidden Layer 1
         #Adding first Hidden layer
         classifier_model.add(Dense(256, kernel_initializer='he_normal'))
         #Adding Activation function
         classifier_model.add(Activation('relu'))

         #Hidden Layer 2
         #Adding second Hidden layer
         classifier_model.add(Dense(128, kernel_initializer='he_normal'))
         #Adding Activation function
         classifier_model.add(Activation('relu'))

         #Hidden Layer 3
         #Adding third Hidden layer
         classifier_model.add(Dense(64, kernel_initializer='he_normal'))
         #Adding Activation function
         classifier_model.add(Activation('relu'))

         #Hidden Layer 4
         #Adding fourth Hidden layer
         classifier_model.add(Dense(32, kernel_initializer='he_normal'))
         #Adding Activation function
         classifier_model.add(Activation('relu'))

         # Output Layer
         #Adding output layer which is of 10 nodes (digits)
         classifier_model.add(Dense(10))
         #Adding Activation function
         # Here, we are using softmax function because we have multiclass␣
     ↪classsification
         classifier_model.add(Activation('softmax'))

         #Compiling the ANN with RMSprop optimizer and binary cross entropy loss␣
     ↪function
```

```
    optimizer = tf.keras.optimizers.Adam(0.001)
    classifier_model.
↪compile(loss='binary_crossentropy',optimizer=optimizer,metrics=['accuracy'])
    return classifier_model
```

```
[ ]: #Importing classback API
from keras import callbacks
es_cb = callbacks.EarlyStopping(monitor='val_loss', min_delta=0.001, patience=5)
model_e=create_model()
# compiling the ANN classifier
classifier_model.compile(optimizer = 'SGD', loss = 'categorical_crossentropy',␣
↪metrics = ['accuracy'])
#Fitting the ANN with batch_size = 200 and 100 epochs
history_e = model_e.
↪fit(X_train,y_train1,batch_size=200,epochs=100,verbose=1,validation_data=(X_test,y_test1),c
```

Training stopped at epoch 26 since val loss is starting to increase

```
[ ]: #Plotting Train Loss vs Validation Loss
plt.plot(history_e.history['loss'])
plt.plot(history_e.history['val_loss'])
plt.title('model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
[ ]: #Store the accuracy results for each model in a dataframe for final comparison
tempResultsDf = pd.DataFrame({'Method':['NN + SGD + 100Epoch +EStop'],␣
↪'accuracy': [results[1]]},index={'2'})
results_on_test= pd.concat([results_on_test, tempResultsDf])
results_on_test = results_on_test[['Method', 'accuracy']]
results_on_test
```

```
[ ]:
```

```
[ ]:
```

Predicting the results using NN classifier on test data

```
[ ]: #Showing the image
plt.imshow(X_test[5].reshape(32,32),cmap='gray')
```

```
[ ]: #Predicting the digits
prediction = classifier_model.predict(X_test[5])
print(prediction.argmax())
```

TESTING RESULTS

[ ]:
```
results_on_test
```

- The best result was obtained with neural networks, sigmoid optimizer and 100 Epochs
- Not listed but KNN withouth NN amounted to 53% accuracy
- There is no significant difference in between SGD and Adam
- Early stop provides as good accuracy as using 100 Epochs
- I tried testing the predictions of the model but I could not find the right function to do so

CONCLUSIONS

The use of KNN gives an accuracy of about 50%

- By adding several layers of neural networks (NN) it is possible to increase accuracy up to 83% depending on optimizer utilized
- The use of SGD sigmoid or ADAM optimizers resulted in changes in accuracy of 3-4% points
- The amount of epochs utilized is important up to a certain number. I urilized early stop after using a full 100 epochs in several cases. Ther early stop aided in reducing the use of time and ended at 26 epochs
- I utlizied the full set instead of reducing samples to 2000
- It is important to mention that the set included a test and validation set which it probably was better to use for training and validate final results on the testing set