MAULANA AZAD
NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL INDIA, 462003

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SIGN LANGUAGE TO TEXT CONVERSION FOR DEAF AND MUTE

Minor Project Report

Semester 6

Submitted by:

ARPIT VERMA	181112205
KARANDEEP SINGH LAMBA	181112201
NITIN KUMAR	181112230
NIKHIL SANORA	181112227

Under the Guidance of
DR. MANASI GYANCHANDANI
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Session: 2018-22

MAULANA AZAD
NATIONAL INSTITUTE OF TECHNOLOGY
BHOPAL INDIA, 462003

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the project report carried out on "**SIGN LANGUAGE TO TEXT CONVERSION FOR DUMB AND DEAF**" by the 3rd year students:

ARPIT VERMA	181112205
KARANDEEP SINGH LAMBA	181112201
NITIN KUMAR	181112230
NIKHIL SANORA	181112227

Have successfully completed their project in partial fulfilment of their Degree in Bachelor of Technology in Computer Science and Engineering.

DR. MANASI GYANCHANDANI
(Minor Project Mentor)

DECLARATION

We, hereby declare that the following report which is being presented in the Minor Project Documentation Entitled as "**SIGN LANGUAGE TO TEXT CONVERSION FOR DUMB AND DEAF**" is an authentic documentation of our own original work and to best of our knowledge. The following project and its report, in part or whole, has not been presented or submitted by us for any purpose in any other institute or organization. Any contribution made to the research by others, with whom we have worked at Maulana Azad National Institute of Technology, Bhopal or elsewhere, is explicitly acknowledged in the report.

ARPIT VERMA	181112205
KARANDEEP SINGH LAMBA	181112201
NITIN KUMAR	181112230
NIKHIL SANORA	181112227

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected guide and coordinator Dr. Manasi Gyanchandani, for her valuable help and guidance. We are thankful for the encouragement that he has given us in completing this project successfully.

It is imperative for us to mention the fact that the report of minor project could not have been accomplished without the periodic suggestions and advice of our project guide Dr. Manasi Gyanchandani and project coordinators Dr. Dhirendra Pratap Singh and Dr. Jay Trilok Choudhary.

We are also grateful to our respected director Dr. N. S. Raghuwanshi for permitting us to utilize all the necessary facilities of the college.

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind cooperation and help. Last but certainly not the least; we would like to express our deep appreciation towards our family members and batch mates for providing the much needed support and encouragement.

ABSTRACT

Sign language is one of the oldest and most natural form of language for communication, but since most people do not know sign language and interpreters are very difficult to come by we have come up with a real time method using neural networks for conversion of Indian sign language. In our method, the hand is first passed through a filter and after the filter is applied the hand is passed through a classifier which predicts the class of the hand gestures. Our method provides 92.3 % accuracy for the 26 letters of the alphabet.

TABLE OF CONTENTS

Certificate **ii**
Declaration **iii**
Acknowledgement **iv**
Abstract **v**
1. Introduction.....7-9.

1.1 Sub section as per required

2. Literature review and survey……. 9-15.

3. Gaps identified……15-16.

4. Proposed work and methodology…….16-32.

4.1 Proposed Work

4.2 Methodology

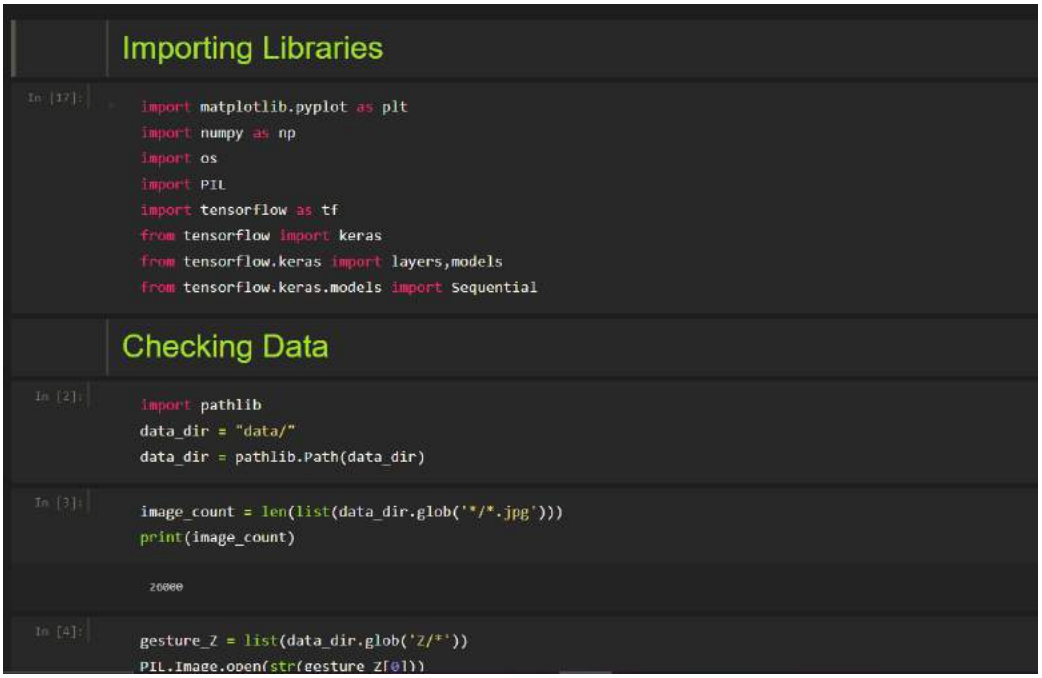
5. Results and Discussion……...…… 34-40.

6. Conclusion…….…. 41.

7. References……...…… 42.

LIST OF FIGURES

Figure 1- **Image Count:**



Importing Libraries

```
In [17]:
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers,models
from tensorflow.keras.models import Sequential
```

Checking Data

```
In [2]:
import pathlib
data_dir = "data/"
data_dir = pathlib.Path(data_dir)

In [3]:
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)

20000

In [4]:
gesture_Z = list(data_dir.glob('Z/*'))
PIL.Image.open(str(gesture_Z[0]))
```

Figure 2- Gesture image:

```
In [4]:
gesture_Z = list(data_dir.glob('Z/*'))
PIL.Image.open(str(gesture_Z[0]))
```

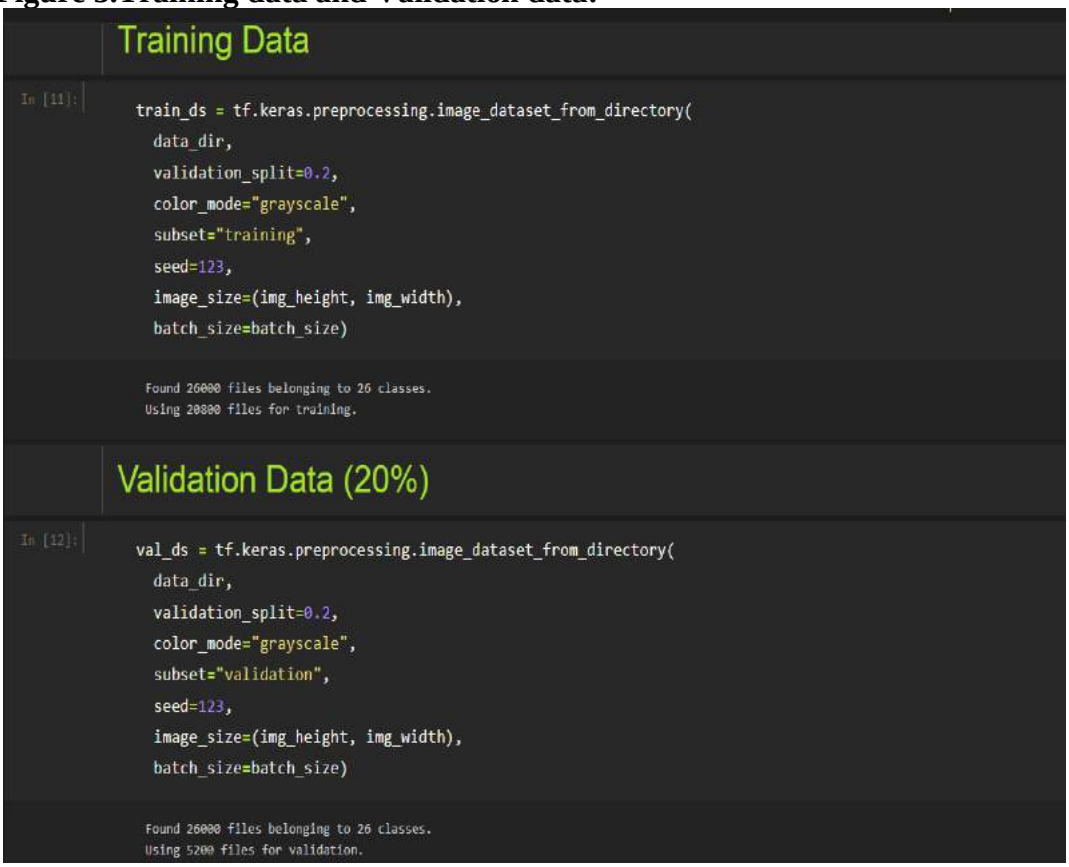

Image and Batch Configuration

```
In [5]:
batch_size = 32
img_height = 128
img_width = 128
```

Training Data

```
In [11]:
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  color_mode="grayscale",
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
```

Figure 3.Training data and Validation data:



Training Data

In [11]:

```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  color_mode="grayscale",
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 26000 files belonging to 26 classes.
Using 20800 files for training.
```

Validation Data (20%)

In [12]:

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  color_mode="grayscale",
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 26000 files belonging to 26 classes.
Using 5200 files for validation.
```

Figure-4. **Building CNN model:**

Building CNN Model

```
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
```

```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))
```

```
0.0 1.0
```

Figure-5. **Training Model:**

Training Model

```
epochs = 5
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/5
650/650 [==============================] - 230s 354ms/step - loss: 0.9529 - accuracy: 0.7290 - val_loss: 0.0022 - val_accuracy: 0.99
96
Epoch 2/5
650/650 [==============================] - 222s 341ms/step - loss: 0.0073 - accuracy: 0.9980 - val_loss: 7.2559e-04 - val_accuracy:
0.9998
Epoch 3/5
650/650 [==============================] - 239s 366ms/step - loss: 9.8578e-04 - accuracy: 0.9998 - val_loss: 4.5500e-05 - val_accura
cy: 1.0000
Epoch 4/5
650/650 [==============================] - 230s 353ms/step - loss: 9.8371e-05 - accuracy: 1.0000 - val_loss: 1.1324e-05 - val_accura
cy: 1.0000
Epoch 5/5
650/650 [==============================] - 237s 364ms/step - loss: 8.7645e-04 - accuracy: 0.9998 - val_loss: 6.5473e-05 - val_accura
cy: 1.0000
```

Figure-6. **Visualization:**

Visualization

```
In [66]:
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

LIST OF TABLES

Model: "sequential_14"

Layer (type)	Output Shape	Param #
rescaling_15 (Rescaling)	(None, 128, 128, 1)	0
conv2d_36 (Conv2D)	(None, 128, 128, 20)	520
max_pooling2d_24 (MaxPooling	(None, 42, 42, 20)	0
conv2d_37 (Conv2D)	(None, 42, 42, 32)	16032
max_pooling2d_25 (MaxPooling	(None, 7, 7, 32)	0
dropout_14 (Dropout)	(None, 7, 7, 32)	0
flatten_14 (Flatten)	(None, 1568)	0
dense_28 (Dense)	(None, 26)	40794
dense_29 (Dense)	(None, 26)	702

Total params: 58,048

Trainable params: 58,048

Non-trainable params: 0

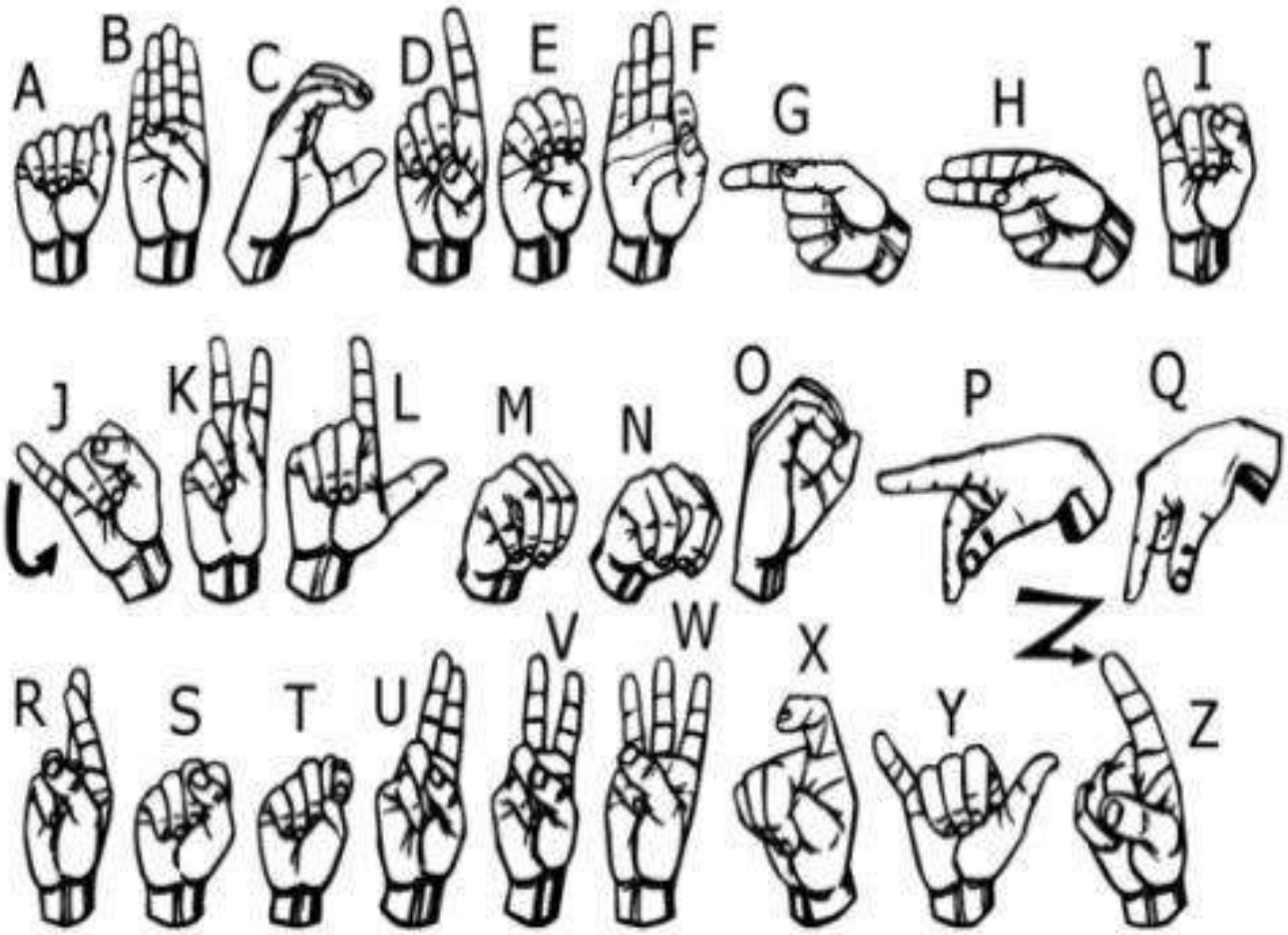
Introduction

Indian sign language is a predominant sign language Since the only disability D&M people have is communication related and they cannot use spoken languages hence the only way for them to communicate is through sign language. Communication is the process of exchange of thoughts and messages in various ways such as speech, signals, behavior and visuals. Deaf and dumb(D&M) people make use of their hands to express different gestures to express their ideas with other people. Gestures are the nonverbally exchanged messages and these gestures are understood with vision. This nonverbal communication of deaf and dumb people is called sign language.

Sign language is a visual language and consists of 3 major components:

Fingerspelling	Word level sign vocabulary	Non-manual features
Used to spell words letter by letter .	Used for the majority of communication.	Facial expressions and tongue, mouth and body position.

In our project we basically focus on producing a model which can recognizes Fingerspelling based hand gestures in order to form a complete word by combining each gesture. The gestures we aim to train are as given in the image below.



For interaction between normal people and D&M people a language barrier is created as sign language structure which is different from normal text. So they depend on vision based communication for interaction.

If there is a common interface that converts the sign language to text the gestures can be easily understood by the other people. So research has been made for a vision based interface system where D&M people can enjoy communication without really knowing each other's language.

The aim is to develop a user friendly human computer interfaces (HCI) where the computer understands the human sign language. There are various sign languages all over the world, namely Indian Sign Language(ISL),American Sign Language (ASL), French Sign Language, British Sign Language (BSL), Indian Sign language, Japanese Sign Language and work has been done on other languages all around the world.

Literature Survey And Review

In the recent years there has been tremendous research done on the hand gesture recognition.
With the help of literature survey done we realized the basic steps in hand gesture recognition are :-

- Data acquisition
- Data preprocessing
- Feature extraction
- Gesture classification

Data acquisition:
The different approaches to acquire data about the hand gesture can be done in the following ways:

1. **Use of sensory devices**

   It uses electromechanical devices to provide exact hand configuration, and position. Different glove based approaches can be used to extract information .But it is expensive and not user friendly.

2. **Vision based approach**

   In vision based methods computer camera is the input device for observing the information of hands or fingers. The Vision Based methods require only a camera, thus realizing a natural interaction between humans and computers without the use of any extra devices. These systems tend to complement biological vision by describing

artificial vision systems that are implemented in software and/or hardware.The main challenge of vision-based hand detection is to cope with the large variability of human hand's appearance due to a huge number of hand movements, to different skin-colour possibilities as well as to the variations in view points, scales, and speed of the camera capturing the scene.

Data preprocessing and Feature extraction for vision based approach:

- In the approach for hand detection combines threshold-based color detection with background subtraction.We can use Adaboost face detector to differentiate between faces and hands as both involve similar skin-color.

- We can also extract necessary image which is to be trained by applying a filter called Gaussian blur. The filter can be easily applied using open computer vision also known as OpenCV .

- For extracting necessary image which is to be trained we can use instrumented gloves . This helps reduce computation time for preprocessing and can give us more concise and accurate data compared to applying filters on data received from video extraction.

- We tried doing the hand segmentation of an image using color segmentation techniques but as mentioned in the research paper skin color and tone is highly dependent on the lighting conditions due to which output we got for the segmentation we tried to do were no so great. Moreover we have a huge number of symbols to be trained for our project many of which look similar to each other like the gesture for symbol 'V' and digit '2', hence we decided that in order to produce better accuracies for our large number of symbols, rather than

segmenting the hand out of a random background we keep background of hand a stable single color so that we don't need to segment it on the basis of skin color . This would help us to get better results.

Key Words and Definitions

.

Feature Extraction and Representation:

The representation of an image as a 3D matrix having dimension as of height and width of the image and the value of each pixel as depth (1 in case of Grayscale and 3 in case of RGB). Further, these pixel values are used for extracting useful features using CNN.

Artificial Neural Networks:

Artificial Neural Network is a connections of neurons, replicating the structure of human brain. Each connection of neuron transfers information to another neuron. Inputs are fed into first layer of neurons which processes it and transfers to another layer of neurons called as hidden layers. After processing of information through multiple layers of hidden layers, information is passed to final output layer.

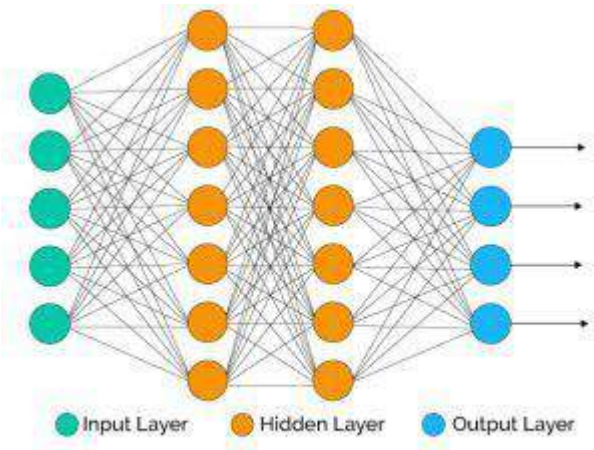


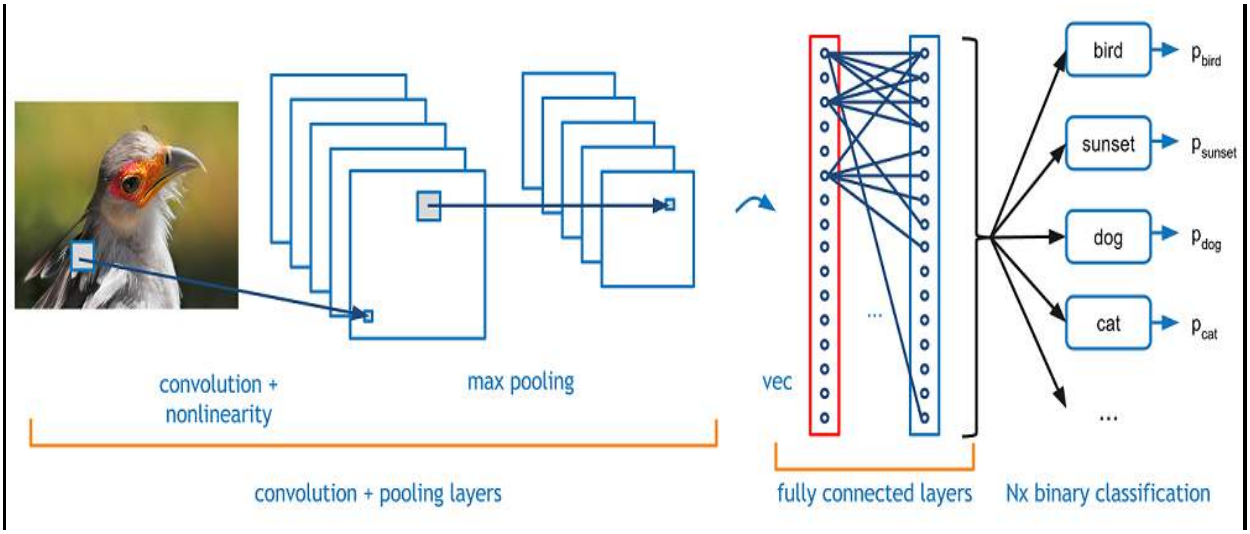
Figure 5.1: Artificial neural networks

There are capable of learning and they have to be trained. There are different learning strategies :

1. Unsupervised Learning
2. Supervised Learning
3. Reinforcement Learning

Convolution Neural Network :

Unlike regular Neural Networks, in the layers of CNN, the neurons are arranged in 3 dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer (window size) before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would have dimensions (number of classes), because by the end of the CNN architecture we will reduce the full image into a single vector of class scores.

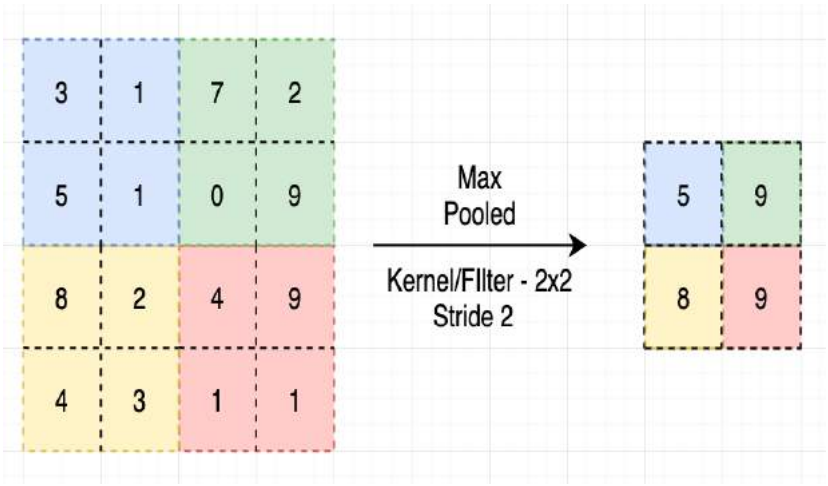


1. **Convolution Layer :** In convolution layer we take a small window size [typically of length 5*5] that extends to the depth of the input matrix. The layer consist of learnable filters of window size. During every iteration we slid the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position. As we continue this process well create a 2-Dimensional activation matrix that gives the response of that matrix at every spatial position. That is, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color

2. **Pooling Layer :** We use pooling layer to decrease the size of activation matrix and ultimately reduce the learnable parameters. There are two type of pooling :

    a) Max Pooling : In max pooling we take a window size [for example window of size 2*2], and only take the maximum of 4 values. Well lid this window and continue this process, so well finally get a activation matrix half of its original Size.

    b) Average Pooling : In average pooling we take average of all value in a window



3. **Fully Connected Layer :** In convolution layer neurons are connected only to a local region, while in a fully connected region, well connect the all the inputs to neurons.

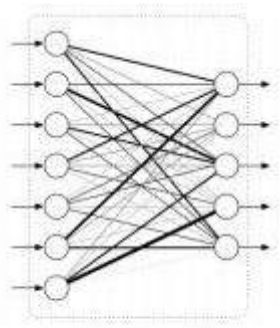


Figure 5.4: Fully Connected Layer

4. **Final Output Layer :** After getting values from fully connected layer, well connect them to final layer of neurons[having count equal to total number of classes], that will predict the probability of each image to be in different classes.

TensorFlow :

Tensorflow is an open source software library for numerical computation. First we define the nodes of the computation graph, then inside a session, the

actual computation takes place. TensorFlow is widely used in Machine Learning.

Keras :

Keras is a high-level neural networks library written in python that works as a wrapper to TensorFlow. It is used in cases where we want to quickly build and test the neural network with minimal lines of code. It contains implementations of commonly used neural network elements like layers, objective, activation functions, optimizers, and tools to make working with images

OpenCV :

OpenCV (Open Source Computer Vision) is an open source library of programming functions used for real-time computer-vision. It is mainly used for image processing, video capture and analysis for features like face and object recognition. It is written in C++ which is its primary interface, however bindings are available for Python, Java, MATLAB/OCTAVE.

GAPS IDENTIFIED

- Situation RELEVANCY
- SIGN LANGUAGE
- Teaching and Learning CURVE
- And most important LACK OF PRACTICE RESOURCES

Proposed Work And Methodology

The system is a vision based approach. All the signs are represented with bare hands and so it eliminates the problem of using any artificial devices for interaction.

Data Set Generation

For the project we tried to find already made datasets but we couldn't find dataset in the form of raw images that matched our requirements. All we could find were the datasets in the form of RGB values. Hence we decided to create our own data set. Steps we followed to create our data set are as follows.

We used Open computer vision(OpenCV) library in order to produce our dataset.Firstly we captured around 800 images of each of the symbol in ASL for training purposes and around 200 images per symbol for testing purpose.

First we capture each frame shown by the webcam of our machine. In the each frame we define a region of interest (ROI) which is denoted by a blue bounded square as shown in the image below.

From this whole image we extract our ROI which is RGB and convert it into gray scale Image as shown below.

Finally we apply our gaussian blur filter to our image which helps us extracting various features of our image. The image after applying gaussian blur looks like below.

Data Generation:

```
import cv2
import os
import time


# Creating the Directory Structure for storing all image classes(if not present
already)
if not os.path.exists("data"):
    os.makedirs("data")

    for i in range(65,91):
        os.makedirs(f'data/{chr(i)}')

    for i in range(65,91):
        os.makedirs(f'data/{chr(i)}')

# Function to Extract useful Features from ROI by applying Different Filters and
Techniques
def convert(frame):
    minValue = 70
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(5,5),2)

    th3 =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,
11,2)
    ret, res = cv2.threshold(th3, minValue, 255,
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    return res



# MODE OF CURRENT WINDOW
_MODE=" DATA GENERATION "
directory="data/"

capture=cv2.VideoCapture(0)

while True:
    ret,frame=capture.read()
    frame=cv2.flip(frame,1)
    if ret==False:
        continue

    #Storing # of Images in Each Label
    count = [len(os.listdir(f'{directory}/{chr(i)}')) for i in range(65,91)]

    cv2.putText(frame, f'--- MODE : {_MODE.capitalize()} ---', (10, 30),
cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 2)
    cv2.putText(frame, "--- IMAGE COUNT ---", (10, 50), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 2)


    #Displaying Count of All Labels
    x_coordinate=10
    y_coordinate=80
    for i in range(65,91):
```

```
   cv2.putText(frame,f' {chr(i)} : {count[i-65]}
',(x_coordinate,y_coordinate),cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 2)
        y_coordinate+=15



    # Coordinates of the ROI
    x1 = int(0.5*frame.shape[1])
    y1 = 10
    x2 = frame.shape[1]-10
    y2 = int(0.5*frame.shape[1])


    # Drawing the ROI
    # The increment/decrement by 1 is to compensate for the bounding box
    cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (0,255,0) ,1)

    # Extracting the ROI
    roi = frame[y1:y2, x1:x2]
    roi = cv2.resize(roi, (128, 128))

    cv2.imshow("Frame", frame)

    #Extracing and Converting
    roi=convert(roi)

    cv2.imshow("ROI", roi)

    #Detecting Key Interrupts
    interrupt = cv2.waitKey(10)

    if interrupt & 0xFF == 27: # esc key
        break

    # Save Image under label and Update
    else:
        for i in range(65,91):
            if (interrupt & 0xFF == i) or (interrupt & 0xFF == i+32):
                cv2.imwrite(f'{directory}/{chr(i)}/{count[i-65]}.jpg', roi)
                count[i-65]+=1

capture.release()
cv2.destroyAllWindows()
```

GESTURE CLASSIFICATION

The approach which we used for this project is :

Our approach uses a single layers of algorithm to predict the final symbol of the user.

Algorithm Layer 1:

1. Apply gaussian blur filter and threshold to the frame taken with opencv to get the processed image after feature extraction.
2. This processed image is passed to the CNN model for prediction and if a letter is detected and then the letter is printed.

Layer 1: CNN Model

1. **1st Convolution Layer :**The input picture has resolution of 128x128 pixels. It is first processed in the first convolutional layer using 32 filter weights (3x3 pixels each). This will result in a 126X126 pixel image, one for each Filter-weights.

2. **1st Pooling Layer :** The pictures are down sampled using max pooling of 2x2 i.e we keep the highest value in the 2x2 square of array. Therefore, our picture is down sampled to 63x63 pixels.

3. **2nd Convolution Layer :**Now, these 63 x 63 from the output of the first pooling layer is served as an input to the second convolutional layer.It is processed in the second convolutional layer using 32 filter weights (3x3 pixels each).This will result in a 60 x 60 pixel image.

4. **2nd Pooling Layer :** The resulting images are down sampled again using max pool of 2x2 and is reduced to 30 x 30 resolution of images.

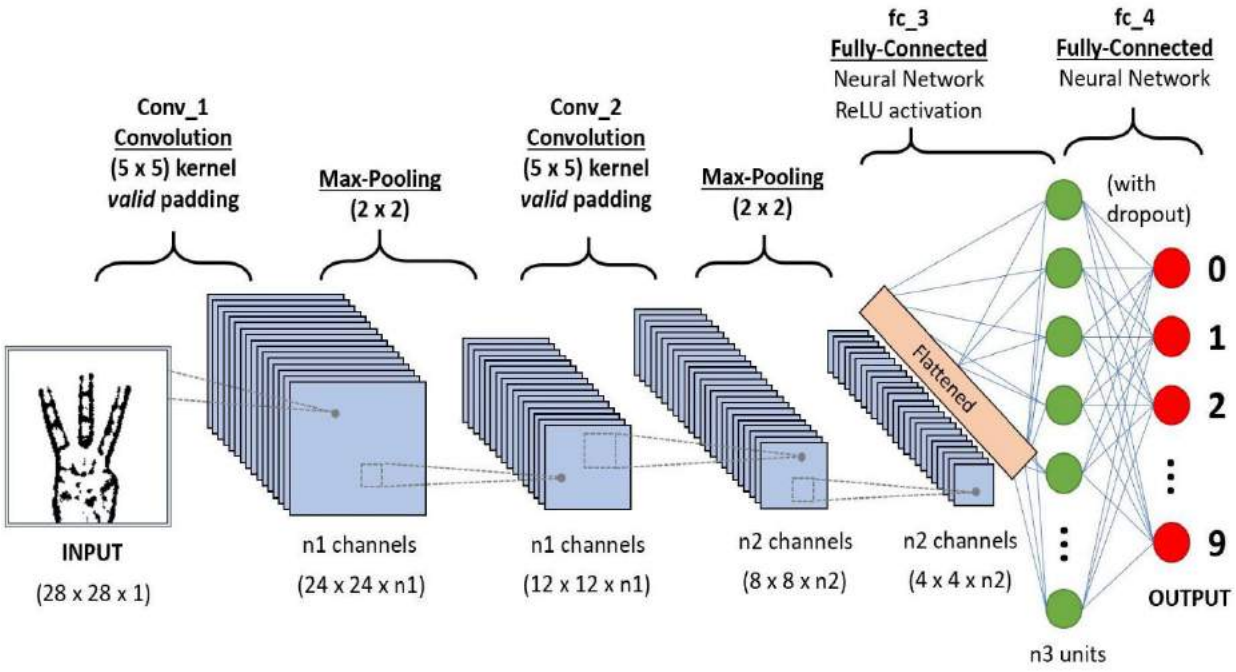
5. **1st Densely Connected Layer :** Now these images are used as an input to a fully connected layer with 128 neurons and the output from the second convolutional layer is reshaped to an array of 30x30x32 =28800 values. The input to this layer is an array of 28800 values. The output of these layer is fed to the 2nd Densely Connected Layer. We are using a dropout layer of value 0.5 to avoid overfitting.

6. **2nd Densely Connected Layer :**

Now the output from the 1st Densely Connected Layer are used as an input to a fully connected layer with 96 neurons.

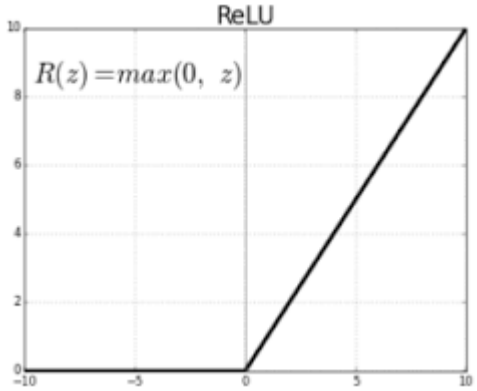
8.Final layer:

The output of the 2nd Densely Connected Layer serves as an input for the final layer which will have the number of neurons as the number of classes we are classifying (alphabets + blank symbol)



Activation Function :

We have used ReLu (Rectified Linear Unit) in each of the layers(convolutional as well as fully connected neurons). ReLu calculates max(0,z) for each input pixel. This adds nonlinearity to the formula and helps to learn more complicated features.It helps in removing the vanishing gradient problem and speeding up the training by reducing the computation time.



Pooling Layer :

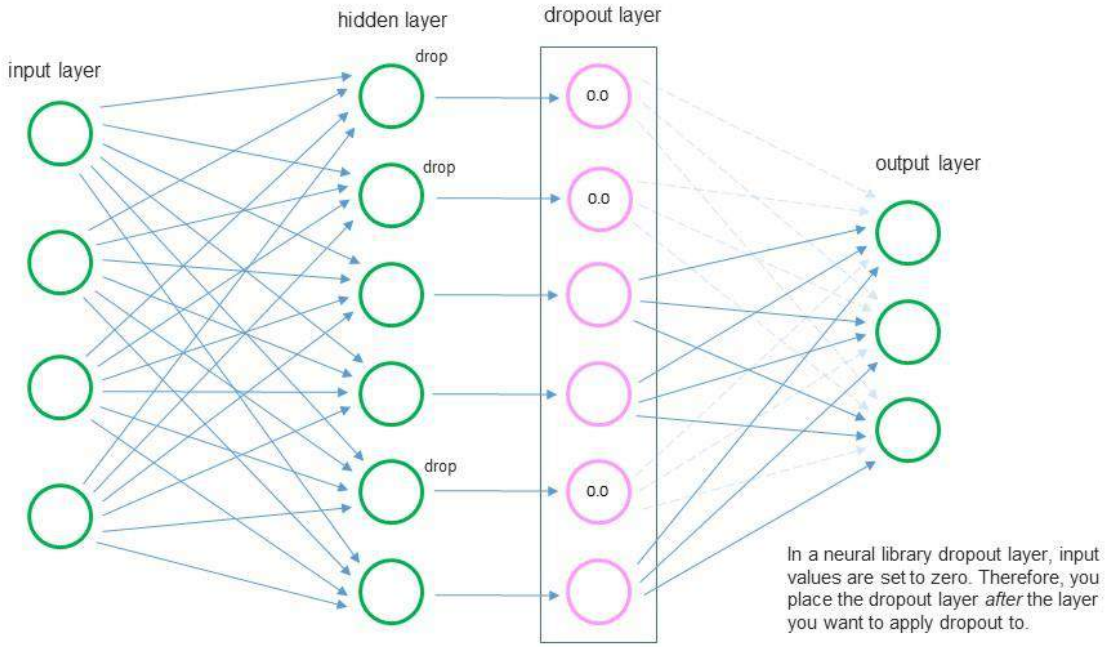
We apply **Max** pooling to the input image with a pool size of (2, 2) with relu activation function.This reduces the amount of parameters thus lessening the computation cost and reduces overfitting.

Dropout Layers:

Applying dropout to a neural network amounts to sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout (Figure . A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks. These networks all share weights so that the total number of parameters is still $O(n^2)$, or less. For each presentation of each training case, a new thinned network is sampled and trained. So training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all.

At test time, it is not feasible to explicitly average the predictions from exponentially many thinned models. However, a very simple approximate averaging method works well in practice. The idea is to use a single neural net at test time without dropout. The weights of this network are scaled-down versions of the trained weights. If a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test

time as shown in Figure . This ensures that for any hidden unit the *expected* output (under the distribution used to drop units at training time) is the same as the actual output attest time. By doing this scaling, 2^n networks with shared weights can be combined into a single neural network to be used at test time. We found that training a network with dropout and using this approximate averaging method at test time leads to significantly lower generalization error on a wide variety of classification problems compared to training with other regularization methods.



Training and Testing :

We convert our input images(RGB) into grayscale and apply gaussian blur to remove unnecessary noise.We apply adaptive threshold to extract our hand from the background and resize our images to 128 x 128.

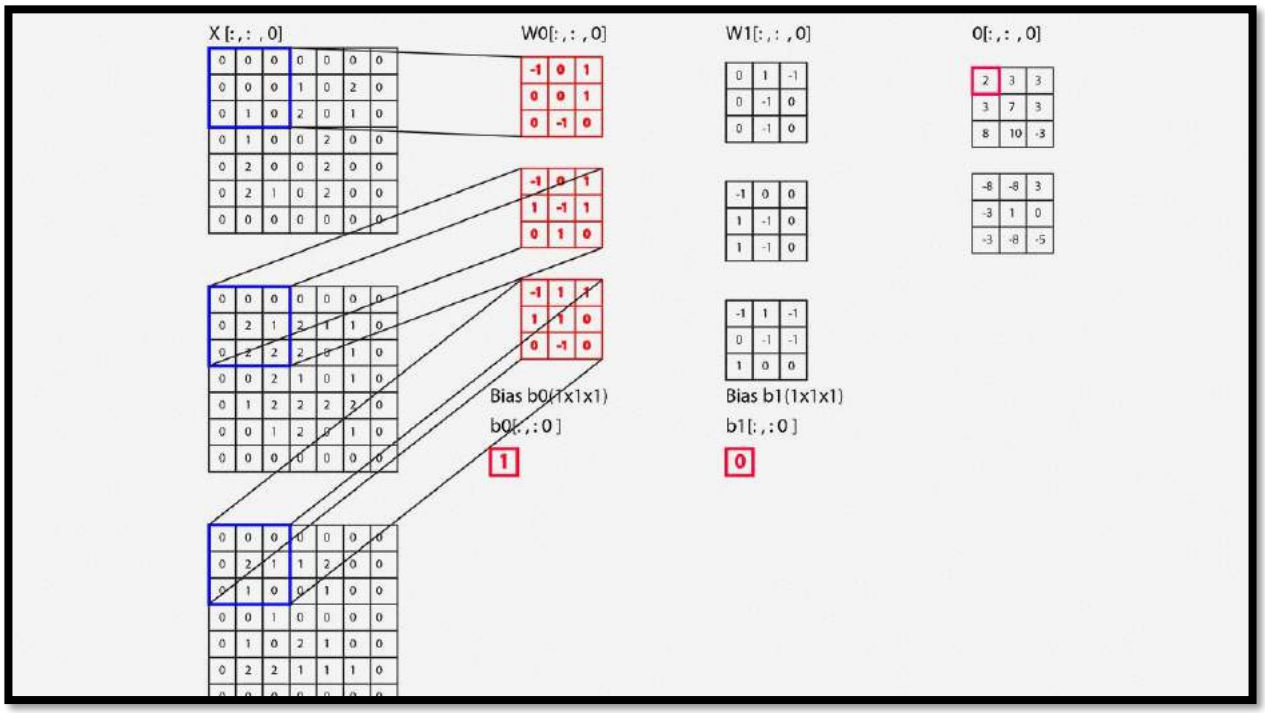
We feed the input images after preprocessing to our model for training and testing after applying all the operations mentioned above.

The prediction layer estimates how likely the image will fall under one of the classes. So the output is normalized between 0 and 1 and such that the sum of each values in each class sums to 1. We have achieved this using

softmax function.

At first the output of the prediction layer will be somewhat far from the actual value. To make it better we have trained the networks using labeled data. The cross-entropy is a performance measurement used in the classification. It is a continuous function which is positive at values which is not same as labeled value and is zero exactly when it is equal to the labeled value. Therefore we optimized the cross-entropy by minimizing it as close to zero. To do this in our network layer we adjust the weights of our neural networks. TensorFlow has an inbuilt function to calculate the cross entropy.

As we have found out the cross entropy function, we have optimized it using Gradient Descent in fact with the best gradient descent optimizer is called Adam Optimizer.



```
#!/usr/bin/env python
# coding: utf-8

# ## Importing Libraries

import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers,models
from tensorflow.keras.models import Sequential


# ## Checking Data

import pathlib
data_dir = "data/"
data_dir = pathlib.Path(data_dir)

image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)

output:26000

gesture_Z = list(data_dir.glob('Z/*'))
PIL.Image.open(str(gesture_Z[0]))

Output:
```

```
# ## Image and Batch Configuration

batch_size = 32
img_height = 128
img_width = 128


# ## Training Data

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  color_mode="grayscale",
  subset="training",
```

```
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

output:
Found 26000 files belonging to 26 classes.
Using 20800 files for training.


# ## Validation Data (20%)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  color_mode="grayscale",
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

output:
Found 26000 files belonging to 26 classes.
Using 5200 files for validation.


# ## Actual Labels
class_names = train_ds.class_names
print(class_names)

output:
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']

# ## Dataset Dimensions

for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
(32, 128, 128, 1)
(32,)


# ## Building CNN Model
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)

normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))

output:
0.0 1.0

num_classes = 26
model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height,
img_width, 1)),
  layers.Conv2D(20, (5,5), padding='same', activation='relu'),
```

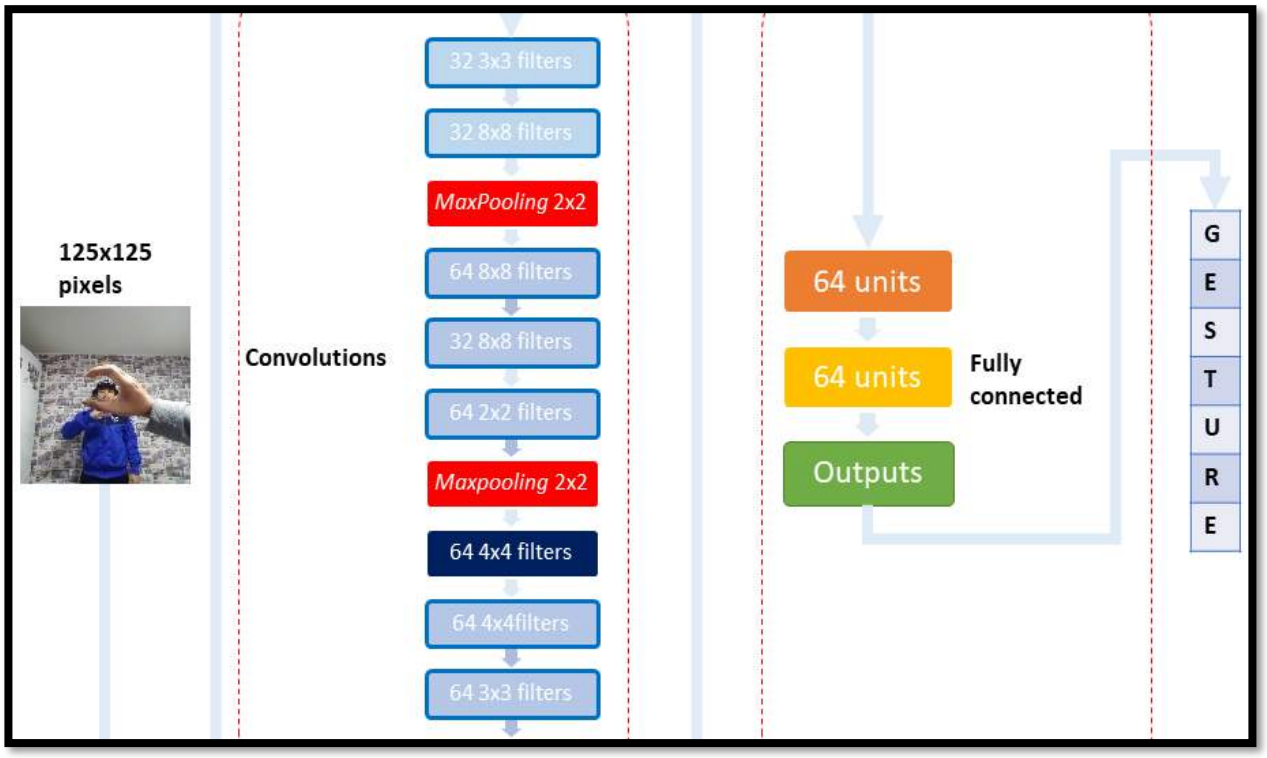
```
  layers.MaxPooling2D(3,3),
  layers.Conv2D(32, (5,5), padding='same', activation='relu'),
  layers.MaxPooling2D(6,6),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(26, activation='relu'),
  layers.Dense(num_classes)
])

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.summary()



# ## Training Model

epochs = 5
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
Epoch 1/5
650/650 [==============================] - 230s 354ms/step - loss: 0.9529 -
accuracy: 0.7290 - val_loss: 0.0022 - val_accuracy: 0.9996
Epoch 2/5
650/650 [==============================] - 222s 341ms/step - loss: 0.0073 -
accuracy: 0.9980 - val_loss: 7.2559e-04 - val_accuracy: 0.9998
Epoch 3/5
650/650 [==============================] - 239s 366ms/step - loss: 9.8578e-04
- accuracy: 0.9998 - val_loss: 4.5500e-05 - val_accuracy: 1.0000
Epoch 4/5
650/650 [==============================] - 230s 353ms/step - loss: 9.8371e-05
- accuracy: 1.0000 - val_loss: 1.1324e-05 - val_accuracy: 1.0000
Epoch 5/5
650/650 [==============================] - 237s 364ms/step - loss: 8.7645e-04
- accuracy: 0.9998 - val_loss: 6.5473e-05 - val_accuracy: 1.0000


# ## Saving Model

model.save('saving_model/myModel.h5')
INFO:tensorflow:Assets written to: saving_model/myModelNative\assets


# ## Visualization

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)
```

```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Model Summary:

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_15 (Rescaling)     (None, 128, 128, 1)       0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 128, 128, 20)      520
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 42, 42, 20)        0
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 42, 42, 32)        16032
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 7, 7, 32)          0
_________________________________________________________________
dropout_14 (Dropout)         (None, 7, 7, 32)          0
_________________________________________________________________
flatten_14 (Flatten)         (None, 1568)              0
_________________________________________________________________
dense_28 (Dense)             (None, 26)                40794
_________________________________________________________________
dense_29 (Dense)             (None, 26)                702
=================================================================
Total params: 58,048
Trainable params: 58,048
Non-trainable params: 0
_________________________________________________________________
```

Sign Language Prediction:



Sign Language Predictor:

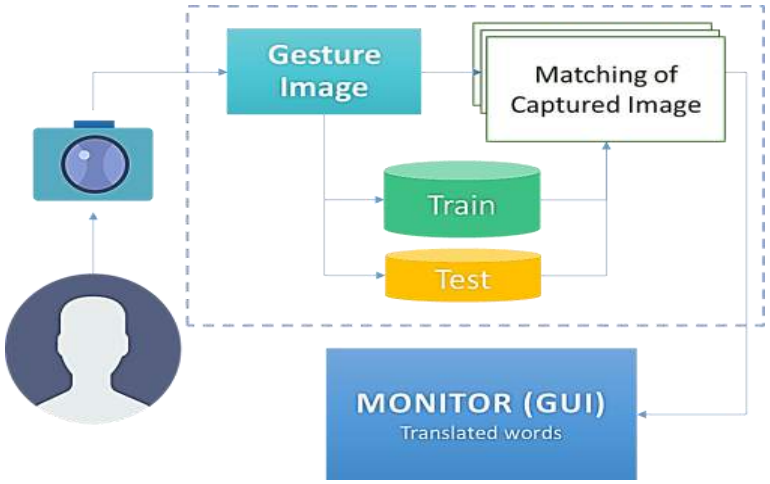
```
import matplotlib.pyplot as plt
import numpy as np
import os
import cv2
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

# Function to Extract useful Features from ROI by applying Different Filters and
Techniques
def convert(frame):
    minValue = 70
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(5,5),2)

    th3 =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV
,11,2)
    ret, res = cv2.threshold(th3, minValue, 255,
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    return res

# Loading the Trained CNN Model
model = tf.keras.models.load_model('saving_model/myModel.h5')

# Original Labels
class_names=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
'O', 'P', 'Q', 'R','S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']


# Function to produce Label using CNN Model and feeding ROI
def prediction(image):
    img_array = keras.preprocessing.image.img_to_array(image)
    img_array = tf.expand_dims(img_array, 0) # Create a batch

    predictions = model.predict(img_array)
    score = tf.nn.softmax(predictions[0])
    return class_names[np.argmax(score)]



capture = cv2.VideoCapture(0)

while True:
    ret,frame=capture.read()
    frame=cv2.flip(frame,1)
    if ret==False:
        continue

    cv2.putText(frame, f'--- SIGN LANGUAGE ---', (10, 30), cv2.FONT_HERSHEY_PLAIN,
1, (0,255,255), 2)

    # Coordinates of the ROI
```

```
    x1 = int(0.5*frame.shape[1])
    y1 = 10
    x2 = frame.shape[1]-10
    y2 = int(0.5*frame.shape[1])

    # Drawing the ROI
    # The increment/decrement by 1 is to compensate for the bounding box
    cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (0,255,0) ,1)

    # Extracting the ROI
    roi = frame[y1:y2, x1:x2]
    roi = cv2.resize(roi, (128, 128))

    #Extracing and Converting
    roi=convert(roi)
    cv2.imshow("ROI", roi)

    #Display Prediction
    cv2.putText(frame, prediction(roi), (10, 120), cv2.FONT_HERSHEY_PLAIN, 3,
(0,255,255), 2)
    cv2.imshow("Frame", frame)

    #Detecting Key Interrupts
    interrupt = cv2.waitKey(10)

    if interrupt & 0xFF == 27: # esc key
        break




capture.release()
cv2.destroyAllWindows()
```

Testing Procedure:

Before the actual recognition of the signs, the user must calibrate the light to ensure that the skin masking of the handis detected and has less noise; the calibration can be done bymoving the lampshade sideways. It is recommended that the light is not directly hitting the hand. The system is sensitive tolight; thus, determining the proper place of the lamp should be considered. If the edges of the hand in the masking are detected clearly, the user may begin to use the translator. For the signs to be recognized, the hand should in front ofthe camera. The detection can only be done if the hand is inside the box that can be seen on the screen of a computer'smonitor. Since the size of the hand of each individual is different, a user may move his/her hand back and forth to fit inside the virtual box. The user should then wait for the system to generate the desired equivalent of the signs in textual form. It also recommended that the user's hand does not make any movement until the system generates the output.

Conceptual Framework of the System:



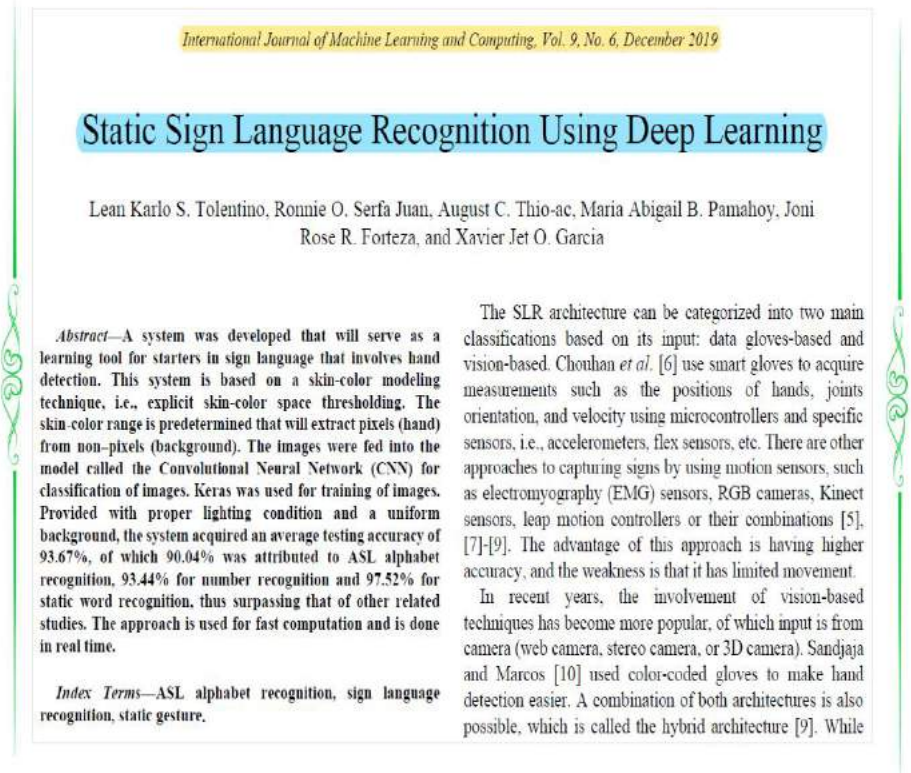
Results and Discussion:

Final Output:

There were many challenges faced by us during the project. The very first issue we faced was of dataset. We wanted to deal with raw images and that too square images as CNN in Keras as it was a lot more convenient working with only square images. We couldn't find any existing dataset for that hence we decided to make our own dataset. Second issue was to select a filter which we could apply on our images so that proper features of the images could be obtained and hence then we could provided that image as input for CNN model. We tried various filter including binary threshold, canny edge detection, gaussian blur etc. but finally we settled with gaussian blur filter. More issues were faced relating to the accuracy of the model we trained in earlier phases which we eventually improved by increasing the input image size and also by improving the dataset.

MODEL HIGHLIGHTS

RESEARCH WORK FOR CNN



International Journal of Machine Learning and Computing, Vol. 9, No. 6, December 2019

Static Sign Language Recognition Using Deep Learning

Lean Karlo S. Tolentino, Ronnie O. Serfa Juan, August C. Thio-ac, Maria Abigail B. Pamahoy, Joni Rose R. Forteza, and Xavier Jet O. Garcia

Abstract—A system was developed that will serve as a learning tool for starters in sign language that involves hand detection. This system is based on a skin-color modeling technique, i.e., explicit skin-color space thresholding. The skin-color range is predetermined that will extract pixels (hand) from non-pixels (background). The images were fed into the model called the Convolutional Neural Network (CNN) for classification of images. Keras was used for training of images. Provided with proper lighting condition and a uniform background, the system acquired an average testing accuracy of 93.67%, of which 90.04% was attributed to ASL alphabet recognition, 93.44% for number recognition and 97.52% for static word recognition, thus surpassing that of other related studies. The approach is used for fast computation and is done in real time.

Index Terms—ASL alphabet recognition, sign language recognition, static gesture.

The SLR architecture can be categorized into two main classifications based on its input: data gloves-based and vision-based. Chouhan *et al.* [6] use smart gloves to acquire measurements such as the positions of hands, joints orientation, and velocity using microcontrollers and specific sensors, i.e., accelerometers, flex sensors, etc. There are other approaches to capturing signs by using motion sensors, such as electromyography (EMG) sensors, RGB cameras, Kinect sensors, leap motion controllers or their combinations [5], [7]-[9]. The advantage of this approach is having higher accuracy, and the weakness is that it has limited movement.

In recent years, the involvement of vision-based techniques has become more popular, of which input is from camera (web camera, stereo camera, or 3D camera). Sandjaja and Marcos [10] used color-coded gloves to make hand detection easier. A combination of both architectures is also possible, which is called the hybrid architecture [9]. While

1.DATA SET:


FEATURES AUTOMATICALLY EXTRACTED USING SCALING AND FILTERS
CAPTURING RAW IMAGE
GRAY SCALING
FILTERING


RESULT :
SELF GENERATABLE
AND
USER-UPDATEABLE
DATA SET.

2.DROPOUTS:

Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.

Editor: Yoshua Bengio

Abstract

Deep neural nets with a large number of parameters are very powerful machine learning

Editor: Yoshua Bengio

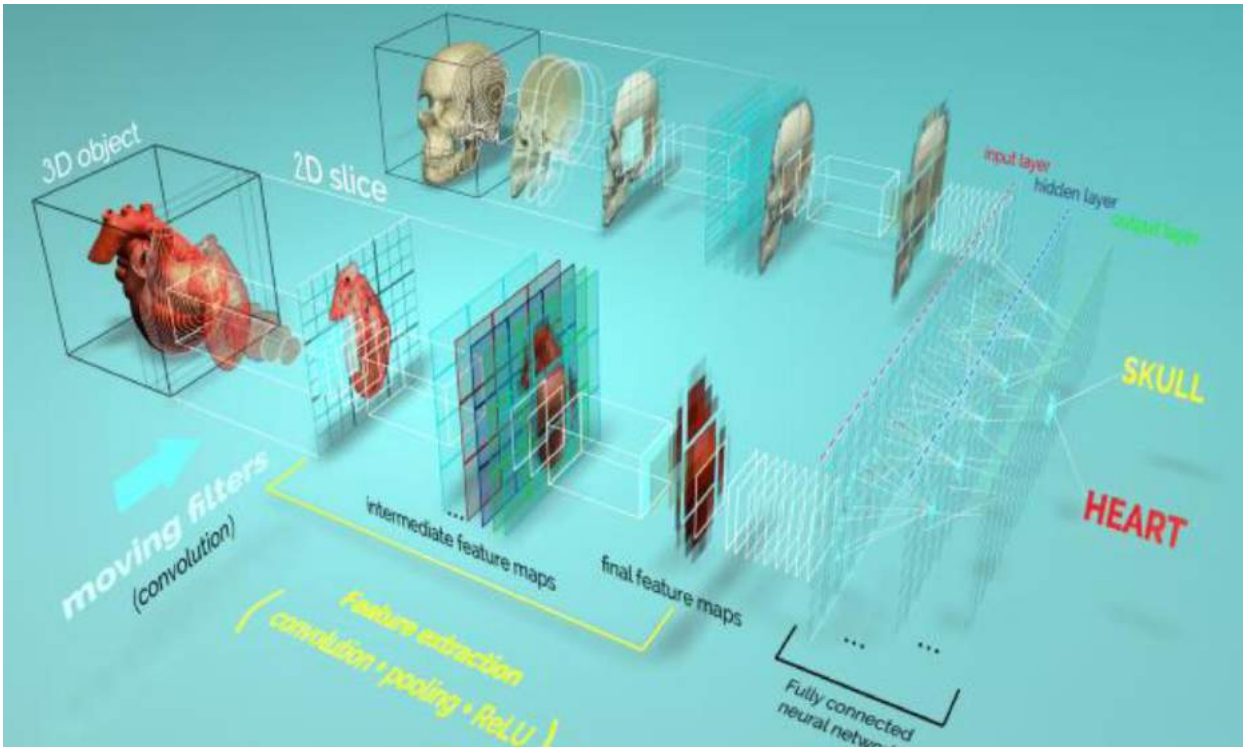
Abstract

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

Keywords: neural networks, regularization, model combination, deep learning

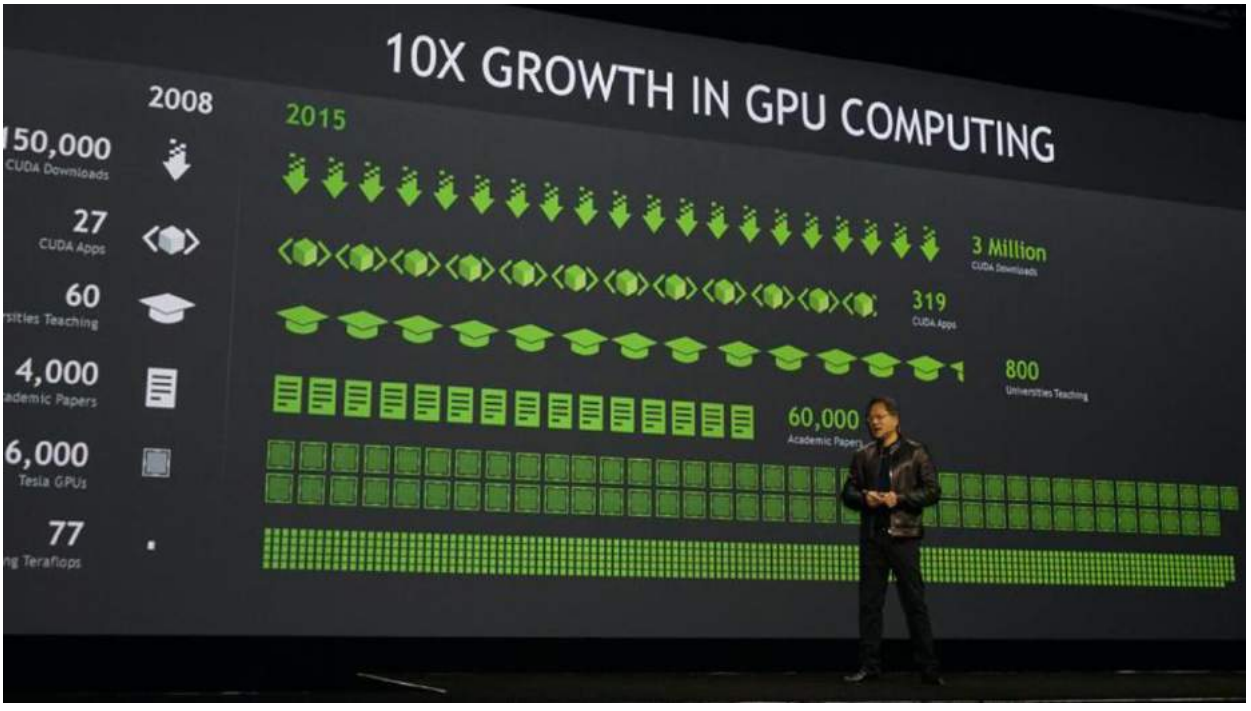
1. Introduction

Deep neural networks contain multiple non-linear hidden layers and this makes them very expressive models that can learn very complicated relationships between their inputs and outputs. With limited training data, however, many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data even if it is drawn from the same distribution. This leads to overfitting and many methods have been developed for reducing it. These include stopping the training as soon as

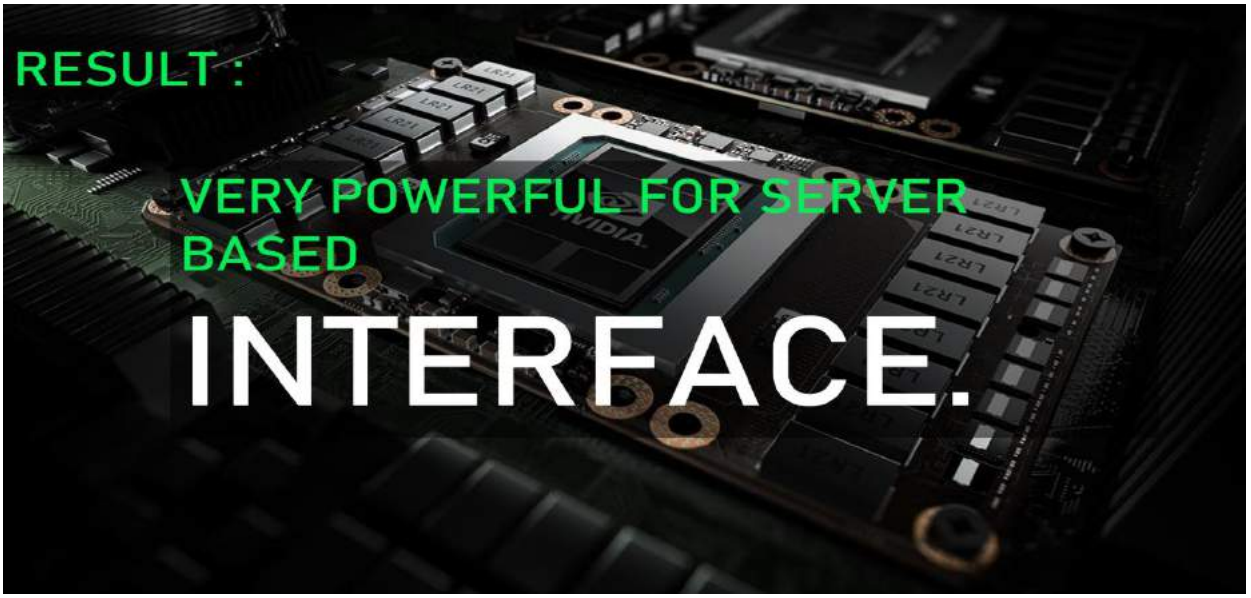

3D object
2D slice
input layer
hidden layer
SKULL
HEART
moving filters
(convolution)
intermediate feature maps
final feature maps
Feature extraction
(convolution + pooling + ReLU)
Fully connected


RESULT :
4X
EFFICIENY
IN FEATURE DETECTION

4.CuDNN: CUDA FOR DEEP NEURAL NETWORK




RESULT :
HIGHLY
SCALABLE
SOLUTION.


RESULT :
VERY POWERFUL FOR SERVER
BASED
INTERFACE.


RESULT :
UPTO
12000%
FASTER
TO
TRAIN .

Conclusion :

We have achieved an accuracy of 92.3% in our model . One thing should be noted that our model doesn’t uses any background subtraction algorithm whiles some of the models do. So once we try to implement background subtraction in our project the accuracies may vary. On the other hand most of the sign language projects use kinect devices but our main aim was to create a project which can be used with readily available resources. A sensor like kinect not only isn’t readily available but also is expensive for most of audience to buy and our model uses a normal webcam of the laptop

References :

[1]https://www.researchgate.net/publication/337285019_Static_Sign_Language_Recognition_Using_Deep_Learning

[2] Mohammed Waleed Kalous, Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language.

[3] aeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convol utional-Neural-Networks-Part-2/

[4] http://www-i6.informatik.rwth-aachen.de/~dreuw/database.php

[5] Pigou L., Dieleman S., Kindermans PJ., Schrauwen B. (2015) Sign Language Recognition Using Convolutional Neural Networks. In: Agapito L., Bronstein M., Rother C. (eds) Computer Vision - ECCV 2014 Workshops. ECCV 2014. Lecture Notes in Computer Science, vol 8925.
Springer, Cham

[6] Zaki, M.M., Shaheen, S.I.: Sign language recognition using a combination of new vision based features. Pattern Recognition Letters 32(4), 572–577 (2011)

[7] N. Mukai, N. Harada and Y. Chang, "Japanese Fingerspelling Recognition Based on Classification Tree and Machine Learning," *2017 Nicograph International (NicoInt)*, Kyoto, Japan, 2017, pp. 19-24.
doi:10.1109/NICOInt.2017.9

[8] Byeongkeun Kang , Subarna Tripathi , Truong Q. Nguyen ”Real-time sign language fingerspelling recognition using convolutional neural networks from depth map” 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)

[9] https://opencv.org/

[12] https://en.wikipedia.org/wiki/TensorFlow

[13] https://en.wikipedia.org/wiki/Convolutional_neural_network

[14] https://developer.nvidia.com/cudnn

[15] https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Technologies Used: Python, Convolutional Neural Network (CNN), OpenCV, TensorFlow, Keras.