GESTURE TRANSLATION INTO ASL
Increment 1

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

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

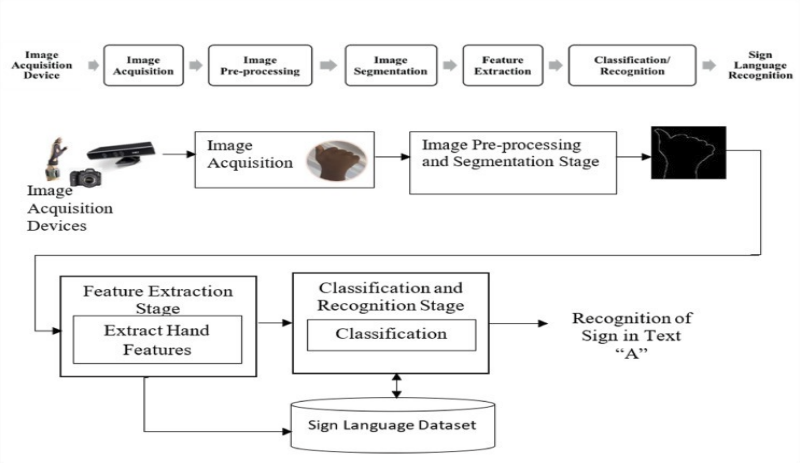
In this, we used canny edge detection which helps to detect the range of the edge in the image.

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.



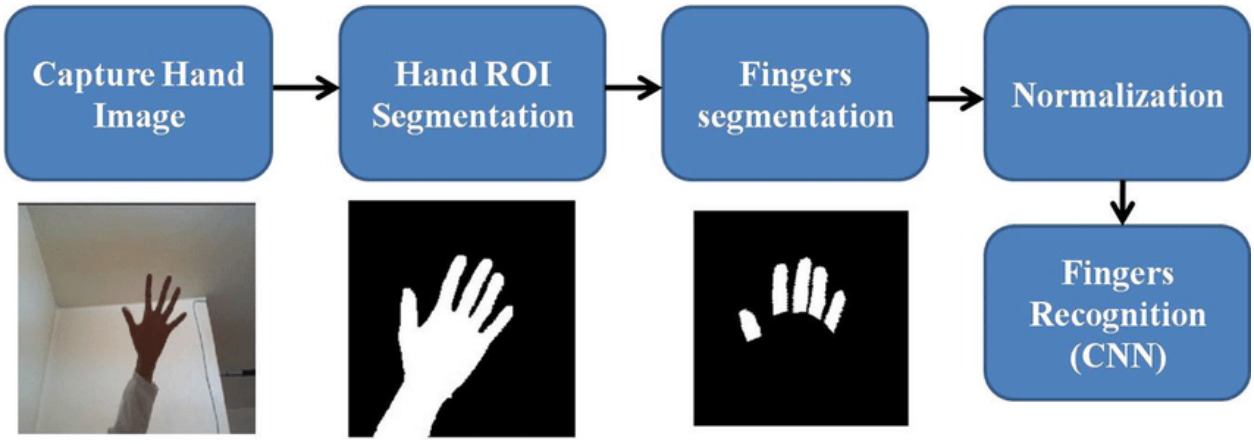
DATASET

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

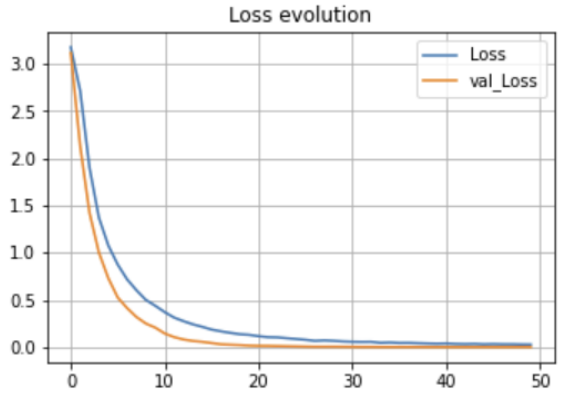


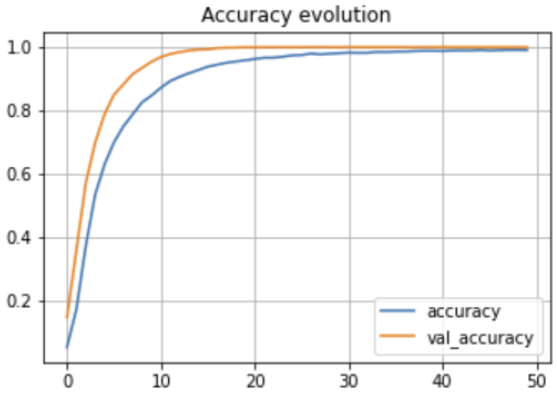
ANALYSIS OF DATA

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:





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

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.

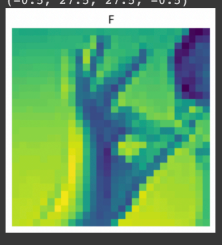
For the future preprocessing will convert this data frames into arrays.

```
[ ] # 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' ]
```

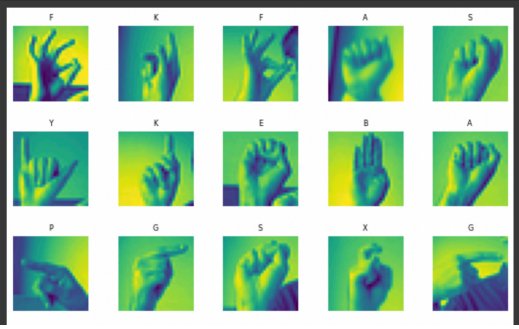
```
#Check a random image for class label verification
i = random.randint(1,27455)
plt.imshow(train_set[i,1:].reshape((28,28)))
plt.imshow(train_set[i,1:].reshape((28,28)))
label_index = train["label"][i]
plt.title(f"{class_names[label_index]}")
plt.axis('off')
```

```
(-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.

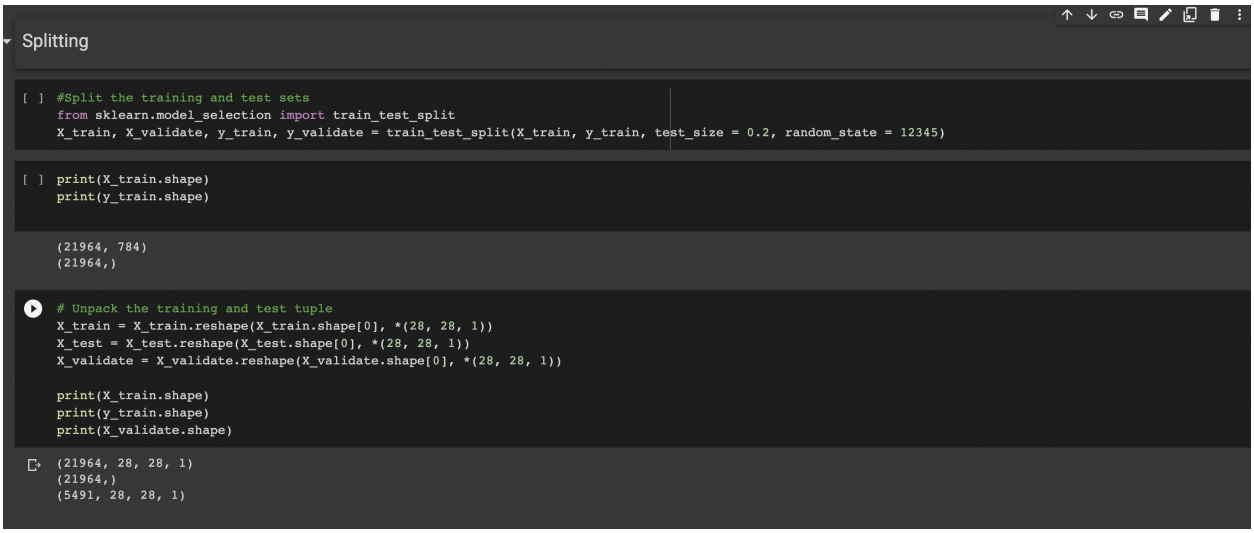
```
# Define the dimensions of the plot grid
W_grid = 5
L_grid = 5
fig, axes = plt.subplots(L_grid, W_grid, figsize = (10,10))
axes = axes.ravel() # flaten the 15 x 15 matrix into 225 array
n_train = len(train_set) # get the length of the train dataset
# Select a random number from 0 to n_train
for i 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)
```



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()
```

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)
```

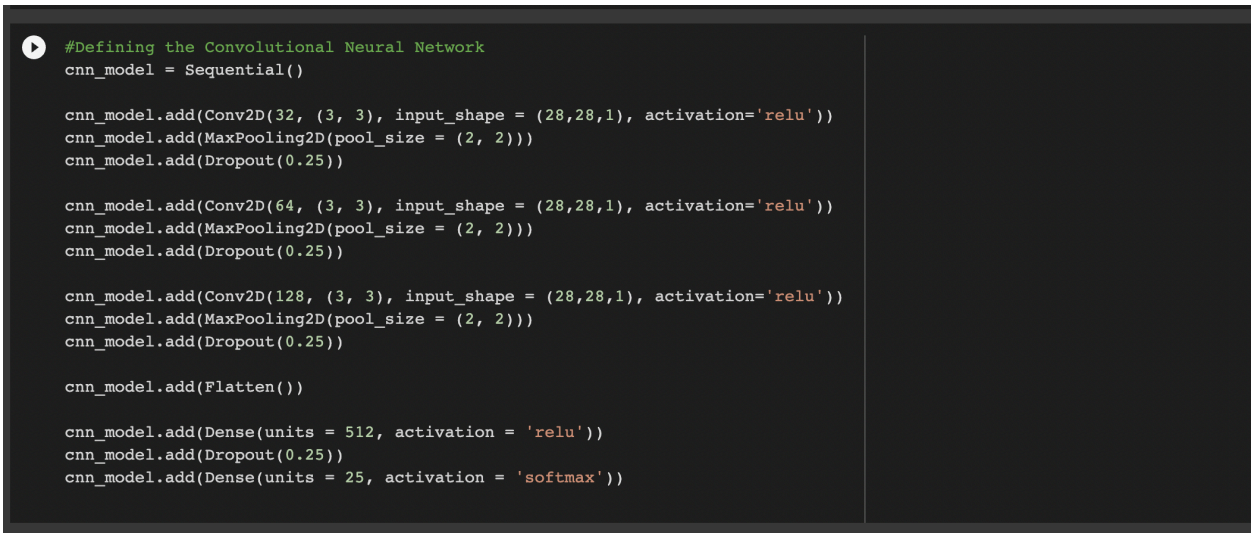
```
(21964, 784)
(21964,)
```

```
# 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'))
```

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.

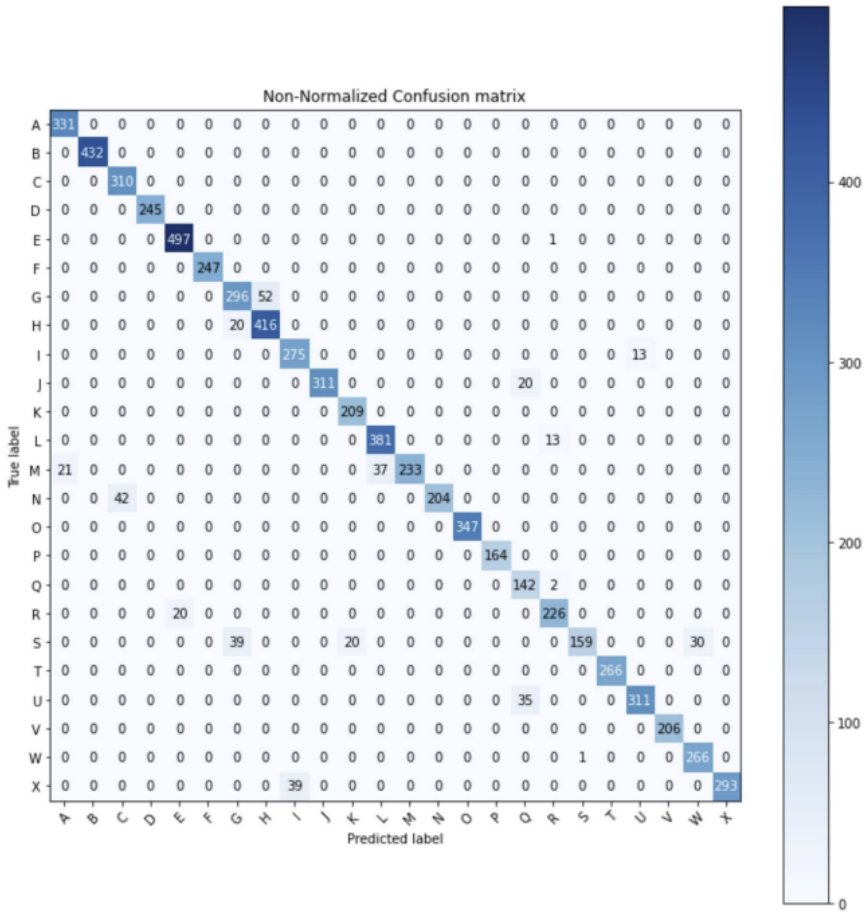
```
# perform the canny edge detector to detect image edges
edges = cv2.Canny(gray, threshold1=30, threshold2=100)
plt.imshow(edges)
```

```
<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

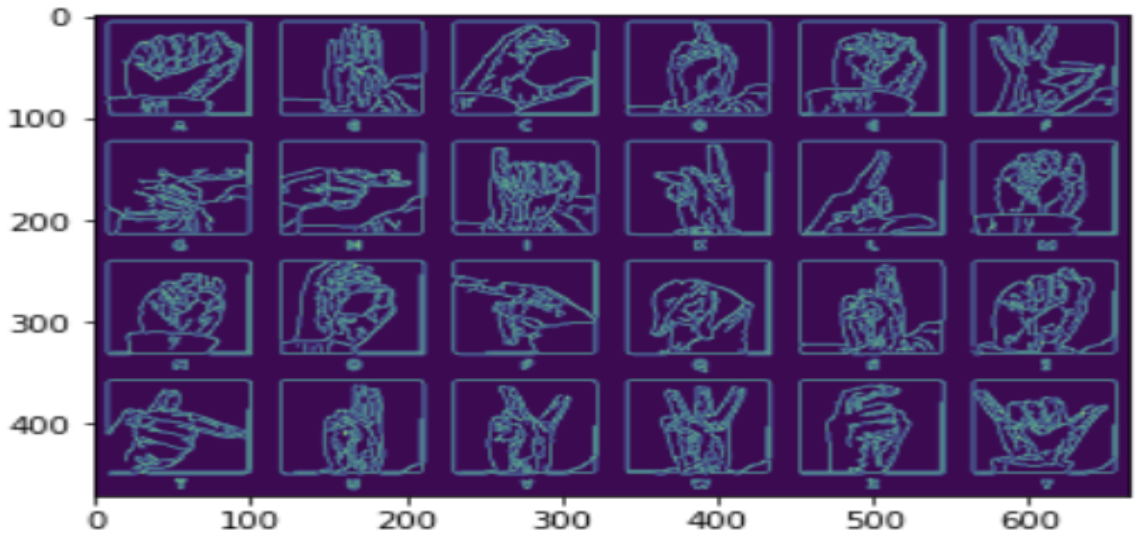
PRELIMINARY RESULTS

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).

Input Image

Converting to Grayscale

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 using the CNN model will implement the project with the help of video capturing.

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-language-classification-cnn-99-40-accuracy. Accessed 10 Nov. 2022.