A PROJECT REPORT

on

Sign language to speech

Submitted to

KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE IN
INFORMATION TECHNOLOGY

BY

Sachin Kumar	21051162
Abhinav Bisht	21051704
Aditya Raj	21051709
Arnav Kumar dey	21051801
Nitin Kumar	21051828

UNDER THE GUIDANCE OF
Dr. Banchhanidhi Dash

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024

May 2020

KIIT Deemed to be University

School of Computer Engineering
Bhubaneswar, ODISHA 751024

CERTIFICATE

This is certify that the project entitled

Sign language to speech

submitted by

Sachin Kumar	21051162
Abhinav Bisht	21051704
Aditya Raj	21051709
Arnav Kumar	21051801
Nitin Kumar	21051828

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering OR Information Technology) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2023-24, under our guidance.

Date: 07 / 04 / 2024

Dr. Banchhanidhi Dash
Project Guide

Acknowledgements

We are profoundly grateful to **Dr. Banchhanidhi Dash** of **Affiliation** for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion. .....

Sachin kumar
Abhinav Bisht
Aditya Raj
Arnav kumar dey
Nitin kumar

ABSTRACT

This project explores the application of Convolutional Neural Networks (CNN) in facilitating real-time translation from sign language to speech, aimed at enhancing communication accessibility for individuals with hearing impairments. Leveraging the power of deep learning, specifically utilizing the Keras framework, our study focuses on developing a robust and efficient model capable of accurately recognizing and interpreting a wide range of sign language gestures.

The objective of this project is to bridge the communication gap between individuals proficient in sign language and those who rely on spoken language, thereby fostering inclusivity and accessibility. To achieve this, we employed a dataset comprising diverse sign language gestures, meticulously curated to encompass various hand configurations, movements, and facial expressions.

Our methodology involves preprocessing the dataset to enhance feature extraction, followed by the implementation of a CNN architecture within the Keras framework. Through extensive training and optimization, the model learns to discern intricate patterns and features characteristic of different sign language gestures, facilitating accurate classification.

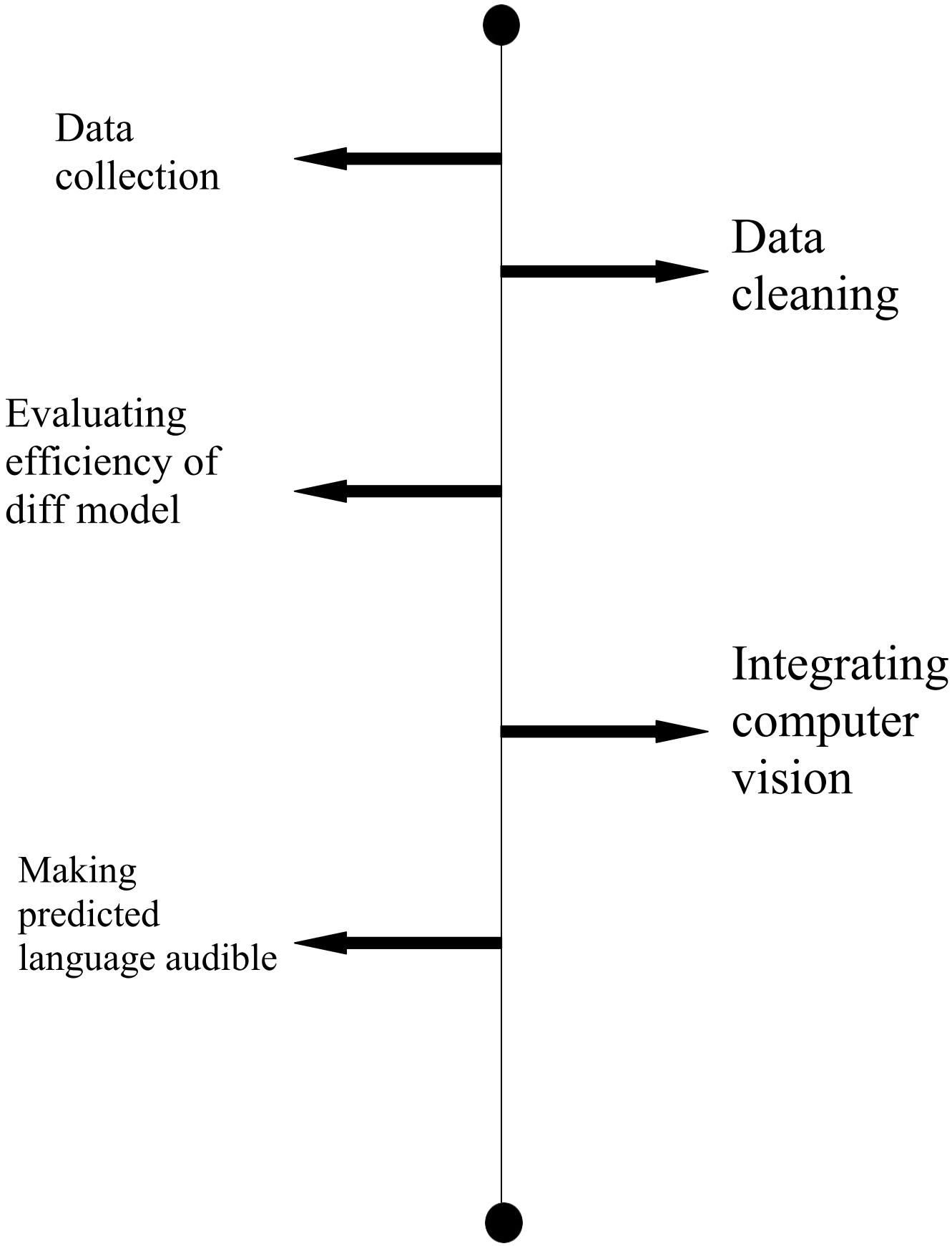
Evaluation of the model's performance showcases promising results, with high levels of accuracy achieved in real-world scenarios. Furthermore, our project emphasizes the importance of user-centric design, with a focus on developing intuitive interfaces and seamless integration into existing communication devices.

The implications of this research extend beyond the realm of academia, offering practical solutions for empowering individuals with hearing impairments to communicate effectively and independently in various social and professional settings. By harnessing the capabilities of machine learning and CNNs, our project contributes to advancing assistive technology and promoting inclusivity in society.

Keywords: Sign language, Speech recognition, Convolutional Neural Networks (CNN), Keras, OpenCV ,Accessibility, Deep Learning, Communication, Assistive Technology.

Content :-

Road-map


Data collection
Data cleaning
Evaluating efficiency of diff model
Integrating computer vision
Making predicted language audible

Keywords

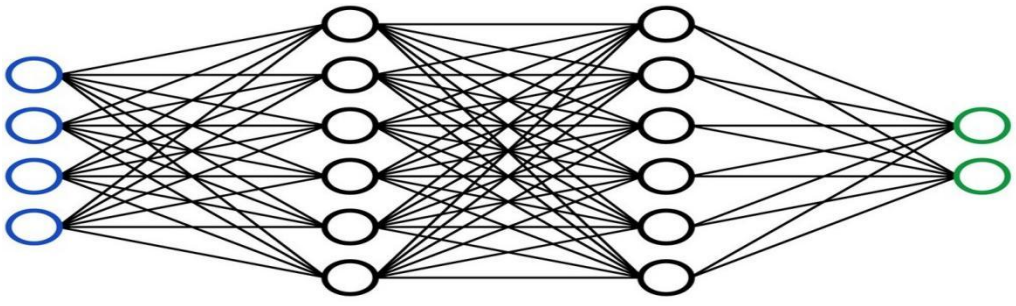
{ **TensorFlow** is an open-source machine learning framework developed by Google. It enables developers to build and deploy machine learning models efficiently, utilizing dataflow graphs to represent computational tasks. With extensive support for deep learning, TensorFlow is widely used for tasks like image recognition, natural language processing, and more.}[2]

OpenCV :- OpenCV (Open Source Computer Vision Library) is a popular open-source computer vision and machine learning software library. It provides a wide range of functionalities for tasks such as image and video processing, object detection and tracking, feature extraction, and more. OpenCV is widely used in various industries, including robotics, healthcare, automotive, and entertainment.

Keras :- Keras is a high-level neural networks API written in Python and capable of running on top of TensorFlow, Theano, or Microsoft Cognitive Toolkit. It provides a user-friendly interface for building and training deep learning models with minimal code. Keras emphasizes simplicity, flexibility, and ease of use, making it popular among both beginners and experts in machine learning.

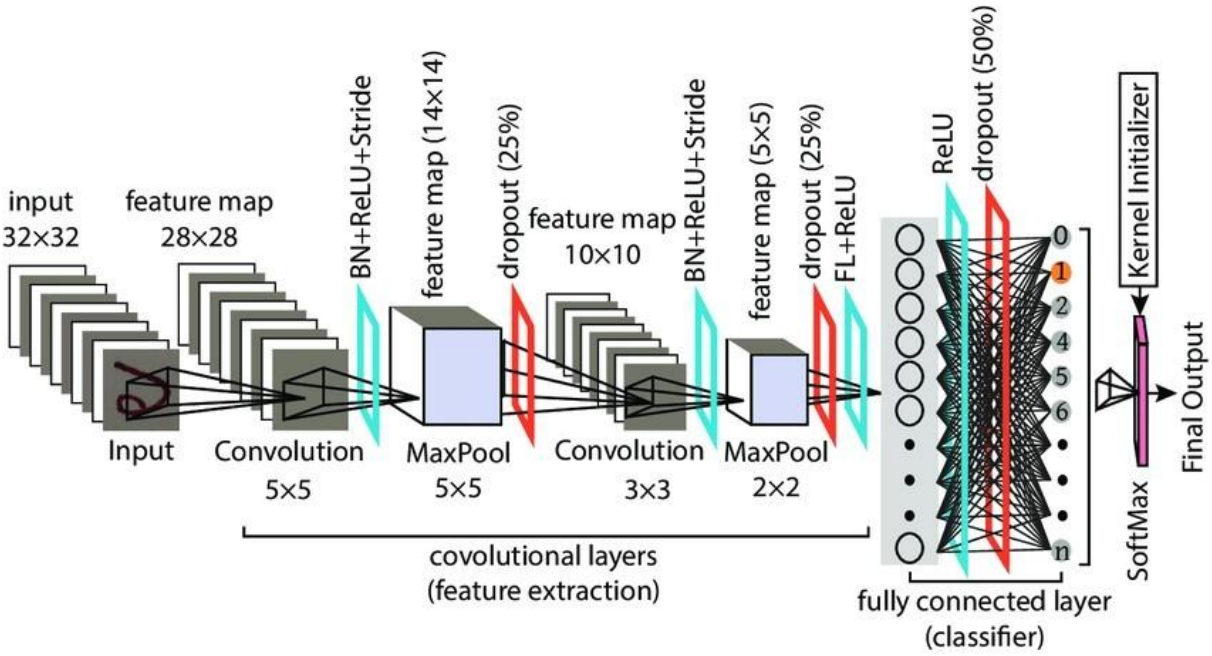
NumPy :- NumPy is a powerful Python library for numerical computing. It provides support for multidimensional arrays, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy is widely used in scientific computing, data analysis, and machine learning due to its speed, versatility, and ease of use.

Neural Network :- A neural network is a computational model inspired by the structure and functioning of the human brain. It consists of interconnected nodes, or neurons, organized in layers. Each neuron processes input data, applies a mathematical operation, and produces an output. Through training, neural networks learn to recognize patterns and make predictions from data, making them powerful tools for tasks like image recognition, natural language processing, and more.

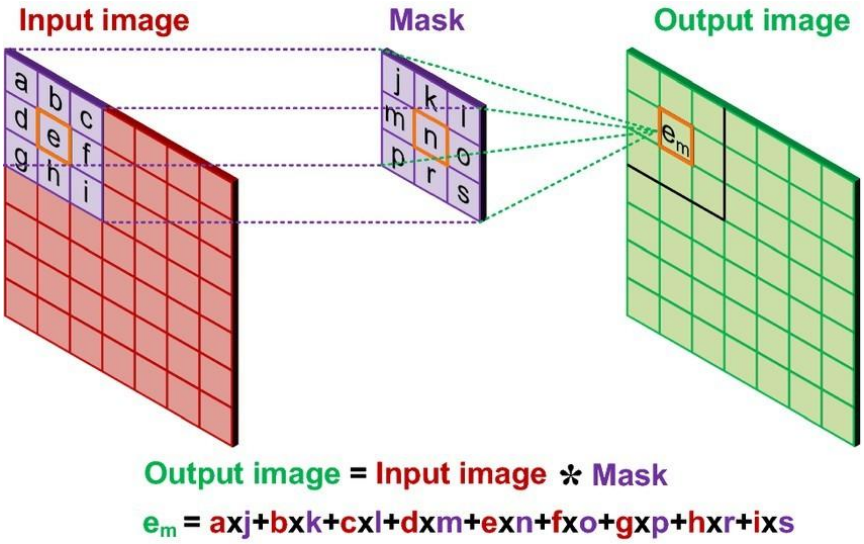


Sequential model :- A sequential model is a type of neural network architecture in which layers are stacked sequentially, one after the other. Each layer in a sequential model receives input from the previous layer and passes its output to the next layer. Sequential models are simple to create and easy to understand, making them suitable for many common deep learning tasks. They are often used in scenarios where the data flows in a linear manner through the network, such as in simple feedforward neural networks. Keras, a popular deep learning library, provides support for building sequential models..

Convolutional Neural Network (CNN) :- A Convolutional Neural Network (CNN) is a type of neural network architecture commonly used for image recognition and classification tasks. CNNs are designed to automatically and adaptively learn spatial hierarchies of features from input images through the application of convolutional layers. These layers use filters to extract local patterns and features, which are then combined and processed by subsequent layers to make predictions. CNNs are particularly effective for tasks like object detection, facial recognition, and medical image analysis. They have revolutionized computer vision and are widely used in various applications due to their ability to capture complex patterns in images.

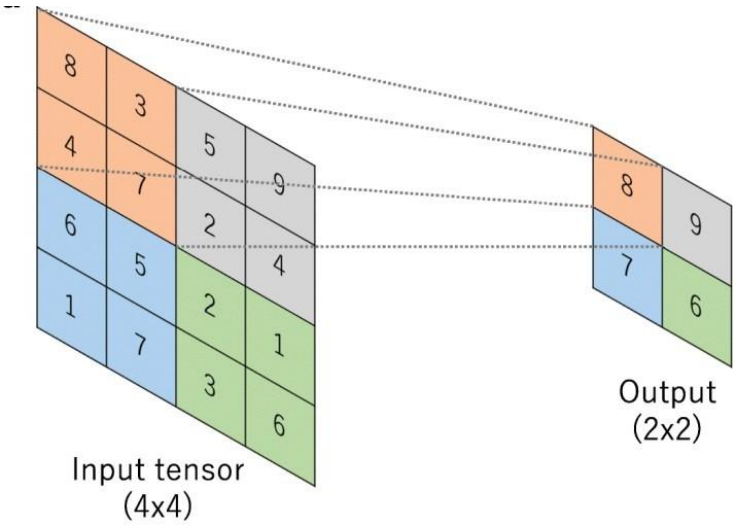


Convolution :- Convolution is a mathematical operation that combines two functions to produce a third function. In the context of Convolutional Neural Networks (CNNs), convolution refers to the process of applying a filter (also known as a kernel) to an input image to produce a feature map. This operation involves sliding the filter over the input image and computing the dot product between the filter and the overlapping region of the image at each position. Convolution plays a crucial role in CNNs by extracting spatial patterns and features from input images, enabling the network to learn hierarchical representations of the data.



{ **Max pooling** :- Max pooling is a downsampling operation used in Convolutional Neural Networks (CNNs) to reduce the dimensionality of feature maps while retaining the most important information. It involves partitioning the input feature map into non-overlapping regions and selecting the maximum value from each region. This process effectively reduces the spatial size of the feature map, making subsequent operations more computationally efficient and reducing overfitting by providing a form of spatial invariance. Max pooling is typically applied after convolutional layers in CNN architectures.

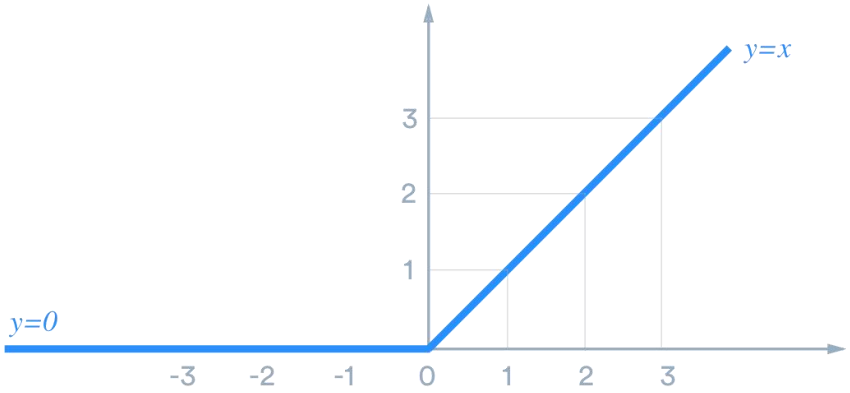
}6



Activation function :- An activation function is a mathematical operation applied to the output of a neuron in a neural network. It introduces non-linearity into the network, allowing it to learn complex patterns in data.

Common activation functions include :-

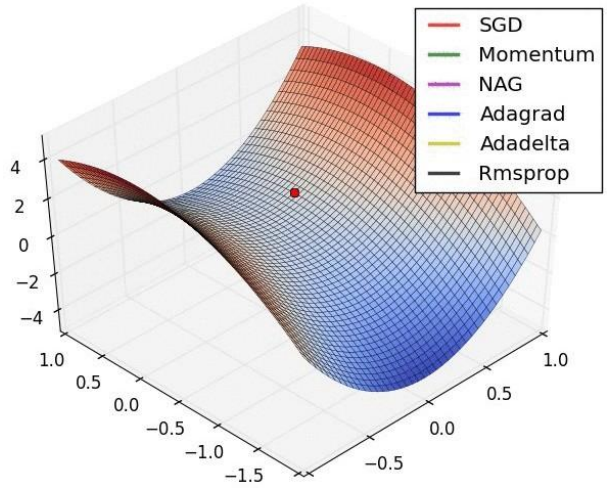
- **ReLU (Rectified Linear Activation):** commonly used due to its simplicity and effectiveness in training deep neural networks.



- ***Softmax****:used in the output layer of multi-class classification tasks to obtain probability distributions over multiple classes.*

Optimizer :- An optimizer is an algorithm used to minimize the loss function during the training of a neural network. It adjusts the parameters (weights and biases) of the network based on the gradients of the loss function with respect to those parameters.

- ***Adam****: Adaptive Moment Estimation, combines the advantages of both AdaGrad and RMSProp. It maintains per-parameter learning rates that are adapted based on the estimates of first and second moments of the gradients.*



Loss :- Loss, also known as cost or objective function, is a measure of the difference between the predicted output of a neural network and the actual target output. It quantifies how well the network is performing on a specific task during training.

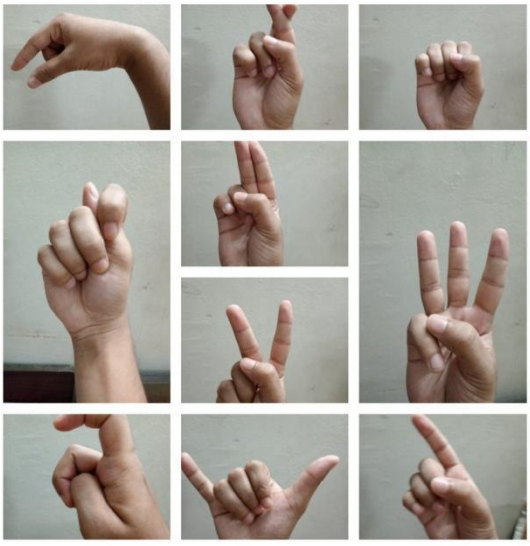
- ***Log Loss:*** *Binary Cross-Entropy (or Log Loss) measures the difference between predicted probabilities and true binary labels..*

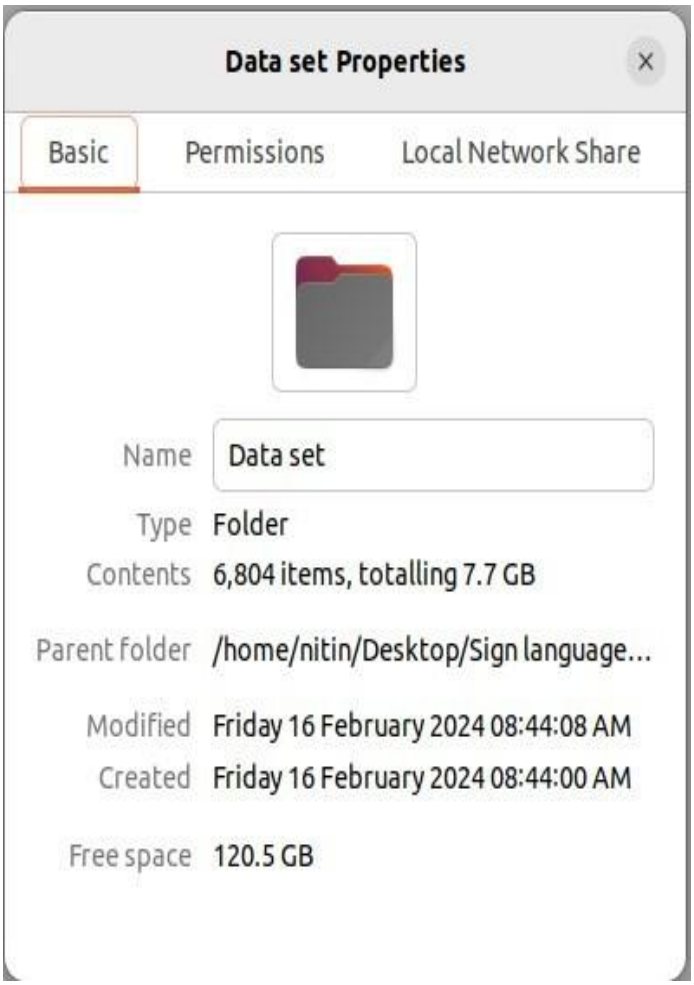
Matrices :- Matrices are rectangular arrays of numbers arranged in rows and columns. They are fundamental mathematical objects used in various fields, including computer science, physics, engineering, and machine learning. In the context of machine learning and neural networks, matrices are commonly used to represent data, weights, biases, and activations.

Data set Generation

The dataset used in this project consists of a diverse collection of sign language gestures captured in various settings and contexts. It includes video recordings of individuals performing sign language gestures, covering a wide range of vocabulary and expressions commonly used in everyday communication.

We have captured pictures of each alphabet, resulting in a total of 6,508 images captured. To introduce diversity and variation into the dataset, different individuals were asked to demonstrate the following symbols, ensuring a broad range of hand shapes, movements, and expressions in each image.





Size :

Applying the following shell command we have rename the file is serial order. The provided shell command iterates through each file in the directory, renames them in sequential order, and updates the filenames accordingly.

Each class in the dataset consists of approximately 250 images, resulting in a total of 6,508 images captured. Prior to processing, the dimensions of each image are 4624 pixels in width by 2080 pixels in height, with three color channels (RGB), representing the red, green, and blue color components. These images provide detailed visual representations of the alphabet signs, enabling accurate recognition and interpretation by machine learning models.

```
count=1
for file in *; do
    if [[ -f "$file" ]]; then
        extension="${file##*.}"
        mv "$file" "$count.$extension"
        ((count++))
    fi
done
```

The code work in the following manner :-

- count=1: Initializes a variable count with the value 1. This variable will be used to generate numerical prefixes for the files.
- for file in *; do ... done: Loops through each file in the current directory. The * wildcard matches all files in the directory.
- if [[-f "$file"]]; then ... fi: Checks if the current item in the loop is a regular file. The -f flag is used to test if the file exists and is a regular file.
- extension="${file##*.}": Extracts the file extension from the current file name using bash parameter expansion. This syntax removes everything before the last dot in the file name, leaving only the extension.
- mv "$file" "$count.$extension": Renames the current file by moving it to a new name consisting of the count variable followed by a dot and the original file extension. The mv command is used for renaming files in Unix-like systems.

- ((count++)): Increments the count variable by 1 after each file is renamed. This ensures that each file receives a unique numerical

Data Processing

Overview :

In our efforts to eliminate noise from the dataset, we utilized OpenCV to convert the images to grayscale, standardizing their format for subsequent processing.
Additionally, we resized each image to a uniform size of 480x640 pixels to ensure consistency across the dataset. Following grayscale conversion and resizing, we applied a Gaussian blur filter with a kernel size of 5x5 to reduce noise and smooth out irregularities in the images. we applied an adaptive thresholding technique to further enhance the contrast between the foreground and background, effectively segmenting the hand gestures from the surrounding noise.After applying this sequence of image processing operations, we observed significant noise reduction and improved clarity in the hand gesture images.

Implementation :

```
[1]: import cv2
     import os

     def convert_to_grayscale(input_path, output_path):
         image = cv2.imread(input_path)
         resized_image = cv2.resize(image, (128, 128))
         gray = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
         Gaussian = cv2.GaussianBlur(gray ,(5,5) , 2 )

         th3 = cv2.adaptiveThreshold(Gaussian,
      255 ,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,
      11,2)
         ret, res = cv2.threshold(th3, 70, 255,
      cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
         cv2.imwrite(output_path, res)

     def convert_images_in_folder(folder_path, output_folder):
         if not os.path.exists(output_folder):
             os.makedirs(output_folder)

         for root, dirs, files in os.walk(folder_path):
             for filename in files:
                 if filename.endswith(('.jpg', '.jpeg', '.png')):
                     input_path = os.path.join(root, filename)
                     output_path = os.path.join(output_folder, os.path.
      relpath(input_path, folder_path))
                     output_dir = os.path.dirname(output_path)
                     os.makedirs(output_dir, exist_ok=True)
                     convert_to_grayscale(input_path, output_path)

     if__name__== "__main__":
         folder_path = 'Data set'
         output_folder = 'Data set edited'

         convert_images_in_folder(folder_path, output_folder)
```

Filter applied :

- **Resize :** This function is used to resize images to a specified width and height. This function allows you to scale images up (enlargement) or down (reduction) while preserving the aspect ratio or specifying a new aspect ratio if desired.
  Syntax:
  resized_image = cv2.resize(image, (width, height))

- **Grayscale :** Grayscale conversion in OpenCV is a process where an image, which typically consists of three color channels RGB is transformed into a single-channel image representing the intensity of each pixel. This intensity is often described as the brightness or luminance of thecorresponding color in the primary img.
  Syntax:
  grayimm = cv2.cvtColor(imm, cv2.COLOR_BGR2GRAY)

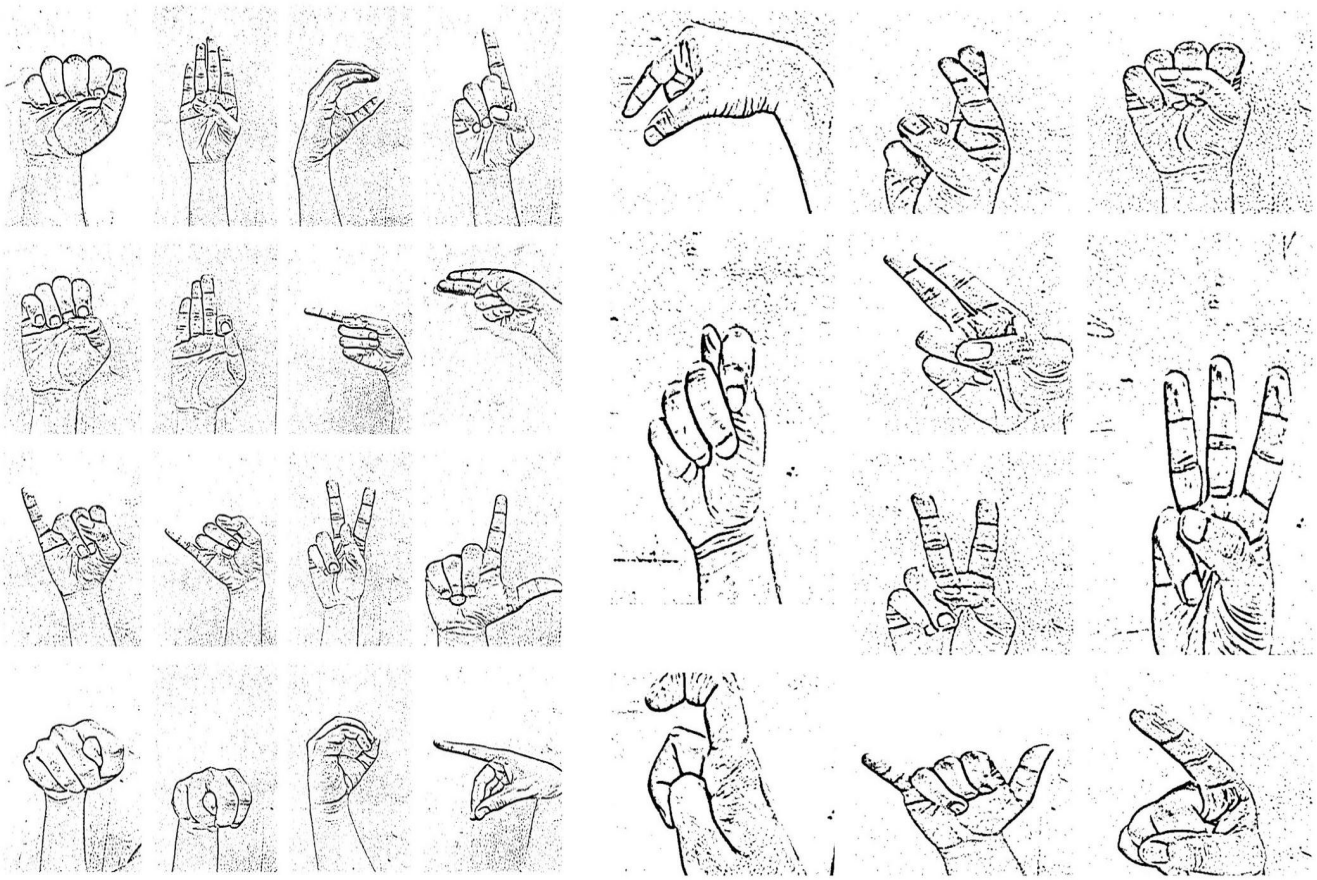
- **GaussianBlur :** This applies a Gaussian smoothing filter to an image, reducing noise and detail while preserving edges. The function takes parameters for kernel size and standard deviation. Kernel size must be odd and positive, determining the extent of blurring. Standard deviation controls the amount of blur applied; higher values result in more blur.
  Syntax:
  blurred_image = cv2.GaussianBlur(image, (kernel_size_width, kernel_size_height), sigma_x)

- **adaptive thresholding :** This is used to apply adaptive thresholding to an image, adjusting the threshold value for each pixel based on local image characteristics. Parameters include the input image, maximum pixel value, adaptive method, threshold type, block size, and constant. It's commonly employed in binarization tasks to segment images with varying illumination.
  Syntax:
  retval, dst = cv2.adaptiveThreshold(src, mae, Method,tType, Size, C)

Data after processing :

After applying the following sequence of filters – resizing, grayscale conversion, Gaussian blur, and adaptive thresholding – our data resembles the preprocessed form. This transformation enhances the dataset's uniformity, reduces noise, and emphasizes essential features for subsequent analysis or processing tasks.

The images looks as follow :-



Test and train split of data

Overview :

{ Train-test split divides a preprocessed dataset into two subsets: training and testing. Comprising a majority of the data, is used to train the machine learning model. The test set, remaining unseen during training, evaluates the model'sperformance on new data. This process ensures the model's ability to generalize to unseen instances. Evaluation metrics such as accuracy, precision, and recall assess model performance. Iterative refinement based on evaluation results improves model effectiveness. Additionally, techniques like cross-validation enhance robustness.
Train-test splitting is vital for developing reliable machine learning models with practical applicability.}[4]

Implementation :

```
import os
      import random
      from shutil import copyfile

      def split_data(source, training, testing, split_size):
          if not os.path.exists(training):
              os.makedirs(training)
          if not os.path.exists(testing):
              os.makedirs(testing)

          files = os.listdir(source)
          random.shuffle(files)

          split_index = int(len(files) * split_size)
          training_files = files[:split_index]
          testing_files = files[split_index:]

          for file in training_files:
              source_file = os.path.join(source, file)
              destination_file = os.path.join(training, file)
              copyfile(source_file, destination_file)

          for file in testing_files:
              source_file = os.path.join(source, file)
              destination_file = os.path.join(testing, file)
              copyfile(source_file, destination_file)

      source_dir = 'Data set edited/Z'
      training_dir = 'training_data/Z'
      testing_dir = 'testing_data/Z'

      split_size = 0.8

      split_data(source_dir, training_dir, testing_dir, split_size)

      print("Data split completed.")

     Data split completed.
```

Explanation :

1. Imports:
- os: Operating system module for interacting with the file system.
- random: Module for generating random numbers.
- copyfile: Function from shutil module for copying files.

2. split_data Function:
- Parameters:
  - source: The directory containing the original data.
  - training: The directory where the training data will be copied.
  - testing: The directory where the testing data will be copied.
  - split_size: The proportion of data to be allocated to the training set (e.g., 0.8 for 80% training data and 20% testing data).
- The function first checks if the training and testing directories exist. If not, it creates them using os.makedirs.
- It then lists all the files in the source directory and shuffles them randomly using random.shuffle.
- It calculates the split index based on the split_size.
- It slices the list of files into two parts: training files and testing files.
- For each file in the training set, it copies the file from the source directory to the training directory using copyfile.
- Similarly, for each file in the testing set, it copies the file from the source directory to the testing directory.

3. Directory Paths:
- source_dir: Path to the directory containing the original data.
- training_dir: Path to the directory where training data will be copied.
- testing_dir: Path to the directory where testing data will be copied.

4. Split Size:
- split_size: The proportion of data to be allocated to the training set. In this case, it's set to 0.8, indicating an 80/20 split between training and testing data.

Model prototype

```
[3]:{ model =
        Sequential([ Flatten(input_shape=(4
        80, 640, 3)),Dense(26,
        activation='softmax')
    ])

    model.compile( optimizer='sg
        d',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    datagen =
    ImageDataGenerator(rescale=1./255)
    train_generator =
    datagen.flow_from_directory(
        'training_data',
        target_size=(480, 640),
        batch_size=10,
        class_mode='categorical',
        classes=['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']
    )

    model.fit(train_generator, epochs=15)

    model.save('model_prototype.h5')}7
```

Structure of Neural network used :-

- *No. of neuron 480 * 640 * 3 = 921600*
  *where 480*640 are the dimension of image and 3 is the color channel i.e RGB.*
- *Activation function used is Relu*
- *Optimizer is Stochastic gradient descent*
- *Log loss function is used.*

If we train our model {Neural network} using the sequential model having only two layer i.e Input and output and no hidden layer.

While training the accuracy and loss comes to be 0.9928 and 0.8921 respectively

```
Found 5119 images belonging to 26 classes.
/home/nitin/anaconda3/lib/python3.11/site-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
Epoch 1/15
/home/nitin/anaconda3/lib/python3.11/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
512/512 ━━━━━━━━━━━━━━━━━━━━ 63s 121ms/step - accuracy: 0.0422 - loss: 4008.1897
Epoch 2/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 61s 117ms/step - accuracy: 0.1190 - loss: 2955.7231
Epoch 3/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 59s 115ms/step - accuracy: 0.2293 - loss: 1932.9410
Epoch 4/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 61s 118ms/step - accuracy: 0.4753 - loss: 855.4783
Epoch 5/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 61s 117ms/step - accuracy: 0.6399 - loss: 423.9066
Epoch 6/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 60s 117ms/step - accuracy: 0.7954 - loss: 170.3431
Epoch 7/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 61s 118ms/step - accuracy: 0.9209 - loss: 35.0356
Epoch 8/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 58s 112ms/step - accuracy: 0.9010 - loss: 47.4757
Epoch 9/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 61s 118ms/step - accuracy: 0.9488 - loss: 19.7713
Epoch 10/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 60s 116ms/step - accuracy: 0.9866 - loss: 2.2658
Epoch 11/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 60s 116ms/step - accuracy: 0.9964 - loss: 0.5068
Epoch 12/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 60s 115ms/step - accuracy: 0.9973 - loss: 0.6479
Epoch 13/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 61s 118ms/step - accuracy: 0.9906 - loss: 6.3966
Epoch 14/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 59s 115ms/step - accuracy: 0.9927 - loss: 1.5997
Epoch 15/15
512/512 ━━━━━━━━━━━━━━━━━━━━ 59s 113ms/step - accuracy: 0.9928 - loss: 0.8921
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

While testing the accuracy and loss comes to be 0.9670 and 4.8730 respectively

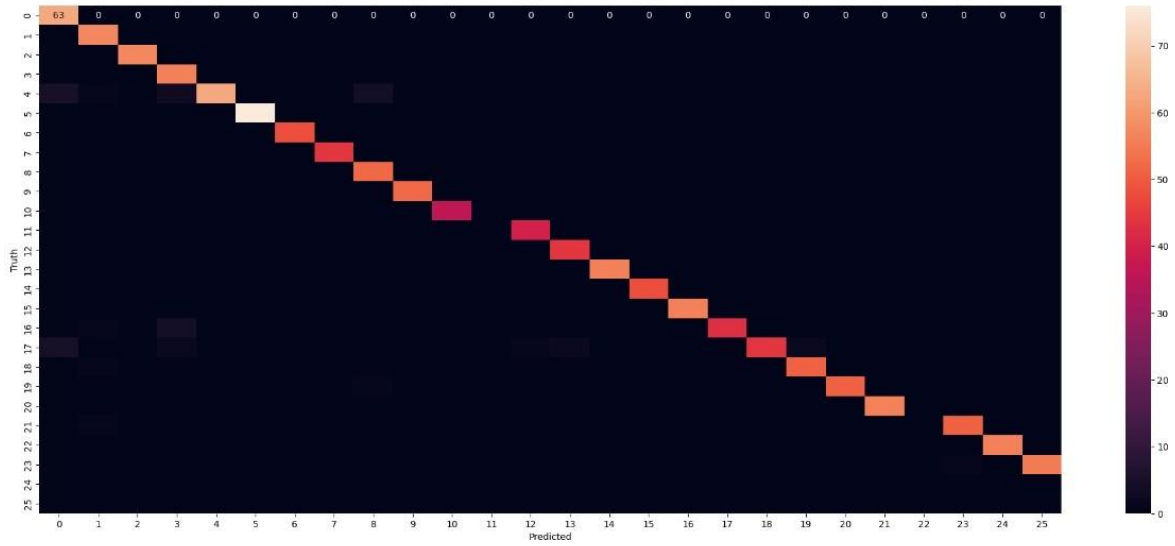
```
datagen = ImageDataGenerator(rescale=1./255)
test_generator = datagen.flow_from_directory(
    'testing_data',
    target_size=(480, 640),
    batch_size=10,
    class_mode='categorical',
    classes=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z
)

loss, accuracy = model.evaluate(test_generator)
```

```
Found 1289 images belonging to 26 classes.
129/129 ━━━━━━━━━━━━━━━━━━━━ 11s 85ms/step - accuracy: 0.9670 - loss: 4.8730
```

The confusion matrix calculated are as follow :-

```
[[63  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0 57  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0 57  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 56  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 5  1  0  3 63  0  0  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0 76  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 48  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 44  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 52  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 52  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 36  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0 40  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0 44  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0 56  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0 48  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 56  0  0  0  0  0  0  0  0  0  0]
 [ 0  1  0  4  0  0  0  0  0  0  0  0  0  0  0  0 43  0  0  0  0  0  0  0  0  0]
 [ 5  0  0  2  0  0  0  0  0  0  0  0  1  2  0  0  0  0 44  2  0  0  0  0  0  0]
 [ 0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 51  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0 51  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 56  0  0  0  0]
 [ 0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 51  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 56  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0 55]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]]
```

The graphical representation for the same is



Drawback of Sequential model :-

{The model attains a commendable accuracy rate of around 96% on test data. However, it encounters challenges with alphabet images characterized by substantial differences in starting and ending coordinates. In these instances, the model's predictive performance suffers due to the variations in pixel values resulting from spatial disparities.}[5]

Adaptive measure to overcome this drawback :-

Instead of opting for a sequential model like an artificial neural network (ANN), it's more advisable to utilize a convolutional neural network (CNN). CNNs are tailored to effectively manage spatial relationships within images, making them a superior choice for tasks such as alphabet recognition where spatial variations play a crucial role in accurate predictions.

Model using CNN

Implementation :

[2]:
```
from keras.models import
Sequential from keras.layers
import Convolution2D from
keras.layers import
MaxPooling2D from keras.layers
import Flatten
from keras.layers import Dense , Dropout
```

[3]:
```
from tensorflow.keras.models import
Sequential from tensorflow.keras.layers
import Flatten, Dense from
tensorflow.keras.optimizers import SGD
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import tensorflow as tf
```

[4]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
```

[5]:
```
test_datagen = ImageDataGenerator(rescale = 1./255)
```

[6]:
```
training_set =
train_datagen.flow_from_directory('trainingData',
                                                target_size = (128,
                                                128),
                                                batch_size = 10,
                                                color_mode =
                                                'grayscale',
                                                class_mode =
                                                'categorical')
```

```
Found 12846 images belonging to 27 classes.
```

[7]:
```
test_set = test_datagen.flow_from_directory('testingData',
                                            target_size = (128, 128),
                                            batch_size = 10,
                                            color_mode = 'grayscale',
                                            class_mode =
                                            'categorical')
```

```
Found 4268 images belonging to 27 classes.
```

[8]:
```
classifier = tf.keras.models.Sequential()
```

[9]:
```
classifier.add(tf.keras.layers.Conv2D(filters=32,
                                      kernel_size=3,
                                      padding="same",
                                      activation="relu",
                                      input_shape=[128, 128, 1]))
```

```
/home/nitin/anaconda3/lib/python3.11/site-packages/keras/src/
layers/convolutional/base_conv.py:99: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(
```

[10]:
```
classifier.add(tf.keras.layers.MaxPool2D(pool_size=2,
                                         strides=2,
                                         padding='valid'))
```

[11]:
```
classifier.add(tf.keras.layers.Conv2D(filters=32,
                                      kernel_size=3,
                                      padding="same",
                                      activation="relu"))

classifier.add(tf.keras.layers.MaxPool2D(pool_size=2,
                                         strides=2,
                                         padding='valid'))
```

[12]:
```
classifier.add(tf.keras.layers.Flatten())
```

[13]:
```
classifier.add(tf.keras.layers.Dense(units=128,
                                     activation='relu'))
classifier.add(tf.keras.layers.Dropout(0.40))
classifier.add(tf.keras.layers.Dense(units=96, activation='relu'))
classifier.add(tf.keras.layers.Dropout(0.40))
classifier.add(tf.keras.layers.Dense(units=64, activation='relu'))
classifier.add(tf.keras.layers.Dense(units=27, activation='softmax')) # softmax
  for more than 2
```

[14]:
```
classifier.compile(optimizer = 'adam',
                   loss = 'categorical_crossentropy',
                   metrics = ['accuracy'])
```

[15]:
```
classifier.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param
conv2d (Conv2D)	(None, 128, 128, 32)	320
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_1 (Conv2D)	(None, 64, 64, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 32)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 128)	4,194,432
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 96)	12,384
dropout_1 (Dropout)	(None, 96)	0
dense_2 (Dense)	(None, 64)	6,208
dense_3 (Dense)	(None, 27)	1,755

Total params: 4,224,347 (16.11 MB)

Trainable params: 4,224,347 (16.11 MB)

Non-trainable params: 0 (0.00 B)

[14]:
```
classifier.fit(training_set,
               epochs =15,
               validation_data = test_set)
```

Epoch 1/15

1285/1285 **141s** 109ms/step
- accuracy: 0.2284 - loss: 2.5790 - val_accuracy: 0.9217 - val_loss: 0.2927

```
     Epoch 2/15
     1285/1285          141s 110ms/step
-      accuracy: 0.7333 - loss: 0.7965 - val_accuracy: 0.9770 - val_loss:
0.0828 Epoch 3/15
     1285/1285164s 127ms/step
-      accuracy: 0.8250 - loss: 0.5227 - val_accuracy: 0.9904 - val_loss:
0.0341 Epoch 4/15
     1285/1285163s 127ms/step
-      accuracy: 0.8660 - loss: 0.4012 - val_accuracy: 0.9955 - val_loss:
0.0218 Epoch 5/15
     1285/1285144s 112ms/step
-      accuracy: 0.9029 - loss: 0.3023 - val_accuracy: 0.9965 - val_loss:
0.0149 Epoch 6/15
     1285/1285148s 115ms/step
-      accuracy: 0.9121 - loss: 0.2765 - val_accuracy: 0.9967 - val_loss:
0.0140 Epoch 7/15
     1285/1285148s 115ms/step
-      accuracy: 0.9228 - loss: 0.2375 - val_accuracy: 0.9991 - val_loss:
0.0049 Epoch 8/15
     1285/1285145s 112ms/step
-      accuracy: 0.9313 - loss: 0.2123 - val_accuracy: 0.9993 - val_loss:
0.0044 Epoch 9/15
     1285/1285157s 122ms/step
-      accuracy: 0.9386 - loss: 0.1907 - val_accuracy: 0.9988 - val_loss:
0.0050 Epoch 10/15
     1285/1285158s 122ms/step
-      accuracy: 0.9450 - loss: 0.1809 - val_accuracy: 0.9995 - val_loss:
0.0025 Epoch 11/15
     1285/1285157s 122ms/step
-      accuracy: 0.9440 - loss: 0.1786 - val_accuracy: 0.9988 - val_loss:
0.0036 Epoch 12/15
     1285/1285145s 112ms/step
-      accuracy: 0.9474 - loss: 0.1732 - val_accuracy: 0.9995 - val_loss:
0.0021 Epoch 13/15
     1285/1285133s 104ms/step
-      accuracy: 0.9573 - loss: 0.1355 - val_accuracy: 0.9995 - val_loss:
0.0025 Epoch 14/15
     1285/1285141s 109ms/step
-      accuracy: 0.9579 - loss: 0.1365 - val_accuracy: 0.9995 - val_loss:
0.0011 Epoch 15/15
     1285/1285145s 112ms/step
-      accuracy: 0.9591 - loss: 0.1337 - val_accuracy: 0.9995 - val_loss: 0.0021

[16]: <keras.src.callbacks.history.History at 0x7375b47dd1d0>[17]:

classifier.save('modelCNN.h5')

     WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
```

[29]:
```
classifier.save('modelCNN.keras')
```

[ ]:
```

```

Model Explain:-

{ The provided model is a Convolutional Neural Network (CNN) designed for image classification tasks. Let's break down the components and their functions:} [1]

Data Preprocessing :-

{ImageDataGenerator: *Used for real-time data augmentation. It rescales pixel values to be between 0 and 1, applies shear and zoom transformations, and enables horizontal flipping for the training data.}*[1]

Data Loading :-

training set: *Loads training images from the 'trainingData' directory. It resizes images to 128x128 pixels, sets batch size to 10, uses grayscale color mode, and employs categorical class mode for multi-class classification.*

test_set: *Loads testing images from the 'testingData' directory with similar configurations as the training set.*

Model Architecture:

• Sequential Model: A linear stack of layers.

• Conv2D Layers: These are convolutional layers with 32 filters, a kernel size of 3x3, 'relu' activation function, and 'same' padding. The first layer also specifies the input shape as 128x128x1 (1 channel for grayscale).

• MaxPool2D Layers: Pooling layers with a pool size of 2x2, a stride of 2, and 'valid' padding to downsample the feature maps.

• Flatten Layer: Flattens the 2D feature maps into a 1D vector for the fully connected layers.

• Dense Layers: Fully connected layers with 128, 96, and 64 units respectively, using 'relu' activation functions. Dropout layers with 40% dropout rate are added after the first two dense layers to prevent overfitting.

• Output Layer: A dense layer with 27 units (assuming 27 classes) and 'softmax' activation for multi-class classification.

Model Compilation:

• Optimizer: Uses the 'adam' optimizer for gradient descent.
• Loss Function: Categorical Crossentropy, suitable for multi-class classification.
• Metrics: Accuracy is used to evaluate model performance during training.

Model Training:

• Fit Method: Trains the model on the training data for 15 epochs, using the validation data from the test set to monitor performance.

Implementing model :-

1. By taking image as a input

Implementation :

```
import cv2
    import numpy as np
    from tensorflow.keras.models import
    load_model cnn =
    load_model('model_prototype.keras')
```

[17]:
```
img = cv2.imread('preprocessed_image.jpg')
resized_image = cv2.resize(img, (128, 128))
resized_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)


                                        2
resized_image =
resized_image.astype('float32') / 255.0
resized_image = np.expand_dims(resized_image,
axis=0) resized_image =
cv2.flip(resized_image, 1)

# Make predictions
predictions = cnn.predict(resized_image)
```

```
1/1             0s 23ms/step
```

[18]:
```
predictions
```

```
[18]: array([[1.8455510e-29, 9.5807602e-09, 1.6274704e-28,
                                            3.7778295e-06,
 4.6792753e-31, 2.4386653e-18, 5.7638909e-16, 2.4103343e-
                                                      07,
        8.0304297e-08, 8.2691570e-31, 9.9999583e-01, 1.6451087e-
        37,
        3.1864112e-11, 2.1890652e-28, 8.9945519e-18, 4.6281798e-
        18,
        5.0488424e-12, 1.0263721e-09, 7.2374585e-31, 4.4063287e-
        18,
        8.5516568e-12, 0.0000000e+00, 1.6764347e-31, 3.2964128e-
        18,
        5.5701946e-22, 9.9732349e-12, 3.8129015e-14]],
        dtype=float32)
```

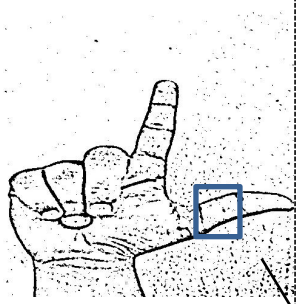
[19]:
```
np.max(predictions)
```

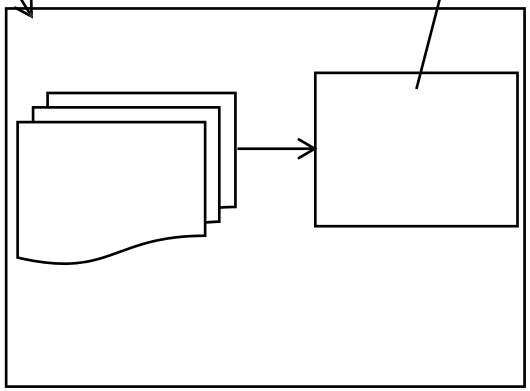
```
[19]: 0.9999958
```

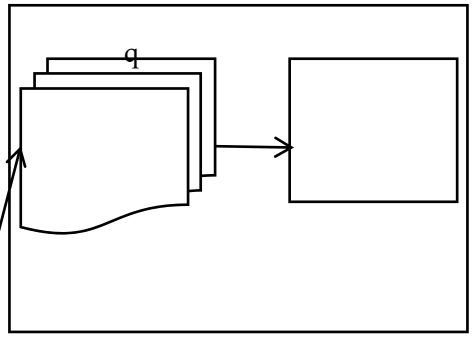
[20]:
```
class_labels = ['0','A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
 'Y', 'Z']
# Decode the predictions
predicted_class_index =
np.argmax(predictions) predicted_class =
class_labels[predicted_class_index]

print("Predicted Class:", predicted_class)
```

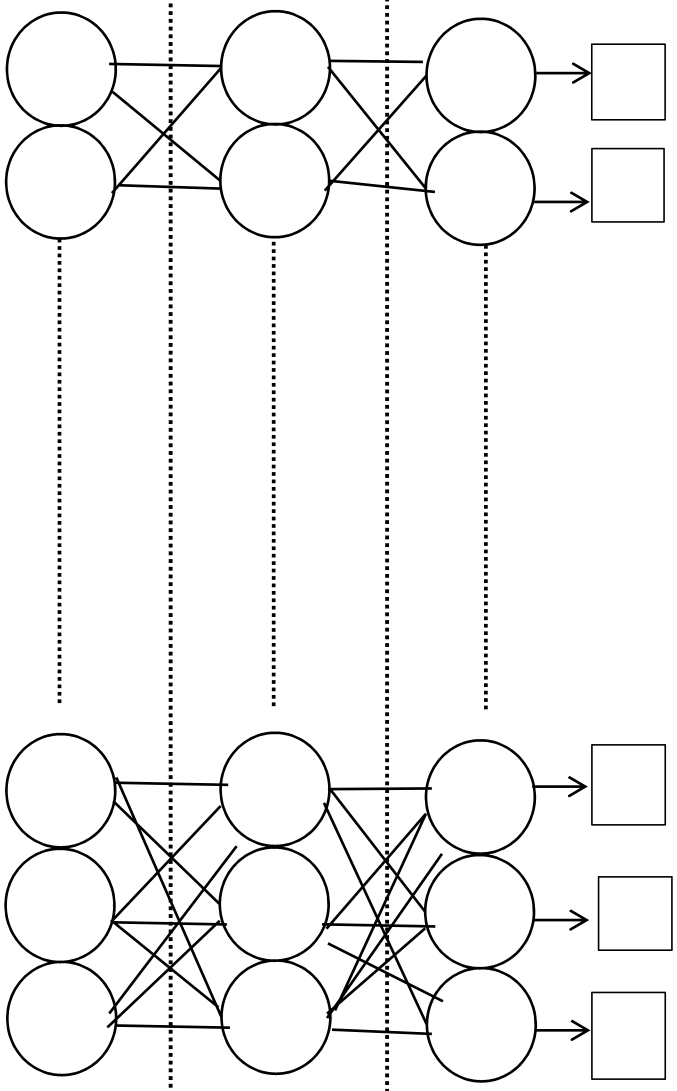
Predicted Class: J

1









CNN

Input Layer

Hidden layer

Output layer

Explain:-

Image Preprocessing:

- *Reads an image named 'preprocessed_image.jpg' using OpenCV (cv2.imread).*
- *Resizes the image to a shape of (128, 128) using bilinear interpolation (cv2.resize).*
- *Converts the resized image to grayscale (cv2.cvtColor).*
- *Normalizes the pixel values to the range [0, 1] by dividing by 255.0.*
- *Expands the dimensions of the image to make it suitable for input to the CNN (np.expand_dims).*
- *Flips the image horizontally (cv2.flip). The reason for this augmentation is unclear from the provided code alone, but it's commonly used to increase the robustness of the model by providing additional training examples.*

Making Predictions:

Uses the CNN model (cnn) to predict the class probabilities of the preprocessed image (cnn.predict).

***Reshaping Input Data**:*

A comment suggests that if the input shape expected by the model is different from the current shape (1, 32768), you may need to reshape the input data accordingly before making predictions.

1. Using computer-vision

[1]:
```
import pyttsx3
text_speech=pyttsx3.init()
text_speech.setProperty('rate', 10)
```

[2]:
```
import cv2
from keras.models import load_model
from keras.preprocessing.image import load_img, img_to_array
import numpy as np
import tensorflow as tf
import keras
```

[3]:
```
model = keras.models.load_model("modelCNN.h5")
```

```
WARNING:absl:Compiled the loaded model, but the compiled
metrics have yet to be built. `model.compile_metrics` will be
empty until you train or evaluate the model.
```

[4]:
```
labels_dict = {0:'0',
               1:'A',
               2:'B',
               3:'C',
               4:'D',
               5:'E',
               6:'F',
               7:'G',


               8:'H',
               9:'I',
               10:'H',
               11:'K',
               12:'L',
               13:'M',
               14:'N',
               15:'O',
               16:'P',
               17:"Q",
               18:'R',
               19:'S',
               20:'T',
               21:'U',
               22:'V',
               23:'W',
               24:'X',
               25:'Y',
               26:'Z'}
color_dict=(0,255,0)
x=0
```

```
img_size=128 minValue = 70 source=cv2.VideoCapture(0) count = 0

    strin
    g = "
    "
    prev
    = " "
    prev_
    val =
    0
    while
    (True
    ):
        ret,img=source.read()
        gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        #cv2.rectangle(img,(x,y),
        (x+w,y+h),color_dict,2) cv2.rectangle(img,
        (24,24),(250 , 250),color_dict,2)
        crop_img=gray[24:250,24:250]
        count = count + 1
        if(count % 100
            == 0):
            prev_val =
            count
     1. cv2.putText(img, str(prev_val//100), (300,
        150),cv2.FONT_HERSHEY_SIMPLEX,1.  5,(255,255,255),2)
        blur = cv2.GaussianBlur(crop_img,(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)
    resized=cv2.resize(res,(img_size,img_size))
    resized = cv2.flip(resized, 1)
    normalized=resized/255.0
    reshaped=np.reshape(normalized,(1,img_size,img_size,1))
    #reshaped = cv2.flip(reshaped, 1)
    result = model.predict(reshaped)
    #print(result)
    label=np.argmax(result,axis=1)[0]
    #text_speech.say(label)
    #text_speech.runAndWait()

    if(count == 300):
        count = 99
        prev= labels_dict[label]
        if(label == 0):
            string = string + " "

        else:
            string = string + prev
            text_speech.say(prev)
            text_speech.runAndWait()
    cv2.putText(img, prev, (24, 14),cv2.FONT_HERSHEY_SIMPLEX,0.
  8,(255,255,255),2)

    cv2.putText(img, string, (275, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.8,

 (0, 0,_ 255), 2)
    #cv2.putText(img, string, (275,
 50),cv2.FONT_HERSHEY_SIMPLEX,0.  8,(200,200,200),2)
    cv2.imshow("Gray",res)
    cv2.imshow('LIVE',img)
    key=cv2.waitKey(1)


    if(key==27):#press Esc. to exit
        break
print(string)
cv2.destroyAllWindows()
source.release()

cv2.destroyAllWindow
```

Explain:-

{ Loading the Model:

- *Loads a pre-trained CNN model named "modelCNN.h5" using Keras' load_model function.*

Label Dictionary:

- *Defines a dictionary (labels_dict) mapping class indices to corresponding characters (e.g., 0 to '0', 1 to 'A', etc.).*

Color Dictionary and Region of Interest (ROI):

- *Defines a color dictionary (color_dict) for drawing rectangles.*
- *Defines the region of interest (ROI) coordinates (x, y, w, h).*

Camera Setup:

- *Initializes a video capture object (source) from the default camera (index 0). }*[3]

Main Loop:

- *Enters a loop to continuously capture frames from the camera.*
- *Converts each frame to grayscale.*
- *Defines a ROI within the frame and extracts it.*
- *Applies Gaussian blur and adaptive thresholding to the ROI to enhance hand gesture features.*
- *Resizes the processed ROI to the expected input size of the CNN model.*
- *Normalizes the resized image and reshapes it to match the model's input shape.*
- *Makes a prediction using the loaded CNN model and extracts the predicted label.*
- *Updates a string variable string with the predicted character, forming a sequence of recognized characters.*
- *Displays the processed frame with the recognized character sequence (string) and the last recognized character (prev).*

Exiting the Loop:

- *Pressing the Esc key (key code 27) breaks the loop, terminating the program.*

Cleaning Up:

- *Releases the video capture object and closes all OpenCV windows.*

Conclusion

Overview:

The development of sign language to speech translation systems marks a significant milestone in fostering accessibility and communication for the deaf and hard of hearing community. These systems leverage cutting-edge technology, including computer vision and natural language processing, to translate sign language gestures into spoken language in real-time. By bridging the communication gap between signers and non-signers, these systems enhance accessibility in various settings, such as educational institutions, workplaces, and public spaces. Key conclusions drawn from this research include improved accessibility, real-time communication capabilities, technological advancements, considerations for user experience, and ongoing challenges.

Future Scope:

Looking ahead, the future of sign language to speech translation holds promising opportunities for further innovation and impact. Areas of future research and development include enhancing accuracy through deep learning algorithms and multimodal fusion techniques, integrating additional modalities such as facial expressions and body movements, personalizing models to adapt to individual signing styles, improving accessibility and inclusivity through affordability and scalability, and addressing ethical considerations related to privacy, bias mitigation, and cultural sensitivity. By advancing technology, fostering interdisciplinary collaboration, and prioritizing user-centered design, we can continue to break down communication barriers and promote inclusivity and diversity in society.

Sign language to Speech

Sachin kumar
21051162

Abstract:

Collaborated on drafting the abstract, emphasizing the importance of robust dataset creation and preprocessing in developing a reliable sign language to speech translation system.

Individual contribution and findings:

- Played a crucial role in creating the dataset by capturing images of sign language gestures, ensuring a diverse and comprehensive collection.
- Assisted in the removal of noise from the dataset through techniques such as converting images to grayscale, applying Gaussian blur, and adding thresholding.
- Contributed to the train-test split of the dataset, ensuring an appropriate division for model training and evaluation.

Report Preparation:

- Contributed to the methodology section detailing the dataset creation process, noise removal techniques, and data splitting methodology.
- Assisted in compiling and organizing the dataset section, providing insights into the significance of diverse and comprehensive data collection.

PPT Preparation:

- Prepared slides outlining the dataset creation process, including sample images and statistics.
- Provided visuals illustrating the effectiveness of noise removal techniques, before-and-after comparisons, and the impact on model performance.

Full Signature of Supervisor: Full signature of the student:

…… ……..

Sign language to Speech

Abhinav Bisht
21051704

Abstract:

Collaborated on drafting the abstract, emphasizing the practical application of the sign language to speech translation system and its potential impact on accessibility.

Individual contribution and findings:

- Contributed to the testing phase of the project by devising strategies for evaluating the model's performance using live camera input.
- Developed scripts and algorithms for real-time testing of the sign language to speech translation system, ensuring seamless integration with camera hardware.
- Worked closely with team members to analyze testing results, identify areas for improvement, and iterate on the model implementation accordingly.

Report Preparation:

- Led the testing methodology section, outlining the approach for evaluating the model using live camera input.
- Contributed to the discussion on testing metrics and result analysis, providing insights into the real-world performance of the system.

PPT Preparation:

- Created slides detailing the testing process, including screenshots of live camera testing and demonstration videos.
- Provided visuals illustrating the system's performance in real-time scenarios, showcasing its effectiveness in translating sign language gestures.

Full Signature of Supervisor: Full signature of the student:

…… ……..

Sign language to Speech

Aditya Raj
21051709

Abstract:

Drafted the abstract section highlighting the use of CNN techniques for model training and emphasizing the importance of experimentation for optimizing model performance.

Individual contribution and findings:

- Played a pivotal role in training the model using convolutional neural network (CNN) techniques, leveraging their expertise in deep learning methodologies.
- Conducted experimentation with various CNN architectures and hyperparameters to enhance model accuracy and generalization.
- Collaborated with team members to troubleshoot training issues and refine the training process for optimal results.

Report Preparation:

- Contributed to the methodology section on model training, detailing the CNN architecture and training process.
- Provided insights into the experimental setup, including hyperparameter selection and training/validation strategies.

PPT Preparation:

- Prepared slides outlining the CNN architecture used for model training, including diagrams and explanations of convolutional layers.
- Provided visuals illustrating the training process, including loss curves and accuracy metrics.

Full Signature of Supervisor: Full signature of the student:

…… ……..

Sign language to Speech

Arnav kumar Dey
21051801

Abstract:

Played a key role in summarizing the model implementation approach in the abstract, highlighting the use of sequential models for sign language translation.

Individual contribution and findings:

- Implemented the model using a sequential approach, orchestrating the architecture and layers required for the sign language to speech translation system.
- Collaborated with team members to fine-tune the model architecture, ensuring compatibility with the dataset and computational resources.
- Provided expertise in neural network design and implementation to optimize model performance.

Report Preparation:

- Led the model implementation section, detailing the sequential model architecture and its components.
- Contributed to the discussion on model optimization strategies, including parameter tuning and regularization techniques.

PPT Preparation:

- Created slides outlining the sequential model architecture, layer configurations, and activation functions used.
- Provided visuals illustrating the flow of data through the sequential model and the role of each layer in the translation process.

Full Signature of Supervisor: Full signature of the student:

…… ……..

Sign language to Speech

Nitin kumar
21051828

Abstract:

Collaborated on drafting the abstract, emphasizing the collaborative effort in developing the sign language to speech translation system and its implications for accessibility and inclusivity.

Individual contribution and findings:

- Provided critical support throughout the project, including data preprocessing, model implementation, and testing phases.
- Assisted in refining the dataset and preprocessing techniques to improve data quality and model robustness.
- Collaborated with team members to document project progress, results, and insights, ensuring comprehensive reporting and knowledge sharing.

Report Preparation:

- Assisted in data preprocessing, including refining dataset quality and optimizing preprocessing techniques.
- Contributed to the discussion on project progress and challenges faced, providing valuable insights into potential solutions.

PPT Preparation:

- Prepared slides summarizing project milestones, challenges, and key learnings.
- Provided visuals highlighting the collaborative nature of the project and the contributions of each team member.

Full Signature of Supervisor: Full signature of the student:

…… ……..

Reference

1. *https://www.researchgate.net/publication/362742399_Convolutional_Neural_Network_CNN_A_comprehensive_overview {24}*

2. *https://www.tensorflow.org/tutorials/images/cnn {1]*

3. *https://www.researchgate.net/publication/301590571_OpenCV_for_Computer_Vision_Applications {32}*

4. *https://www.geeksforgeeks.org/ml-understanding-data-processing/ {12}*

5. *https://keras.io/guides/sequential_model/ {1}*

6. *https://medium.com/@ilyurek/what-is-max-pooling-and-why-do-we-need-max-pooling-57247a3fbca9 {3}*

7. *https://www.tensorflow.org/tutorials/customization/custom_training_walkthrough {14}*

8. *https://www.researchgate.net/publication/276151976_Conversation_of_Sign_Language_to_Speech_with_Human_Gestures*