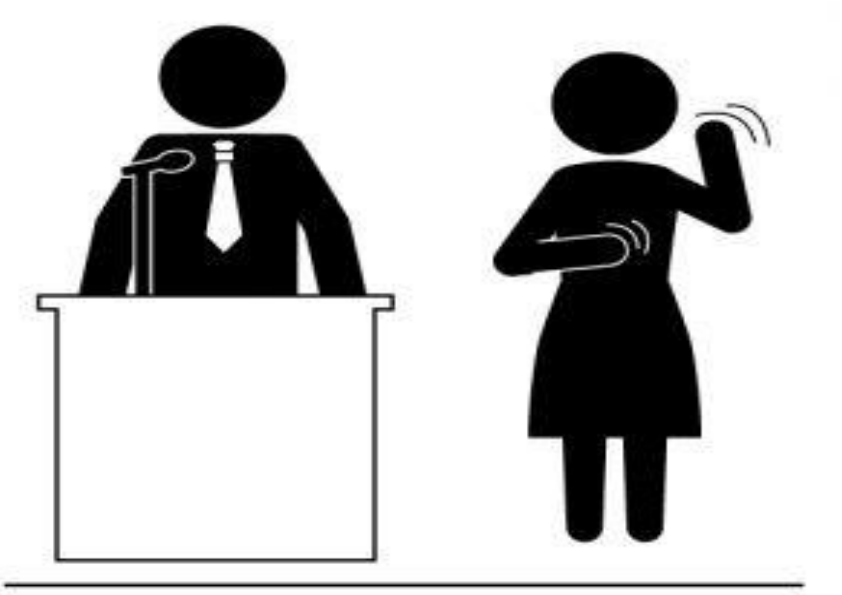
PYTHON PROJECT

SIGN LANGUAGE RECOGNITION



SUBMITTED BY: -

HRITIK RAO (RA1611003010365)

ACKNOWLEDGEMENT

The success and the final outcome of this project required guidance and assistance from different sources and we feel extremely fortunate to have got this all along the completion of our project. Whatever we have done is largely due to such guidance and assistance and we would not forget to thank them. We would give our sincere gratitude to **MS. D. AISHWARYA** for guiding us to complete this project.

We express our sincere thanks to the Head of the Department, Department of Computer Science and Engineering, **Dr. B. Amutha** for all the help and infrastructure provided to us to complete this project successfully and his valuable guidance.

And last but not the least, I would like to thank my classmate and friends for their support and willingness to participate in group discussions.

Table of content

ABSTRACT

Communication is the process of exchanging information, views and expressions between two or more persons, in both verbal and non-verbal manner. Hand gestures are the nonverbal method of communication used along with verbal communication.

A more organized form of hand gesture communication is known as sign language. In this language each alphabet of the English vocabulary is assigned a sign. The physically disabled person like the deaf and the dumb uses this language to communicate with each other. The idea of this project is to design a system that can understand the sign language accurately so that the less fortunate people may communicate with the outside world without the need of an interpreter.

By keeping this in mind, the fact that in normal cases every human being has the same hand shape with four fingers and one thumb, this project aims at designing a real time system for the recognition of some meaningful shapes made using hands.

Problem statement

How do we ensure that everyone has a voice? --- Even those who don’t have one.

There are approximately 70 million people, who are hearing and speech impaired.

Static hand gestures are the most essential facets of gesture recognition. Sign language is a natural way of interaction for such people, and is vital for their everyday functioning.

Static hand gesture recognition does without any supplementary devices which are used to give instructions to a machine.

Hand Gesture Recognition, a person instructs the machine using his bare hands, whereas images of the persons hand gestures are captured and analysed in order to determine the meaning of the hand gesture.

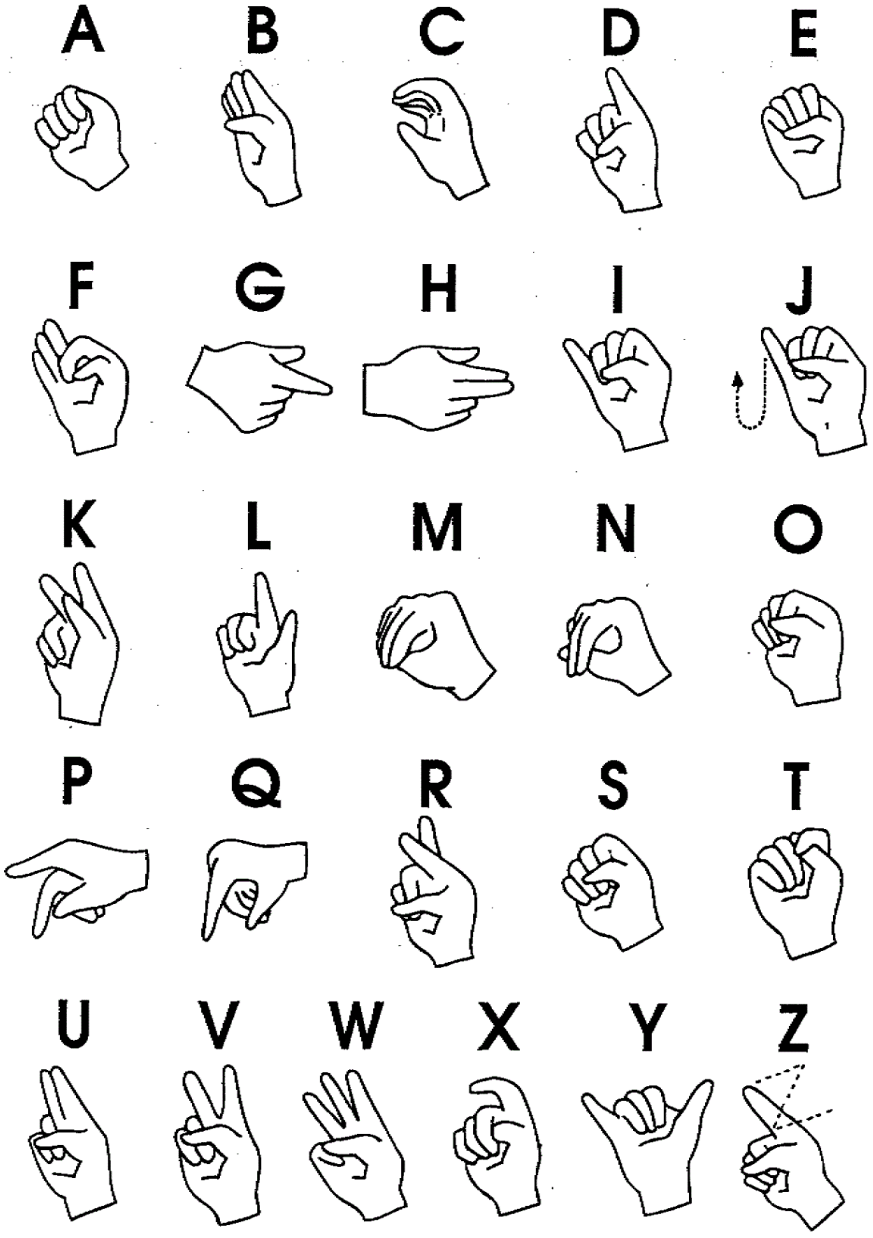
Find and extract features that can be used to determine the meaning of a given gesture - should uniquely describe the gesture in order to achieve a reliable recognition

Introduction

A gesture may be defined as a movement, usually of hand or face that expresses an idea, sentiment or emotion e.g. rising of eyebrows, shrugging of shoulders are some of the gestures we use in our day to day life. Sign language is a more organized and defined way of communication in which every word or alphabet is assigned some gesture. In American Sign Language (ASL) each alphabet of English vocabulary, A-Z, is assigned a unique gesture. Sign language is mostly used by the deaf, dumb or people with any other kind of disabilities. With the rapid advancements in technology, the use of computers in our daily life has increased manifolds. Our aim is to design a Human Computer Interface (HCI) system that can understand the sign language accurately so that the signing people may communicate with the non-signing people without the need of an interpreter. It can be used to generate speech or text. Unfortunately, there has not been any system with these capabilities so far. A huge population in India alone is of the deaf and dumb. It is our social responsibility to make this community more independent in life so that they can also be a part of this growing technology world. In this work a sample sign language has been used for the purpose of testing.

No one form of sign language is universal as it varies from region to region and country to country and a single gesture can carry a different meaning in a different part of the world. Various available sign languages are American Sign Language (ASL), British Sign Language (BSL), Turkish Sign Language (TSL), Indian Sign Language (ISL) and many more. There is a total of 26 alphabets in the English vocabulary. Each alphabet may be assigned a unique gesture. In our project, the image of the hand is captured using a simple web camera. The acquired

image is then processed and some features are extracted. These features are then used as input to a classification algorithm for recognition. The recognized gesture may then be used to generate speech or text. Few attempts have been made in the past to recognize the gestures made using hands but with limitations of recognition rate and time. This project aims at designing a fully functional system with significant improvement from the past works. The simple sign language is shown in the figure.



Simple sign language chart

LITERATURE REVIEW

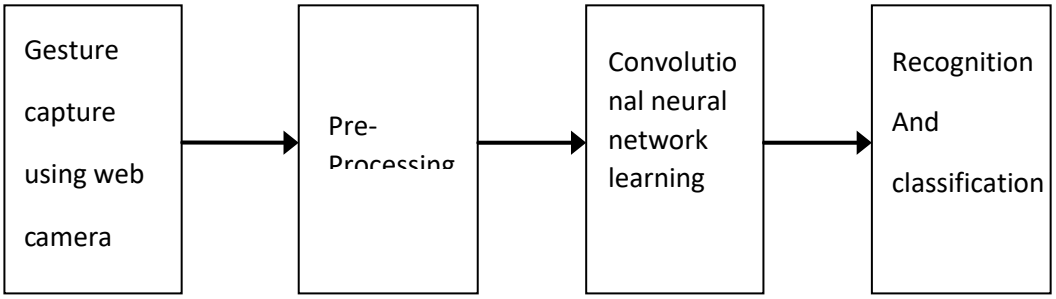
The importance of sign language may be understood from the fact that early humans used to communicate by using sign language even before the advent of any vocal language. Since then it has been adopted as an integral part of our day to day communication. We make use of hand gestures, knowingly or unknowingly, in our day to day communication. Now, sign languages are being used extensively as international sign use for the deaf and the dumb, in the world of sports by the umpires or referees, for religious practices, on road traffic boards and also at work places.

Gestures are one of the first forms of communication that a child learns to express whether it is the need for food, warmth and comfort. It increases the impact of spoken language and helps in expressing thoughts and feelings effectively. Christopher Lee and Yangsheng Xu developed a glove-based gesture recognition system that was able to recognize 14 of the letters from the hand alphabet, learn new gestures and able to update the model of each gesture in the system in online mode, with a rate of 10Hz. Over the years advanced glove devices have been designed such as the Sayre Glove, Dexterous Hand Master and Power Glove. The most successful commercially available glove is by far the VPL Data Glove as shown in figure 2

It was developed by Zimmerman during the 1970's. It is based upon patented optical fiber sensors along the back of the fingers. Star-ner and Pentland developed a glove-environment system capable of recognizing 40 signs from the American Sign Language (ASL) with a rate of 5Hz. Hyeon-Kyu Lee and Jin H. Kim presented work on real-time hand-gesture recognition using HMM (Hidden Markov Model). P. Subha Rajam and Dr. G. Balakrishnan proposed a system for the recognition of south Indian Sign Language. The system assigned a binary 1 to each finger detected. The accuracy of this system was 98.12%. Olena Lomakina studied various approaches for the development of a gesture recognition system. Etsuko Ueda and Yoshio Matsumoto proposed a hand-pose estimation technique that can be used for vision-based human interfaces. Claudia Nölker and Helge Ritter presented a hand gesture recognition modal based on recognition of finger tips, in their approach they find full identification of all finger joint angles and based on that a 3D modal of hand is prepared and using neural network. In 2011, Meenakshi Panwar proposed a shape-based approach for hand gesture recognition with several steps including smudges elimination orientation detection, thumb detection, finger counts etc. Visually Impaired people can make use of hand gestures for writing text on electronic document like MS Office, notepad etc. The recognition rate was improved up to 94% from 92%.

SIGN LANGUAGE RECOGNITION SYSTEM

The sign language recognition done using cameras may be regarded as vision-based analysis system. The idea may be implemented using a simple web camera and a computer system. The web camera captures the gesture image with a resolution of 28x28 pixels. The captured image is then processed for recognition purpose. The idea may be represented by the block diagram as shown in figure.



1 Gesture capture using web camera

The first step towards image processing is to acquire the image. The acquired image that is stored in the system windows needs to be connected to the software automatically. This is done by creating an object. With the help of high-speed processors available in computers today, it is possible to trigger the camera and capture the images in real time. The image is stored in the buffer of the object. As has been already discussed, the image is acquired using a simple web camera. Image acquisition devices typically support multiple video formats. When we create a video input object, we can specify the video format that you want the device to use. If the video format as an argument is not specified, the video input function uses the default format.

The acquired image is RGB image and needs to be processed before its features are extracted and recognition is made.

. Acquired image, gesture ‘d’

2 Pre-Processing

The captured image is a RGB image. This image is first converted into grayscale as some of the pre-processing operations can be applied on grayscale image only.

Grey Scale Image, gesture ‘D’

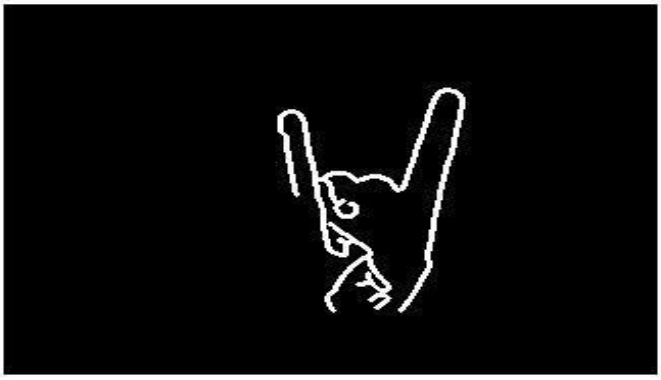
3 Edges & peaks detection

The edges are detected in the binary image. A number of edge detection techniques may be used in CNN. The Edge Detection block finds the edges in an input image by approximating the gradient magnitude of the image. The block convolves the input matrix with the Sobel, Prewitt, or Roberts kernel. The block outputs two gradient components of the image, which are the result of this convolution operation. Alternatively, the block can perform a thresholding operation on the gradient magnitudes and output a binary image, which is a matrix of Boolean values. If a pixel value is 1, it is an edge. For Canny, the Edge Detection block finds edges by looking for the local maxima of the gradient of the input image. It calculates the gradient using the derivative of the Gaussian filter. The Canny method uses two thresholds to detect strong and weak edges. It includes the weak edges in the output only if they are connected to strong edges. As a result, the method is more robust to noise, and more likely to detect true weak edges. In this project we have used canny edge detection.

The purpose of edge detection in general is to significantly reduce the amount of data in an image, while preserving the structural properties to be used for further image processing. Canny edge detection was developed by John F. Canny (JFC) in 1986. Even though it is quite old, it has become one of the standard edge detection methods and it is still used in research. The Canny edge detector works on gray scale image. In image processing finding edge is fundamental problem because edge defines the boundaries of different objects. Edge can be defined as sudden or strong change in the intercity or we can say sudden jump in intensity from one pixel to another pixel.

By finding the edge in any image we are just reducing some amount of data but we are preserving the shape. The Canny edge detection algorithm is known as the optimal edge detector. Canny, improved the edge detection by following a list of criteria. The first is low error rate. Low error rate means edges occurring in images should not be missed and that there are NO responses to non-edges. The second criterion is that the edge points be well localized. In other words, the distance between the edge pixels as found by the detector and the actual edge is to be at a minimum. A third criterion is to have only one response to a single edge. This was implemented because the first 2 were not substantial enough to completely eliminate the possibility of multiple responses to an edge. Based on these criteria, the canny edge detector first smoothest the image to eliminate and noise. It then finds the image gradient to highlight regions with high spatial derivatives. The algorithm then tracks along these regions and suppresses any pixel that is not at the maximum (non-maximum suppression). The gradient array is now further reduced by hysteresis. Hysteresis is used to track along the remaining pixels that have not been suppressed. Hysteresis uses two thresholds and if the magnitude is below the first threshold, it is set to zero (made a non-edge). If the magnitude is above the high threshold, it is made an edge. And if the magnitude is between the 2 thresholds, then it is set to zero. The resulting image contains a number of discrete objects. The discontinuities are joined using k-

Nearest Neighbour search. The k-nearest neighbour (kNN) search helps to find the k closest points in X to a query point or set of points. The kNN search technique and kNN-based algorithms are widely used as benchmark learning rules—the relative simplicity of the kNN search technique makes it easy to compare the results from other classification techniques to kNN results. They have been used in various areas such as bioinformatics, image processing and data compression, document retrieval, computer vision, multimedia database, and marketing data analysis. You can use kNN search for other machine learning algorithms, such as kNN classification, local weighted regression, missing data imputation and interpolation, and density estimation.



After Canny’s Edge Detection

Watershed Transform may be used in place of kNN search. Watershed transform computes a label matrix identifying the watershed regions of the input matrix A, which can have any dimension. The elements of L are integer values greater than or equal to 0. The elements labelled 0 do not belong to a unique watershed region. These are called watershed pixels. Once the edges are detected, our aim is to detect the finger tips. Wavelet family method is one of the techniques that may be used to detect the peaks. Wavelet analysis consists of decomposing a signal or an image into a hierarchical set of approximations and details. The levels in the hierarchy often

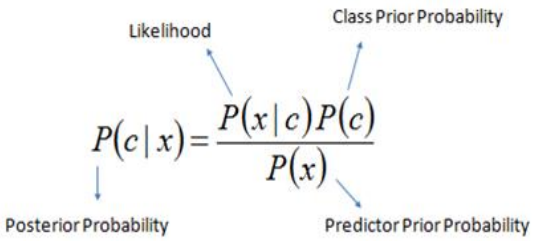
correspond to those in a dyadic scale. From the signal analyst's point of view, wavelet analysis is a decomposition of the signal on a family of analysing signals, which is usually an orthogonal function method. From an algorithmic point of view, wavelet analysis offers a harmonious compromise between decomposition and smoothing techniques. Unlike conventional techniques, wavelet decomposition produces a family of hierarchically organized decompositions. The selection of a suitable level for the hierarchy will depend on the signal and experience. Often the level is chosen based on a desired low-pass cut-off frequency. The finger tips are detected by finding the 1s at the minimum rows. The width and height of the finger is predefined.

4 Recognition

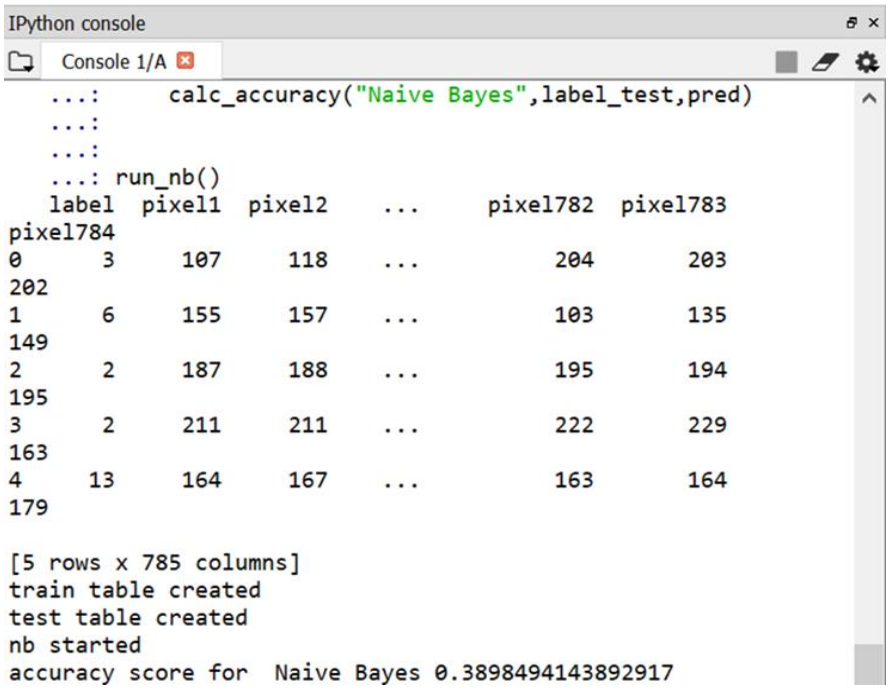
This is a list of the models we have used to process and transform the data

- **NAIVE BAYES**

  In ML, Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong independence assumptions between the features.
  Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.



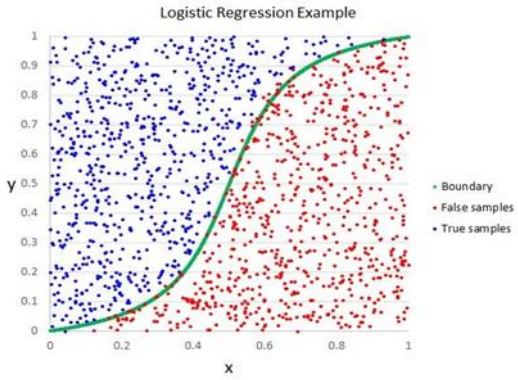
$$P(c|\mathrm{X}) = P(x_1|c)\times P(x_2|c)\times\cdots\times P(x_n|c)\times P(c)$$



- **LOGISTIC REGRESSION**

  Logistic Regression measures the relationship between one variable (what we want to predict) and the other variables (our features), by estimating probabilities using it's underlying logistic function. Basically, it finds the probability of a certain element belonging to a particular category.

  It is a form of supervised learning, and is very commonly used for classification. It has wide applications in Healthcare , Social Sciences & various ML for advanced research & analytics.

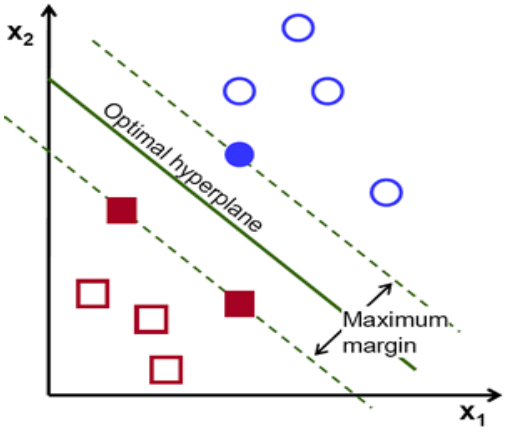



```
IPython console
Console 1/A
regression",label_test,pred)
   ...:
   ...:
   ...: run_lr()
   label  pixel1  pixel2    ...     pixel782  pixel783
pixel784
0      3     107     118    ...          204       203
202
1      6     155     157    ...          103       135
149
2      2     187     188    ...          195       194
195
3      2     211     211    ...          222       229
163
4     13     164     167    ...          163       164
179

[5 rows x 785 columns]
train table created
test table created
lr started
accuracy score for  Logistic regression 0.6820970440602342
```

- **SUPPORT VECTOR MACHINE**

  "Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges.

  SVM has a technique called the **kernel trick**. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels.

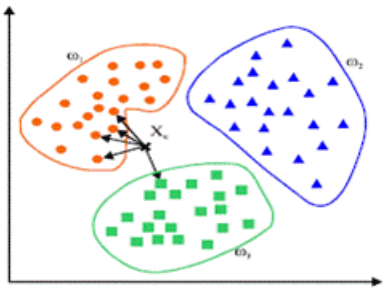



```
IPython console
Console 1/A
   ...:     calc_accuracy("SVM",label_test,pred)
   ...:
   ...:
   ...: run_svm()
   label  pixel1  pixel2    ...     pixel782  pixel783
pixel784
0      3     107     118    ...          204       203
202
1      6     155     157    ...          103       135
149
2      2     187     188    ...          195       194
195
3      2     211     211    ...          222       229
163
4     13     164     167    ...          163       164
179

[5 rows x 785 columns]
train table created
test table created
svm started
accuracy score for  SVM 0.6968767428890128
```

- **K- NEAREST NEIGHBOUR**

  KNN is a non-parametric, lazy learning algorithm. Its purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point.

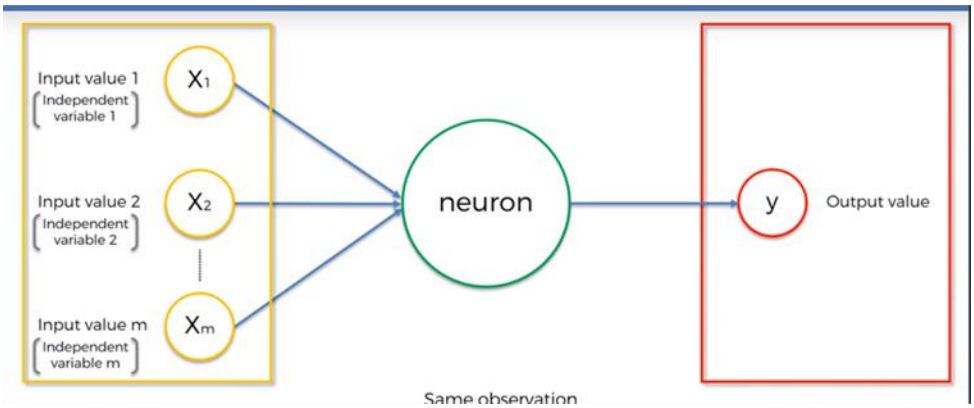
  KNN Algorithm is based on feature similarity: How closely out-of-sample features resemble our training set determines how we classify a given data point.



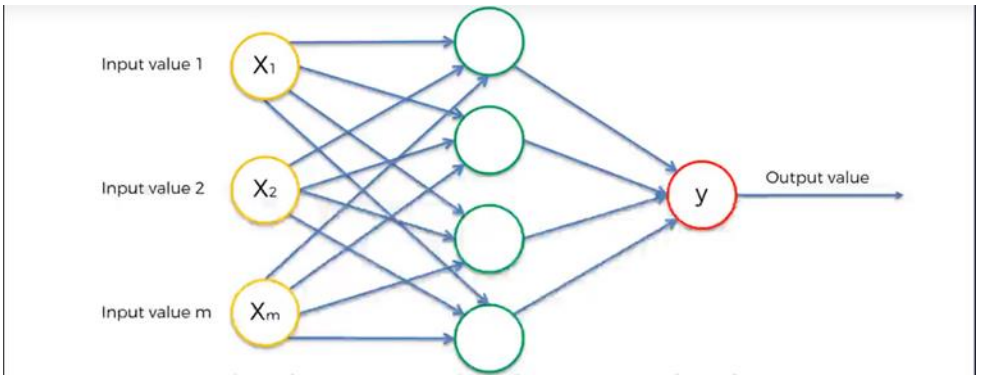
```
IPython console
Console 1/A
neighbours",label_test,pred)
   ...:
   ...:
   ...: run_knn()
   label  pixel1  pixel2    ...     pixel782  pixel783
pixel784
0      3     107     118    ...          204       203
202
1      6     155     157    ...          103       135
149
2      2     187     188    ...          195       194
195
3      2     211     211    ...          222       229
163
4     13     164     167    ...          163       164
179

[5 rows x 785 columns]
train table created
test table created
knn started
accuracy score for  K nearest neighbours 0.8039598438371445
```

- **CONVOLUTION NEURAL NETWORKS**

  CNN are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output.



  The convolution layer comprises of a set of independent filters.Parameter sharing is sharing of weights by all neurons in a particular feature map.



Modules used:

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation

Python Data Analysis Library

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Pandas is a NumFOCUS sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project, and makes it possible to donate to the project.

sklearn.model_selection.train_test_split

`sklearn.model_selection.` **`train_test_split`** (**arrays*, ***options*) [source]

Split arrays or matrices into random train and test subsets

Quick utility that wraps input validation and `next(ShuffleSplit().split(X, y))` and application to input data into a single call for splitting (and optionally subsampling) data in a oneliner.

```
#Cross validation technique (Splitting training set in 4:1 as train and validation set)
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, Y, test_size=0.2, stratify=Y, shuffle=True, random_state=42)
```

sklearn.preprocessing.OneHotEncoder

Encode categorical integer features as a one-hot numeric array.

The input to this transformer should be an array-like of integers or strings, denoting the values taken on by categorical (discrete) features. The features are encoded using a one-hot (aka 'one-of-K' or 'dummy') encoding scheme. This creates a binary column for each category and returns a sparse matrix or dense array.

By default, the encoder derives the categories based on the unique values in each feature. Alternatively, you can also specify the categories manually. The OneHotEncoder previously assumed that the input features take on values in the range [0, max(values)). This behaviour is deprecated.

This encoding is needed for feeding categorical data to many scikit-learn estimators, notably linear models and SVMs with the standard kernels.

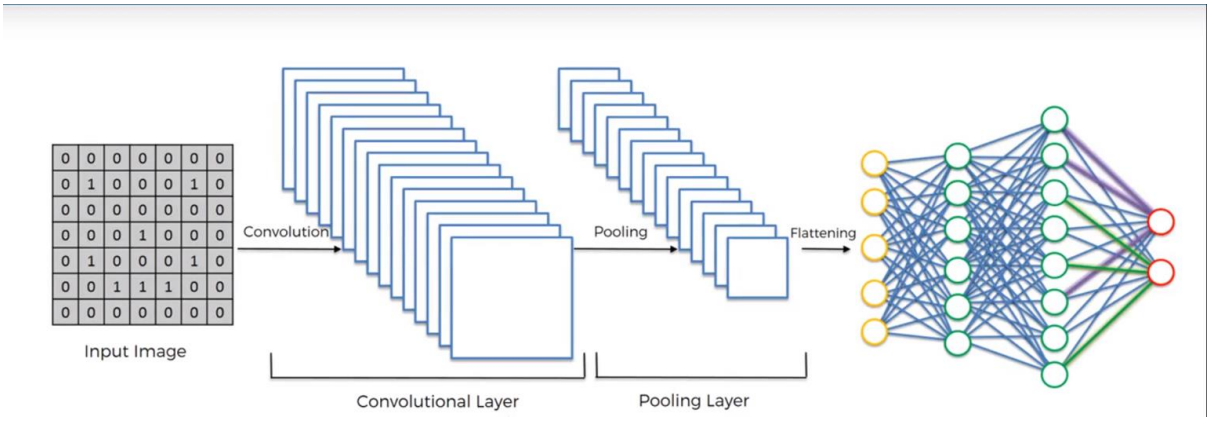
Sample	Value	Numerical	Value_Red	Value_Blue	Value_Green
1	Red	1	1	0	0
2	Green	3	0	0	1
3	Blue	2	0	1	0
4	Green	3	0	0	1

Conv2D

2D convolution layer (e.g. spatial convolution over images).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the sample axis), e.g. `input_shape=(128, 128, 3)` for 128x128 RGB pictures in `data_format="channels_last"`.



CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function.

Usage of callbacks

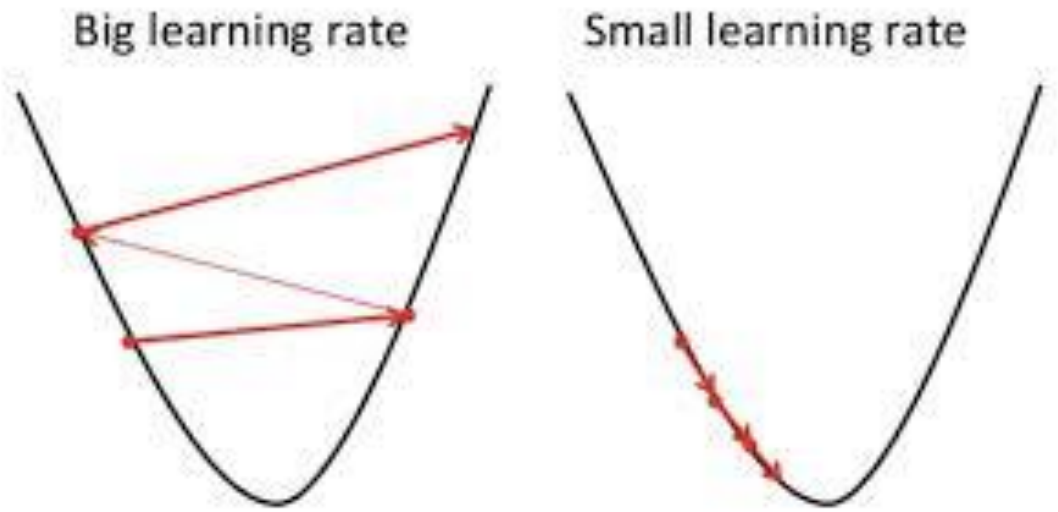
A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training. You can pass a list of callbacks (as the keyword argument `callbacks`) to the `.fit()` method of the `Sequential` or `Model` classes. The relevant methods of the callbacks will then be called at each stage of the training.

```
Epoch 00019: val_loss improved from 0.00002 to 0.00002, saving model to model.h5
Epoch 20/50
21964/21964 [==============================] - 59s 3ms/step - loss: 4.8541e-04 - acc: 0.9998 - val_loss: 2.0200e-05 - val_acc: 1.0000

Epoch 00020: ReduceLROnPlateau reducing learning rate to 1.0000000656873453e-06.

Epoch 00020: val_loss did not improve from 0.00002
Epoch 21/50
21964/21964 [==============================] - 59s 3ms/step - loss: 3.3540e-04 - acc: 1.0000 - val_loss: 1.9757e-05 - val_acc: 1.0000

Epoch 00021: val_loss did not improve from 0.00002
Epoch 22/50
21964/21964 [==============================] - 60s 3ms/step - loss: 9.0231e-04 - acc: 0.9998 - val_loss: 2.0023e-05 - val_acc: 1.0000

Epoch 00022: val_loss did not improve from 0.00002
Epoch 00022: early stopping
```

Gradient Descent

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.



Modal Summary

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 28, 28, 1)	0
conv2d_1 (Conv2D)	(None, 28, 28, 64)	640
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 64)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 7, 7, 64)	0
conv2d_3 (Conv2D)	(None, 7, 7, 64)	36928
max_pooling2d_3 (MaxPooling2	(None, 3, 3, 64)	0
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 128)	73856
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 24)	3096

```
Tconfm = {'tp':0, 'fp':0}
for n in range(df2.shape[0]):
  if Y_test.argmax(axis=-1)[n]==Tpreds[n]:
    Tconfm['tp'] +=1
  if Y_test.argmax(axis=-1)[n]!=Tpreds[n]:
    Tconfm['fp'] +=1
print(Tconfm)
```

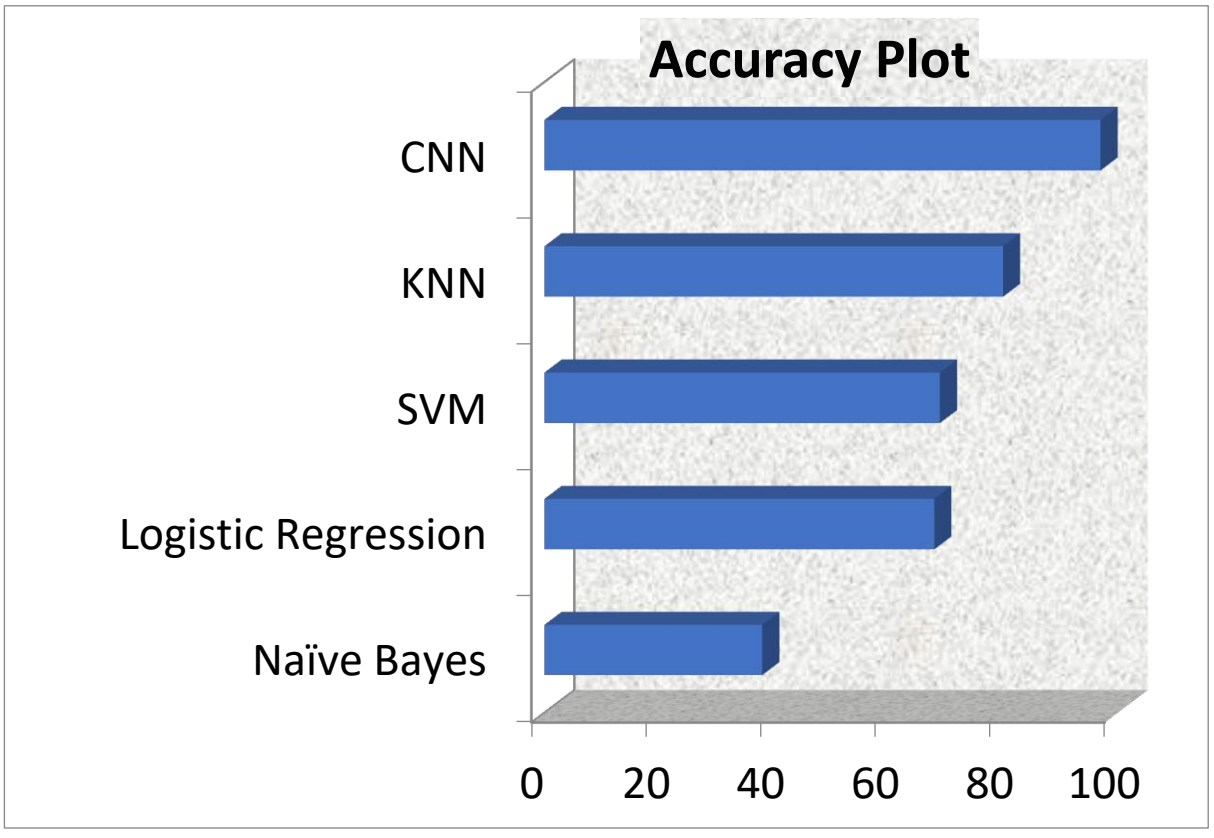
```
{'tp': 6844, 'fp': 328}
```

```
#Test accuracy
score, acc = model.evaluate(X_test, Y_test, batch_size=32)
#print('Test score:', score)
print('Test accuracy:', acc)
```

```
7172/7172 [==============================] - 5s 762us/step
Test accuracy: 0.9542665923034022
```

Result

This result was achieved by following simple steps without the need of any gloves or any specifically coloured backgrounds. This work may be extended to recognizing all the characters of the standard keyboard by using two hand gestures. The recognized gesture may be used to generate speech as well as text to make the software more interactive. This is an initiative in making the less fortunate people more independent in their life. Much is needed to be done for their upliftment and the betterment of the society as a whole.



SOURCE CODE

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
#numpy.random.seed(0)

#Sample data output
df=pd.read_csv("sign_mnist_train.csv")
df.head()

#Image Dimensions
x=df.iloc[0,1:].values
x=x.reshape(28,28)
x.shape

#Sample image output
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
imgplot = plt.imshow(x)

#Generating array of images
def create(df):
    import cv2
    w = np.empty((0,28,28))
    x=[df.iloc[i,1:].values.reshape(28,28) for i in range(df.shape[0])]
    n = 0     #Progress var
    for _ in x:
        w = np.append(w, [_], axis=0)
        n+= 1 #Progress update
        if n % 1000 == 0:
            print(n)
    del cv2
    print('!')
    return w

#Calling create()
x = create(df)

#Generating label var
y=df.iloc[:,0]
labels = pd.get_dummies(y).values

#Splitting training-set into train and validate sets
from sklearn.model_selection import train_test_split
x = x.reshape(df.shape[0], 28, 28, 1)
X_train, X_test, y_train, y_test = train_test_split(x, labels, test_size=0.20,
                                                    random_state=42)

```

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense  #, Dropout


learning_rate=0.01
# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Conv2D(32, (3, 3), input_shape = (28, 28, 1),
                      activation = 'relu'))

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a second convolutional layer
classifier.add(Conv2D(64, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a third convolutional layer
classifier.add(Conv2D(128, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
#classifier.add(Dense(units = 256, activation = 'relu'))
#classifier.add(Dense(units = 256, activation = 'relu'))
#classifier.add(Dense(units = 512, activation = 'relu'))
#classifier.add(Dense(units = 512, activation = 'relu'))
classifier.add(Dense(units = 24, activation = 'softmax'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
                   metrics = ['accuracy'])

from keras.preprocessing.image import ImageDataGenerator
import keras
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)

```

```
# compute quantities required for featurewise normalization
# (std, mean, and principal components if ZCA whitening is applied)
datagen.fit(X_train)

callback = [keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0,
                                          patience=2, verbose=0, mode='auto')]


# fits the model on batches with real-time data augmentation:
classifier.fit_generator(datagen.flow(X_train, y_train, batch_size=35),
                    steps_per_epoch=len(X_train)/32, epochs=15,
                    validation_data=(X_test, y_test), verbose=1)

#Generating test-set var
df_test = pd.read_csv("sign_mnist_test.csv")
x_final = create(df_test)
x_final = x_final.reshape(df_test.shape[0], 28, 28, 1)
y_final=df_test.iloc[:,0]
y_final = pd.get_dummies(y_final).values

#Test accuracy
score, acc = classifier.evaluate(x_final, y_final,
                             batch_size=32)
print('Test score:', score)
print('Test accuracy:', acc)



#Hyper parameter tuning
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

def create_model(optimizer='adam', learn_rate=0.01, momentum=0,
                 activation='softmax', neurons=2, init_mode='uniform',
                 dropout_rate=0.0, weight_constraint=0):
    # Initialising the CNN
    classifier = Sequential()

    # Step 1 - Convolution
    classifier.add(Conv2D(32, (3, 3), input_shape = (28, 28, 1),
                          activation = 'relu'))

    # Step 2 - Pooling
    classifier.add(MaxPooling2D(pool_size = (2, 2)))

    # Adding a second convolutional layer
    classifier.add(Conv2D(64, (3, 3), activation = 'relu'))
    model.add(Dropout(dropout_rate))
    classifier.add(MaxPooling2D(pool_size = (2, 2)))

```

```
    # Step 3 - Flattening
    classifier.add(Flatten())

    # Step 4 - Full connection
    classifier.add(Dense(units = neurons, kernel_initializer=init_mode,
                         activation = 'relu',
                         kernel_constraint=maxnorm(weight_constraint)))
    classifier.add(Dense(units = 24, activation = activation))

    # Compiling the CNN
    classifier.compile(optimizer = optimizer, loss = 'categorical_crossentropy'
                       , metrics = ['accuracy'])
    return classifier

#Tuning Batch and epochs
# create model
model = KerasClassifier(build_fn=create_model, epochs=15, batch_size=35,
                        verbose=0)
batch_size = [40, 32, 35]
epochs=[25,15,20]
# define the grid search parameters
param_grid = dict(epochs=epochs,batch_size=batch_size)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
grid_result = grid.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_)
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

#Tuning optimizers
# create model
model = KerasClassifier(build_fn=create_model, epochs=20, batch_size=30,
                        verbose=0)
# define the grid search parameters
optimizers = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax',]
param_grid = dict(optimizer=optimizers)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
grid_result = grid.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_)
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
#Tuning learning rate and momemtum
# create model
model = KerasClassifier(build_fn=create_model, epochs=35, batch_size=15,
                        verbose=0)
# define the grid search parameters
learn_rate = [0.001, 0.01, 0.1, 0.2, 0.3]
momentum = [0.0, 0.2, 0.4, 0.6, 0.8, 0.9]
param_grid = dict(learn_rate=learn_rate, momentum=momentum)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
grid_result = grid.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_)
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

#Tuning network weight initialization
# create model
model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10,
                        verbose=0)
# define the grid search parameters
init_mode = ['uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal',
             'glorot_uniform', 'he_normal', 'he_uniform']
param_grid = dict(init_mode=init_mode)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
grid_result = grid.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_)
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

#Tuning neural activation function
model = KerasClassifier(build_fn=create_model, epochs=20, batch_size=32,
                        verbose=0)
activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid',
              'hard_sigmoid', 'linear']
param_grid = dict(activation=activation)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
grid_result = grid.fit(X_tain, y_train)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_)
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

```

```
#Tuning dropout regularization
# create model
model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# define the grid search parameters
weight_constraint = [1, 2, 3, 4, 5]
dropout_rate = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
param_grid = dict(dropout_rate=dropout_rate, weight_constraint=weight_constraint)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
grid_result = grid.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))

#Tuning number of neurons in hidden layer
# create model
model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# define the grid search parameters
neurons = [1, 5, 10, 15, 20, 25, 30]
param_grid = dict(neurons=neurons)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1)
grid_result = grid.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

Output screenshots

```
[ ]  Tconfm = {'tp':0, 'fp':0}
     for n in range(df2.shape[0]):
       if Y_test.argmax(axis=-1)[n]==Tpreds[n]:
         Tconfm['tp'] +=1
       if Y_test.argmax(axis=-1)[n]!=Tpreds[n]:
         Tconfm['fp'] +=1
     print(Tconfm)
```

```
{'tp': 6966, 'fp': 206}
```

```
[ ]
     #Test accuracy
     score, acc = model.evaluate(X_test, Y_test, batch_size=32)
     print('Test score:', score)
     print('Test accuracy:', acc)
```

```
7172/7172 [==============================] - 1s 136us/step
Test score: 0.14279215571731874
Test accuracy: 0.9712771890686002
```

Applications

- Apps that can facilitate direct feed conversion of sign language to letters/ sounds. (Human Computer Interface).
- Emergency First Responders
- Robot control
- Sign language in terrorism

Conclusion and future enhancement

Sign Language Recognition can be quite useful and important, if it can be merged with various facets of society.

While the current solution can only be used to recognize static hand gestures, there are many ways this project can be taken forward.

We can improve this to include Live Feed Sign Language recognition, that can be made available for everyday use.

We can also include various other forms of sign languages, and can also involve other hand gestures, that can be informative of their nature.

From helping aid with a person’s everyday activity, to enhancing security and helping save lives, Sign Language Recognition has quite the future