**CSEE5590 - Python and Deep Learning for Engineering & Science**

**Project Increment –** 4(Final)

**Team number**: 5

**Project Title**

Sign Language for Numbers

**Team Members**

Vasim Shaikh

Shireesha Maddi

Harshita Patil

Prabhanjan Trivedi

**University of Missouri-Kansas City**

MS – Computer Science, 2020-21

5000 Holmes St,

Kansas City,

MO 64110

## The Story and its details:

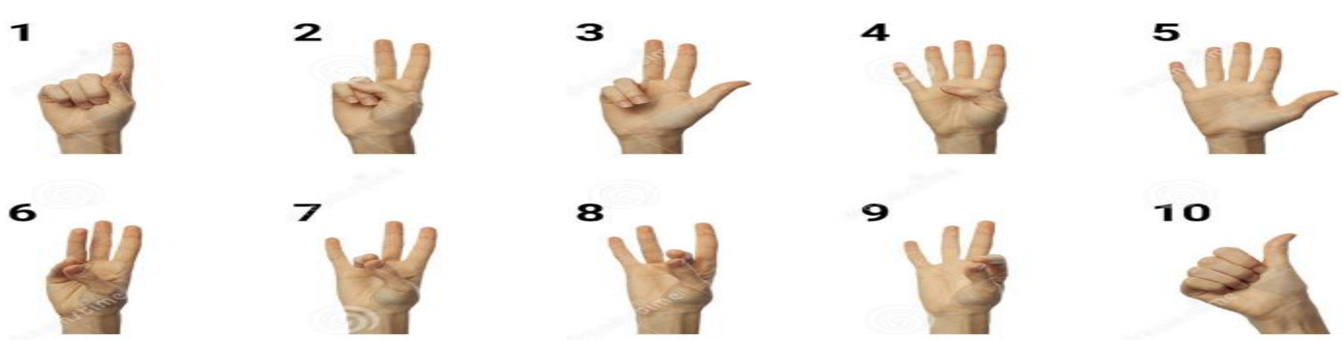
There have been several milestone reached in innovation and a great deal of technology advancement and research has been done to help the individuals who are physically challenged. Supporting the reason, Deep learning and computer vision can be utilized in an extensive way for community of people who can't speak or listen.

The Sign language is very important for people who have hearing and speaking deficiency generally called Deaf and Mute. It is the only mode of communication for such people to convey their messages and it becomes very important for people to understand their language.

This project proposes the method or algorithm for an application which would help to recognize the different signs using palm/finger gestures. This will ease and make a better communication channel to rest of the world.

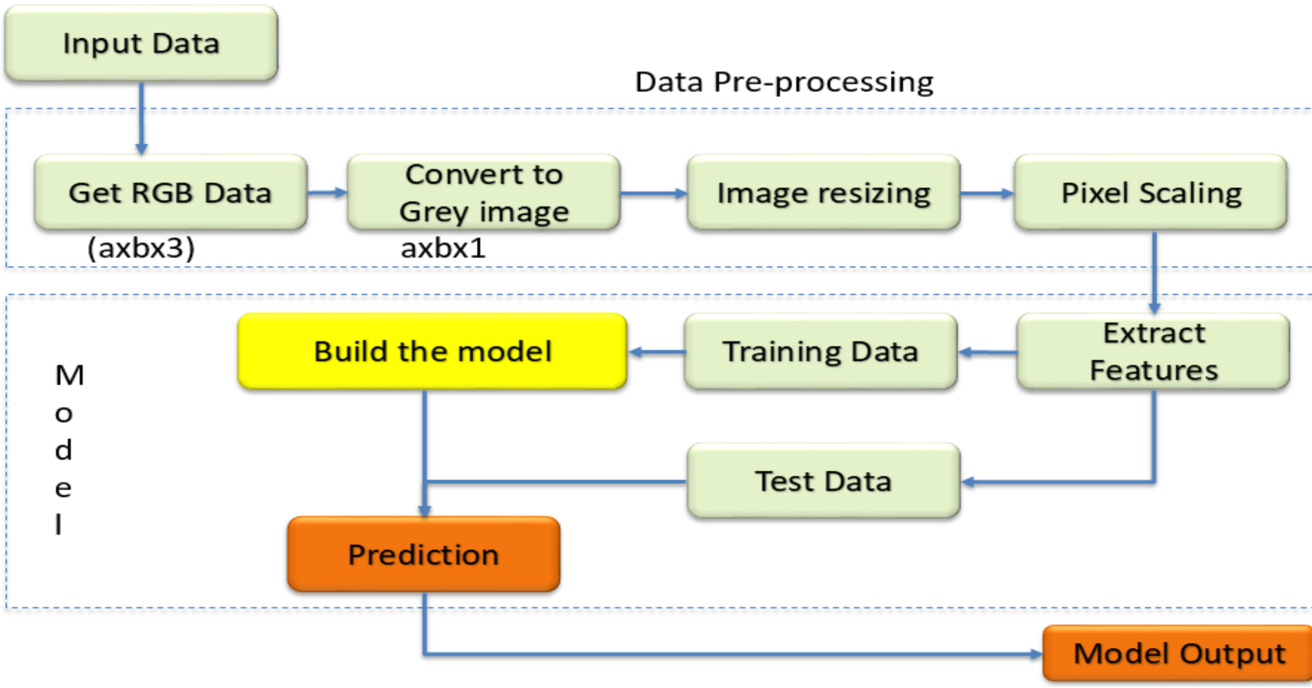
## The Data and its details:

- Sign language datasets for hand gesture recognition for numbers
- Source of data – Kaggle – Link in reference section
- Sample hand signs are in below picture
- Total number of training images – 1500 images for each sign = 16,500
- Covering hand images 0 – 9 and unknown (Total classes – 11)
- Input data type – RGB Images(3 channels)
- Assumption – All input images are front facing



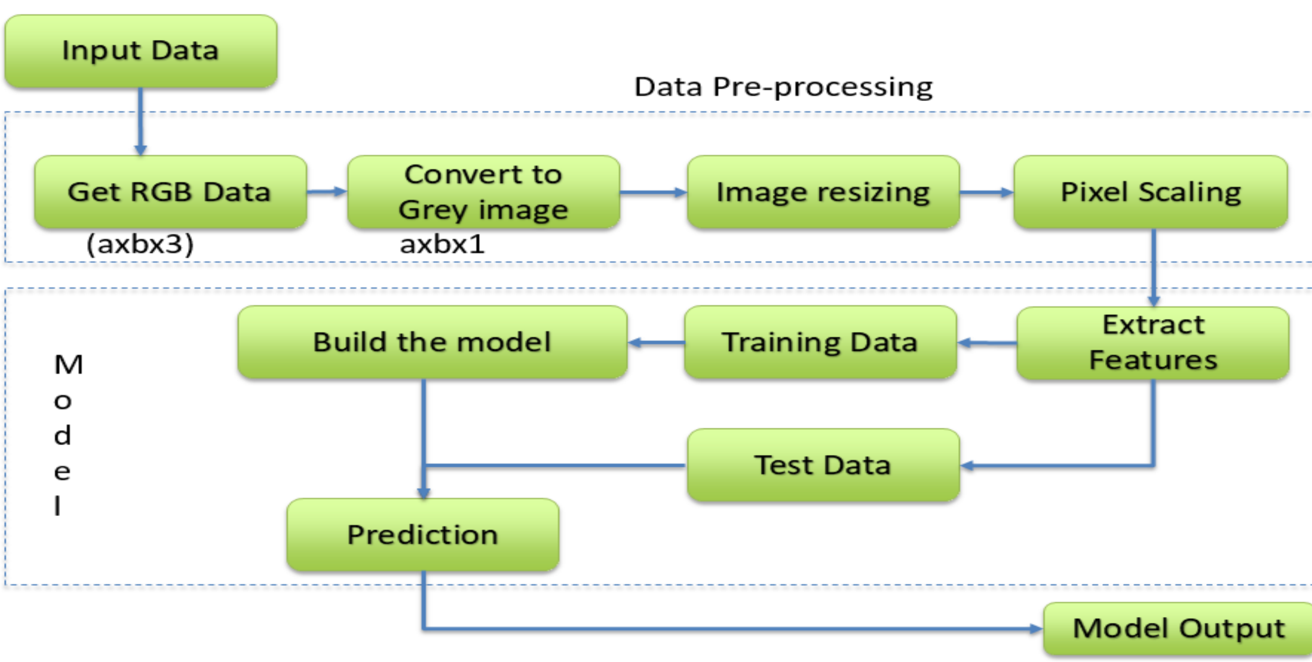
**The Model Building Blocks:**

**Increment 3 state -**




1. Blocks highlighted in green are completed
2. Build the model - Model 1 (CNN)implementation is complete and evaluation is in progress
3. Prediction and ensemble modelling – Next steps

**Increment 4 – Final state –**



1. All building blocks are implemented successfully
2. SVM , KNN and CNN models built and executed successfully
3. Above model results are compared

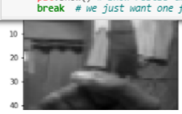
## Working screens from project:

### Reading Input Data and Preprocessing

```
In [1]: # importing required libraries
import os
import cv2
from tqdm import tqdm
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
In [2]: #Storing path of the file to a variable filepath.
dir = '/Users/vshaikh/OneDrive - Capgemini/Desktop/Py/Project/Sign Language for Numbers'
#Setting the image size
IMG_SIZE = 64
CATEGORIES = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "unknown"]
#Use one folder for testing
#CATEGORIES = ["0"]
```

```
In [3]: #Only for Visualizing the first image of each category
for category in CATEGORIES:
    path = os.path.join(dir,category)  # create path to categories
    for img in os.listdir(path):
      # iterate over each image
      # convert to array
      print('Label of below image is {}'.format(category))
      img_array = cv2.imread(os.path.join(path,img))
      print('Image size of {} is {}'.format(img, img_array.shape))
      plt.imshow(img_array, cmap = None)  # graph it
      plt.show()  # display!

      print('Printing the graysacle image')
      img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)
      plt.imshow(img_array, cmap='gray')  # graph it
      plt.show()  # display!
      print('Resized image')
      img_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
      plt.imshow(img_array, cmap='gray')
      plt.show() # Show resize image
      break  # we just want one for now so break
```



```
        class_num = CATEGORIES.index(category)  # get the classification
        print('Processing folder {}'.format(category))
        # Iterate through each image in folder
        for file in tqdm(os.listdir(path)):
          # Get the path name of the image
          img_path = os.path.join(path, file)
          # Open and read the image in grayscale
          image_array = cv2.imread(img_path,cv2.IMREAD_GRAYSCALE)
          # Resize the image
          image_resize = cv2.resize(image_array, (IMG_SIZE, IMG_SIZE))
          #print('type of Image', type(Images))
          #print('type of image', type(image))
          # Append the image and its corresponding label to the output
          output.append([image_resize, class_num])
          #writing data to csv file
          flat_pixel = (image_resize.flatten())
          flat_pixel = " ".join(map(str, flat_pixel))
          with open('output_file_with_pixels.csv', 'a') as f:
              f.write(file+',')
              f.write(flat_pixel)
              f.write(','+str(class_num))
              f.write('\n')
        #Images = np.array(Images, dtype = 'float32')
        #Labels = np.array(Labels, dtype = 'int32')
    return output
```

```
In [5]: image_with_label = get_images(dir)

  0%|                                                  | 0/1500 [00:00<?, ?it/s]
Loading /Users/vshaikh/OneDrive - Capgemini/Desktop/Py/Project/Sign Language for Numbers
Processing folder 0
100%|██████████████████████████████████████████| 1500/1500 [00:26<00:00, 56.18it/s]
  1%|▌                                          | 12/1500 [00:00<00:14, 101.97it/s]
Processing folder 1
100%|██████████████████████████████████████████| 1500/1500 [00:12<00:00, 116.44it/s]
  1%|▌                                          | 12/1500 [00:00<00:12, 119.13it/s]
Processing folder 2
100%|██████████████████████████████████████████| 1500/1500 [00:15<00:00, 95.95it/s]
  1%|▌                                          | 9/1500 [00:00<00:17, 83.93it/s]
Processing folder 3
100%|██████████████████████████████████████████| 1500/1500 [00:16<00:00, 90.25it/s]
  1%|▌                                          | 10/1500 [00:00<00:16, 87.96it/s]
Processing folder 4
```

### SVM - Machine Learning Model - 1

```
In [13]: # imported the necessary libraries.
# Pandas is a library use for data manipulation and analysis
import numpy as np
from sklearn.metrics import classification_report
import pandas as pd
from sklearn.model_selection import train_test_split
# To implement the Support Vector Machines we will use Scikit-Learn and will import our SVM
from sklearn import svm
from sklearn.metrics import accuracy_score
```

```
In [14]: # Separate features and target columns from input dataset
# We have only one feature i.s Pixel column
# Format Pixel column as list of data items
X = np.array(data['Pixel'].tolist())

# For SVM model it is must that we should have X and y i.e features and target less than or equal to 2 dimension.
X.shape

Out[14]: (16500, 4096)
```

```
In [15]: # Our target columns is Label. We have total 11 categories.
y = np.array(data['Label'], dtype=int)
```

```
In [16]: # Split X and y into Train and Test datasets
X_train_ML, X_test_ML, Y_train_ML, Y_test_ML = train_test_split( X, y, test_size = 0.4, random_state=40)
```

```
In [17]: # Create SVM model. We are using Gaussian's rbf kernel.
ModelSVM = svm.SVC(kernel="rbf")
```

```
In [18]: # Fitting the Model
ModelSVM.fit(X_train_ML, Y_train_ML)

Out[18]: SVC()
```

```
In [*]: # Predict the target using test dataset
predSVM = ModelSVM.predict(X_test_ML)
```

```
In [*]: # Prediction result
predSVM
```

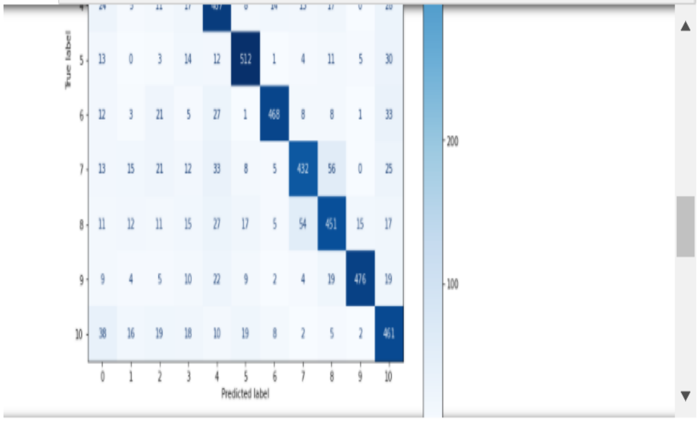
```
In [*]: # Actual Value
Y_test
```

```
In [*]: # Calculage the accuracy score
```

```
In [18]: print(metrics.classification_report(Y_test, Y_pred1))

              precision    recall  f1-score   support

           0       0.78      0.88      0.83       567
           1       0.86      0.87      0.87       591
           2       0.77      0.79      0.78       585
           3       0.84      0.80      0.82       609
           4       0.75      0.75      0.75       627
           5       0.87      0.81      0.84       612
           6       0.89      0.80      0.84       595
           7       0.73      0.69      0.71       612
           8       0.72      0.76      0.74       615
           9       0.94      0.84      0.89       587
          10       0.69      0.81      0.75       601

    accuracy                           0.80      6601
   macro avg       0.80      0.80      0.80      6601
weighted avg       0.80      0.80      0.80      6601
```

```
In [19]: #Create confusion Matrix
from sklearn.metrics import plot_confusion_matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
class_names = ['0','1','2','3','4','5','6','7','8','9','10']
for title, normalize in titles_options:
    fig, ax = plt.subplots(figsize=(11, 11))
    disp = plot_confusion_matrix(ModelSVM, X_test, Y_test,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,ax=ax,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()

Confusion matrix, without normalization
[[497   5  10   1   7   9  10   1   2   0  25]
 [ 11 517  17   3   8   4   3  10   7   1  10]
 [ 12  17 463   7  15   1  10  11   7   0  42]
 [  7  19  21 487  11  11   1  27  10   2  13]
 [ 33   4  19  13 471   7  11  15  24   2  28]
 [ 15   3   3  24  15 498   1   3  23   9  18]
 [  9   3  24   3  27   5 475  10  11   2  26]
 [ 11  10  21  15  27   2   5 424  64   2  31]
 [  9   7  11   8  13   6   2  72 467  14   6]
```

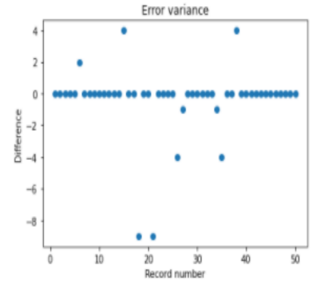
In [27]:
```python
#create confusion matrix
from sklearn.metrics import plot_confusion_matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
class_names = ['0','1','2','3','4','5','6','7','8','9','10']
for title, normalize in titles_options:
    fig, ax = plt.subplots(figsize=(11, 11))
    disp = plot_confusion_matrix(ModelSVM, X_test_ML, Y_test_ML,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,ax=ax,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()
```



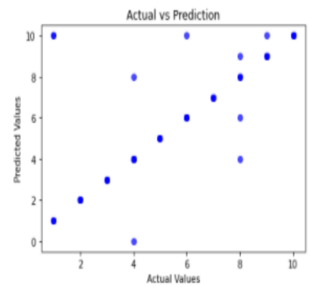

In [28]:
```python
print(np.array(predSVM.tolist()))
```

```
[4 6 3 ... 7 1 3]
```

In [32]:
```python
# This graph shows the error variance between actual value and predicted value for first 50 records.
g=plt.plot(fil_df['Actual'] - fil_df['Prediction'],marker='o',linestyle='')
# Naming x and y labels.
plt.xlabel('Record number')
plt.ylabel('Difference')
# Printing title
plt.title('Error variance')
plt.show()
```




In [33]:
```python
# Scatter plit for co relation between actual values and predicted values.
plt.scatter(fil_df['Actual'], fil_df['Prediction'], alpha=.7, color = 'b')
# Naming x and y label
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
# # Printing title
plt.title('Actual vs Prediction')
plt.show()
```




## KNN - Machine Learning Model - 2

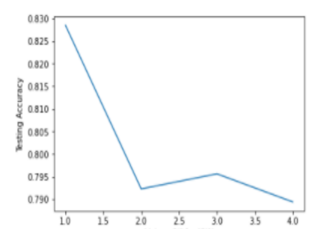
In [40]:
```python
# import knearest neighbors classifier model
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
# Setup a knn classifier with k neighbors
# Try K=1 through K=10 and record testing accuracy
k_range = range(1, 5)

# We can create Python dictionary using [] or dict()
scores = []

# We are using a loop through the range 1 to 10
# We can append the scores in the dictionary
for k in k_range:
    # Create KNN Classifier
    Model_knn = KNeighborsClassifier(n_neighbors=k)
    # Train the model using the training sets
    # Fit the model
    Model_knn.fit(X_train_ML, Y_train_ML)
    # Predict the response for test dataset
    predKNN = Model_knn.predict(X_test_ML)
    # In case of classification algorithms score method represents an accuracy.
    # Evaluating the test accuracy
    scores.append(metrics.accuracy_score(Y_test_ML, predKNN))
print ('Accuracy:')
print(scores)
```

```
Accuracy:
[0.8284848484848485, 0.7922727272727272, 0.7956060606060606, 0.7893939393939394]
```

In [41]:
```python
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Classification report
print('Classification Report: \n',metrics.classification_report(Y_test_ML, predKNN))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.91      0.88       566
           1       0.77      0.91      0.83       589
           2       0.69      0.83      0.76       587
           3       0.76      0.83      0.80       610
           4       0.72      0.81      0.76       624
           5       0.82      0.87      0.85       605
           6       0.76      0.80      0.78       587
           7       0.84      0.75      0.79       620
           8       0.83      0.79      0.81       635
           9       0.92      0.82      0.87       579
          10       0.72      0.36      0.48       598

    accuracy                           0.79      6600
   macro avg       0.79      0.79      0.78      6600
weighted avg       0.79      0.79      0.78      6600
```

Out[42]: Text(0, 0.5, 'Testing Accuracy')




In [43]:
```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import plot_confusion_matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
Class_names= ['0','1','2','3','4','5','6','7','8','9','10']
for title, normalize in titles_options:
  fig,ax =  plt.subplots(figsize=(11,11))
  disp = plot_confusion_matrix(Model_knn, X_test_ML, Y_test_ML,
                               display_labels= Class_names,
                               cmap= plt.cm.Blues,ax=ax,
                               normalize = normalize)
  disp.ax_.set_title(title)

  print(title)
  print(disp.confusion_matrix)
plt.show()
```

# CNN - Deep Learning Model - 3

```
In [48]: # Importing required libraries for CNN implementation
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten,BatchNormalization
from tensorflow.keras.layers import Dense, MaxPooling2D,Conv2D
from tensorflow.keras.layers import Input,Activation,Add
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
```

```
In [49]: # Function for convolution layer
def Convolution(input_tensor,filters):

    x = Conv2D(filters=filters,kernel_size=(3, 3),padding = 'same')(input_tensor)
    x= Activation('relu')(x)

    return x
```

```
In [50]: # CNN model - The default structure for the convolutional layers in the model is
# based on a Conv2D layer with a ReLU activation function, followed by a BatchNormalization
# layer, a MaxPooling and a Dropout layer. Each of these default layers is then followed by
# the final layer for each feature, composed by a Dense layer.

def modelCNN(input_shape):
  inputs = Input((input_shape))

  conv_1= Convolution(inputs,32)
  maxp_1 = MaxPooling2D(pool_size = (2,2)) (conv_1)
  conv_2 = Convolution(maxp_1,64)
  maxp_2 = MaxPooling2D(pool_size = (2, 2)) (conv_2)
  conv_3 = Convolution(maxp_2,64)
  maxp_3 = MaxPooling2D(pool_size = (2, 2)) (conv_3)
  flatten= Flatten() (maxp_3)
  dense_1= Dense(128,activation='relu')(flatten)
  hand = Dense(11,activation="softmax",name='hand')(dense_1)

  modelCNN = Model(inputs=inputs, outputs = hand)

  modelCNN.compile(optimizer='rmsprop',
             loss='categorical_crossentropy',
             metrics=['accuracy'])

  return modelCNN
```

```
In [51]: # Converting input image pixel data to an array and reshaping it.
X = np.array(data['Pixel'].tolist())
X = X.reshape(X.shape[0],64,64,1)
```

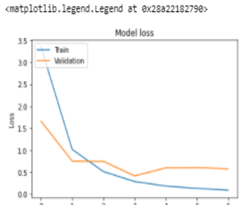
```
In [52]: # Converting labels to category
from keras.utils import to_categorical
labels_f=to_categorical(np.array(data['Label'], dtype=int))
```

```
Epoch 00006: val_loss did not improve from 0.41431
181/181 [==============================] - 64s 353ms/step - loss: 0.1298 - accuracy: 0.9629 - val_loss: 0.6048 - val_accuracy:
0.8495
Epoch 7/500
181/181 [==============================] - ETA: 0s - loss: 0.0894 - accuracy: 0.9740
Epoch 00007: val_loss did not improve from 0.41431
181/181 [==============================] - 65s 361ms/step - loss: 0.0894 - accuracy: 0.9740 - val_loss: 0.5727 - val_accuracy:
0.8901
```

```
In [58]: # Evaluating CNN model on test data
ModelCNN.evaluate(X_test,Y_test)

155/155 [==============================] - 6s 39ms/step - loss: 0.4143 - accuracy: 0.8901

Out[58]: [0.41431161761283875, 0.8901010155677795]
```

```
In [59]: # Model prediction on test data
predCNN=ModelCNN.predict(X_test)
print(predCNN)

[[6.47551133e-07 2.89460101e-12 1.04322744e-05 ... 1.19897950e-05
  4.94795386e-05 3.00872841e-07]
 [6.08528990e-06 1.06076811e-07 1.83149171e-03 ... 2.26420013e-07
  1.05925824e-10 1.99286951e-08]
 [1.21651333e-09 2.49985189e-18 6.20423445e-14 ... 4.73028630e-11
  3.70272710e-17 5.56299729e-11]
 ...
 [4.86658809e-16 7.82166943e-19 3.21158829e-18 ... 1.35070576e-14
  7.37304466e-20 1.00000000e+00]
 [7.14494614e-03 4.56885709e-06 3.58600516e-07 ... 1.21191770e-01
  7.11808035e-09 7.59708211e-02]
 [1.35744671e-07 7.07626457e-10 1.42070758e-05 ... 1.96131378e-08
  1.87961220e-08 2.78481765e-10]]
```

```
In [60]: # Plotting CNN model loss
plt.plot(History.history['loss'])
plt.plot(History.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

Out[60]: <matplotlib.legend.Legend at 0x28a22182790>
```




```
In [54]: # Initializing the CNN model
ModelCNN=modelCNN((64,64,1))
```

```
In [55]: # Prininting CNN model
ModelCNN.summary()

Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 64, 64, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 64, 64, 32)        320
_________________________________________________________________
activation (Activation)      (None, 64, 64, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 128)               524416
_________________________________________________________________
hand (Dense)                 (None, 11)                1419
=================================================================
Total params: 581,579
Trainable params: 581,579
Non-trainable params: 0
_________________________________________________________________
```

```
In [56]: # Defining model checkpoints and callback
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow as tf
fle_s='hand_ges_CNN.h5'
checkpointer = ModelCheckpoint(fle_s, monitor='val_loss',verbose=1,save_best_only=True,save_weights_only=False, mode='auto',save
Early_stop=tf.keras.callbacks.EarlyStopping(patience=3, monitor='val_loss',restore_best_weights=True),
callback_list=[checkpointer,Early_stop]
```

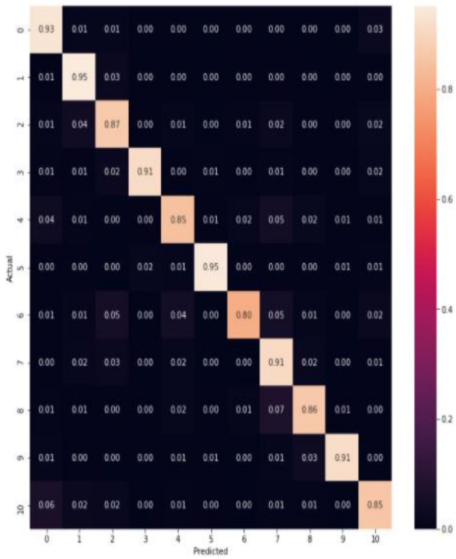
```
In [*]: # Fitting CNN model on training data
History=ModelCNN.fit(X_train,Y_train,batch_size=64,validation_data=(X_test,Y_test),epochs=500,callbacks=[callback_list])

Epoch 1/500
 73/181 [===========>.................] - ETA: 32s - loss: 5.7022 - accuracy: 0.2307
```

```
In [62]: # Collecting all predicted values in a list
i=0
Pred_l_CNN=[]
while(i<len(predCNN[0])):
  Pred_l_CNN.append(int(np.round(predCNN[0][i])))
  i+=1
```

```
In [63]: # CNN model evaluation - classification report
from sklearn.metrics import classification_report
hand_gesture_class = classification_report(np.argmax(Y_test,axis=1),np.argmax(ModelCNN.predict(X_test), axis=1))
print(hand_gesture_class)

              precision    recall  f1-score   support

           0       0.84      0.93      0.89       422
           1       0.87      0.95      0.91       450
           2       0.83      0.87      0.85       428
           3       0.97      0.91      0.94       460
           4       0.88      0.85      0.86       486
           5       0.96      0.95      0.95       443
           6       0.94      0.80      0.86       448
           7       0.80      0.91      0.85       463
           8       0.89      0.86      0.88       464
           9       0.97      0.91      0.94       432
          10       0.88      0.85      0.87       454

    accuracy                           0.89      4950
   macro avg       0.89      0.89      0.89      4950
weighted avg       0.89      0.89      0.89      4950
```

```
In [64]: # CNN model evaluation - confusion matrix
from sklearn.metrics import confusion_matrix
hand_gesture_confu = confusion_matrix(np.argmax(Y_test,axis=1),np.argmax(ModelCNN.predict(X_test), axis=1))
print(hand_gesture_confu)

[[394   6   5   0   0   1   2   1   1   0  12]
 [  3 428  13   0   1   1   0   2   0   0   2]
 [  5  19 372   2   5   0   4  10   2   0   9]
 [  4   3  11 419   2   5   0   4   2   2   8]
 [ 18   5   0   2 411   3   8  23  10   3   3]
 [  2   0   0   8   5 419   0   2   0   3   4]
 [  3   5  24   0  20   0 358  22   5   1  10]
 [  1   8  12   0   7   0   1 421  10   0   3]
 [  4   3   2   0   7   2   6  34 401   3   2]
 [  5   2   0   1   3   3   1   5  15 395   2]
 [ 29  11  10   0   4   2   1   4   4   1 388]]
```

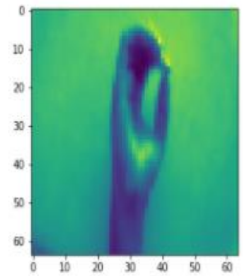
In [65]:
```python
# Plotting confusion matrix using seaborn
import seaborn as sns
cm = confusion_matrix(np.argmax(Y_test,axis=1),np.argmax(ModelCNN.predict(X_test), axis=1))
# Normalise
cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
fig, ax = plt.subplots(figsize=(11,11))
sns.heatmap(cmn, annot=True, fmt='.2f')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show(block=False)
```




In [66]:
```python
# Defining a function for test image prediction using model
def validate(ind,Model):
  plt.imshow(data['Pixel'].iloc[ind].reshape(64,64))
  test = np.array(data['Pixel'].iloc[ind])
  test = test.reshape(64,64)
  pred_1=Model.predict(np.array([test]))
  print(pred_1)

  hand=int(np.round(np.argmax(pred_1[0])))
  print("Predicted Number: "+ str(hand))
  print("Actual Number: "+ str(np.array(data['Label'].iloc[ind])))
```

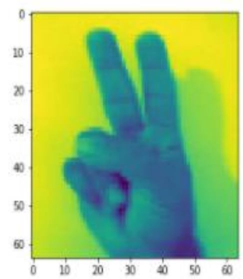
In [67]:
```python
validate(100,ModelCNN)
```

```
[[1.0000000e+00 2.2549336e-10 2.4730658e-11 7.5459283e-10 5.4509632e-09
  1.5477954e-09 1.4255460e-08 2.0766985e-08 1.5771899e-11 3.1831897e-13
  8.1886536e-10]]
Predicted Number: 0
Actual Number: 0
```



In [68]:
```python
validate(3000,ModelCNN)
```

```
[[1.0263575e-06 9.7538467e-04 9.9895918e-01 2.0739399e-06 1.0699740e-08
  2.5675016e-13 9.8719038e-06 5.2438914e-05 8.9878107e-08 2.9238187e-09
  1.9113500e-08]]
Predicted Number: 2
Actual Number: 2
```

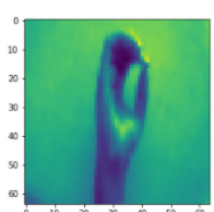


## Ensemble Model

In [94]:
```python
def ensembleModel(ind,Model1,Model2,Model3):
  plt.imshow(data['Pixel'].iloc[ind].reshape(64,64))
  test = np.array(data['Pixel'].iloc[ind])
  test = test.reshape(64,64)
  test_ML = test.reshape(4096)

  pred_1=Model1.predict(np.array([test_ML]))
  print(pred_1)
  hand_1=int(np.round(np.argmax(pred_1[0])))

  pred_2=Model2.predict(np.array([test_ML]))
  print(pred_2)
  hand_2=int(np.round(np.argmax(pred_2[0])))

  pred_3=Model3.predict(np.array([test]))
  print(pred_3)
  hand_3=int(np.round(np.argmax(pred_3[0])))


  final_pred = np.array([])
  for i in range(0,len(X_test)):
     final_pred = np.argmax(np.append(final_pred, np.max([hand_1, hand_2,hand_3])))

  print("Predicted Number: "+ str(final_pred))
  print("Actual Number: "+ str(np.array(data['Label'].iloc[ind])))
```

In [95]:
```python
ensembleModel(100,ModelSVM,Model_knn,ModelCNN)
```

```
[0]
[0]
[[1.0000000e+00 2.2549336e-10 2.4730658e-11 7.5459283e-10 5.4509632e-09
  1.5477954e-09 1.4255460e-08 2.0766985e-08 1.5771899e-11 3.1831897e-13
  8.1886536e-10]]
Predicted Number: 0
Actual Number: 0
```



In [ ]:

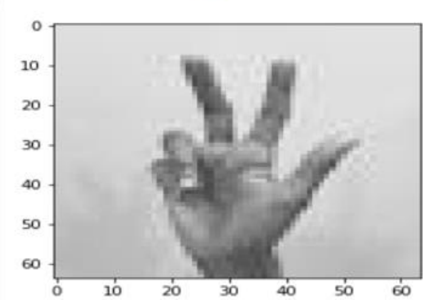
**Improvement from the previous increment –**

1. In increment 3 we worked on data pre-processing and implementing CNN deep learning model
2. In increment 4 –
    a. Converted input image data to pixel and saved to csv. So that it can be used as input for all models instead of pre-processing data again
    b. We built SVM, KNN and improved CNN model implementation
    c. Successfully executed all models on train data
    d. Successfully compared the results from each model
    e. Completed model evaluation using classification report and confusion matrix
    f. Finally, built ensemble model to generate combined output

**Important code snippets from the project -**

## Data Pre-processing

```python
#Only for Visualizing the first image of each category
for category in CATEGORIES:
    path = os.path.join(dir,category)  # create path to categories
    for img in os.listdir(path):
      # iterate over each image
      # convert to array
      print('Label of below image is {}'.format(category))
      img_array = cv2.imread(os.path.join(path,img))
      print('Image size of {} is {}'.format(img, img_array.shape))
      plt.imshow(img_array, cmap = None)  # graph it
      plt.show()  # display!

      print('Printing the graysacle image')
      img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)
      plt.imshow(img_array, cmap='gray')  # graph it
      plt.show()  # display!
      print('Resized image')
      img_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
      plt.imshow(img_array, cmap='gray')
      plt.show() # Show resize image
      break  # we just want one for now so break
```




```
Printing the graysacle image
```

```python
'''
This function will create csv file with header.
Each image will have a header for pixel.
'''
def get_images1(directory):
    output = []
    print("Loading {}".format(directory))
    #open a file to write the header data
    with open('output_file1.csv', 'w+') as f:
      f.write('FileName,Pixel,Label\n')

    # Iterate through each folder corresponding to a category
    for category in CATEGORIES:
        path = os.path.join(directory,category)  # create path to categories
        class_num = CATEGORIES.index(category)  # get the classification
        print('Processing folder {}'.format(category))
        # Iterate through each image in folder
        for file in tqdm(os.listdir(path)):
          # Get the path name of the image
          img_path = os.path.join(path, file)
          # Open and read the image in grayscale
          image_array = cv2.imread(img_path,cv2.IMREAD_GRAYSCALE)
          # Resize the image
          image_resize = cv2.resize(image_array, (IMG_SIZE, IMG_SIZE))
          # Append the image and its corresponding label to the output
          output.append([image_resize, class_num])
          #writing data to csv file
          flat_pixel = (image_resize.flatten())
          flat_pixel = " ".join(map(str, flat_pixel))
          with open('output_file1.csv', 'a') as f:
              f.write(file+',')
              f.write(flat_pixel)
              f.write(','+str(class_num))
              f.write('\n')

    return output
```

```python
'''
Reading the file from csv
'''
data = pd.read_csv('output_file1.csv')
```

```python
data.head()
```

| | FileName | Pixel | Label |
|---|---|---|---|
| 0 | zero_1.jpg | 173 190 209 219 212 192 159 141 137 141 112 10... | 0 |
| 1 | zero_10.jpg | 99 94 98 94 94 57 67 88 101 104 103 97 88 97 1... | 0 |
| 2 | zero_100.jpg | 98 100 102 108 120 128 135 140 145 148 148 153... | 0 |
| 3 | zero_1000.jpg | 220 251 254 254 252 254 251 252 243 209 240 25... | 0 |
| 4 | zero_1001.jpg | 141 137 137 140 142 145 146 149 153 154 155 15... | 0 |

# SVM

### Step 1: Data Load and formatting.

```python
# Imported the necessary libraries.
# Pandas is a library use for data manipulation and analysis
import numpy as np
from sklearn.metrics import classification_report
import pandas as pd
from sklearn.model_selection import train_test_split
# To implement the Support Vector Machines we will use Scikit-learn and will import our SVM
from sklearn import svm
from sklearn.metrics import accuracy_score
```

```python
df = pd.read_csv('output_file1.csv')
df.head()
```

| | FileName | Pixel | Label |
|---|---|---|---|
| 0 | zero_1353.jpg | 141 138 138 140 142 145 146 148 150 151 152 15... | 0 |
| 1 | zero_1450.jpg | 142 140 138 138 141 144 145 146 150 150 149 15... | 0 |
| 2 | zero_1409.jpg | 139 136 134 135 136 139 141 143 145 146 149 14... | 0 |
| 3 | zero_1351.jpg | 139 139 138 139 143 144 146 147 149 151 153 15... | 0 |
| 4 | zero_1390.jpg | 139 135 134 136 137 138 142 145 149 149 151 15... | 0 |

```python
# Format Pixel column in each row into arrays of float datatype values.
df['Pixel']=df['Pixel'].apply(lambda x:  np.array(x.split(), dtype="float32"))
```

```python
df['Pixel']
```

```
0        [141.0, 138.0, 138.0, 140.0, 142.0, 145.0, 146...
1        [142.0, 140.0, 138.0, 138.0, 141.0, 144.0, 145...
2        [139.0, 136.0, 134.0, 135.0, 136.0, 139.0, 141...
```

### Step 2: Create, Train & Fit model.

```python
[ ] # Separate features and target columns from input dataset
    # We have only one feature i.s Pixel column
    # Format Pixel column as list of data items
    X = np.array(df['Pixel'].tolist())

    # For SVM model it is must that we should have X and y i.e features and target less than or equal to 2 dimension.
    X.shape

    (16501, 4096)

[ ] # Our target columns is Label. We have total 11 categories.
    y = np.array(df['Label'], dtype=int)

[ ] # Split X and y into Train and Test datasets
    X_train, X_test, Y_train, Y_test = train_test_split( X, y, test_size = 0.4, random_state=40)

[ ] # Create SVM model. We are using Gaussian's rbf kernel.
    Model = svm.SVC(kernel="rbf")

[ ] # Fitting the Model
    Model.fit(X_train, Y_train)
```

# SVM

### Prediction:

```python
# Predict the target using test dataset
Y_pred1 = ModelSVM.predict(X_test)
```

```python
# Prediction result
Y_pred1
```

```
array([8, 3, 7, ..., 7, 4, 9])
```

```python
# Actual Value
Y_test
```

```
array([8, 3, 7, ..., 7, 4, 9])
```

### Accuracy:

```python
# Calculage the accuracy score
accuracy = round(accuracy_score(Y_test,Y_pred1)*100,2)
```

```python
# Accuracy of svc model
accuracy
```

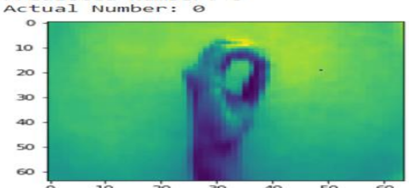
```
79.99
```

# SVM

Validation on Test Images

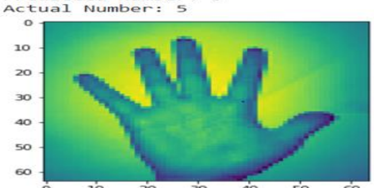
```python
validate(50,Model)
```

```
[0]
Predicted Number: 0
Actual Number: 0
```




```python
validate(9000,Model)
```

```
[5]
Predicted Number: 5
Actual Number: 5
```



# KNN

## Step 1: Data load and formatting

```
'''
Reading the file from csv
'''
data = pd.read_csv('output_file1.csv')
```

```
data.head()
```

| | FileName | Pixel | Label |
|---|---|---|---|
| 0 | zero_593.jpg | 55 56 56 56 57 59 60 60 62 63 63 63 64 65 65 6... | 0 |
| 1 | zero_601.jpg | 124 133 140 147 150 152 159 163 166 168 170 17... | 0 |
| 2 | zero_598.jpg | 35 35 37 38 40 45 53 66 74 91 107 115 124 133 ... | 0 |
| 3 | zero_59.jpg | 131 133 132 136 132 117 118 146 145 121 127 12... | 0 |
| 4 | zero_592.jpg | 52 54 55 56 56 57 59 60 60 61 62 63 64 65 65 6... | 0 |

```
data['Pixel']
```

```
0        [55.0, 56.0, 56.0, 56.0, 57.0, 59.0, 60.0, 60....
1        [124.0, 133.0, 140.0, 147.0, 150.0, 152.0, 159...
2        [35.0, 35.0, 37.0, 38.0, 40.0, 45.0, 53.0, 66....
3        [131.0, 133.0, 132.0, 136.0, 132.0, 117.0, 118...
4        [52.0, 54.0, 55.0, 56.0, 56.0, 57.0, 59.0, 60....
                               ...
16495    [93.0, 92.0, 89.0, 89.0, 91.0, 97.0, 103.0, 10...
16496    [114.0, 111.0, 110.0, 105.0, 105.0, 107.0, 112...
16497    [49.0, 51.0, 52.0, 52.0, 53.0, 53.0, 54.0, 54....
16498    [36.0, 36.0, 35.0, 33.0, 32.0, 28.0, 26.0, 26....
16499    [78.0, 83.0, 85.0, 82.0, 79.0, 70.0, 69.0, 83....
Name: Pixel, Length: 16500, dtype: object
```

## Step 2: Create, Train/Test and Fitting Model

```
# We have only one feature in dataset which represents pixel
# Convert the pixel column as list of data items,
X = np.array(data['Pixel'].tolist())
```

```
# Printing the x variable shape.
# For knn model x and y variable pixel size less than are equal to two dimensions.
X.shape
```

```
(16500, 4096)
```

```
# our target column is label.
# Assigning labels into y variable
y = np.array(data['label'], dtype=int)
```

```
#  Splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test= train_test_split(X,y,test_size=0.30, random_state=40)
```

```
# Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors=k)
# Train the model using the training sets
# Fit the model
knn.fit(X_train, Y_train)
```

# KNN

No. of K neighbors

Knn classifier

Fit the model

Prediction

Accuracy

```
# Import knearest neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
# Setup a knn classifier with k neighbors
# Try K=1 through K=10 and record testing accuracy
k_range = range(1, 10)

# We can create Python dictionary using [] or dict()
scores = []

# We are using a loop through the range 1 to 10
# We can append the scores in the dictionary
for k in k_range:
    # Create KNN Classifier
    knn = KNeighborsClassifier(n_neighbors=k)
    # Train the model using the training sets
    # Fit the model
    knn.fit(X_train, Y_train)
    # Predict the response for test dataset
    Y_pred1 = knn.predict(X_test)
    # In case of classification algorithms score method represents an accuracy.
    # Evaluating the test accuracy
    scores.append(metrics.accuracy_score(Y_test, Y_pred1))
print(Y_test.shape)
print ('Accuracy:')
print(scores)
```

# KNN

## Accuracy values for KNN, with K values from 1 to 10

```
Accuracy:
[0.8432323232323232, 0.8022222222222222, 0.8157575757575758, 0.8054545454545454, 0.8018181818181818, 0.7977777777777778, 0.7925252525252525, 0.7828282828282829, 0.7739393939393939]
```

```
Y_pred1
```

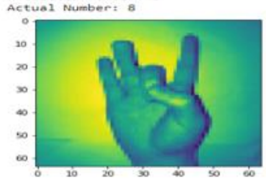
```
array([10,  6,  3, ...,  6,  9,  7])
```

```
Y_test
```

```
array([ 4,  6,  3, ..., 10,  8,  7])
```

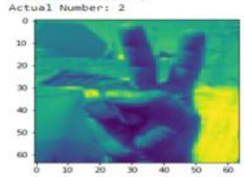
```
validate(12700,knn)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]]
Predicted Number: 8
Actual Number: 8
```



```
validate(3000,knn)
```

```
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
Predicted Number: 2
Actual Number: 2
```



### CNN - Deep Learning Model - 3

```
In [ ]:
# importing required libraries for CNN implementation
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten,BatchNormalization
from tensorflow.keras.layers import Dense, MaxPooling2D,Conv2D
from tensorflow.keras.layers import Input,Activation,Add
from tensorflow.keras.models import Model
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
```

```
In [ ]:
# Function for convolution layer
def Convolution(input_tensor,filters):

    x = Conv2D(filters=filters,kernel_size=(3, 3),padding = 'same')(input_tensor)
    x= Activation('relu')(x)

    return x
```

```
In [ ]:
# CNN model - The default structure for the convolutional layers in the model is
# based on a Conv2D layer with a ReLU activation function, followed by a BatchNormalization
# layer, a MaxPooling and a Dropout layer. Each of these default layers is then followed by
# the final layer for each feature, composed by a Dense layer.

def modelCNN(input_shape):
  inputs = Input((input_shape))

  conv_1= Convolution(inputs,32)
  maxp_1 = MaxPooling2D(pool_size = (2,2)) (conv_1)
  conv_2 = Convolution(maxp_1,64)
  maxp_2 = MaxPooling2D(pool_size = (2, 2)) (conv_2)
  conv_3 = Convolution(maxp_2,64)
  maxp_3 = MaxPooling2D(pool_size = (2, 2)) (conv_3)
  flatten= Flatten() (maxp_3)
  dense_1= Dense(128,activation='relu')(flatten)
  hand = Dense(11,activation="softmax",name='hand')(dense_1)

  modelCNN = Model(inputs=inputs, outputs = hand)

  modelCNN.compile(optimizer='rmsprop',
            loss='categorical_crossentropy',
            metrics=['accuracy'])

  return modelCNN
```

```
In [ ]: # Defining a function for test image prediction using model
        def validate(ind,Model):
          plt.imshow(data['Pixel'].iloc[ind].reshape(64,64))
          test = np.array(data['Pixel'].iloc[ind])
          test = test.reshape(64,64)
          pred_1=Model.predict(np.array([test]))
          print(pred_1)

          hand=int(np.round(np.argmax(pred_1[0])))
          print("Predicted Number: "+ str(hand))
          print("Actual Number: "+ str(np.array(data['Label'].iloc[ind])))
```

```
In [ ]: validate(100,ModelCNN)
```

```
In [ ]: validate(3000,ModelCNN)
```

```
In [ ]: validate(15000,ModelCNN)
```

```
In [ ]: validate(10000,ModelCNN)
```

#### Ensemble Model

```
In [ ]: def ensembleModel(ind,Model1,Model2,Model3):
          plt.imshow(data['Pixel'].iloc[ind].reshape(64,64))
          test = np.array(data['Pixel'].iloc[ind])
          test = test.reshape(64,64)
          test_ML = test.reshape(4096)

          pred_1=Model1.predict(np.array([test_ML]))
          print(pred_1)
          hand_1=int(np.round(np.argmax(pred_1[0])))

          pred_2=Model2.predict(np.array([test_ML]))
          print(pred_2)
          hand_2=int(np.round(np.argmax(pred_2[0])))

          pred_3=Model3.predict(np.array([test]))
          print(pred_3)
          hand_3=int(np.round(np.argmax(pred_3[0])))


          final_pred = np.array([])
          for i in range(0,len(x_test)):
              final_pred = np.append(final_pred, mode([hand_1, hand_2,hand_3))

          print("Predicted Number: "+ str(final_pred))
          print("Actual Number: "+ str(np.array(data['Label'].iloc[ind])))
```

```
In [ ]: ensembleModel(100,modelSVM,Model_knn, ModelCNN)
```

**Work sharing/Module sharing between teammates:**

Prabhajan Trivedi –

1. Worked on prepressing of data
2. Converting input image data to .csv file
3. Worked with other team members on model implementation

Harshita Patil –

1. Worked on SVM model and successfully completed it
2. Completed SVM model evaluation
3. Calculated SVM model accuracy
4. Prediction of test data images using SVM model
5. Worked on final code base merging and reporting / presentation

Shireesha Maddi –

1. Worked on KNN model and successfully completed it
2. Completed KNN model evaluation
3. Calculated KNN model accuracy
4. Prediction of test data images using KNN model
5. Worked on final code base merging and reporting / presentation

Vasim Shaikh –

1. Worked on CNN model and successfully completed it
2. Completed CNN model evaluation
3. Calculated CNN model accuracy
4. Prediction of test data images using CNN model
5. Worked on final code base merging and reporting / presentation

**Any issues, blockages with the project:**

1. KNN model execution with different values of ranging from 1 to 10 took a lot of time compared to SVM and CNN model – Time consuming
2. Unable to upload data to Github due to size issue

**Conclusion and Future work–**

1. Successful implemented all the models – SVM , KNN and CNN to identify Sign Language for Numbers
2. Were able to get comparable results to solutions which were implemented with separate models for predicting each of the categories
3. Improved performance with Deep learning model as compared to machine learning models
4. Future work – implement different deep learning models and ensemble model

**Github Link** – https://github.com/vasimshaikh39/python2020

**Codebooks-**
https://github.com/vasimshaikh39/python2020/blob/main/Python_Project_Sign_Language_For_Numbers_Group_5_Code.ipynb

**Presentation Link -**
https://github.com/vasimshaikh39/python2020/blob/main/Python_Project_Presentation_Final_Group_5_v1.0.pptx

**Video Link -**

https://github.com/vasimshaikh39/python2020/blob/main/Python_Project_Group_5_Video_recording.mp4

## References:


1. https://www.kaggle.com/muhammadkhalid/sign-language-for-numbers
2. https://heartbeat.fritz.ai/introduction-to-machine-learning-model-evaluation-fa859e1b2d7f
3. https://data-flair.training/blogs/sign-language-recognition-python-ml-opencv/
4. https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234
5. https://www.kaggle.com/joshbeau/numerical-sign-language-recognition-tensorflow