# SIGN LANGUAGE RECOGNITION

*This project report is submitted to*

*Silicon Institute of Technology, Bhubaneswar*

*in partial fulfillment of the requirements for the award of the degree of*

**Bachelor of Technology**

**in**

**Computer Science and Engineering**

*Submitted by*

**ASHISH SUREKA** (180310424)

*Under the Esteemed Supervision of*

**Prof. Pradyumna Kumar Tripathy**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
**SILICON INSTITUTE OF TECHNOLOGY**
**SILICON HILLS, BHUBANESWAR – 751024, ODISHA, INDIA**
**May, 2021**

# ACKNOWLEDGEMENTS

We would like to express our sincere gratitude towards our advisors and mentors throughout this project. Without their support, advice and guidance, this project would not have been ever possible. We are highly grateful to our esteemed faculty Mr. Pradyumna Kumar Tripathy for he provided us with essential information and resources required for this project. Also, We like to thank Mr. Jiten Kumar Mohanty for his precise guidance and regular supervision. Throughout this development, constant help and patient advice of our mentors kept us building and improving this project. We also feel appreciative towards our family and friends for their encouragement and motivation for this project and our institution for giving us an opportunity to explore in this field.

Thank you everyone.

**ASHISH SUREKA**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
**SILICON INSTITUTE OF TECHNOLOGY**
**BHUBANESWAR – 751024**

# ABSTRACT

Conversing to a person with hearing disability is always a major challenge. Sign language has indelibly become the ultimate panacea and is a very powerful tool for individuals with hearing and speech disability to communicate their feelings and opinions to the world. It makes the integration process between them and others smooth and less complex. However, the invention of sign language alone, is not enough. There are many strings attached to this boon. The sign gestures often get mixed and confused for someone who has never learnt it or knows it in a different language. However, this communication gap which has existed for years can now be narrowed with the introduction of various techniques to automate the detection of sign gestures. Here, we introduce a Sign Language recognition using American Sign Language.

For this, the user must be able to capture images of the hand gesture using web camera and the system shall predict and display the name of the captured image. We have manually created a dataset from web camera using OpenCV. Each alphabet contains nearly 2600 images making the size of the dataset to 70000. Each image in the dataset went through background subtraction method that include a series of processing steps which include various Computer vision techniques such as the conversion to grayscale, thresholding, Gaussian smoothing and eroding operation. And the region of interest which, in our case is the hand gesture is segmented. The features extracted are the binary pixels (black & white) of the images. We make use of Convolutional Neural Network (CNN) for training and to classify the images. We are able to recognise all the American Sign gesture alphabets with high accuracy. Our model has achieved a remarkable accuracy of above 99.9%.

***Keywords:*** *Sign Language, ASL, Hearing disability, Convolutional Neural Network (CNN), Gesture recognition, Sign language recognition, Background subtraction technique.*

# LIST OF ABBREVIATIONS

| Abbreviation | Description |
|---|---|
| ASL | American Sign Language |
| CNN | Convolutional Neural Network |
| SLR | Sign Language Recognition |

# LIST OF FIGURES

# CONTENTS

# CHAPTER 1

# INTRODUCTION

## 1.1 BACKGROUND

The world is hardly live without communication, no matter whether it is in the form of texture, voice or visual expression. The communication among the people with impaired hearing and speech is carried by text and visual expressions. Sign language is a form of communication used by them as a means of non-verbal communication to express their thoughts and emotions. But normal people find it extremely difficult to understand, hence trained person are needed in order for communication to take place between a normal people and a hearing-impaired people. The goal of this project was to build a neural network using deep learning technique that will be able to classify the letter of the American Sign Language (ASL) alphabet when shown using camera and show the result to the normal person. So that fruitful communication can take place between them Such a software would also lower the barrier for many deaf and mute individuals to be able to better communicate with others in day-to-day interactions.

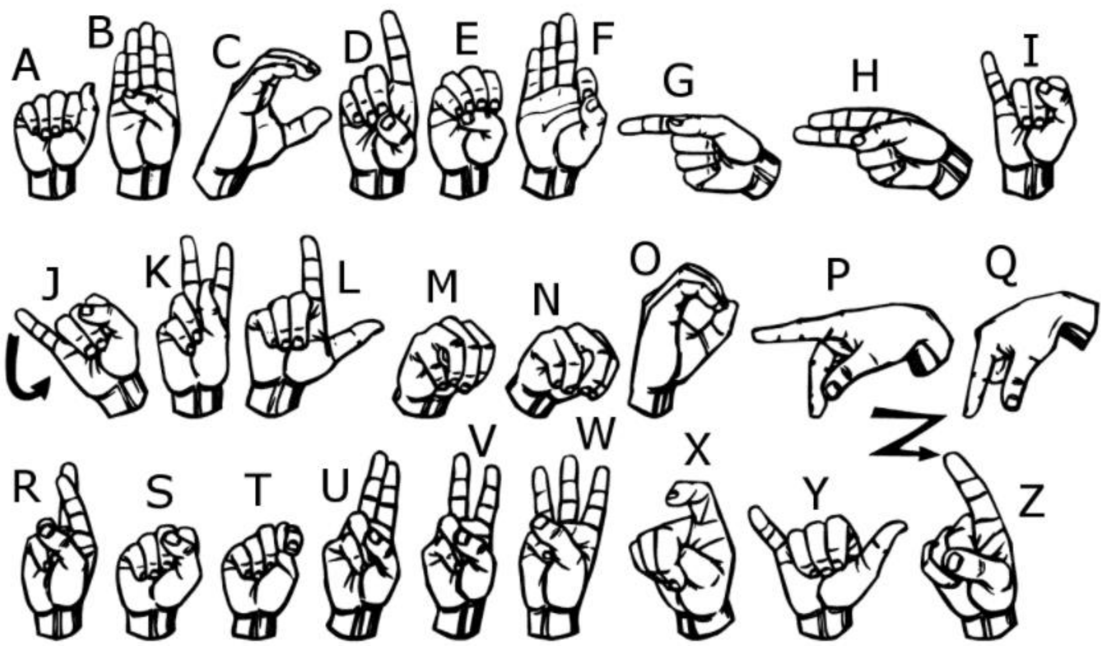



Figure 1.1: American Sign Language

## 1.2 PROBLEM STATEMENT

To Design a real time software system that will be able to recognize the sign language performed by movement of hand using deep learning technique Convolutional Neural Network. This project aims to predict the ‘alphabets’ of the American Sign Language.

## 1.3 OBJECTIVE AND MOTIVATION

Sign language is learned by deaf and dumb, and usually it is not known to normal people, so it becomes a challenge for communication between a normal and hearing-impaired person. To bridge the gap between hearing impaired and normal people to make the communication easier such software's can help. Sign language recognition (SLR) system takes an input expression from the hearing-impaired person gives output to the normal person in the form text. The objective of this project is to identify the sign expression when shown in camera and show the result in the form of text so that the communication between a normal and hearing-impaired person can take place easily.

## 1.4 PROPOSED METHOD

### Convolutional Neural Network

CNN is used here because of its popularity in image classification. It mainly consists of 3 layers: Convolutional layer, pooling layer and fully connected layer. The main advantage of this method is that is automatically deconstruct an image and detects the important features without any human supervision. For example, given many pictures of cat and dogs it learns distinctive features for each class by itself. It is also computationally efficient because it uses special convolutional and pooling operation and performs parameter sharing which enables the CNN models to run effectively. We will be training a CNN having 27 classes and is 7 layers deep.

### 1.4.1 Method to Improve the Performance of Discovery

The model trained on the dataset directly captured from camera was performing well in the test dataset (accuracy close to 99%) but during the real time prediction the model was not able to classify the examples correctly. This was due to the reason that the skin tone and background was very much different in the training and real time data.

Therefore, background subtraction technique was applied which first finds the difference between the current frame and the reference frame (called background frame) and then use thresholding (In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally 255)) followed by Gaussian smoothing (help to reduce noise) and erosion (help in removing noise and isolation of individual elements and joining disparate elements in an image). Using this technique greatly increase the accuracy of prediction in real time.

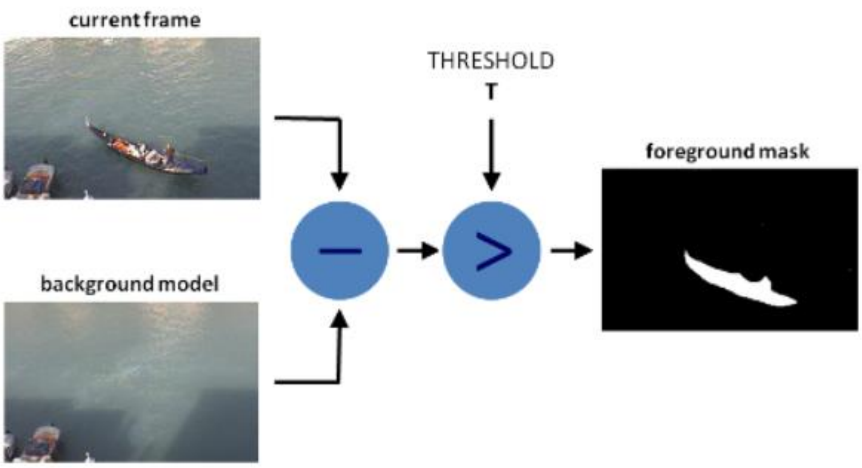



Figure 1.2: Background subtraction technique

### 1.4.2 Evaluation Parameter - PRECISION, RECALL, F1-SCORE

**Precision:** Precision quantifies the number of positive class predictions that actually belong to the positive class.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$

Figure 1.3: Precision Formula

**Recall:** Recall quantifies the number of positive class predictions made out of all positive examples in the dataset.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

Figure 1.4: Recall Formula

**<u>F1-Score:</u>** F1-Score provides a single score that balances both the concerns of precision and recall in one number.

$$\text{F1} = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

Figure 1.5: F1-Score Formula

## 1.5 PROJECT ORGANIZATION

**Chapter 1** titled, *"Introduction",* presents the general overview of the sign language and the method we will be using to classify the sign in the sign language. It also contains the objective and motivation behind the work.

**Chapter 2** titled, *"Literature review"* presents the renowned works earlier performed by well-known personalities in the area of image processing. This chapter furthermore contains the positives and negatives of the existing works done.

**Chapter 3** named as, *"Methodology",* presents the step-by-step process followed to accomplish our aim.

**Chapter 4** titled, *"Result, Conclusion and Recommendation",* presents the Result of our project, conclusion and the future scope and improvements

## SUMMARY

In this chapter we discussed about:

- Introduction, motive and objective of our project.
- Brief overview of CNN and background removal technique.
- Parameters for evaluating the results like F1-Score, Precision and Recall

# CHAPTER 2

## LITERATURE REVIEW

**Convolutional Neural Networks based Sign Language Recognition [4]:** This paper aims to recognize the hand sign through camera and predict the alphabet corresponding to a particular hand sign with the help of Convolutional Neural Network (CNN) in real time. The system which implements predictive model that gives result the overall 91% accuracy in real-time recognition.

**Using Deep Convolutional Networks for Gesture Recognition in American Sign Language [5]:** The use of Deep Convolutional Networks to classify images of American Sign Language. This paper has also mentioned Background Subtraction Technique for removing the background from each of the images. They achieved an accuracy of 82.5% on the alphabet gestures and 97% validation set accuracy on digit gestures.

**Hand Gesture Recognition for Sign Language Recognition: A Review [6]:** Authors presented various approaches of sign language and hand gesture recognition. That were proposed in the past by various researchers for expressing emotions and thoughts to another person by sign language. There are broadly three approaches mentioned in this paper which are Approach based on Vision, Approach based on Instrumental Glove and Approach based on Colored Marker.

**Hand Gesture Recognition Using a Convolutional Neural Network [7]:** In this paper the approach towards hand sign recognition is based on Convolutional Neural Network (CNN) and Microsoft Kinect SDK. Microsoft Kinect Sensor is used for capturing the raw skeletal joint motion data. Authors presented the comparison of the proposed approach to previous work on hand-crafted features.

# CHAPTER 3

# METHODOLOGY

## 3.1 DATASET PREPARATION

The Dataset was prepared using the web camera of laptop and python library OpenCV. There is total 27 class in our dataset which corresponds to alphabet from A-Z of ASL and an extra class “Nothing” is added which is shown when no sign is seen in front of the camera. For each class around 1300 images was captured using right hand. After capturing the images for each class. The images of each class were flipped horizontally in order to create the dataset for left hand too. Thus, each class now has 2600 images making the size of the dataset to around 70000 images. After Capturing, each image has undergone background subtraction technique which first finds the difference between the current frame and the reference frame (called background frame) and then use thresholding (In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally 255)) followed by Gaussian smoothing (help to reduce noise) and erosion (help in removing noise and isolation of individual elements and joining disparate elements in an image). The image was then resized into 50X50 pixel and saved.

After performing the above process, we get a folder gesture which contain 27 sub-folders A-Z and “Nothing” each folder containing around 2600 images. Then a python code was written which help to convert the folder into a numpy array having 2501 columns and nearly 70000 rows. Each row corresponds to a single image in which first 2500 column were 50X50 pixel array flattened to 2500 and the last column contain the label for each image. The array was then stored as .npy file.

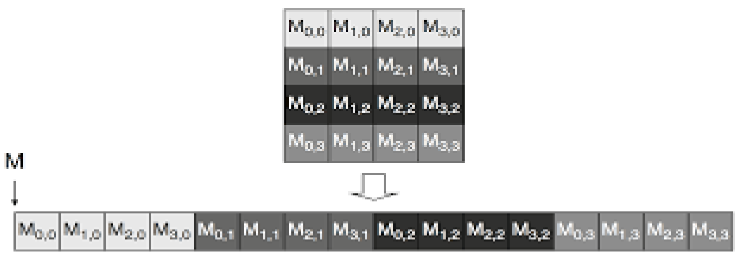



Figure 3.1: Flattening 2D array to 1D visualization


Figure 3.2: Samples from Dataset for each class

## 3.2 CNN MODEL

The above classification is a difficult task but deep learning algorithm CNN has proved to be beneficial in extracting complicated features. Convolutional Neural Network uses the property of convolution, mainly devised for analyzing visual imagery. It consists of one input layer and one output layer and numerous hidden layers in between. The hidden layer consists of convolutional layers that compute the dot product between the weights and regions of the input image. This is usually followed by ReLU (an activation function) and

Max Pooling (for down sampling and reduce the output volume). The advantage of using CNN over ordinary ANN is the reduction of number of parameters to train, feature sharing etc. Although high computation power is required for training it is much faster than ordinary ANN. Fig 3.3 depicts the actual architecture of our CNN model and Fig 3.4 describes the model summary.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 50, 50, 32)        320
_________________________________________________________________
batch_normalization_4 (Batch (None, 50, 50, 32)        128
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 25, 25, 64)        18496
_________________________________________________________________
dropout_5 (Dropout)          (None, 25, 25, 64)        0
_________________________________________________________________
batch_normalization_5 (Batch (None, 25, 25, 64)        256
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 13, 13, 128)       73856
_________________________________________________________________
dropout_6 (Dropout)          (None, 13, 13, 128)       0
_________________________________________________________________
batch_normalization_6 (Batch (None, 13, 13, 128)       512
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 7, 7, 256)         295168
_________________________________________________________________
dropout_7 (Dropout)          (None, 7, 7, 256)         0
_________________________________________________________________
batch_normalization_7 (Batch (None, 7, 7, 256)         1024
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               1048832
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 27)                3483
=================================================================
Total params: 1,474,971
Trainable params: 1,474,011
Non-trainable params: 960
_________________________________________________________________
```

Figure 3.3: Model Summary

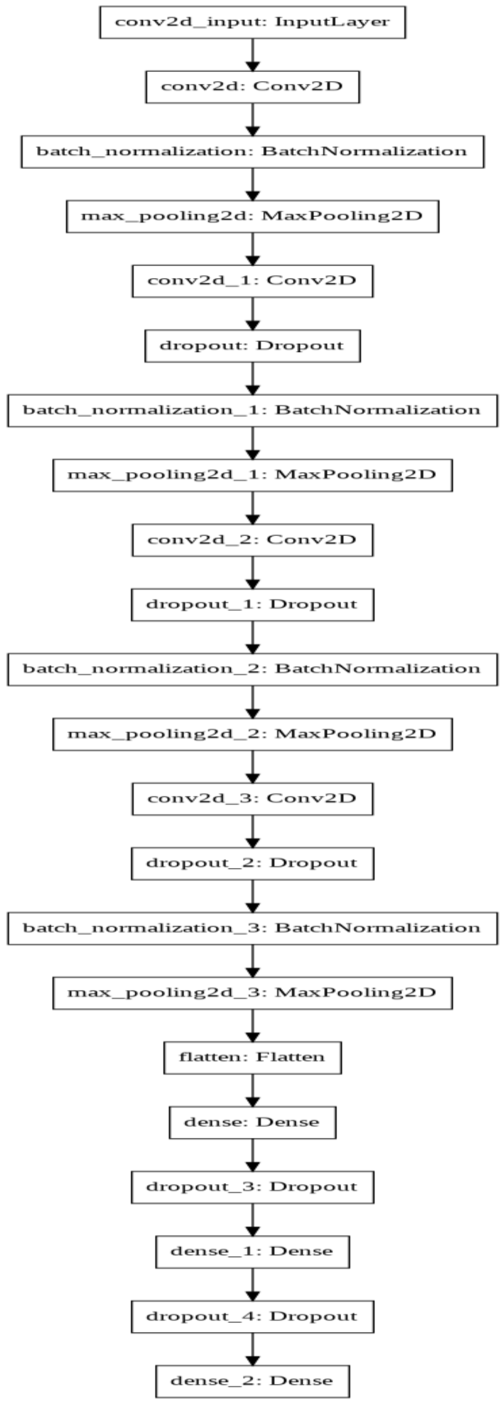



Figure 3.4:  Model Architecture

## 3.3 Training and saving the model

The dataset which consists of nearly 70000 images is split into training, testing and validation test in the ratio of 0.83:0.1:0.07.  The training data then has gone through data augmentation step (Data augmentation is a technique that help to create random variation in the training dataset like randomly rotating image to some angle, zooming in, shift image vertically or horizontally so that it generalizes well in the real-life data.). Then we are feeding the result of the data augmentation to our CNN model (optimizer – ADAM optimizer and metrics- Cross Entropy Metrics) with a mini-batch size (batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters) of 256 i.e., each training set is split into nearly 228 mini-batches and feed to the model. The process is repeated for 50 epochs (The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset) .

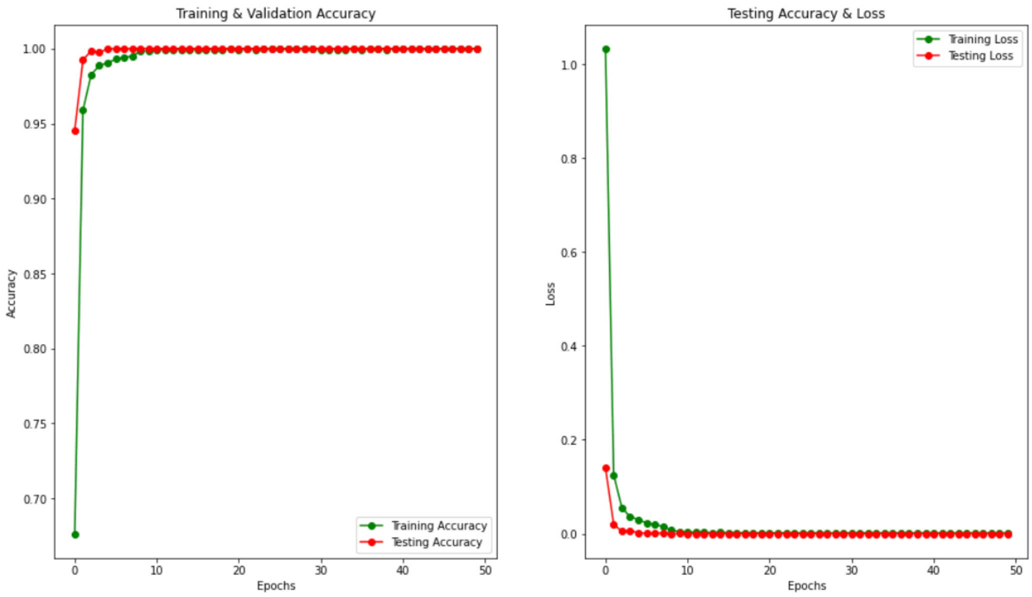



Figure 3.5 variation of losses and accuracy with epochs

The model was saved as a h5 format file, and this was called during real time testing. For real time implementation, each video frame of the camera was processed and segmented based on background removal technique, finally passed through the called trained model for its classification.

## SUMMARY

In this chapter we discussed about:

- The preparation of Dataset.
- Designing the structure of the model.
- Step by step procedure followed to train the model and saving it.

# CHAPTER 4

## RESULT, CONCLUSION AND RECOMMENDATION

### 4.1 Result

The training accuracy came out to be 99.98% and when the trained model was used to classify the images in the test dataset the test accuracy came out to 100% i.e., the model was performing quite well and was classifying each image correctly.
The precision, Recall and F1-score for each class came out to be 1. Each class in the test contain at least 230 examples making the size of test dataset around 7k.
The model was also performing very well when used for prediction in real time.

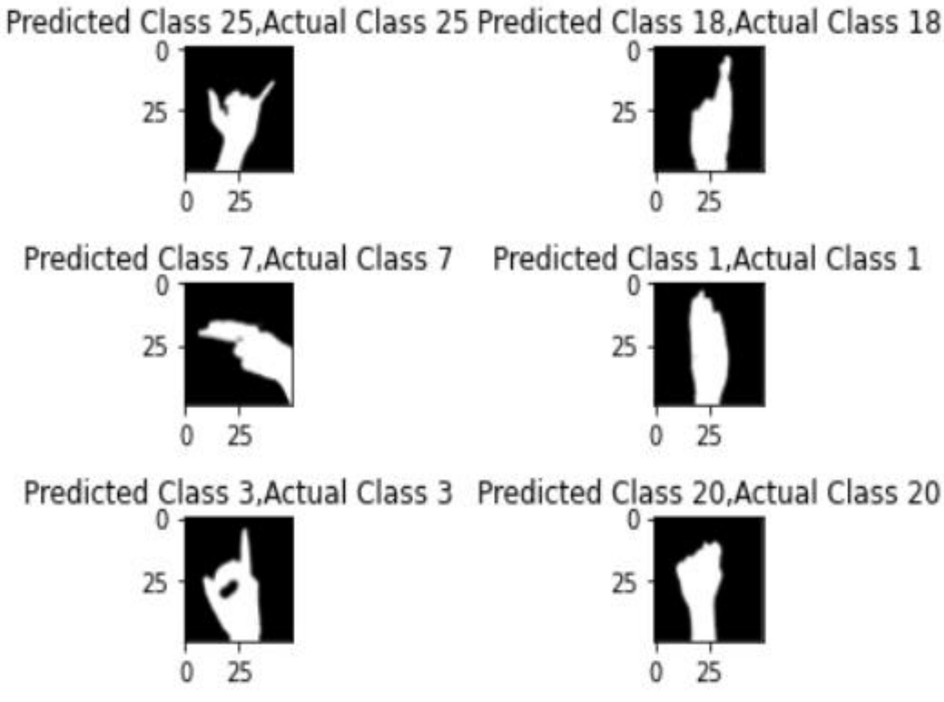



Figure 4.1 predicted class vs Actual class

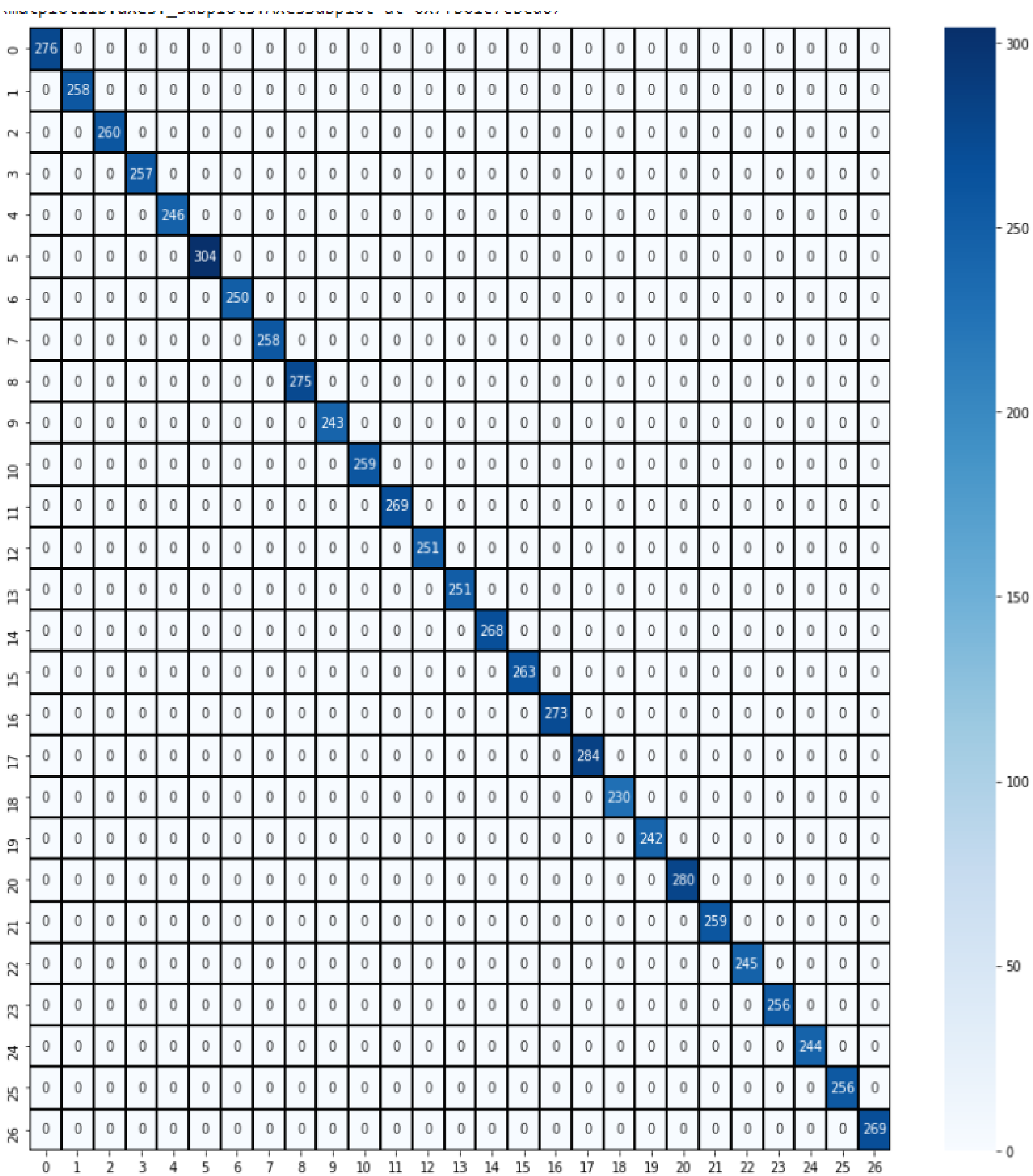


Figure 4.2: Confusion Matrix

## 4.2 CONCLUSION

The aim of the project was to predict the sign of the American Sign Language. Our above work shows that we were successfully in getting a high accuracy in the test dataset as well as prediction during the real time.

There are some limitations in our approach:

- The project can only predict alphabet A-Z of the American sign Language.

- Only the hand must be positioned in the Region of Interest (ROI) of the frame and nothing else should come in the ROI
- The background must be solid.
- The camera must be in the static position.

## 4.3 FUTURE SCOPE AND IMPROVEMENTS

- We can include digits (1-10) and other gestures like Hello, Hmm, Yes, No, I love you, Thumb up etc. in our model.
- We can use transfer learning technique on AlexNet, Inception Network which are trained on millions of images and provide accuracy close to human level performance to train our model.
- We can also improve the technique of hand detection by using more advance techniques.
- We can develop a model for ISL word and sentence level recognition. This will require a system that can detect changes with respect to the temporal space.
- We can develop a complete product that will help the speech and hearing-impaired people, and thereby reduce the communication gap.

## SUMMARY

In this chapter we discussed about:

- Outcome and accuracy of the trained model
- Future improvements that can be made.

# REFERENCES

[1] Singha, J. and Das, K. “Hand Gesture Recognition Based on Karhunen-Loeve Transform”, Mobile and Embedded 232 Technology International Conference (MECON), January 17-18, 2013, India. 365-371.

[2] D. Aryanie, Y. Heryadi. American Sign Language-Based Finger-spelling Recognition using k-Nearest Neighbors Classifier. 3rd International Conference on Information and Communication Technology (2015) 533-536. [

3] R. Sharma et al. Recognition of Single Handed Sign Language Gestures using Contour Tracing descriptor. Proceedings of the World Congress on Engineering 2013 Vol. II, WCE 2013, July 3 - 5, 2013, London, U.K.

[4] Karthick Arya, Jayesh Kudase, “Convolutional Neural Networks based Sign Language Recognition”, International Journal of Innovative Research in Computer and Communication Engineering, Volume 5, Issue 10, October 2017.

[5] Vivek and N. Dianna Radpour, “Using Deep Convolutional Networks for Gesture Recognition in American Sign Language”, Department of Computer Science, Department of Linguistics, State University of New York at Buffalo, 2017.

[6] Pratibha Pandey and Vinay Jain, “International Journal of Science, Engineering and Technology Research (IJSETR), Volume 4, Issue 3, March 2015.

[7] Eirini Mathe, Alexandros Mitsou, Evaggeles Spyrou and Phives Mylonas, “Hand Gesture Recognition Using a Convolutional Neural Network”, 13th International workshop on Semantic and Social Media Adaptation and Personalization (SMAP), 2018.

[8] Y.F. Admasu, and K. Raimond, Ethiopian Sign Language Recognition Using Artificial Neural Network. 10th International Conference on Intelligent Systems Design and Applications, 2010. 995-1000. [

9] J. Atwood, M. Eicholtz, and J. Farrell. American Sign Language Recognition System. Artificial Intelligence and Machine Learning for Engineering Design. Dept. of Mechanical Engineering, Carnegie Mellon University, 2012.

# APPENDIX – A

## SNAPSHOT OF SOURCE CODE

Source Code Link: click here

### Importing Necessary Libraries

```python
import pandas as pd
import numpy as np
import io
```

```python
import matplotlib.pyplot as plt
import plotly.express as px
import random
import keras
from keras.models import Sequential ,save_model, load_model
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout , BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from keras.callbacks import ReduceLROnPlateau
```

```python
import os
import cv2
from sklearn.model_selection import train_test_split
```

### Mounting Google Drive

```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

### Code For Creating Dataset for each alphabet from A-Z

### Code For Creating Dataset for each alphabet from A-Z

```python
import cv2
import os
import numpy as np

num_frames=0
cap = cv2.VideoCapture(0)
img_counter=1
num_frames=0
while(1):
    ret,image = cap.read()
    image = cv2.flip(image, 1)
    if not ret:
        break
    k = cv2.waitKey(5) & 0xFF
    if k== ord('c'):
        num_frames=0
    if num_frames<=30:
        background_image = image
        num_frames+=1
    if num_frames>30:
#         cv2.imshow('background',background_image)
        current_image= image
        diff = cv2.absdiff(background_image,current_image)
        mask = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
        th, mask_thresh = cv2.threshold(mask, 10, 255, cv2.THRESH_BINARY)
        mask_blur = cv2.GaussianBlur(mask_thresh, (3, 3), 10)
        mask_erosion = cv2.erode(mask_blur, np.ones((5,5), dtype=np.uint8), iterations=1)
        mask_erosion = mask_erosion[150:400,150:400]
        cv2.imshow('final',mask_erosion)
    if k==ord('z'):
        img_name = "gestures/J/{}.png".format(img_counter)
        final = cv2.resize(mask_erosion, (50, 50),interpolation = cv2.INTER_AREA)
        cv2.imwrite(img_name, final)
        img_counter+=1
    if img_counter>1301:
        break
    cv2.rectangle(image, (150, 150), (400, 400), (255, 255,0), 3)
    cv2.imshow('frame',image)
    if k == 27:
        break

cv2.destroyAllWindows()
cap.release()
```

### Code for Flipping each image from A-Z

```python
import cv2, os

def flip_images():
    gest_folder = r"E:\skill lab\gestures"
    images_labels = []
    images = []
    labels = []

    for g_id in os.listdir(gest_folder):
        print(g_id)
        for i in range(1301):
            path = gest_folder+"/"+g_id+"/"+str(i+1)+".png"
            new_path = gest_folder+"/"+g_id+"/"+str(i+1+1301)+".png"
            print(path)
            img = cv2.imread(path, 0)
            img = cv2.flip(img, 1)
            cv2.imwrite(new_path, img)
flip_images()
```

Code for creating a NPY file which store each image in a row and its corresponding label making the dimesion of each row to 2501 (50X50 Image resized to 2500 + label) and the column corresponds to the number of example in the dataset which is around 70K.

```
def create_dataset(img_folder):
    img_data_array=[]
    class_name=[]
    for dir1 in os.listdir(img_folder):
        print()
        print(dir1,end=' ')
        i=0
        for file in os.listdir(os.path.join(img_folder, dir1)):
            if i%1000==0:
                print(i,end=" ")
            image_path= os.path.join(img_folder, dir1,  file)
            image= cv2.imread(image_path)
            image=np.array(image,dtype=np.uint8)
            img_data_array.append(image)
            class_name.append(dir1)
            i+=1
    return img_data_array, class_name
# extract the image array and class name
img_data, class_name =create_dataset(r"E:\skill lab\gestures")
```

Mapping each alphabet to a number for training purpose i.e A-N is mapped to 0-13 , Nothing to 14 and N-Z to 15-26

```
img_data = np.array(img_data)
class_name=np.array(class_name)
target_dict={k: v for v, k in enumerate(np.sort(np.unique(class_name)))}
class_name=  [target_dict[class_name[i]] for i in range(len(class_name))]
class_name = np.array(class_name)
display(target_dict)
class_name = class_name.reshape(class_name.shape[0],1)
img_data = img_data.reshape(img_data.shape[0],-1)
print(img_data.shape)
print(class_name.shape)
dataset = np.concatenate((img_data,class_name),axis=1)
dataset.shape
dataset = np.asarray(dataset,dtype = np.uint8)
```

Saving the file in drive

```
# np.save(r"E:\New folder\gestures\dataset.npy", dataset)
```

Loading the file from google drive

```
dataset = np.load("/content/drive/MyDrive/DATASET/My_dataset/dataset1.npy")
```

## Code for producing image of each alphabet with its label in dataset

```
def get_pred_text(pred_class):
    dictionary={0:'A', 1:'B', 2:'C', 3:'D', 4:'E', 5:'F', 6:'G', 7:'H', 8:'I', 9:'J', 10:'K', 11:'L', 12:'M',
                13:'N', 14:'Nothing', 15:'O',16:'P', 17:'Q', 18:'R', 19:'S', 20:'T',21:'U', 22:'V', 23:'W',
                24:'X', 25:'Y', 26:'Z'}
    return dictionary[pred_class]
```

```
import matplotlib.gridspec as gridspec
plt.figure(figsize = (8,8))
gs1 = gridspec.GridSpec(4, 4)
gs1.update(wspace=0.025, hspace=0.1)
i = 0
l=0
for _ in range(27):
    plt.subplot(6,5,i+1)
    k = np.asarray(dataset[l,:2500].reshape(50,50),dtype='float32')
    k = cv2.resize(k, (200, 200))
    fig =plt.imshow(k, cmap="gray")
    fig.axes.get_xaxis().set_visible(False)
    fig.axes.get_yaxis().set_visible(False)
    # plt.text(1, 1, get_pred_text(i),fontsize=15)
    plt.title("{}".format(get_pred_text(i)))


    l += 2605
    i += 1
plt.tight_layout()
plt.show()
```

Randomly shuffling the dataset and then splitting into training and testing set

```
np.random.shuffle(dataset)
train_dataset, test_dataset = train_test_split(dataset,test_size=0.1)
```

```
print(train_dataset.shape)
print(test_dataset.shape)
```

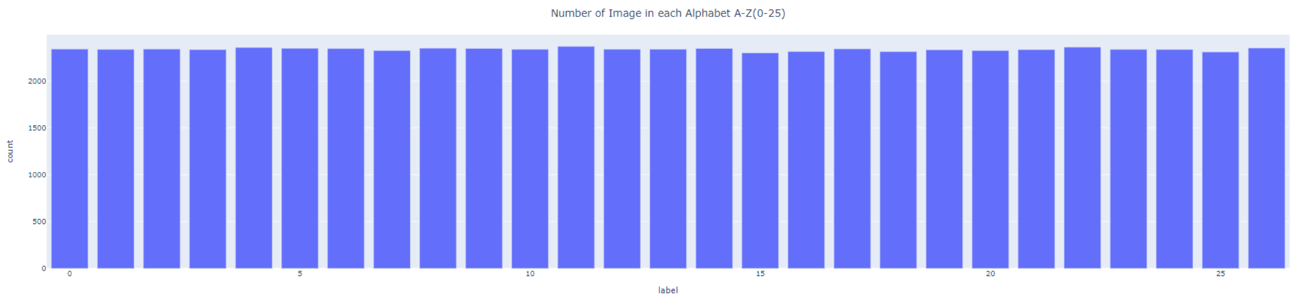
```
(63228, 2501)
(7026, 2501)
```

```
# dataset=np.array([1])
```

A bar graph which shows how much image is present in each alplabet in training set

```
train_dataset_label = train_dataset[:,-1]
unique_elements, counts_elements = np.unique(train_dataset_label, return_counts=True)
df = pd.DataFrame({'label': unique_elements, 'count':counts_elements}, columns=['label', 'count'])
```

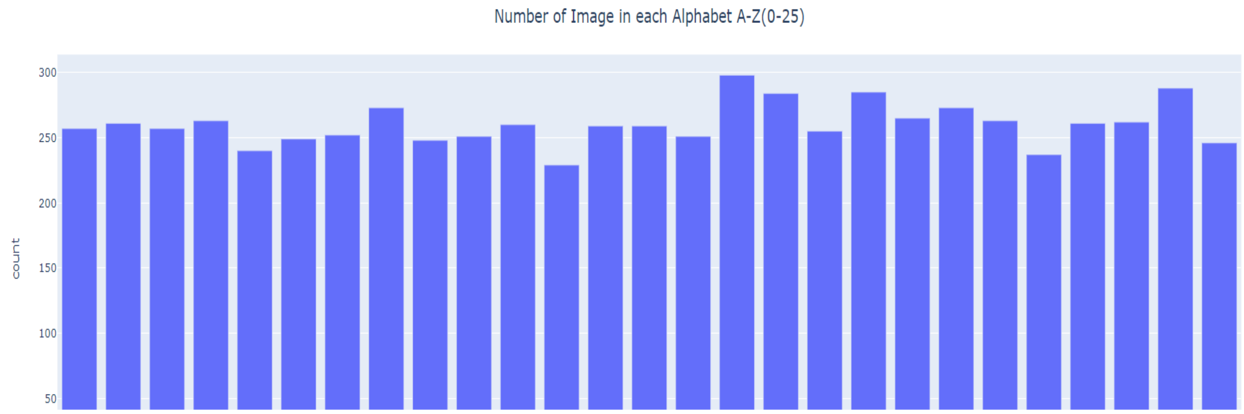
```
fig = px.bar(df,x='label', y='count')
fig.update_layout(
    title={
        'text': "Number of Image in each Alphabet A-Z(0-25)",
        'x':0.5,
        })
fig.show()
```




A bar graph which shows how much image is present in each alplabet in test set

```
test_dataset_label = test_dataset[:,-1]
unique_elements, counts_elements = np.unique(test_dataset_label, return_counts=True)
df = pd.DataFrame({'label': unique_elements, 'count':counts_elements}, columns=['label', 'count'])
```

```
fig = px.bar(df,x='label', y='count')
fig.update_layout(
    title={
        'text': "Number of Image in each Alphabet A-Z(0-25)",
        'x':0.5,
        })
fig.show()
```




Separating the parameters and response variable

```
x_train = train_dataset[:,0:2500]
x_test = test_dataset[:,0:2500]
y_train = train_dataset[:,-1]
y_test = test_dataset[:,-1]
```

```
# train_dataset =np.array([1])
# test_dataset=np.array([1])
```

```
y = y_test
```

```
# # Reshaping the array to required format

x_train = x_train.reshape(-1,50,50,1)
x_test = x_test.reshape(-1,50,50,1)
```

```
[ ]
```

```
[ ] # converting the label from 0-26 to binary array of 0 & 1 i.e a array for each category

from sklearn.preprocessing import LabelBinarizer
label_binarizer = LabelBinarizer()
y_train = label_binarizer.fit_transform(y_train)
y_test = label_binarizer.fit_transform(y_test)
```

```
# shape after preprocessing

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(63228, 50, 50, 1)
(63228, 27)
(7026, 50, 50, 1)
(7026, 27)
```

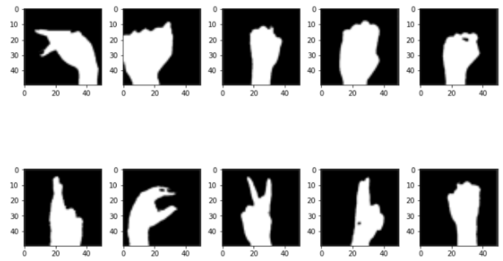
## Splitting the training set into training and validation dataset

```
[ ] from sklearn.model_selection import train_test_split

x_train, x_val, y_train, y_val = train_test_split(x_train,y_train,test_size=0.08)

print(x_train.shape)
print(x_val.shape)
print(y_train.shape)
print(y_val.shape)
```

```
(58169, 50, 50, 1)
(5059, 50, 50, 1)
(58169, 27)
(5059, 27)
```

## Displaying 10 images randomly from the training dataset

```
f, ax = plt.subplots(2,5)
f.set_size_inches(10, 10)
for i in range(2):
    for j in range(5):
        k = random.randint(0,x_train.shape[0])
        ax[i,j].imshow(np.asarray(x_train[k].reshape(50, 50),dtype='float32') , cmap = "gray")
    plt.tight_layout()
```




## Data Augmentation

```
[ ] # With data augmentation to prevent overfitting

datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=20,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.05, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(x_train)
```

## Training the model

```
[ ] # This is used to reduce the learning rate. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates.
# This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy', patience = 2, verbose=1,factor=0.4, min_lr=0.00001)
```

## Building the architecture of the model

```
## Our model consist of 6 hidden layer i.e first four layers consist of  convolution operation
## followed by max pooling and the last two layer consist of fully connected layer
## The output layer consist of softmax layer of 27 nodes depicting the output from A-Z and nothing


model = Sequential()
model.add(Conv2D(32 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu' , input_shape = (50,50,1)))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))

model.add(Conv2D(64 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))

model.add(Conv2D(128 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))

model.add(Conv2D(256 , (3,3) , strides = 1 , padding = 'same' , activation = 'relu'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(MaxPool2D((2,2) , strides = 2 , padding = 'same'))

model.add(Flatten())
model.add(Dense(units = 256 , activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(units = 128 , activation = 'relu'))
model.add(Dropout(0.15))
model.add(Dense(units = 27 , activation = 'softmax'))
```

## Compiling the model

```
[ ] # We have used adam optimizer and loss function is categorical cross entropy

model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
```

+ Code + Text

```
model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
```

```
# printing the details of our layers in a table with the sizes of its inputs/outputs

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 50, 50, 32)        320
_________________________________________________________________
batch_normalization (BatchNo (None, 50, 50, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 25, 25, 64)        18496
_________________________________________________________________
dropout (Dropout)            (None, 25, 25, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 25, 25, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 128)       73856
_________________________________________________________________
dropout_1 (Dropout)          (None, 13, 13, 128)       0
_________________________________________________________________
batch_normalization_2 (Batch (None, 13, 13, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 256)         295168
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 256)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 7, 7, 256)         1024
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1048832
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 27)                3483
=================================================================
Total params: 1,474,971
Trainable params: 1,474,011
Non-trainable params: 960
_________________________________________________________________
```

+ Code + Text

## Fitting the model

```
train_model = model.fit(datagen.flow(x_train, y_train, batch_size=256),validation_data = (x_val, y_val), epochs=50,callbacks = [learning_rate_reduction])
```

```
Epoch 1/50
228/228 [==============================] - 57s 103ms/step - loss: 1.9542 - accuracy: 0.4394 - val_loss: 0.1399 - val_accuracy: 0.9456
Epoch 2/50
228/228 [==============================] - 23s 100ms/step - loss: 0.1619 - accuracy: 0.9461 - val_loss: 0.0199 - val_accuracy: 0.9923
Epoch 3/50
228/228 [==============================] - 23s 100ms/step - loss: 0.0660 - accuracy: 0.9788 - val_loss: 0.0051 - val_accuracy: 0.9986
Epoch 4/50
228/228 [==============================] - 23s 102ms/step - loss: 0.0387 - accuracy: 0.9879 - val_loss: 0.0067 - val_accuracy: 0.9976
Epoch 5/50
228/228 [==============================] - 23s 101ms/step - loss: 0.0326 - accuracy: 0.9893 - val_loss: 0.0020 - val_accuracy: 0.9996
Epoch 6/50
228/228 [==============================] - 23s 102ms/step - loss: 0.0260 - accuracy: 0.9917 - val_loss: 6.6076e-04 - val_accuracy: 1.0000
Epoch 7/50
228/228 [==============================] - 23s 102ms/step - loss: 0.0203 - accuracy: 0.9936 - val_loss: 5.5245e-04 - val_accuracy: 1.0000
Epoch 8/50
228/228 [==============================] - 23s 101ms/step - loss: 0.0136 - accuracy: 0.9956 - val_loss: 0.0017 - val_accuracy: 0.9998

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.0004000000189989805.
Epoch 9/50
228/228 [==============================] - 23s 100ms/step - loss: 0.0093 - accuracy: 0.9976 - val_loss: 2.7780e-05 - val_accuracy: 1.0000
Epoch 10/50
228/228 [==============================] - 23s 101ms/step - loss: 0.0042 - accuracy: 0.9987 - val_loss: 1.6607e-04 - val_accuracy: 1.0000

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.00016000000759959222.
Epoch 11/50
228/228 [==============================] - 23s 100ms/step - loss: 0.0040 - accuracy: 0.9988 - val_loss: 2.8197e-05 - val_accuracy: 1.0000
Epoch 12/50
228/228 [==============================] - 23s 100ms/step - loss: 0.0031 - accuracy: 0.9990 - val_loss: 1.3020e-04 - val_accuracy: 1.0000

Epoch 00012: ReduceLROnPlateau reducing learning rate to 6.40000042039901e-05.
Epoch 13/50
228/228 [==============================] - 23s 100ms/step - loss: 0.0026 - accuracy: 0.9993 - val_loss: 3.9202e-05 - val_accuracy: 1.0000
Epoch 14/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0021 - accuracy: 0.9993 - val_loss: 2.2562e-05 - val_accuracy: 1.0000

Epoch 00014: ReduceLROnPlateau reducing learning rate to 2.560000284574926e-05.
Epoch 15/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0033 - accuracy: 0.9991 - val_loss: 3.4319e-05 - val_accuracy: 1.0000
Epoch 16/50
228/228 [==============================] - 22s 99ms/step - loss: 0.0021 - accuracy: 0.9994 - val_loss: 4.0995e-05 - val_accuracy: 1.0000

Epoch 00016: ReduceLROnPlateau reducing learning rate to 1.0240000847261399e-05.
Epoch 17/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0030 - accuracy: 0.9993 - val_loss: 2.9973e-05 - val_accuracy: 1.0000
Epoch 18/50
228/228 [==============================] - 23s 100ms/step - loss: 0.0020 - accuracy: 0.9993 - val_loss: 2.3007e-05 - val_accuracy: 1.0000

Epoch 00018: ReduceLROnPlateau reducing learning rate to 1e-05.
Epoch 19/50
228/228 [==============================] - 22s 98ms/step - loss: 0.0021 - accuracy: 0.9992 - val_loss: 2.2587e-05 - val_accuracy: 1.0000
Epoch 20/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0014 - accuracy: 0.9995 - val_loss: 1.2691e-05 - val_accuracy: 1.0000
```

```
Epoch 22/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0012 - accuracy: 0.9996 - val_loss: 1.5306e-05 - val_accuracy: 1.0000
Epoch 23/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0017 - accuracy: 0.9995 - val_loss: 1.8207e-05 - val_accuracy: 1.0000
Epoch 24/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0018 - accuracy: 0.9994 - val_loss: 3.0136e-05 - val_accuracy: 1.0000
Epoch 25/50
228/228 [==============================] - 23s 100ms/step - loss: 0.0013 - accuracy: 0.9997 - val_loss: 1.8571e-05 - val_accuracy: 1.0000
Epoch 26/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0012 - accuracy: 0.9996 - val_loss: 1.8392e-05 - val_accuracy: 1.0000
Epoch 27/50
228/228 [==============================] - 22s 99ms/step - loss: 0.0020 - accuracy: 0.9995 - val_loss: 1.5907e-05 - val_accuracy: 1.0000
Epoch 28/50
228/228 [==============================] - 23s 100ms/step - loss: 0.0018 - accuracy: 0.9994 - val_loss: 9.6378e-06 - val_accuracy: 1.0000
Epoch 29/50
228/228 [==============================] - 22s 99ms/step - loss: 0.0014 - accuracy: 0.9997 - val_loss: 1.2032e-05 - val_accuracy: 1.0000
Epoch 30/50
228/228 [==============================] - 23s 100ms/step - loss: 0.0016 - accuracy: 0.9993 - val_loss: 7.3221e-06 - val_accuracy: 1.0000
Epoch 31/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0014 - accuracy: 0.9995 - val_loss: 1.6961e-05 - val_accuracy: 1.0000
Epoch 32/50
228/228 [==============================] - 23s 100ms/step - loss: 0.0015 - accuracy: 0.9996 - val_loss: 6.7681e-06 - val_accuracy: 1.0000
Epoch 33/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0017 - accuracy: 0.9994 - val_loss: 8.4714e-06 - val_accuracy: 1.0000
Epoch 34/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0016 - accuracy: 0.9994 - val_loss: 1.3106e-05 - val_accuracy: 1.0000
Epoch 35/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0013 - accuracy: 0.9996 - val_loss: 1.1366e-05 - val_accuracy: 1.0000
Epoch 36/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0015 - accuracy: 0.9995 - val_loss: 1.4608e-05 - val_accuracy: 1.0000
Epoch 37/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0011 - accuracy: 0.9997 - val_loss: 1.6948e-05 - val_accuracy: 1.0000
Epoch 38/50
228/228 [==============================] - 22s 98ms/step - loss: 0.0012 - accuracy: 0.9995 - val_loss: 1.3140e-05 - val_accuracy: 1.0000
Epoch 39/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0018 - accuracy: 0.9994 - val_loss: 1.7573e-05 - val_accuracy: 1.0000
Epoch 40/50
228/228 [==============================] - 23s 100ms/step - loss: 0.0011 - accuracy: 0.9998 - val_loss: 4.7039e-06 - val_accuracy: 1.0000
Epoch 41/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0015 - accuracy: 0.9996 - val_loss: 5.1331e-06 - val_accuracy: 1.0000
Epoch 42/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0013 - accuracy: 0.9996 - val_loss: 6.0380e-06 - val_accuracy: 1.0000
Epoch 43/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0015 - accuracy: 0.9995 - val_loss: 5.2489e-06 - val_accuracy: 1.0000
Epoch 44/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0012 - accuracy: 0.9996 - val_loss: 4.7798e-06 - val_accuracy: 1.0000
Epoch 45/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0016 - accuracy: 0.9995 - val_loss: 4.1214e-06 - val_accuracy: 1.0000
Epoch 46/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0010 - accuracy: 0.9997 - val_loss: 6.0936e-06 - val_accuracy: 1.0000
Epoch 47/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0012 - accuracy: 0.9995 - val_loss: 6.5207e-06 - val_accuracy: 1.0000
Epoch 48/50
228/228 [==============================] - 23s 99ms/step - loss: 9.9192e-04 - accuracy: 0.9996 - val_loss: 6.3514e-06 - val_accuracy: 1.0000
Epoch 49/50
228/228 [==============================] - 23s 99ms/step - loss: 0.0014 - accuracy: 0.9995 - val_loss: 8.5947e-06 - val_accuracy: 1.0000
Epoch 50/50
228/228 [==============================] - 23s 99ms/step - loss: 7.8367e-04 - accuracy: 0.9998 - val_loss: 7.0749e-06 - val_accuracy: 1.0000
```

## finding Accuracy of the model

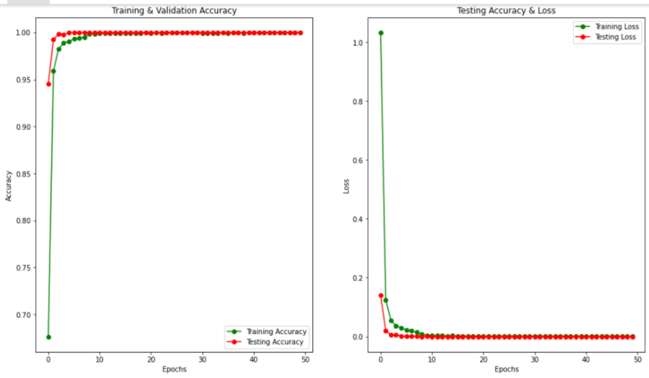
```
[ ] print("Accuracy of the model is - " , model.evaluate(x_test,y_test)[1]*100 , "%")
```

```
220/220 [==============================] - 1s 4ms/step - loss: 8.0004e-06 - accuracy: 1.0000
Accuracy of the model is -  100.0 %
```

## plotting graph of Training & Validation Accuracy and Testing Accuracy & Loss

```
[ ] epochs = [i for i in range(50)]
    fig , ax = plt.subplots(1,2)
    train_acc = train_model.history['accuracy']
    train_loss = train_model.history['loss']
    val_acc = train_model.history['val_accuracy']
    val_loss = train_model.history['val_loss']
    fig.set_size_inches(16,9)

    ax[0].plot(epochs , train_acc , 'go-' , label = 'Training Accuracy')
    ax[0].plot(epochs , val_acc , 'ro-' , label = 'Testing Accuracy')
    ax[0].set_title('Training & Validation Accuracy')
    ax[0].legend()
    ax[0].set_xlabel("Epochs")
    ax[0].set_ylabel("Accuracy")

    ax[1].plot(epochs , train_loss , 'g-o' , label = 'Training Loss')
    ax[1].plot(epochs , val_loss , 'r-o' , label = 'Testing Loss')
    ax[1].set_title('Testing Accuracy & Loss')
    ax[1].legend()
    ax[1].set_xlabel("Epochs")
    ax[1].set_ylabel("Loss")
    plt.show()
```

+ Code + Text Connect




## predicting the label of the test dataset

```
[ ] predictions =np.argmax(model.predict(x_test), axis=1)
```

```
[ ] np.unique(predictions)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26])
```

## Displaying the F1-score, Recall, precision and no of examples for each label

```
classes = ["Class " + str(i) for i in range(27)]
print(classification_report(y, predictions, target_names = classes))
```

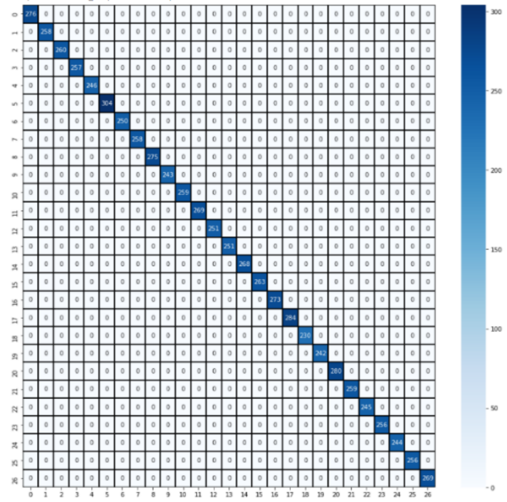
```
              precision    recall  f1-score   support

     Class 0       1.00      1.00      1.00       276
     Class 1       1.00      1.00      1.00       258
     Class 2       1.00      1.00      1.00       260
     Class 3       1.00      1.00      1.00       257
     Class 4       1.00      1.00      1.00       246
     Class 5       1.00      1.00      1.00       304
     Class 6       1.00      1.00      1.00       250
     Class 7       1.00      1.00      1.00       258
     Class 8       1.00      1.00      1.00       275
     Class 9       1.00      1.00      1.00       243
    Class 10       1.00      1.00      1.00       259
    Class 11       1.00      1.00      1.00       269
    Class 12       1.00      1.00      1.00       251
    Class 13       1.00      1.00      1.00       251
    Class 14       1.00      1.00      1.00       268
    Class 15       1.00      1.00      1.00       263
    Class 16       1.00      1.00      1.00       273
    Class 17       1.00      1.00      1.00       284
    Class 18       1.00      1.00      1.00       230
    Class 19       1.00      1.00      1.00       242
    Class 20       1.00      1.00      1.00       280
    Class 21       1.00      1.00      1.00       259
    Class 22       1.00      1.00      1.00       245
    Class 23       1.00      1.00      1.00       256
    Class 24       1.00      1.00      1.00       244
    Class 25       1.00      1.00      1.00       256
    Class 26       1.00      1.00      1.00       269

    accuracy                           1.00      7026
   macro avg       1.00      1.00      1.00      7026
weighted avg       1.00      1.00      1.00      7026
```

## Creating a confusion matrix

```
cm = confusion_matrix(y,predictions)
cm = pd.DataFrame(cm , index = [i for i in range(27)] , columns = [i for i in range(27)])
# display(cm)
```

```
import seaborn as sns
plt.figure(figsize = (15,15))
sns.heatmap(cm,cmap= "Blues", linecolor = 'black' , linewidth = 1 , annot = True, fmt='')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb01c7cbcd0>
```

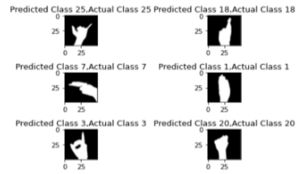



## Showing the Actual and predicted output of some images from test dataset randomly

```
# saving the index for label whose prediciton is true

correct = np.nonzero(predictions == y)[0]
correct
```

```
array([   0,    1,    2, ..., 7023, 7024, 7025])
```

```
i = 0
for _ in correct[:6]:
    plt.subplot(3,2,i+1)
    k = random.randint(0,y.shape[0])
    plt.imshow(x_test[k].reshape(50,50), cmap="gray")
    plt.title("Predicted Class {},Actual Class {}".format(predictions[k], y[k]))
    plt.tight_layout()
    i += 1
```




## Saving the model

```
import h5py
model.save('/content/drive/MyDrive/model/my_model/keras_model.h5')
```

## code for prediction in real time using opencv

```
import cv2, pickle
import numpy as np
import os
import math
from keras.models import load_model

model = load_model('/content/drive/MyDrive/model/my_model/keras_model.h5')



def keras_process_image(img):
    img = cv2.resize(img, (50,50))
    img = np.array(img, dtype=np.uint8)
    img = np.reshape(img, (1,50,50, 1))
    return img

def keras_predict(model, image):
    processed = keras_process_image(image)
    pred_probab = model.predict(processed)[0]
    pred_class = list(pred_probab).index(max(pred_probab))
    return max(pred_probab), pred_class

def get_pred_text(pred_class):
    dictionary={0:'A', 1:'B', 2:'C', 3:'D', 4:'E', 5:'F', 6:'G', 7:'H', 8:'I', 9:'J', 10:'K', 11:'L', 12:'M',
                13:'N', 14:'Nothing', 15:'O',16:'P', 17:'Q', 18:'R', 19:'S', 20:'T',21:'U', 22:'V', 23:'W',
                24:'X', 25:'Y', 26:'Z'}
    return dictionary[pred_class]


def recognize():
    num_frames=0
    cap = cv2.VideoCapture(0)

    num_frames=0
    while(1):
        ret,image = cap.read()
        image = cv2.flip(image, 1)
        if not ret:
            break
        k = cv2.waitKey(5) & 0xFF
        if k== ord('c'):
            num_frames=0
        if num_frames<=30:
            background_image = image
            num_frames+=1
        if num_frames>30:
#             cv2.imshow('background',background_image)
            current_image= image
            diff = cv2.absdiff(background_image,current_image)
            mask = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
            th, mask_thresh = cv2.threshold(mask, 10, 255, cv2.THRESH_BINARY)
            mask_blur = cv2.GaussianBlur(mask_thresh, (3, 3), 10)
            mask_erosion = cv2.erode(mask_blur, np.ones((5,5), dtype=np.uint8), iterations=1)
            mask_erosion = mask_erosion[150:400,150:400]
            cv2.imshow('final',mask_erosion)
            pred_probab, pred_class = keras_predict(model, mask_erosion)
#             print(pred_class)


            if pred_probab*100 > 80:
                    text = get_pred_text(pred_class)
                    print(text)
                    image = cv2.putText(image, text, (150, 100), cv2.FONT_HERSHEY_COMPLEX, 4.0, (255, 255, 255), lineType=cv2.LINE_AA)
            cv2.rectangle(image, (150, 150), (400, 400), (255, 255,0), 3)
            cv2.imshow('frame',image)



        if k == 27:
            break

    cv2.destroyAllWindows()
    cap.release()
```

Department of Computer Science and Engineering
Silicon Institute of Technology,
Silicon Hills, Bhubaneswar –751024,
Odisha, India