GESTURE RECOGNITION TO ASL
Final Report

Course: CSCE 5222 Feature Engineering

Instructor: *Dr. Sayed Khushal Shah*(sayed.shah@unt.edu)

Members:

Sreeja Bellamkonda (sreejabellamkonda@my.unt.edu)

Bhavana Katari (Bhavanakatari@my.unt.edu)

GitHub Link:

https://github.com/SreejaBellamkonda/FE-GestureTranslation.git

Increment 1

GOALS AND MANAGEMENT

Motivation and Significance

There are several disabilities That could cause deafness. One of the most common deafness is both ears and one ear. This might be caused by genetics, infections, or noise exposure. Mutism is another disability that causes the person unable to speak. One person may experience this. Deafness can sometimes lead to mutism or the other way around. Most people today deal with this issue on a regular basis. To express their emotions and communicate with their hands, American Sign Language was developed in the 18th century. There have been numerous inventions since then. There are many inventions, but not all of them are beneficial to people. So, learning sign language is both simple and invaluable. Face expressions and hand gestures help to distinguish this sign language. In this project, pictures are taken, or gesture pictures are gathered, and then using the feature extraction approach, we can tell where the picture is located or what the sign or letter is. Machine learning and deep learning approaches can thus fully extract the image and produce reliable findings. Additionally, we can assess how accurate each image gesture is.

Objectives

The project's objective is translation. American Sign Language hand movements are translated into this language using image classifications, data collection, and model training. I'll then assess the translations' overall correctness rate.

- Although this project involves translating gestures into ASL, we have many projects involving gesture recognition.
- Compile alphabetical data using hand gestures for the signs and letters.

  To recognize the motion of the hand, we use CNN image classifications and deep learning.
- Will evaluate the outcomes of each image, explain the implementation strategies, and make

  adjustments as necessary.

Features

In this, we used canny edge detection which helps to detect the range of the edge in the image.

It helps in Noise reduction, Feature matching, voice, and more.

RELATED WORK (Background)

Technology has been advancing over the last few years in every field. Specifically in the financial and industrial sectors, and the health sectors. To treat illnesses, every hospital is receiving improved technology. Recently developed aids for the disabled make it easier for them to move, speak, and recognize other individuals. Our study focuses on those with disabilities who can only understand us through gestures and who are unable to talk or listen. We will convert the movements

or photographs into American sign language using image classifiers. We do that by utilizing the machine learning-based feature engineering technique.

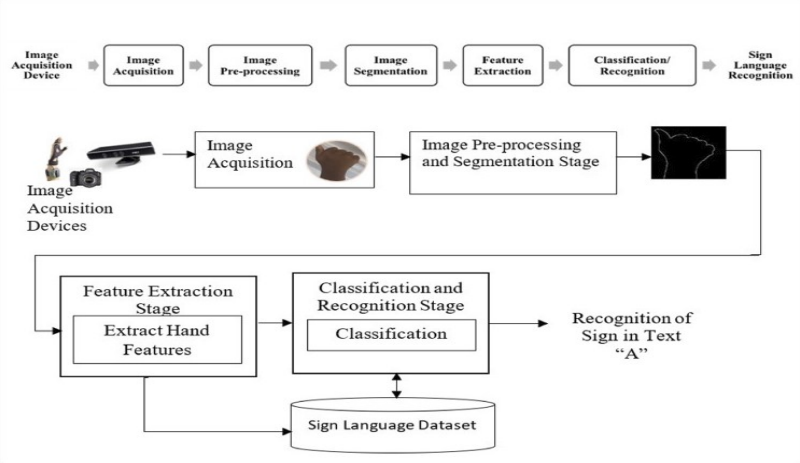


Fig 1: Relatable work classification

DATASET

ASL Alphabet: There are 29 classes in this dataset, which was assembled by Kaggle (26 classes are the alphabet and 3 classes are SPACE, DELETE, and NOTHING). 87,000 100x100 pixel images make up the training set of data, whereas 29 images make up the test set. To train the model to anticipate gestures based on the images of the alphabets being used, this data will be used.

https://www.kaggle.com/grassknoted/asl-alphabet

DETAIL DESIGN OF FEATURES

This will explain in detail how the features work. Firstly, the hand is captured then ROI segmentation is if we are capturing the video ROI is useful to remove the background noise, and then the Fingers segmentation is done then that applying the CNN method we recognize an image, and that image can be trained to connect with the alphabetical letter which converts to ASL

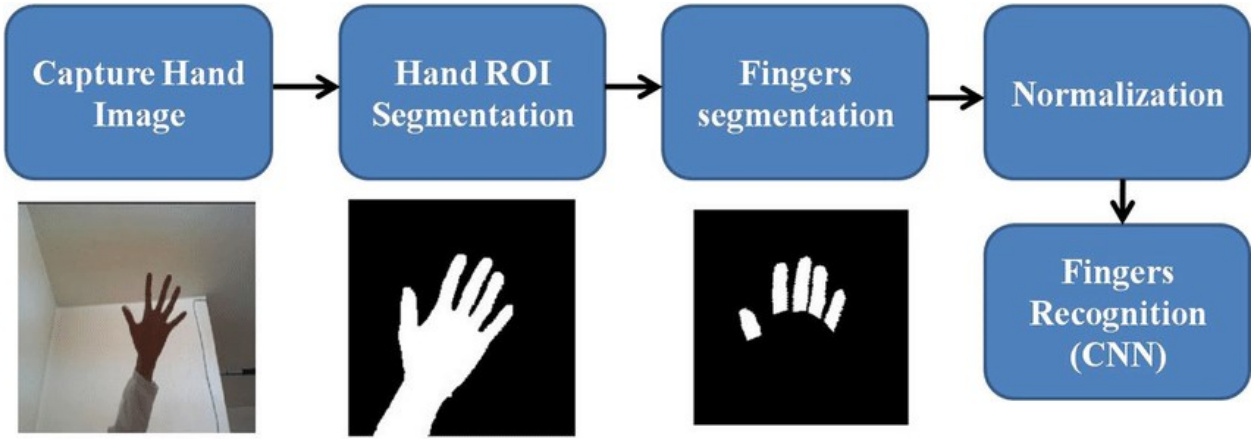


Fig 2: Design of Dataset

ANALYSIS OF DATA

Image Preprocessing

A Greyscale (1D) image is created from a camera-captured RGB format (3D) image. That indicates that the color image input will be changed to a monochrome image.

The above picture represents the greyscale image, which is one-dimensional. Once the training is done, the visualization of the training performance could be better understood through the graph shown below:

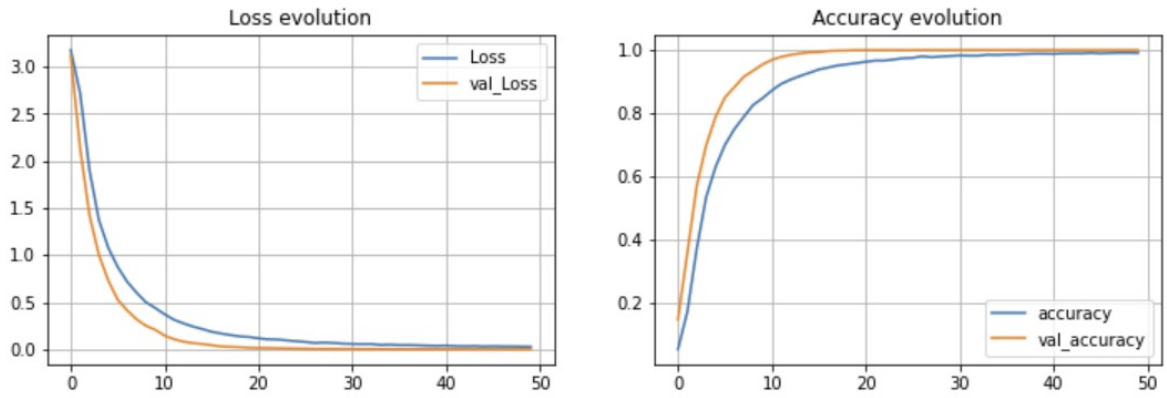


Fig 3: Loss and Accuracy plots

IMPLEMENTATION

For the implementation purpose, we have taken the help of a confusion matrix. A method for summarizing a classification algorithm's performance is the confusion matrix. If the dataset contains more than two classes or has more observations than classes, classification accuracy alone may be deceptive. Therefore we calculate the Confusion matrix which helps us to understand the categorization model's successes and failures. A powerful system for recognizing sign language hand gestures can be created using the CNN model and image processing.

Preprocessing

In the first step, We collect the data or create the dataset with respect to the collection of what we need. And then we do the preprocessing method where we can use the data before we perform an action and check the data. Here we are using the data to train and test the dataset.

```
[ ] train = pd.read_csv('gdrive/My Drive/FE/Project/Dataset/sign_mnist_train.csv')
    test = pd.read_csv('gdrive/My Drive/FE/Project/Dataset/sign_mnist_test.csv')
```

Checking the shape of the train and test file.

```
[ ] print(train.shape)
    print(test.shape)
```

```
(27455, 785)
(7172, 785)
```

Check the pixels of the train dataset.

```
train.head()
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel7
0	3	107	118	127	134	139	143	146	150	153	...	207	207	207	207	206	206	206	
1	6	155	157	156	156	156	157	156	158	158	...	69	149	128	87	94	163	175	
2	2	187	188	188	187	187	186	187	188	187	...	202	201	200	199	198	199	198	
3	2	211	211	212	212	211	210	211	210	210	...	235	234	233	231	230	226	225	
4	13	164	167	170	172	176	179	180	184	185	...	92	105	105	108	133	163	157	

5 rows × 785 columns

Feature Engineering

Now after preprocessing will convert the data frames into the arrays where we create the arrays for the train and the test data. We also specify the labels where hand gesture is represents the letters like A,B,C.. And check the class label image for verification.

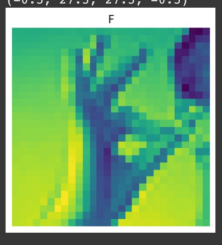
For the future preprocessing will convert this data frames into arrays.

```
[ ] # Creating the training and testing arrays
    train_set = np.array(train, dtype = 'float32')
    test_set = np.array(test, dtype='float32')
```

```
[ ] #Specifying class 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' ]
```

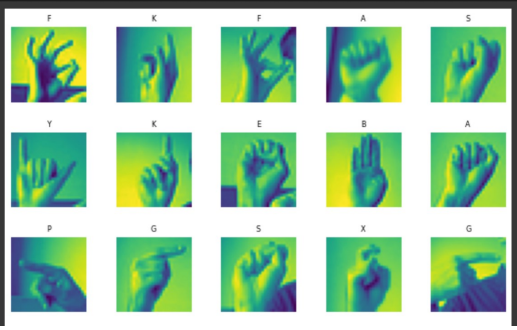
```
#Check a random image for class label verification
i = random.randint(1,27455)
plt.imshow(train_set[i,1:].reshape((28,28)))
plt.imshow(train_set[i,1:].reshape((28,28)))
label_index = train["label"][i]
plt.title(f"{class_names[label_index]}")
plt.axis('off')
```

```
(-0.5, 27.5, 27.5, -0.5)
```



Then define the result of the letters. By hand gesture, we can identify which Alphabet letter it is. As shown in the results the Gesture is represented by each alphabetical letter.

```
# Define the dimensions of the plot grid
W_grid = 5
L_grid = 5
fig, axes = plt.subplots(L_grid, W_grid, figsize = (10,10))
axes = axes.ravel() # flaten the 15 x 15 matrix into 225 array
n_train = len(train_set) # get the length of the train dataset
# Select a random number from 0 to n_train
for i in np.arange(0, W_grid * L_grid): # create evenly spaces variables
    # Select a random number
    index = np.random.randint(0, n_train)
    # read and display an image with the selected index
    axes[i].imshow( train_set[index,1:].reshape((28,28)) )
    label_index = int(train_set[index,0])
    axes[i].set_title(class_names[label_index], fontsize = 8)
    axes[i].axis('off')
plt.subplots_adjust(hspace=0.4)
```



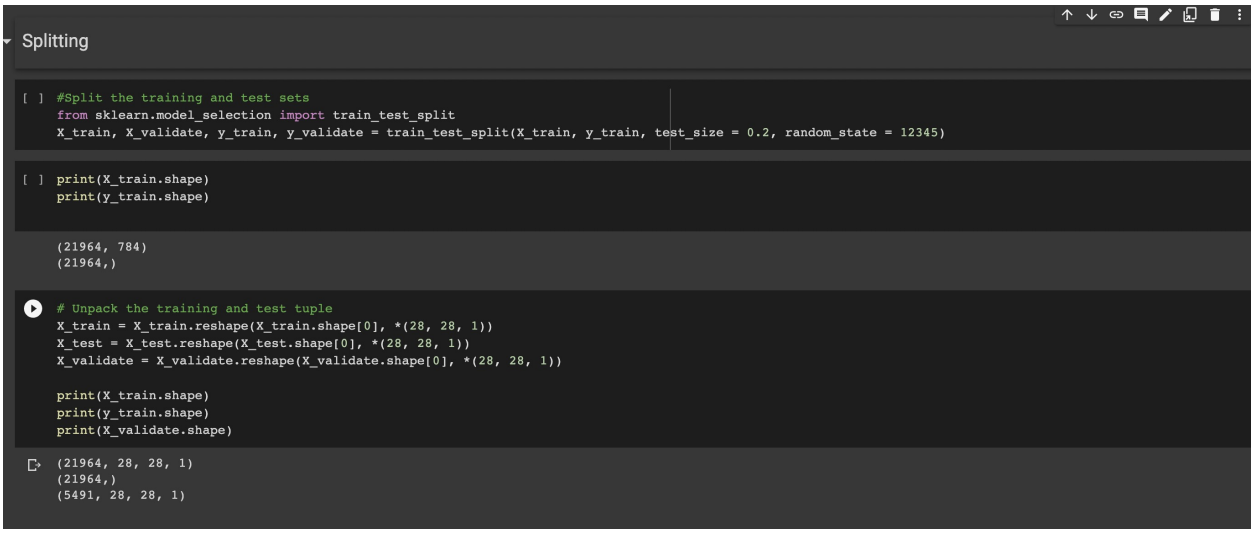
Feature Engineering

We can now visualize the gestures using the gray scale image too. After training the model apply the visualization to the images. Add grid by giving the range and visualize the images. As we can see in the result the representation of the of the gesture is directed by the alphabetical letter

```
#Visualize train images
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_train[i].reshape((28,28)), cmap=plt.cm.binary)
    label_index = int(y_train[i])
    plt.title(class_names[label_index])
plt.show()
```

Feature Engineering

We are using the train and test splitting here in the project this is because it helps in performing the machine learning which applicable to predicting the algorithms. This splitting method is fast and furious so that we can perform the machine learning results. Before the model implementation, splitting method is useful for preparing the model.



Splitting

```
[ ] #Split the training and test sets
    from sklearn.model_selection import train_test_split
    X_train, X_validate, y_train, y_validate = train_test_split(X_train, y_train, test_size = 0.2, random_state = 12345)
```

```
[ ] print(X_train.shape)
    print(y_train.shape)
```

```
(21964, 784)
(21964,)
```

```
# Unpack the training and test tuple
X_train = X_train.reshape(X_train.shape[0], *(28, 28, 1))
X_test = X_test.reshape(X_test.shape[0], *(28, 28, 1))
X_validate = X_validate.reshape(X_validate.shape[0], *(28, 28, 1))

print(X_train.shape)
print(y_train.shape)
print(X_validate.shape)
```

```
(21964, 28, 28, 1)
(21964,)
(5491, 28, 28, 1)
```

Here we are performing the CNN model. Convolution Neural network method. It helps in image processing which also converts into 2D images. CNN is mostly used for image recognition and that includes the preprocessing of the pixel data. We can use many other techniques in image processing but the CNN is the best image processing method to identify and recognize the objects or the image.

```
#Defining the Convolutional Neural Network
cnn_model = Sequential()

cnn_model.add(Conv2D(32, (3, 3), input_shape = (28,28,1), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.25))

cnn_model.add(Conv2D(64, (3, 3), input_shape = (28,28,1), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.25))

cnn_model.add(Conv2D(128, (3, 3), input_shape = (28,28,1), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Dropout(0.25))

cnn_model.add(Flatten())

cnn_model.add(Dense(units = 512, activation = 'relu'))
cnn_model.add(Dropout(0.25))
cnn_model.add(Dense(units = 25, activation = 'softmax'))
```

Feature Engineering

The summary of the CNN model. Will check how does the dataset given the model is working.

```
#CNN Model Summary
cnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 5, 5, 64)          0

 conv2d_2 (Conv2D)           (None, 3, 3, 128)         73856

 max_pooling2d_2 (MaxPooling  (None, 1, 1, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 1, 1, 128)         0

 flatten (Flatten)           (None, 128)               0

 dense (Dense)               (None, 512)               66048

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 25)                12825

=================================================================
```

Below shown is the training model where its training the CNN model.

```
#Training the CNN model
history = cnn_model.fit(X_train, y_train, batch_size = 512, epochs = 50, verbose = 1, validation_data = (X_validate, y_validate))
```

```
Epoch 1/50
43/43 [==============================] - 23s 515ms/step - loss: 3.1782 - accuracy: 0.0522 - val_loss: 3.1197 - val_accuracy: 0.1464
Epoch 2/50
43/43 [==============================] - 23s 533ms/step - loss: 2.7187 - accuracy: 0.1713 - val_loss: 2.1316 - val_accuracy: 0.3580
Epoch 3/50
43/43 [==============================] - 22s 520ms/step - loss: 1.8991 - accuracy: 0.3713 - val_loss: 1.4219 - val_accuracy: 0.5698
Epoch 4/50
43/43 [==============================] - 22s 518ms/step - loss: 1.3714 - accuracy: 0.5341 - val_loss: 1.0012 - val_accuracy: 0.6979
Epoch 5/50
43/43 [==============================] - 24s 562ms/step - loss: 1.0772 - accuracy: 0.6295 - val_loss: 0.7300 - val_accuracy: 0.7880
Epoch 6/50
43/43 [==============================] - 22s 515ms/step - loss: 0.8752 - accuracy: 0.6994 - val_loss: 0.5261 - val_accuracy: 0.8499
Epoch 7/50
43/43 [==============================] - 22s 516ms/step - loss: 0.7183 - accuracy: 0.7495 - val_loss: 0.4137 - val_accuracy: 0.8824
Epoch 8/50
43/43 [==============================] - 22s 515ms/step - loss: 0.6025 - accuracy: 0.7881 - val_loss: 0.3188 - val_accuracy: 0.9151
Epoch 9/50
43/43 [==============================] - 22s 514ms/step - loss: 0.5004 - accuracy: 0.8261 - val_loss: 0.2498 - val_accuracy: 0.9346
Epoch 10/50
43/43 [==============================] - 22s 514ms/step - loss: 0.4382 - accuracy: 0.8472 - val_loss: 0.2067 - val_accuracy: 0.9543
Epoch 11/50
43/43 [==============================] - 29s 673ms/step - loss: 0.3716 - accuracy: 0.8722 - val_loss: 0.1448 - val_accuracy: 0.9692
Epoch 12/50
43/43 [==============================] - 23s 536ms/step - loss: 0.3135 - accuracy: 0.8931 - val_loss: 0.1064 - val_accuracy: 0.9787
Epoch 13/50
43/43 [==============================] - 23s 533ms/step - loss: 0.2753 - accuracy: 0.9066 - val_loss: 0.0812 - val_accuracy: 0.9849
Epoch 14/50
43/43 [==============================] - 22s 516ms/step - loss: 0.2417 - accuracy: 0.9179 - val_loss: 0.0651 - val_accuracy: 0.9893
Epoch 15/50
43/43 [==============================] - 22s 513ms/step - loss: 0.2154 - accuracy: 0.9280 - val_loss: 0.0563 - val_accuracy: 0.9927
Epoch 16/50
43/43 [==============================] - 23s 527ms/step - loss: 0.1861 - accuracy: 0.9386 - val_loss: 0.0432 - val_accuracy: 0.9933
```

Now check the accuracy of the model that we performed. We check the accuracy because the model we run is to find the gesture recognition. According to the data given and performing the CNN model the gesture should represent the alphabetical order. To check if that is correct and we developed that in a correct way we check the accuracy of the model.

```
#Classification accuracy
from sklearn.metrics import accuracy_score
acc_score = accuracy_score(y_test, classes_x)
print('Accuracy Score = ',acc_score)
```

```
Accuracy Score =  0.9435303959843837
```

Here we are applying the Edge Canny detection. Firstly, print the original image and convert that to grayscale image. And then apply the canny edge detection. We use this canny edge detection because the background is eliminated, and the gestures are displayed in the noise removed and we see if we can identify that in edge detection model.

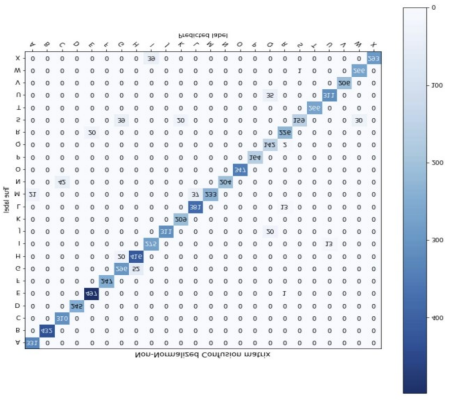
```
# perform the canny edge detector to detect image edges
edges = cv2.Canny(gray, threshold1=30, threshold2=100)
plt.imshow(edges)
```

```
<matplotlib.image.AxesImage at 0x7fb3e25a7bd0>
```

Below is the confusion matrix which defines the classification perfoming.

```
#Non-Normalized Confusion Matrix
plt.figure(figsize=(20,20))
plot_confusion_matrix(y_test, classes_x, classes = class_names, title='Non-Normalized Confusion matrix')
plt.show()
```

```
Confusion matrix, without normalization
```



Non-normalized Confusion Matrix

PRELIMINARY RESULTS

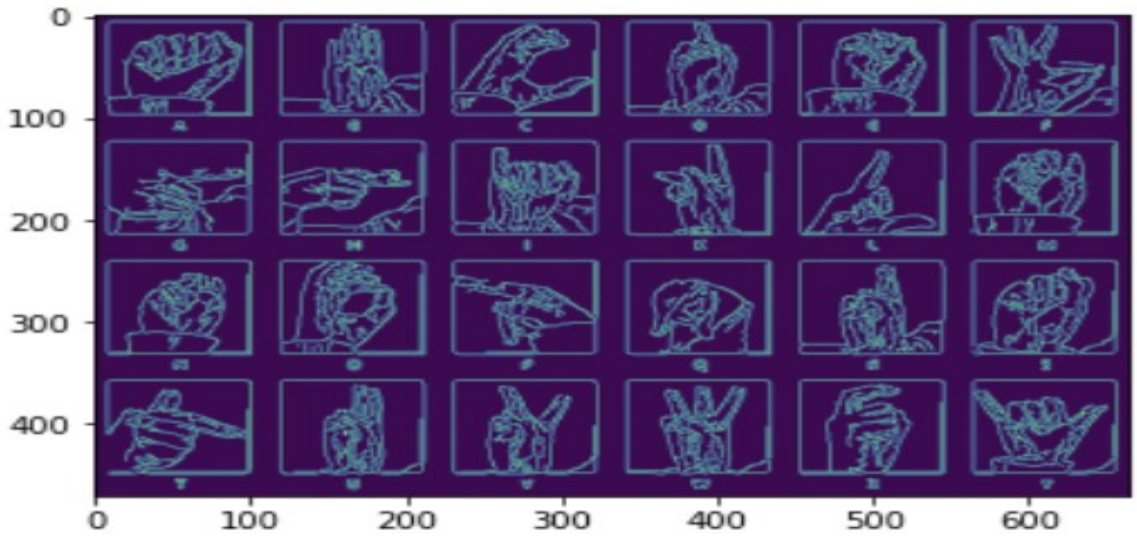
As we discussed before, here we make use of Canny Edge Detection to translate to American Sign Language(ASL). The images can be transformed from RGB to grayscale via image processing before being fed into a deep learning model like a convolutional neural network (CNN).

Fig 4: Input Image

Fig 6: Converting to Grayscale

Canny edge detection:

The Output using the Canny edge detection is shown as below. It is basically used to detect the edges of the image.



Implementation status report

Work completed

Description

In this project, we have completed Gesture recognition by collecting the data. We collected the images of the gestures and the given input arrays of the alphabets where each recognition represents each alphabet respectively. By using CNN image processing and edge detection we recognized the gestures.

Responsibilities & Contribution

Name	Task	Percentage
Bhavana	Data Collection, Preprocessig, Reaserach, Documentaion	50

Sreeja	Implementing the CNN model, Edge detection, Confusion, Documentation	50

Work to be completed

Description

In increment 2 we will collect the data using the video camera. From video capture, we recognize the gestures and convert that into sign language assigning the letters words, and numbers.

Responsibilities

Using a machine learning technique, CNN model will implement the project with the help of video capturing.

Increment 2

INTRODUCTION

In our world the population of those with a disability is high. Some of them have standing and sitting, walking disabilities and some have hearing and visual disabilities. This Hand gesture project is designed specifically for those people who are having a hearing disability. This hand gesture recognition is the system that recognizes the hand gesture using video or webcam videos. Hand gesture is classified with the Neural Networks used. In this project, we have two difficult tasks. Firstly, Capture the gestures using the webcam. Secondly, the dataset is given, and find the gestures accordingly. If it is the Alphabet or numbers detect that gesture. We apply CNN and canny edge detection in this process. To develop this hand gesture recognition, we use image processing filters. This hand gesture recognition is developed using OpenCV and Python with applicable theories and detects the gestures.

BACKGROUND

Related Work:

We have many relatable works on the website. One among those is real-time hand gesture recognition using OpenCV and TensorFlow. In this work, they used the Media Pipe which is the machine learning framework. It is an open-source method. It also has ML solutions that have face detection, hand detection, and object detection. TensorFlow is also a machine learning and deep learning library. It is an open-source environment used in both ML and DL. It is used the perform the task on the neural networks. The development of this project starts with initializing the media pipe and the TensorFlow. Then by giving the ID of the cam, read the frames of the webcam. Detect the key points according to the data given. Recognize the hand gestures and capture the gestures.

Reliably we have many similar project works, but every project has the uniqueness where the output produces the same.

Some of the project's works I referred to and links are given below.

https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/

https://aihubprojects.com/hand-detection-gesture-recognition-opencv-python/

https://github.com/ANANTH-SWAMY/NUMBER-DETECTION-WITH-MEDIAPIPE/blob/main/number_detection.py

MODEL

Architecture diagram with explanation:

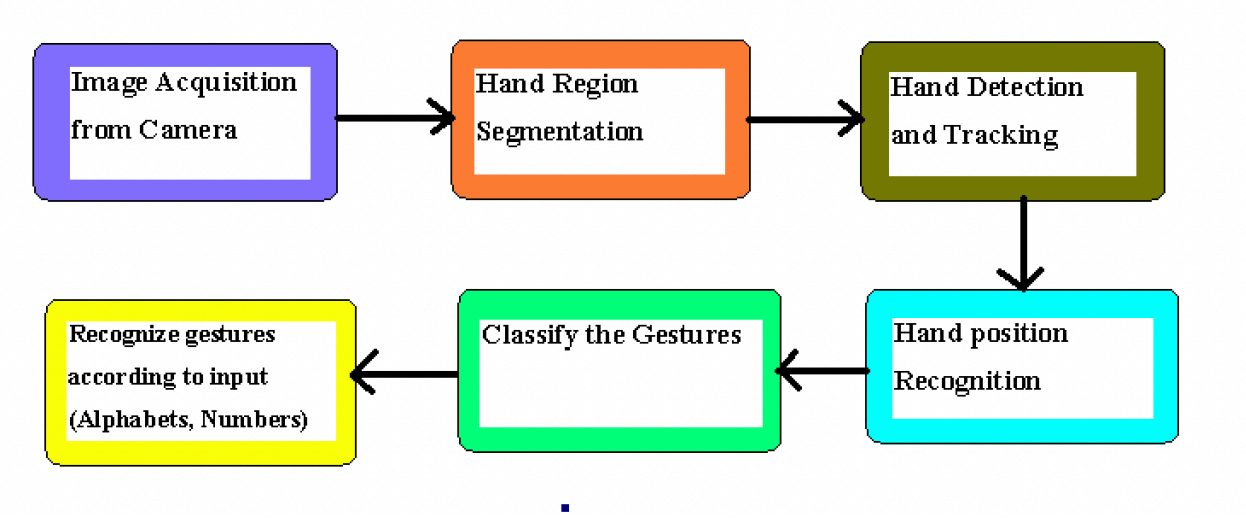


Fig 1. Architecture Diagram of Hand Gesture Recognition

Explanation:

In this project, the architecture diagram explains how to recognize hand gestures. Firstly, read the webcam framework. Then give the hand region segmentation, this is where the background edit section. If we are given a particular area the only hand is detected no matter what runs in the background. Or we can also be given an area where we can place our hands and give the gestures. Then delete the background and track the hand gesture. After the hand gesture read the position and the hand can classify that gesture. After the hand gesture recognition through the webcam, we were given the dataset of the gestures. Collecting the data of the gestures and giving the inputs like alphabets or numbers then according to the input the image pop-ups.

Workflow diagram with explanation:

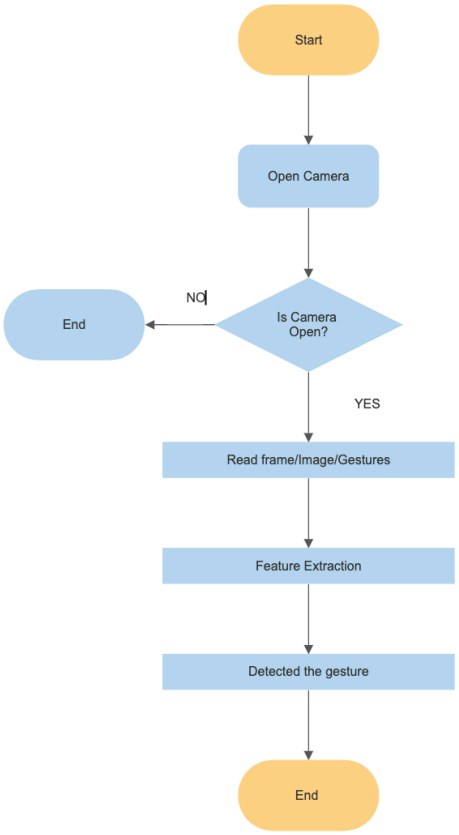


Fig 2: Workflow diagram of Hand gesture recognition

The workflow of the hand gesture recognition process starts with the open camera. Give the webcam frameworks and check whether the camera is working or not. It is not working end the process and find the errors. If the camera is working, then read the frameworks or the gestures of the given input. Here during the feature extraction which means when the webcam is opened gives a small area where only hand features can be read. Give the gestures in that area. Then extract the features of the hand or the hand gestures. Let's say give the gestures as "A" then according to the features it should recognize that the symbol is "A". Same goes with the words and numbers. After that, we have given the dataset with the alphabets collection, and from that, we extract the images to the given input. Let's, say I have given the input as the "M" the output should show me the gesture for the symbol "M". That's how the feature extraction is done using machine learning techniques.

DATASET

Description of Dataset:

In this project, we have collected the dataset from Kaggle for increment 1. Where the data file contains the gestures of the hand with the letters assigned. The MNIST is the handwritten digits that are popular for image processing-based machine learning. For Increment 2, we have the 1000 plus images as the test and the train. The test data file contains the A-Z alphabet letters images, and the train also contains the same data as the test. Before implementing the model, we need to train the model which is called the pre-processing where we set ready before the model implementation. When we collected the data which has the A-Z letters when we give the input it finds the gesture related to that letter. We have the train, and test data files for the preprocessing, and for increment 1 we used the CSV files for the alphabet to recognize the gestures.

Detail design of features with diagram

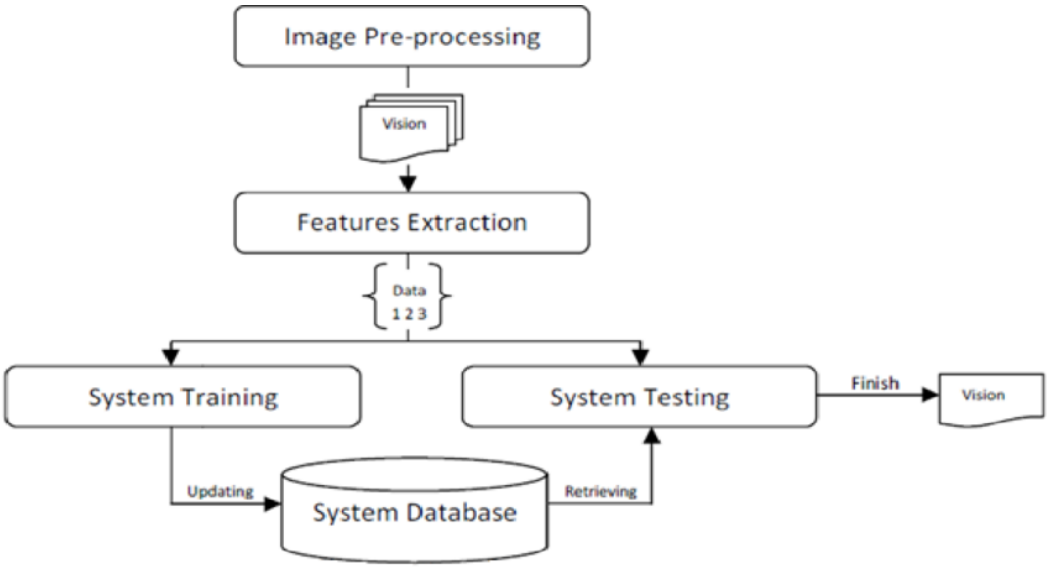


Fig 3: Detail design of the dataset

Firstly, collect the dataset or can create your own dataset. Then knowing about the dataset design features. Initially, the image processing is done nothing but the setting up of the model. Predict the results of the process with the feature extraction. In our project feature extraction is mean recognizing the gestures from the data file given. Then we train them and test the dataset files. First, the training is system training will be done. Training the dataset to get ready before the model implementation. Then we will test the data for the prediction of the result whether it is providing correct results or not. After all these processes we will verify the gestures by giving the input. Recognize the gestures accordingly. If correct results are not produced run back to the model and verify the error.

ANALYSIS OF DATA

Data Preprocessing

Analysis of the data can be done in many ways here in this project we are analyzing the data using the webcam frameworks. Firstly, collect the data and load the given data. Since we are using the webcam for hand gesture recognition check the camera frameworks by giving the id check if the webcam is opening or not if not give the condition as not opened. After checking the webcam frameworks to check if the given data is correct or not print the data provided in the name of the class.

```
# load the hand gesture model

mpHands = mp.solutions.hands
hands = mpHands.Hands(max_num_hands=1, min_detection_confidence=0.7)
mpDraw = mp.solutions.drawing_utils

model = load_model('mp_hand_gesture')

# open the gesture names file
# read the names to the classNames

f = open('gesture.names', 'r')
classNames = f.read().split('\n')
f.close

print(classNames)
```

```
['okay', 'peace', 'thumbs up', 'thumbs down', 'call me', 'stop', 'rock', 'live long', 'fist', 'smile']
```

Fig 4: Data preprocessor using the webcam frameworks

A webcam is one part of the implementation we have another part where we can recognize the gestures or convert the gestures to sign language. For this, we are not using any webcam. By giving the train and test data files we are preprocessing the data and checking the input values which has the class names.

```
#Normalizing the data before feeding to the model
train_datagen = ImageDataGenerator(rescale = 1/255, validation_split = 0.2)
test_datagen = ImageDataGenerator(rescale = 1/255)
```

```
#Loading the train and test data files and
#find the how many classes are present
train_generator = train_datagen.flow_from_directory(
    'Dataset files/Alphabets/Train',
    target_size = (28, 28),
    batch_size = 128,
    class_mode = "sparse",
    color_mode='grayscale',
    #subset = 'training'
    )

test_generator = test_datagen.flow_from_directory(
    'Dataset files/Alphabets/Test',
    target_size = (28, 28),
    batch_size = 128,
    class_mode = "sparse",
    color_mode='grayscale'
    )
```

```
Found 19246 images belonging to 17 classes.
Found 7172 images belonging to 24 classes.
```

```
#Class labels
classes = [char for char in string.ascii_uppercase if char != "J" if char != "Z"]
print(classes, end = " ")
```

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

Fig 5: Data preprocessing without webcam

Graph model with explanation

Since we are using the webcam framework for part one of the code implementations, we are not having any graph model for the webcam framework. Because we are using the live camera or the webcam access and recognizing the gestures, we are not using the graph model. Relatable we used to check the evolution of the data in the previous increment. Were we given the gestures of the images from the A-Z letters process them and checked the evolution of that images? In this increment, we also got to check the loss and accuracy of the model which can be explained in the implementation part.

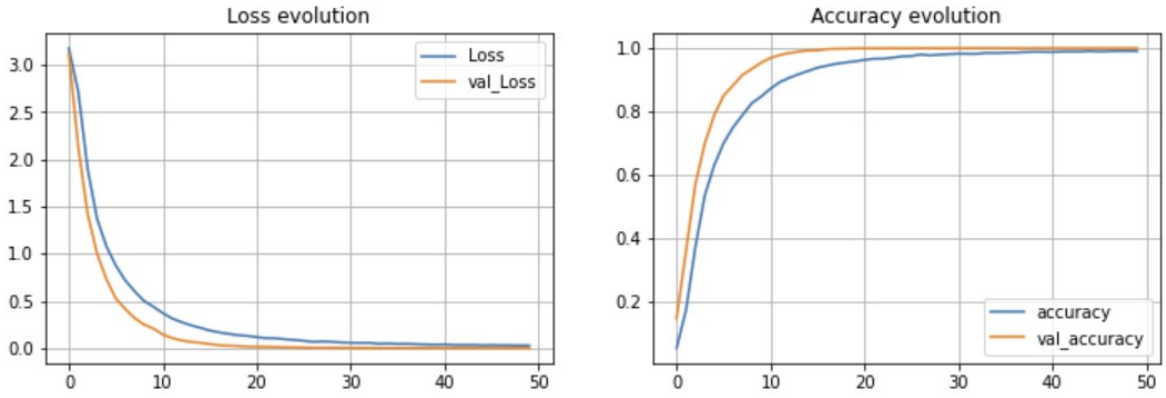


Fig 6: Graph Model

IMPLEMENTATION

Algorithms/Pseudocode

For the implementation of this project, we are using two methods one is with using the webcam, and the other using the Keras model. In the webcam frameworks, we collect the data and check whether the webcam frameworks are working or not and then process the data before running we predict the data and whether we have the correct set of inputs to give. So, in the first part, we just use the webcam frameworks. Secondly, we use the Keras model for the implementation of hand gesture recognition. Before we used the CNN model because CNN is the best image-processing classification we use. It gives accurate results. Here we are using the Keras model which also helps in image processing but does not compare to the CNN model.

Keras is also the neural network model but acts as the API which is integrated with TensorFlow where it is developed by the machine learning process. Keras is simple to use comparatively with CNN Model and user-friendly, but we can only as the API models.

Explanation of Implementation

Implementation using the webcam framework

As we mentioned before we are having two parts. One is with webcam access and the another is without the webcam. The below-given image is the webcam framework implementation. Here, we give the data, process the input values then predict the values. When we run the program the camera window pops up. If we give the gestures, it should recognize the symbols. Here we don't use any other image processing, or image classification models. Because the live webcam can recognize the gestures by the gesture model installation.

```
cap = cv2.VideoCapture(0)

while True:
  # Read each frame to frame from the webcam
  _, frame = cap.read()
  x , y, c = frame.shape

  # Flip the frame vertically
  frame = cv2.flip(frame, 1)
  if cv2.waitKey(1) == ord('q'):
    break
  framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
  # Get landmark prediction of the hand

  result = hands.process(framergb)

  className = ''

  # process the result
  if result.multi_hand_landmarks:
      landmarks = []
      for handslms in result.multi_hand_landmarks:
          for lm in handslms.landmark:
              # print(id, lm)
              lmx = int(lm.x * x)
              lmy = int(lm.y * y)

              landmarks.append([lmx, lmy])

            # Drawing landmarks on frames
          mpDraw.draw_landmarks(frame, handslms, mpHands.HAND_CONNECTIONS)

  # Predict gesture in Hand Gesture Recognition project
      prediction = model.predict([landmarks])
      print(prediction)
      classID = np.argmax(prediction)
      className = classNames[classID]

  # show the predicted text on the frame
      cv2.putText(frame, className, (10, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2, cv2.LINE_AA)

  # Display the final output
  cv2.imshow("Output", frame)

# Quit the webcam and destroy all the active windows
cap.release()
cv2.destroyAllWindows()
```

Fig 7: Model of hand gesture using the webcam

Implementing using the Keras model

In the previous increment, we used the CNN model for accurate results, and we also used canny edge detection. Here we are using the Keras model. Keras model is the API-generated machine learning process that mainly generates and works with TensorFlow. It is user-friendly model than the CNN model which also the similarity between the keras and the CNN model. In the below implementation, we use the Keras model which has layers that can be performed.

```
#A small network of single convolution and 3 Dense layers
model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(64, (3,3), activation = "relu", input_shape = (28,28,1)),
        tf.keras.layers.MaxPool2D((2,2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(256, activation = "relu"),
        tf.keras.layers.Dense(256, activation = "relu"),
        tf.keras.layers.Dense(len(classes), activation = "softmax")
])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 64)        640

 max_pooling2d (MaxPooling2D  (None, 13, 13, 64)       0
 )

 flatten (Flatten)           (None, 10816)             0

 dense (Dense)               (None, 256)               2769152

 dense_1 (Dense)             (None, 256)               65792

 dense_2 (Dense)             (None, 24)                6168

=================================================================
Total params: 2,841,752
Trainable params: 2,841,752
Non-trainable params: 0
_________________________________________________________________
```

Fig 8: Keras Model

Train the model

After the implementation of the Keras model, we train the model. The below we have given the 10 epochs but we have as many epochs as we want. Since I have the bigger data file I am using only the 10 epochs which makes my model train in lesser time.

```
#Traning upto 10 epochs
history = model.fit(
    train_generator,
    epochs=10,
    callbacks = [callback],
    validation_data = test_generator
)
```

```
Epoch 1/10
151/151 [==============================] - 67s 428ms/step - loss: 1.7200 - accuracy: 0.4666 - val_loss: 3.8433 - val_
accuracy: 0.5544
Epoch 2/10
151/151 [==============================] - 34s 223ms/step - loss: 0.3520 - accuracy: 0.8977 - val_loss: 4.0180 - val_
accuracy: 0.6287
Epoch 3/10
151/151 [==============================] - 38s 247ms/step - loss: 0.0740 - accuracy: 0.9833 - val_loss: 4.7225 - val_
accuracy: 0.6525
Epoch 4/10
151/151 [==============================] - 32s 212ms/step - loss: 0.0234 - accuracy: 0.9954 - val_loss: 5.0198 - val_
accuracy: 0.6085
Epoch 5/10
151/151 [==============================] - 33s 221ms/step - loss: 0.0064 - accuracy: 0.9991 - val_loss: 6.0346 - val_
accuracy: 0.6591
Epoch 6/10
151/151 [==============================] - ETA: 0s - loss: 8.3753e-04 - accuracy: 0.9999
Reached 99.6% accuracy so cancelling training!
151/151 [==============================] - 35s 229ms/step - loss: 8.3753e-04 - accuracy: 0.9999 - val_loss: 6.7942 -
val_accuracy: 0.6578
```

Fig 9: Train the model

Accuracy and Loss Validation

After every model is trained, we check for the accuracy and the loss of the model. Since we trained the model. We will check for accuracy and loss by comparing the original images and the predicted images. That is where we find the accurate results.

```
# Retrieve a list of list results on training and test data
# sets for each training epoch
#-----------------------------------------------------------
acc=history.history['accuracy']
val_acc=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']

epochs=range(len(acc)) # Get number of epochs


# Plot training and validation accuracy per epoch
#------------------------------------------------
plt.plot(epochs, acc, 'r', "Training Accuracy")
plt.plot(epochs, val_acc, 'b', "Validation Accuracy")
plt.title('Training and validation accuracy')
plt.figure()


# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot(epochs, loss, 'r', "Training Loss")
plt.plot(epochs, val_loss, 'b', "Validation Loss")


plt.title('Training and validation loss')
```

Fig 10: Accuracy and Loss validation

RESULTS

As we mentioned earlier that we are using the two parts in this increment. The below given are the hand gesture recognition using the webcam framework. In the Images, we have given the gestures accordingly it is showing the results in the red text. The gesture given in image 1 and image 2 is peace according to the gesture using the hand gesture model it recognizes the shape and points of the hand and recognizes the gesture.

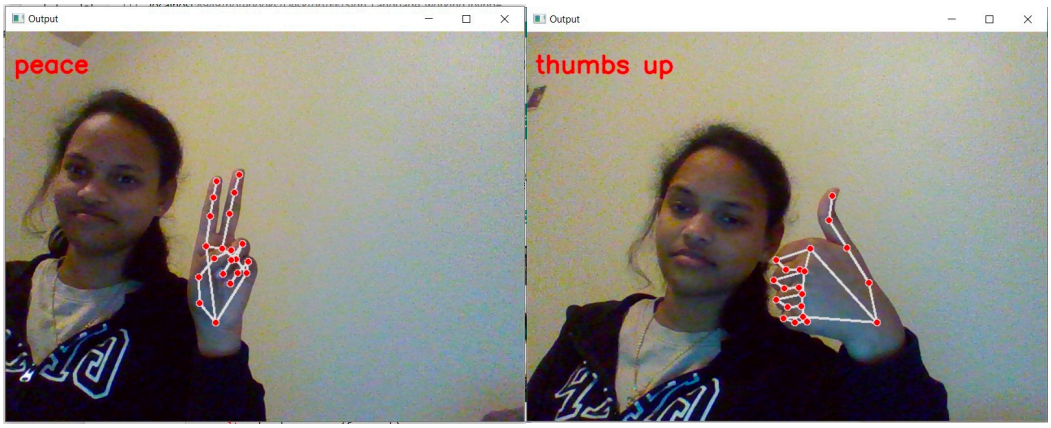


Fig 11: Results for the webcam framework

This is the second part of the implementation using the keras model. After the successful implementation of the model and we have given the input as the "N" then output result gives the gesture symbol "N". Similarly if anyone wants to learn or wants to know the gestures for the letters if we give the input from A-Z we will get the gestures.

```
]: testModel("N")

Dataset files/Alphabets/Test/N\5343_N.jpg
1/1 [==============================] - 1s 802ms/step
```

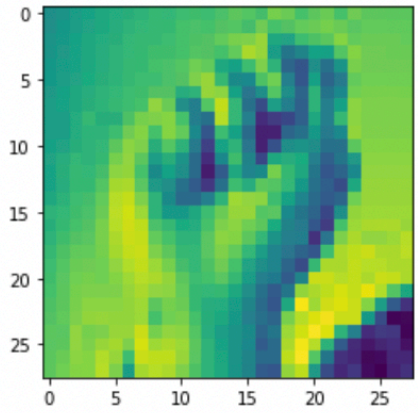


Fig 12: Output Result for the Hand gesture recognition using the keras model

Accuracy and Loss Graph

As mentioned before in the implementation part the accuracy and the loss are calculated to check or to compare the difference between the original image and the predicted image. In the below-shown image, the accuracy is high, and we have the very good prediction results.

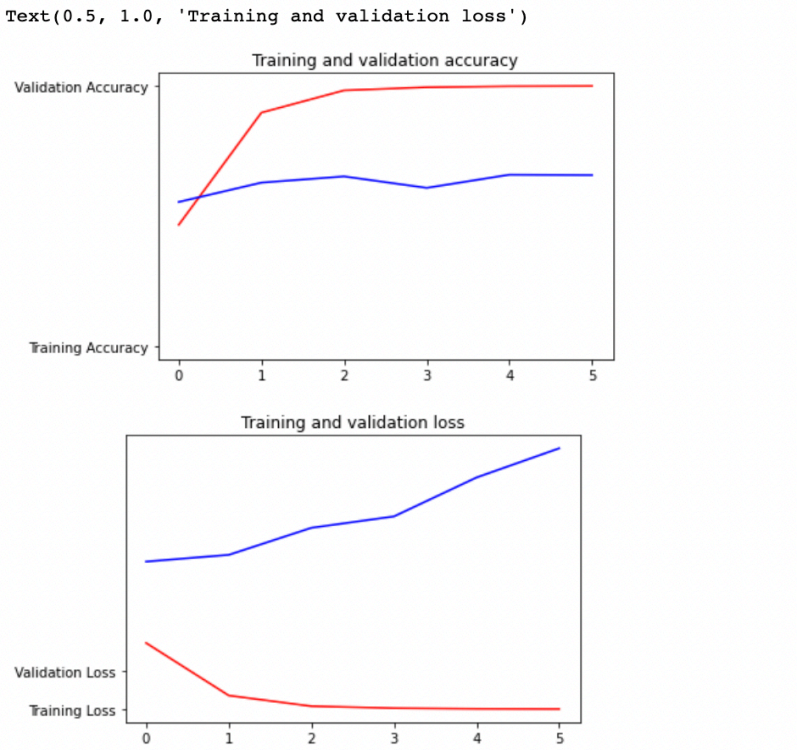


Fig 13: Accuracy and Loss graph

PROJECT MANAGEMENT

Implementation status report

Work Completed

- Description

  We have implemented hand gesture recognition using the webcam framework and hand gesture model, also using the Keras model with TensorFlow. In Increment 1 we implemented hand gesture recognition using the CNN model and the canny edge detection.

- Responsibilities & Contribution

Name	Task	Percentage
Bhavana	Data Collection, Preprocessig, Implemented using the webcam & hand gesture model	50
Sreeja	Data collection, Preprocessing, Training, Keras Model, Documentation	50

- Issues/Concerns

  We tried to capture the image's hand gestures from the live webcam, store the images, and apply the image classifications to them which recognize the hand gestures and convert them into sign language. Unfortunately, due to the framework and the version issues, we are unable to capture the images from the webcam, so we have given another data file.

REFERENCES

1. *Shin, Jungpil, et al. "American Sign Language Alphabet Recognition by Extracting Feature From Hand Pose Estimation - PMC." PubMed Central (PMC), 31 Aug. 2021, www.ncbi.nlm.nih.gov/pmc/articles/PMC8434249.*

2. *Gu, Yutong, et al. "Frontiers | American Sign Language Translation Using Wearable Inertial and Electromyography Sensors for Tracking Hand Movements and Facial Expressions." Frontiers, 1 Jan. 2001, www.frontiersin.org/articles/10.3389/fnins.2022.962141/full.*

3. *"ASL Alphabet." ASL Alphabet | Kaggle, www.kaggle.com/datasets/grassknoted/aslalphabet. Accessed 18 Oct. 2022.*

4. *"Hand Gesture Recognition Database With CNN." Hand Gesture Recognition Database With CNN | Kaggle, www.kaggle.com/code/*

5. "Hand Gesture Recognition Using Image Processing and Feature Extraction Techniques." *Hand Gesture Recognition Using Image Processing and Feature Extraction Techniques - ScienceDirect*, 1 July 2020, www.sciencedirect.com/science/article/pii/S187705092031526X.

6. "Sign-Language Classification CNN (99.40% Accuracy)." *Sign-Language Classification CNN (99.40% Accuracy) | Kaggle*, www.kaggle.com/code/sayakdasgupta/sign-languageclassification-cnn-99-40-accuracy. Accessed 10 Nov. 2022.

7. Team, TechVidvan. "Real-Time Hand Gesture Recognition Using Tensorflow & Opencv." *TechVidvan*, 21 July 2021, 1. https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/.