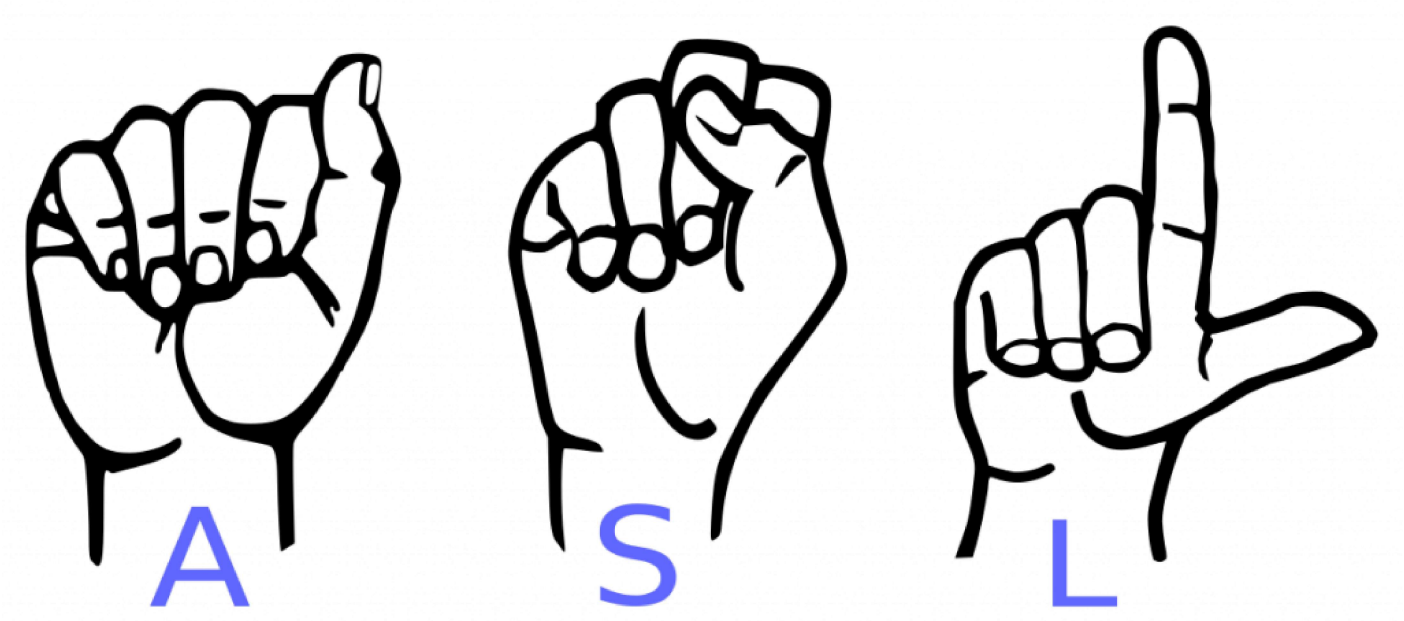
Image Processing - *American Sign Language (ASL)*



Group Members :

Name	Roll No
R.Abhinav	CB.EN.U4CSE19453
P.Kalaiarasan	CB.EN.U4CSE19446
P.Koushik	CB.EN.U4CSE19449
S.Shanthan	CB.EN.U4CSE19459

Review-1

Mounting and Importing

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

All the import statements for the libraries used in the project below lies here.

In [ ]:
```
#Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
import warnings
warnings.filterwarnings('ignore')
```

Exploring the Data

In [ ]:
```
#reading the dataset

test_df = pd.read_csv("/content/drive/MyDrive/Sign Language/datasets/sign_mnist_test.csv")
train_df = pd.read_csv("/content/drive/MyDrive/Sign Language/datasets/sign_mnist_train.csv")
```

In [ ]:
```
train_df.tail()
```

Out[ ]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel1
27450	13	189	189	190	190	192	193	193	193	193	194	19
27451	23	151	154	157	158	160	161	163	164	166	167	16
27452	18	174	174	174	174	174	175	175	174	173	173	17
27453	17	177	181	184	185	187	189	190	191	191	190	19
27454	23	179	180	180	180	182	181	182	183	182	182	18

5 rows × 785 columns

In [ ]:
```
print(type(test_df))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [ ]:
```
test_df.head()
```

Out[ ]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pi
0	6	149	149	150	150	150	151	151	150	151	152	152	
1	5	126	128	131	132	133	134	135	135	136	138	137	
2	10	85	88	92	96	105	123	135	143	147	152	157	
3	0	203	205	207	206	207	209	210	209	210	209	208	
4	3	188	191	193	195	199	201	202	203	203	203	204	

5 rows × 785 columns

In [ ]:
```
test_df.tail()
```

Out[ ]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11
7167	1	135	119	108	102	105	99	61	103	121	133	143
7168	12	157	159	161	164	166	166	171	174	175	176	176
7169	2	190	191	190	191	190	190	192	192	191	192	193
7170	4	201	205	208	209	214	216	218	223	226	229	234
7171	2	173	174	173	174	173	173	175	175	174	175	176

5 rows × 785 columns

In [ ]:
```
print(test_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7172 entries, 0 to 7171
Columns: 785 entries, label to pixel784
dtypes: int64(785)
memory usage: 43.0 MB
None
```

In [ ]:
```
# Showing the first 4 lines of the training set

train_df.iloc[:4,:10]
```

Out[ ]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9
0	3	107	118	127	134	139	143	146	150	153
1	6	155	157	156	156	156	157	156	158	158
2	2	187	188	188	187	187	186	187	188	187
3	2	211	211	212	212	211	210	211	210	210

In [ ]:
```
print(f'Number of images in the training set: {train_df.shape[0]}')
print(f'Number of images in the test set: {test_df.shape[0]}')

d = int((train_df.shape[1] - 1)**0.5)
print(f'Shape of the images: {d} x {d}')
```

```
Number of images in the training set: 27455
Number of images in the test set: 7172
Shape of the images: 28 x 28
```

In [ ]:
```
print(train_df.dtypes)
print("_________________\n")
print(train_df.dtypes.value_counts())
```

```
label       int64
pixel1      int64
pixel2      int64
pixel3      int64
pixel4      int64
            ...
pixel780    int64
pixel781    int64
pixel782    int64
pixel783    int64
pixel784    int64
Length: 785, dtype: object
_________________

int64    785
dtype: int64
```

In [ ]:
```
train_df.describe()
```

Out[ ]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	
count	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	274
mean	12.318813	145.419377	148.500273	151.247714	153.546531	156.210891	1
std	7.287552	41.358555	39.942152	39.056286	38.595247	37.111165	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	6.000000	121.000000	126.000000	130.000000	133.000000	137.000000	1
50%	13.000000	150.000000	153.000000	156.000000	158.000000	160.000000	1
75%	19.000000	174.000000	176.000000	178.000000	179.000000	181.000000	1
max	24.000000	255.000000	255.000000	255.000000	255.000000	255.000000	2

8 rows × 785 columns

Statistical Analysis of data

In [ ]:
```
train_df.mean()
```

Out[ ]:
```
label        12.318813
pixel1      145.419377
pixel2      148.500273
pixel3      151.247714
pixel4      153.546531
               ...
pixel780    162.736696
pixel781    162.906137
pixel782    161.966454
pixel783    161.137898
pixel784    159.824731
Length: 785, dtype: float64
```

In [ ]:
```
#finding the median for the numberical columns in the data set
train_df.median()
```

Out[ ]:
```
label        13.0
pixel1      150.0
pixel2      153.0
pixel3      156.0
pixel4      158.0
            ...
pixel780    184.0
pixel781    184.0
pixel782    182.0
pixel783    182.0
pixel784    182.0
Length: 785, dtype: float64
```

In [ ]:
```
# finding the mode of the numerical columns in the data set

k = train_df.mode()

k[0:1]
```

Out[ ]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pi
0	17.0	158.0	171.0	172.0	171.0	165.0	159.0	157.0	158.0	174.0	164.0	165.0	

1 rows × 785 columns

In [ ]:
```
# finding the standard deviation the dataset for the numerical values
standard_deviation = train_df.std()
print(standard_deviation)
```

```
label        7.287552
pixel1      41.358555
pixel2      39.942152
pixel3      39.056286
pixel4      38.595247
               ...
pixel780    63.444008
pixel781    63.509210
pixel782    63.298721
pixel783    63.610415
pixel784    64.396846
Length: 785, dtype: float64
```

In [ ]:
```
# finding the variance for the numerical values of the dataset
variance = standard_deviation**2
print(variance)
```

```
label         53.108415
pixel1      1710.530097
pixel2      1595.375528
pixel3      1525.393469
pixel4      1489.593076
               ...
pixel780    4025.142186
pixel781    4033.419724
pixel782    4006.728058
pixel783    4046.284910
pixel784    4146.953763
Length: 785, dtype: float64
```

Data preprocessing

- converting array to images(tensors)
- visulaizing lables and making sure that dataset is balanced
- performing encoding for lables

In [ ]:
```
# couting the missing values or cells with empty values

no_of_missing_values = train_df.isnull().sum()
total_missing_values = no_of_missing_values.sum()
print(no_of_missing_values)

print("\n\nTotal missing values in the dataset: {0}".format(total_missing_valu
es))
```

```
label       0
pixel1      0
pixel2      0
pixel3      0
pixel4      0
           ..
pixel780    0
pixel781    0
pixel782    0
pixel783    0
pixel784    0
Length: 785, dtype: int64


Total missing values in the dataset: 0
```

In [ ]:
```
# checking if there are any duplicated values

print("Duplicated Columns :",train_df.duplicated().sum())
```

```
Duplicated Columns : 0
```

In [ ]:
```
train_df
```

Out[ ]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel1
0	3	107	118	127	134	139	143	146	150	153	156	158
1	6	155	157	156	156	156	157	156	158	158	157	158
2	2	187	188	188	187	187	186	187	188	187	186	185
3	2	211	211	212	212	211	210	211	210	210	211	209
4	13	164	167	170	172	176	179	180	184	185	186	188
...	...	...	...	...	...	...	...	...	...	...	...	...
27450	13	189	189	190	190	192	193	193	193	193	194	193
27451	23	151	154	157	158	160	161	163	164	166	167	169
27452	18	174	174	174	174	174	175	175	174	173	173	173
27453	17	177	181	184	185	187	189	190	191	191	190	191
27454	23	179	180	180	180	182	181	182	183	182	182	182

27455 rows × 785 columns

In [ ]:
```
# Create training and testing arrays
train_set = np.array(train_df, dtype = 'float32')
test_set = np.array(test_df, dtype='float32')
```

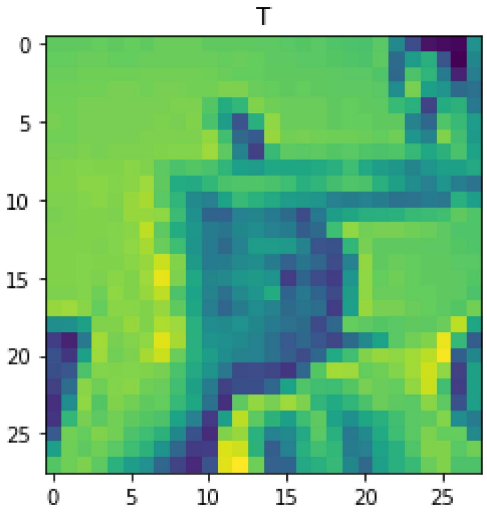
Reason:

- The training and testing input arrays are converted to continuous float values since it allows our model for a more precise learning as compared to discrete values

In [ ]:
```
#Specifying class labels
class_names = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M'
, 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y' ]
```

In [ ]:
```
#See a random image for class label verification
import random
i = random.randint(1,27455)
plt.imshow(train_set[i,1:].reshape((28,28)))
label_index = train_df["label"][i]
plt.title(f"{class_names[label_index]}")
```

Out[ ]:
```
Text(0.5, 1.0, 'T')
```



Inference:

- We also need to reshape the array to (28x28) since the initial shape is just a row array.

Subplots :

- Define the dimensions of the plot grid
- flaten the 15 x 15 matrix into 225 array
- get the length of the train dataset
- Select a random number from 0 to n_train
- create evenly spaces variables
- Select a random number
- read and display an image with the selected index

In [ ]:
```
#
W_grid = 5
L_grid = 5
fig, axes = plt.subplots(L_grid, W_grid, figsize = (10,10))
axes = axes.ravel()
n_train = len(train_set)
for i in np.arange(0, W_grid * L_grid):
    index = np.random.randint(0, n_train)
    axes[i].imshow( train_set[index,1:].reshape((28,28)) )
    label_index = int(train_set[index,0])
    axes[i].set_title(class_names[label_index], fontsize = 8)
    axes[i].axis('off')
plt.subplots_adjust(hspace=0.4)
```

In [ ]:
```
train_x = train_df[train_df.columns[1::]].to_numpy()
train_y = train_df[train_df.columns[0]].to_numpy()

test_x = test_df[test_df.columns[1::]].to_numpy()
test_y = test_df[test_df.columns[0]].to_numpy()

print("SUMMARY OF DATA:")

print("train_x shape: " + str(train_x.shape))
print("train_y shape: " + str(train_y.shape))
print("test_x shape: " + str(test_x.shape))
print("test_y shape: " + str(test_y.shape))
```

```
SUMMARY OF DATA:
train_x shape: (27455, 784)
train_y shape: (27455,)
test_x shape: (7172, 784)
test_y shape: (7172,)
```

- we know that the pixel values lies between 0-255 but it is observed that models performs exceptionally well if we scale pixel values between 0-1

Visualization

In [ ]:
```
#Visualize train images
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_x[i].reshape((28,28)), cmap=plt.cm.binary)
    label_index = int(train_y[i])
    plt.title(class_names[label_index])
plt.show()
```

In [ ]:
```
#Get our training labels:
labels = train_df['label'].values
```

In [ ]:
```
label_frq=pd.value_counts(train_df['label'],ascending=True).reset_index(level=
0)
pd.DataFrame(label_frq)
```

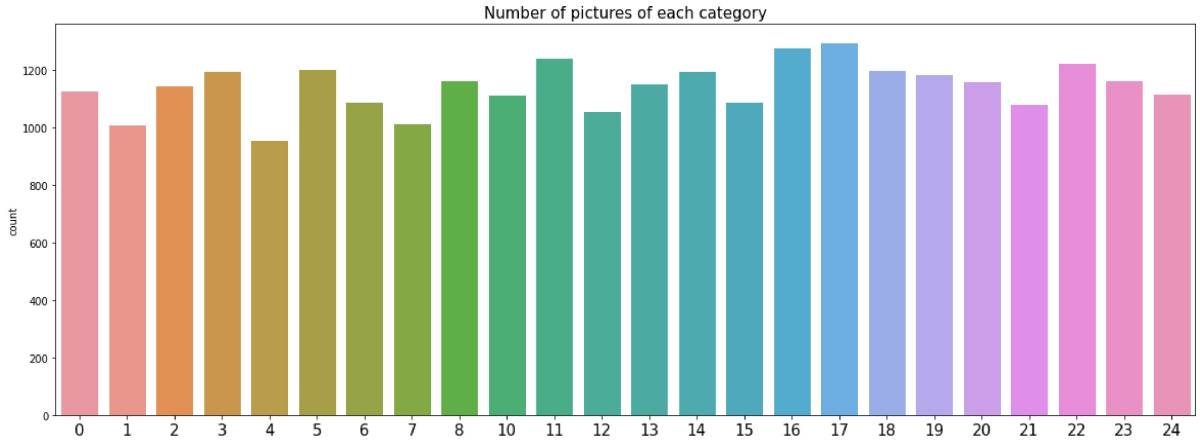
Out[ ]:

	index	label
0	4	957
1	1	1010
2	7	1013
3	12	1055
4	21	1082
5	15	1088
6	6	1090
7	10	1114
8	24	1118
9	0	1126
10	2	1144
11	13	1151
12	20	1161
13	8	1162
14	23	1164
15	19	1186
16	3	1196
17	14	1196
18	18	1199
19	5	1204
20	22	1225
21	11	1241
22	16	1279
23	17	1294

In [ ]:
```
unique_labels = train_df['label'].unique()
unique_labels = np.sort(unique_labels)
unique_labels
```

Out[ ]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24])
```

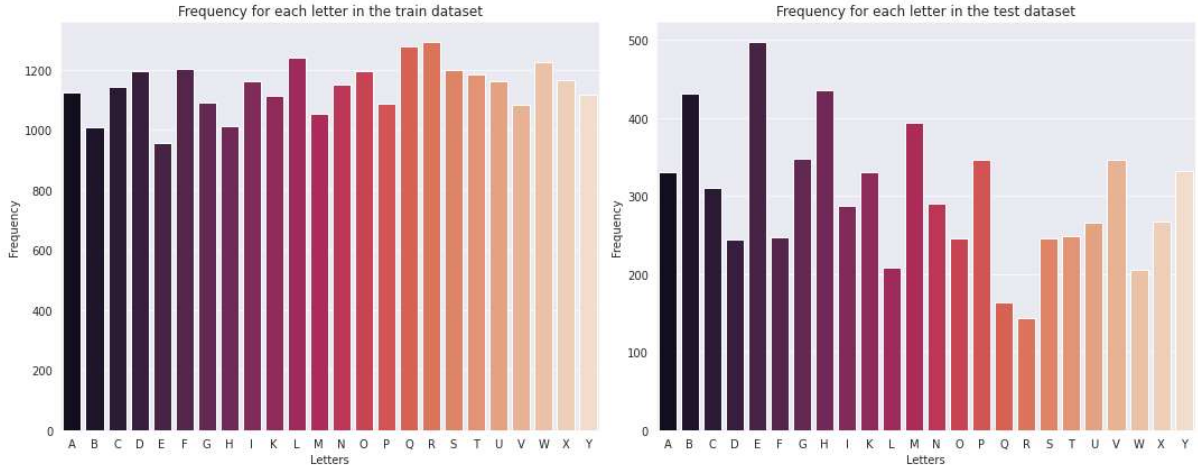
In [ ]:
```
# visulaizing the data

plt.figure(figsize=(20,7))
plt.title("Number of pictures of each category", fontsize = 15)
plt.xticks(fontsize = 15)
sns.set_style("darkgrid");
sns.countplot(train_y);
```



Observation:

- We can see the visual tells us that there are nearly 1000 examples for each class output
- So we can consider this dataset as a balanecd dataset because there's no class suffering from very less or too much examples

In [ ]:
```
list_data = [train_df, test_df]
fig,axes = plt.subplots(nrows=1,ncols=2,figsize=(15,6))

for data, ax, names in zip(list_data, axes.ravel(), ['train', 'test']):
    sns.countplot(data['label'], palette='rocket', ax=ax)
    ax.set_title("Frequency for each letter in the {} dataset".format(names))
    ax.set_xlabel('Letters')
    ax.set_ylabel('Frequency')
    ax.set_xticklabels(['A','B','C','D','E','F','G','H','I','K','L','M','N',
'O','P','Q','R','S',
                        'T','U','V','W','X','Y'])

plt.tight_layout()
```



In [ ]:
```
#Encoding:

from sklearn.preprocessing import LabelBinarizer
labels = train_df['label'].values
label_binarizer = LabelBinarizer()
labels = label_binarizer.fit_transform(labels)
labels
```

Out[ ]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 1, 0]])
```

Observation:

- So basically LabelBinarizer performed `One-Hot Encoding` on our training data.

In [ ]:
```
labels[100]
```

Out[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0])
```

In [ ]:
```
import cv2
df_htrain = pd.read_csv("/content/drive/MyDrive/Sign Language/Hog_Features.csv")
df_htest = pd.read_csv("/content/drive/MyDrive/Sign Language/Hog_Features_test.csv")
htrain = df_htrain[df_htrain.columns[1::]].to_numpy()
htest = df_htest[df_htest.columns[1::]].to_numpy()
print("Training Data size : ",htrain.shape)
print("Test Data size : ",htest.shape)
```

```
Training Data size :  (27455, 1764)
Test Data size :  (7172, 1764)
```

Review-2(Final Review)

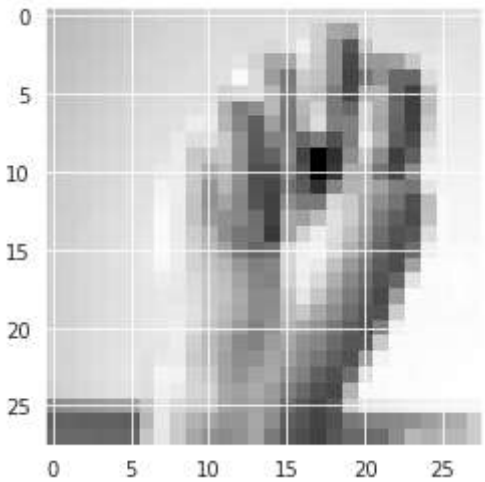
Feature Engineering

In [ ]:
```
train_df_original = train_df.copy()
```

In [ ]:
```
train_x = train_x/255
test_x = test_x/255
```

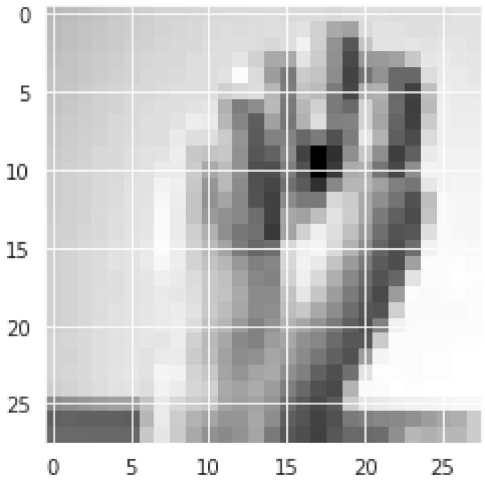
In [ ]:
```
# Split into training, test and validation sets
val_index = int(train_df.shape[0]*0.1)

train_df = train_df_original.iloc[val_index:]
val_df = train_df_original.iloc[:val_index]
```

In [ ]:
```
index = 4
plt.imshow(train_x[index].reshape(28, 28),cmap='gray')
plt.show()
```



In [ ]:
```
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage import data,exposure
import cv2
from sklearn import svm
%matplotlib inline
np.random.seed(1)
```

In [ ]:
```
image = plt.imshow(train_x[index].reshape(28, 28),cmap='gray')
```



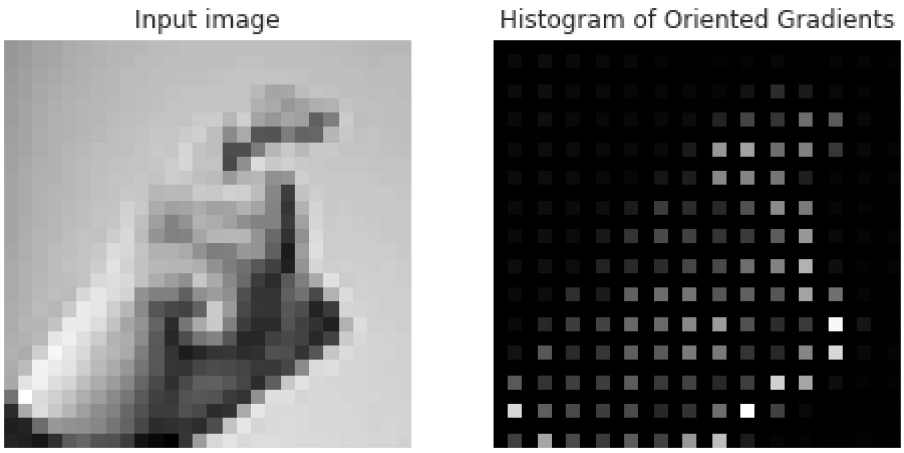
End of the Review☺

In this work, we propose a method to detect American Sign Language by analyzing sign language MNIST image dataset. We will use the pixel level features in each image as well as HOG Features, to be followed with feature selection to improve the performance. Finally we will print different evaluation metrics for the classifier model such as Accuracy, F1 Score, Precision, Recall and plot ROC curves for all the classes.

Feature Extraction Using HOG Features

In [ ]:
```
from skimage.feature import hog
from skimage import data,exposure
def get_hog(image):
    fd,hog_image=hog(image.reshape(28,28),pixels_per_cell=(2,2),
                     cells_per_block=(1, 1),visualize=True,feature_vector=F
alse)
    return(fd,hog_image)

def show_hog(image,hog_image):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 4), sharex=True, sharey=T
rue)
    ax1.axis('off')
    ax1.imshow(image.reshape(28, 28), cmap=plt.cm.gray)
    ax1.set_title('Input image')
    hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10
))

    ax2.axis('off')
    ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)
    ax2.set_title('Histogram of Oriented Gradients')
    plt.show()
```

In [ ]:
```
a,b=get_hog(train_x[180])
show_hog(train_x[180],b)
a=a.reshape(-1)
a=a.reshape(1,len(a))
print(a.shape)
```




```
(1, 1764)
```

Feature Selection using PCA

In [ ]:
```
model = []
# Model names
accuracy = []
# Accuracy of the respective model
```

In [ ]:
```
from sklearn.decomposition import PCA
import sklearn.metrics as metrics
from sklearn.metrics import classification_report
pca = PCA(n_components=30)
htrain_pca = pca.fit_transform(htrain)

htest_pca=pca.transform(htest)

clf = svm.SVC(kernel='rbf') # rbf Kernel
#Train the model using the training sets
clf.fit(htrain_pca,train_y)
y_pred = clf.predict(htest_pca)
#Predict the response for test dataset
x = metrics.accuracy_score(test_y, y_pred)
model.append('SVM')
accuracy.append(x*100)
print(classification_report(test_y, y_pred))
print("SVM's Accuracy is: ", x*100)
```

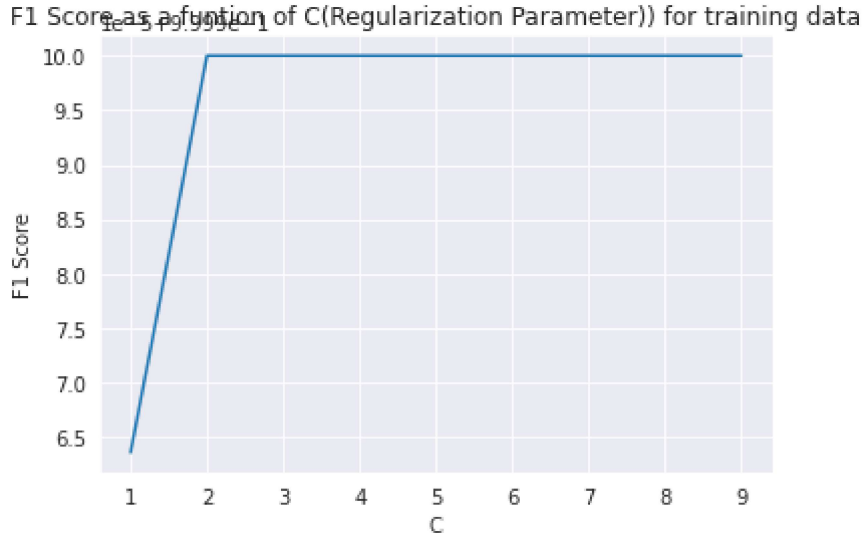
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       331
           1       1.00      0.95      0.97       432
           2       1.00      0.94      0.97       310
           3       0.98      0.93      0.95       245
           4       0.99      1.00      0.99       498
           5       0.96      1.00      0.98       247
           6       0.98      0.92      0.95       348
           7       0.98      0.98      0.98       436
           8       0.89      0.99      0.94       288
          10       0.99      0.87      0.93       331
          11       1.00      1.00      1.00       209
          12       0.89      0.84      0.87       394
          13       0.79      0.88      0.84       291
          14       0.92      0.91      0.92       246
          15       1.00      1.00      1.00       347
          16       0.89      1.00      0.94       164
          17       0.55      0.99      0.70       144
          18       0.93      0.88      0.91       246
          19       0.91      0.91      0.91       248
          20       0.98      0.72      0.83       266
          21       0.92      0.81      0.86       346
          22       0.69      0.94      0.79       206
          23       0.88      0.91      0.89       267
          24       1.00      0.88      0.94       332

    accuracy                           0.93      7172
   macro avg       0.92      0.93      0.92      7172
weighted avg       0.94      0.93      0.93      7172

SVM's Accuracy is:  92.63803680981594
```

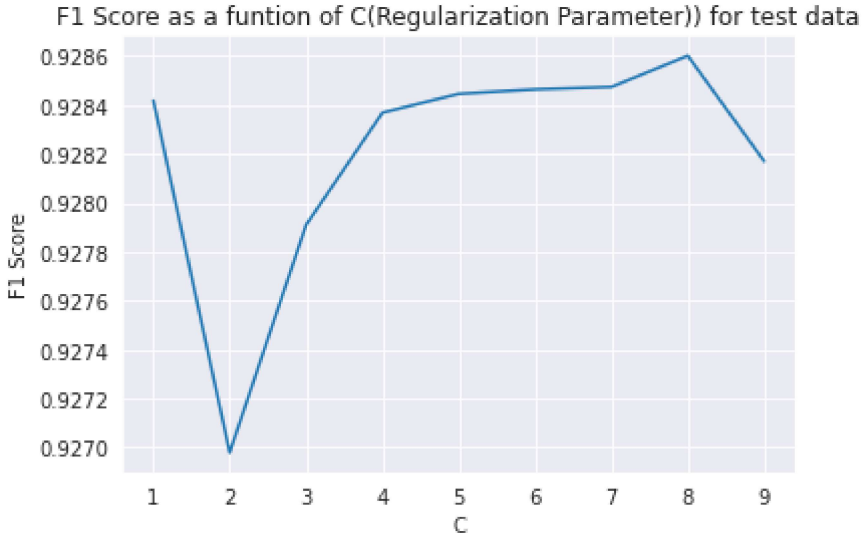
In [ ]:
```
y_pred = clf.predict(htrain_pca)
print("Training Set Metrics : ")
print("Accuracy:",metrics.accuracy_score(train_y, y_pred))
print("F1 Score:",metrics.f1_score(train_y, y_pred, average='weighted'))
print("Precision: ",metrics.precision_score(train_y,y_pred, average='weighted'
))
print("Recall: ",metrics.recall_score(train_y,y_pred,average = 'weighted'))
```

```
Training Set Metrics :
Accuracy: 0.9999635767619741
F1 Score: 0.9999635761187055
Precision:  0.9999636101471603
Recall:  0.9999635767619741
```

In [ ]:
```
x=[]
y=[]
y2=[]
for i in range(1,10):
    clf = svm.SVC(kernel='rbf',C=i)
    clf.fit(htrain_pca,train_y)
    y_pred = clf.predict(htest_pca)
    x.append(i)
    y.append(metrics.f1_score(test_y, y_pred, average='weighted'))
    y2_pred = clf.predict(htrain_pca)
    y2.append(metrics.f1_score(train_y, y2_pred, average='weighted'))
```

In [ ]:
```
plt.plot(x,y2)
plt.title("F1 Score as a funtion of C(Regularization Parameter)) for training
 data")
plt.xlabel("C")
plt.ylabel("F1 Score")
plt.show()

plt.plot(x,y)
plt.title("F1 Score as a funtion of C(Regularization Parameter)) for test dat
a")
plt.xlabel("C")
plt.ylabel("F1 Score")
plt.show()
```





Testing The Model

- For testing we are going to use the test data only

In [ ]:
```
print('True:', test_y[25])
print('Pred:', y_pred[25])
```

```
True: 15
Pred: 15
```

Perceptron:

In [ ]:
```
tran_y=pd.DataFrame(train_y)
```

In [ ]:
```
tran_y.tail()
```

Out[ ]:

	0
27450	13
27451	23
27452	18
27453	17
27454	23

In [ ]:
```
tes_y=pd.DataFrame(test_y)
```

In [ ]:
```
tes_x=pd.DataFrame(test_x)
```

In [ ]:
```
tran_x=pd.DataFrame(train_x)
```

In [ ]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
clf = MLPClassifier()#solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2), r
andom_state=1
clf.fit(tran_x, tran_y)
y_pred=clf.predict(tes_x)
x = metrics.accuracy_score(tes_y, y_pred)
accuracy.append(x*100)
model.append('Perceptron')
print(classification_report(tes_y, y_pred))
print("Perceptron's Accuracy is: ", x*100)
```

```
              precision    recall  f1-score   support

           0       0.84      1.00      0.92       331
           1       0.99      0.91      0.94       432
           2       0.90      0.91      0.91       310
           3       0.84      0.82      0.83       245
           4       0.94      0.91      0.93       498
           5       0.64      0.91      0.75       247
           6       0.85      0.81      0.83       348
           7       0.87      0.72      0.79       436
           8       0.78      0.75      0.76       288
          10       0.70      0.49      0.58       331
          11       0.73      0.77      0.75       209
          12       0.76      0.79      0.78       394
          13       0.84      0.57      0.68       291
          14       0.95      0.63      0.76       246
          15       1.00      0.97      0.98       347
          16       0.58      0.78      0.67       164
          17       0.24      0.43      0.31       144
          18       0.43      0.48      0.45       246
          19       0.47      0.53      0.50       248
          20       0.38      0.49      0.43       266
          21       0.75      0.60      0.67       346
          22       0.64      0.70      0.67       206
          23       0.58      0.68      0.62       267
          24       0.77      0.64      0.70       332

    accuracy                           0.74      7172
   macro avg       0.73      0.72      0.72      7172
weighted avg       0.76      0.74      0.74      7172

Perceptron's Accuracy is:  73.96820970440602
```

Logistic Regression:

Multi class Logistic Regression Using OVR

- Since we are going to use One Vs Rest algorithm, set > multi_class='ovr'
- Note: since we are using One Vs Rest algorithm we must use 'liblinear' solver with it.

In [ ]:
```
from sklearn.linear_model import LogisticRegression
# train a logistic regression model on the training set
from sklearn import metrics
from sklearn import svm
from sklearn.metrics import classification_report

LogReg = LogisticRegression(multi_class='ovr',solver='liblinear')
# instantiate model
LogReg.fit(htrain,train_y)
# fit model
predicted_values = LogReg.predict(htest)
# make class predictions for the testing set
x = metrics.accuracy_score(test_y, predicted_values)
model.append('Logistic Regression')
accuracy.append(x*100)
print(classification_report(test_y, predicted_values))
print("Logistic Regression's Accuracy is: ", x*100)
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       331
           1       0.97      0.90      0.93       432
           2       0.95      0.93      0.94       310
           3       0.81      0.99      0.89       245
           4       0.95      1.00      0.97       498
           5       0.95      0.97      0.96       247
           6       0.82      0.85      0.83       348
           7       0.96      0.94      0.95       436
           8       0.90      0.87      0.88       288
          10       0.91      0.74      0.81       331
          11       0.92      1.00      0.96       209
          12       0.79      0.62      0.70       394
          13       0.75      0.65      0.70       291
          14       0.78      0.79      0.78       246
          15       0.97      1.00      0.99       347
          16       0.76      0.87      0.81       164
          17       0.37      0.64      0.47       144
          18       0.70      0.80      0.74       246
          19       0.71      0.85      0.77       248
          20       0.81      0.61      0.70       266
          21       0.78      0.56      0.65       346
          22       0.58      0.91      0.71       206
          23       0.71      0.72      0.71       267
          24       0.94      0.75      0.83       332

    accuracy                           0.84      7172
   macro avg       0.82      0.83      0.82      7172
weighted avg       0.85      0.84      0.84      7172

Logistic Regression's Accuracy is:  83.63078639152258
```

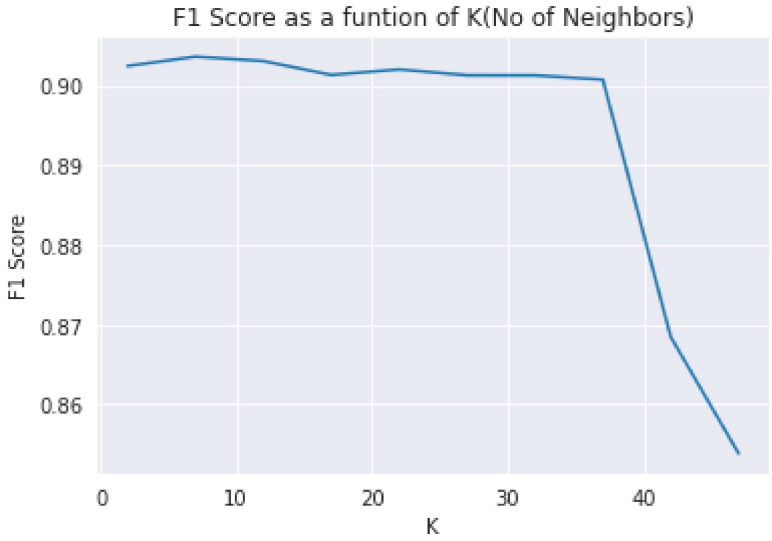
Random Forest:

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier()
RF.fit(train_x,train_y)
predicted_values = RF.predict(test_x)
x = metrics.accuracy_score(test_y, predicted_values)
accuracy.append(x*100)
model.append('Random Forest')
print(classification_report(test_y,predicted_values))
print("RF's Accuracy is: ", x*100)
```

```
              precision    recall  f1-score   support

           0       0.91      1.00      0.96       331
           1       0.96      0.94      0.95       432
           2       0.94      0.98      0.96       310
           3       0.81      0.97      0.88       245
           4       0.89      0.97      0.93       498
           5       0.91      0.93      0.92       247
           6       0.92      0.86      0.89       348
           7       0.99      0.92      0.95       436
           8       0.81      0.70      0.75       288
          10       0.72      0.62      0.67       331
          11       0.81      1.00      0.90       209
          12       0.85      0.67      0.75       394
          13       0.73      0.58      0.65       291
          14       0.97      0.87      0.92       246
          15       0.95      1.00      0.97       347
          16       0.91      1.00      0.95       164
          17       0.29      0.56      0.38       144
          18       0.60      0.81      0.69       246
          19       0.61      0.82      0.70       248
          20       0.65      0.63      0.64       266
          21       0.82      0.58      0.68       346
          22       0.55      0.67      0.60       206
          23       0.86      0.69      0.77       267
          24       0.86      0.61      0.71       332

    accuracy                           0.81      7172
   macro avg       0.80      0.81      0.80      7172
weighted avg       0.83      0.81      0.82      7172

RF's Accuracy is:  81.39988845510318
```

KNN

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
x=[]
y=[]
for i in range(2,50,5):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(htrain_pca,train_y)
    y_pred = neigh.predict(htest_pca)
    x.append(i)
    y.append(metrics.f1_score(test_y, y_pred, average='weighted'))
plt.plot(x,y)
plt.title("F1 Score as a funtion of K(No of Neighbors)")
plt.xlabel("K")
plt.ylabel("F1 Score")
plt.show()
```



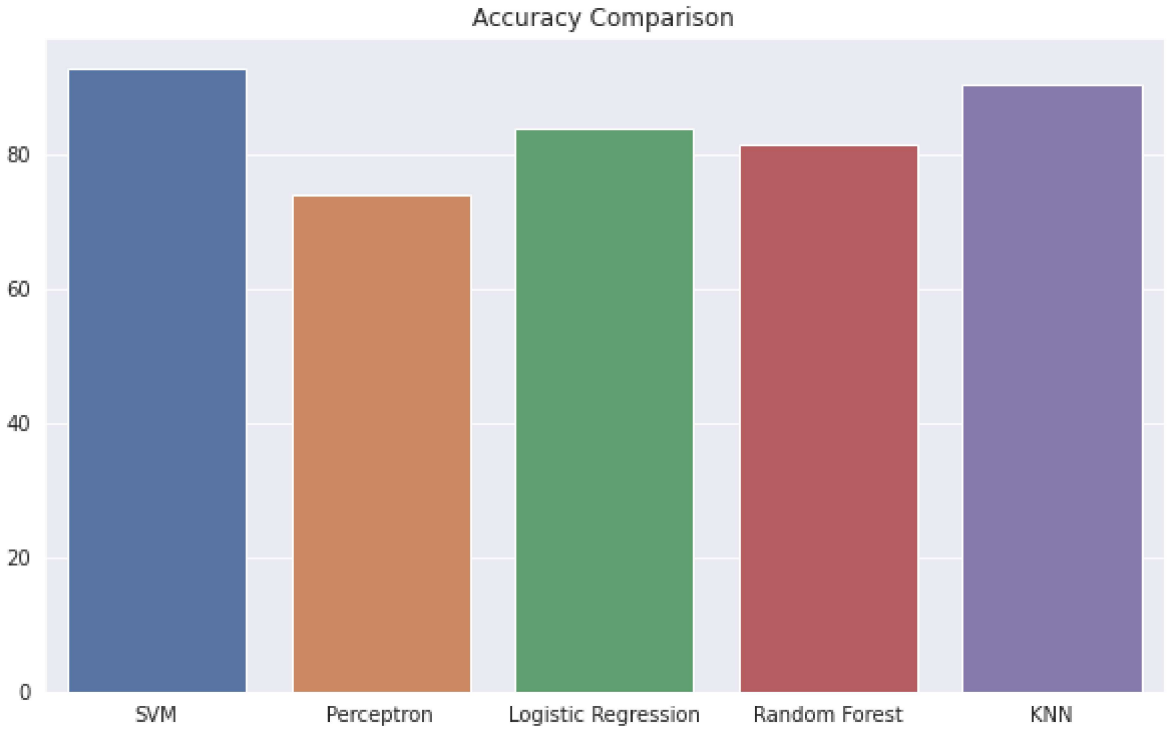
In [ ]:
```
m=max(y)
index = y.index(m)
print("Max value of F1 Score occurs at K = ",x[index])
print("The Evaluation metrics are : ")
neigh = KNeighborsClassifier(n_neighbors=x[index])
neigh.fit(htrain_pca,train_y)
y_pred = neigh.predict(htest_pca)
x = metrics.accuracy_score(test_y, y_pred)
accuracy.append(x*100)
model.append('KNN')
print("Accuracy:",metrics.accuracy_score(test_y, y_pred))
print("F1 Score:",metrics.f1_score(test_y, y_pred, average='weighted'))
print("Precision: ",metrics.precision_score(test_y,y_pred, average='weighted'
))
print("Recall: ",metrics.recall_score(test_y,y_pred,average = 'weighted'))
```

```
Max value of F1 Score occurs at K =  7
The Evaluation metrics are :
Accuracy: 0.9033742331288344
F1 Score: 0.9036218462600369
Precision:  0.9104427818377326
Recall:  0.9033742331288344
```

Model Comparison

In [ ]:
```
plt.figure(figsize=(10,6))
plt.title('Accuracy Comparison')
sns.barplot(x=model, y=accuracy, palette='deep')
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbb03a11090>



In [ ]:
```
for x in range(len(accuracy)):
    print(accuracy[x])
```

```
92.63803680981594
73.96820970440602
83.63078639152258
81.39988845510318
90.33742331288343
```

In [ ]:
```
def Average(accuracy):
    return sum(accuracy) / len(accuracy)
average = Average(accuracy)
print("Average accuracy of classifier models is =", round(average, 2))
```

```
Average accuracy of classifier models is = 84.39
```

To visualize the inner working of the model we use Confusion Matrix

In [ ]:
```
y_predicted = neigh.predict(htest_pca)
```

In [ ]:
```
print('True:', test_y[0:25])
# print the first 25 true and predicted responses
print('Pred:', y_predicted[0:25])
```

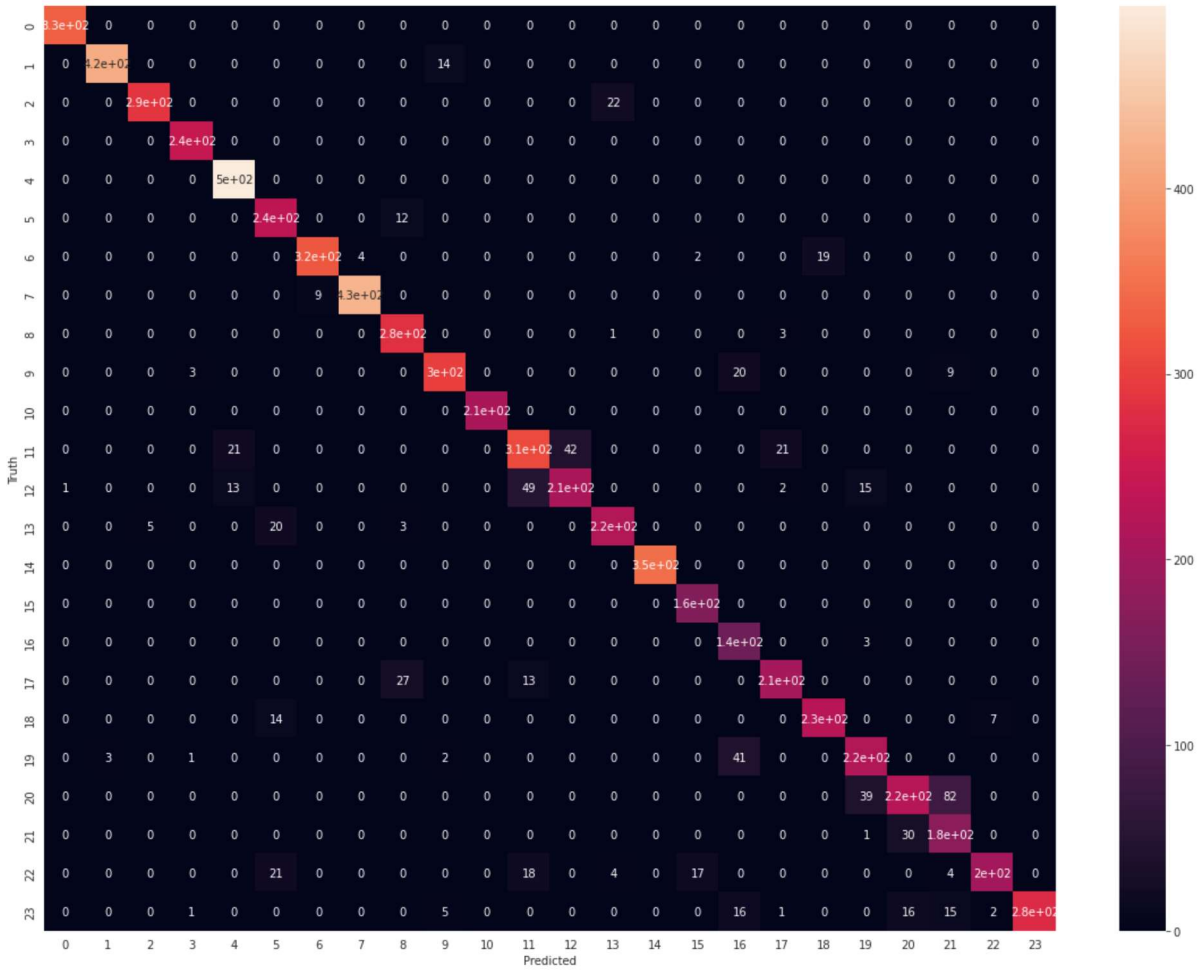
```
True: [ 6  5 10  0  3 21 10 14  3  7  8  8 21 12  7  4 22  0  7  7  2  0 21
4
 10]
Pred: [ 6  5 10  0  3 20 10 14  3  7  8  8 22 12  7  4 22  0  7  7  2  0 21
4
 10]
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
# import confusion matrix class
cm = confusion_matrix(test_y,y_predicted)
# create classifier of confussion_matrix
cm
# it is two dimentional arraym
```

Out[ ]:
```
array([[331,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0, 418,   0,   0,   0,   0,   0,   0,   0,  14,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0, 288,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         22,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0, 245,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0, 498,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0, 235,   0,   0,  12,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0, 323,   4,   0,   0,   0,   0,   0,
          0,   0,   2,   0,   0,  19,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   9, 427,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0, 284,   0,   0,   0,   0,
          1,   0,   0,   0,   3,   0,   0,   0,   0,   0],
       [  0,   0,   0,   3,   0,   0,   0,   0,   0, 299,   0,   0,   0,
          0,   0,   0,  20,   0,   0,   0,   0,   9,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 209,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,  21,   0,   0,   0,   0,   0,   0, 310,  42,
          0,   0,   0,   0,  21,   0,   0,   0,   0,   0],
       [  1,   0,   0,   0,  13,   0,   0,   0,   0,   0,   0,  49, 211,
          0,   0,   0,   0,   2,   0,  15,   0,   0,   0],
       [  0,   0,   5,   0,   0,  20,   0,   0,   3,   0,   0,   0,   0,
        218,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0, 347,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0, 164,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0, 141,   0,   0,   3,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  27,   0,   0,  13,   0,
          0,   0,   0,   0, 206,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,  14,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0, 227,   0,   0,   7,   0],
       [  0,   3,   0,   1,   0,   0,   0,   0,   0,   2,   0,   0,   0,
          0,   0,   0,  41,   0,   0, 219,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,  39, 225,  82,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   1,  30, 175,   0,   0],
       [  0,   0,   0,   0,   0,  21,   0,   0,   0,   0,   0,  18,   0,
          4,   0,  17,   0,   0,   0,   0,   0,   4, 203,   0],
       [  0,   0,   0,   1,   0,   0,   0,   0,   0,   5,   0,   0,   0,
          0,   0,   0,  16,   1,   0,   0,  16,  15,   2, 276]])
```

In [ ]:
```
plt.figure(figsize = (20,15))
sns.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[ ]: Text(159.0, 0.5, 'Truth')



Observation:

- In above matrix, X axis contains `predicted values` , and Y axis contains `actual values` .
- Each value of matrix show the number of times predicted value matched with actual value.

Cross Validation:

In [ ]:
```
X_train = train_df[train_df.columns[1::]]
y_train = train_df[train_df.columns[0]]

X_test = test_df[test_df.columns[1::]]
y_test = test_df[test_df.columns[0]]
```

In [ ]:
```
X_train=X_train/255
X_test=X_test/255
```

In [ ]:
```
from sklearn import svm
clf = svm.SVC(kernel='rbf', C=2).fit(X_train, y_train)
clf.score(X_test, y_test)
```

In [ ]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, X_train, y_train, cv=5)
scores
```

In [ ]:
```
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), sc
ores.std()))
```

Got a mean of 0.98 with std=0.2

.

CONCLUSIONS AND FUTURE WORK

Hand gesture recognition is a difficult problem and the current work is only half way towards achieving the results needed in the field.

- This notebook presented a comparative study of five algorithms applied to two different datasets for static gesture recognition and classification, for human computer interaction.
- Future work will be concerned with the study of different hand feature selection applied to hand gesture recognition, noise reduction in the depth images acquired.

References:

1. https://en.wikipedia.org/wiki/Support-vector_machine (https://en.wikipedia.org/wiki/Support-vector_machine)
2. https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python (https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python)
3. https://link.springer.com/article/10.1007/s10586-021-03282-8 (https://link.springer.com/article/10.1007/s10586-021-03282-8)