COMP-5435-FA Fall 2019

Final Project Report

Project Title: ASL Recognition: A comparison between SIFT and Gabor Filter for Feature Extraction

Instructor: Dr. Shan Du

Group Members:

Student Name	Student ID	Email ID
Abhigya Koirala	1115723	akoirala@lakeheadu.ca
Sagar Deshpande	1109819	sdeshpan@lakeheadu.ca
Isura Thrikawala	1128519	ithrikaw@lakeheadu.ca
Shreya Reddy	1121153	sreddy@lakeheadu.ca

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Many people suffer from hearing disabilities and deaf-mute is one of the major disabilities. Around 360 million people suffer from some kind of hearing disabilities in the world [1]. Sign language (SL) is the medium of communication for deaf-mute people. Sign languages use different hand gestures, orientation and movements of hands as the methods to communicate. Sign language is not a universal language, and different sign languages are used in different countries, like the many spoken languages around the world. Some countries such as Belgium, UK, USA or India may have more than one sign language. They are hundreds of sign languages around the world, for instance, American Sign Language (ASL), British Sign Language (BSL), Pakistani Sign Language (PSL), Japanese Sign Language, Chinese Sign Language (CSL) etc.

Sign language is a visual language and consist of three major components:

- Fingerspelling- Used to spell words letter by letter (Shown in Fig1).
- Word level sign vocabulary- Used for the majority of communication.
- Non-manual features- Facial expressions and tongue, mouth and body positions.

American sign language (ASL) is considered to be one of the most complex but complete sign language. This is most practiced SL around North America and also in many countries who don't have their own SL [2]. ASL consists of 26 signs used to spell the 26 alphabets of English language. The sign uses 19 hand shapes of ASL, for e.g. the sign to represent letter P and K uses the same hand shape but it differs in orientation [2].It also consists of 10 signs for numeric value representation. Fig.1 shows the signs in the ASL.

Figure 1: Signs used in ASL

The availability of the interpreters is very limited. They are expensive as well. This is why we propose to make an American sign language interpreter. There have been many previous works in this project using various methods. Number of input devices has been in the past to achieve some good results such as data-gloved approach, Leap Motion Controller, vision-based approach, and Kinect [3]. But due to lack of its implementation outside the laboratory and high cost of the input devices it's not feasible. Thus, we propose to implement a more vision-based approach which uses different aspects of computer vision such as Camera Initialization & Image Processing, Pre-Processing and Hand Tracking, Gray Scaling & Threshold along with machine learning and deep learning classifiers.

2. Literature Review

For vision-based method, computer camera is the main input device for observing the information of hands or fingers. This method requires only a camera, thus realizing a natural interaction between humans and computers without the use of any extra devices. These systems tend to complement biological vision by describing artificial vision systems that are implemented in software and/or hardware. This poses a challenging problem as these systems need to be background invariant, lighting insensitive, person and camera independent to achieve real time performance. Moreover, such systems must be optimized to meet the requirements, including accuracy and robustness. The vision-based hand gesture recognition system is shown in Fig 2.

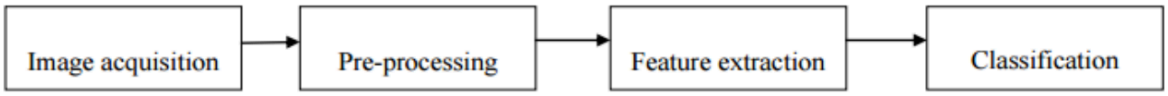

Figure 2: Gesture Recognition system

Since vision-based analysis is based on the way humans perceive information about their surroundings, it is still the most difficult to implement. There has been immense research done on hand sign language gesture recognition. Researchers in Indonesia had done a similar project, Sentence Level Indonesian Sign Language Recognition Using 3D Convolutional Neural Network and Bidirectional Recurrent Neural Network. They used deep learning method by combining Convolutional Neural Network (CNN) and Bidirectional Recurrent Neural Network (Bi-RNN) [3]. They implemented 3D CNN (Shown in Fig 3) to extract the feature from each frame of video and implemented bidirectional-RNN to extract the unique features from the video frame's sequential behavior to generate a possible sentence.

Researchers from India did a similar project called Indian sign language recognition using ANN and SVM classifiers. They only worked on the numerals. They used a self-created dataset of 1000 images with 100 images per numeral signs. They used shape descriptors, Scale Invariant Feature Transform (SIFT) and Histogram of Oriented Gradients (HOG) techniques are used for extracting desired features and used Artificial Neural Networks (ANN) and Support Vector Machine (SVM) classifiers are used to classify the signs. They achieved an accuracy of 99%. [4]

Another group of researched did a similar project called American Sign Language Recognition using Deep Learning and Computer Vision. They created a vison-based application for the translation of the signs. They proposed a model which took video sequences and extract temporal and spatial feature from them. They used a CNN for recognizing the spatial features and a RNN to train the temporal features. They were able to achieve the accuracy of 92% [5] .

Another group of researched did a similar project called Vision-Based Approach for American Sign Language Recognition Using Edge Orientation Histogram. They proposed a real time static ASL recognizer. They divided the ASLR system into six phases, image capturing, image pre-processing, region extraction, feature extraction, feature matching and pattern recognition.

They used Edge Orientation Histogram (EOH). They achieved accuracy of 88.26% with recognition time of 0.5 second in complex background with mixed lightning condition [6].

3. Dataset

The dataset we would be using is ASL alphabet dataset by students at PES University, Bangalore, Karnataka, India. It is a collection of images of alphabets from ASL separated in 29 folders to represent various classes [7]. The classes are 26 letters from A-Z and the other 3 classes represent Space, Delete and Nothing respectively. It contains 87,000 images of 200x200 pixels for training.

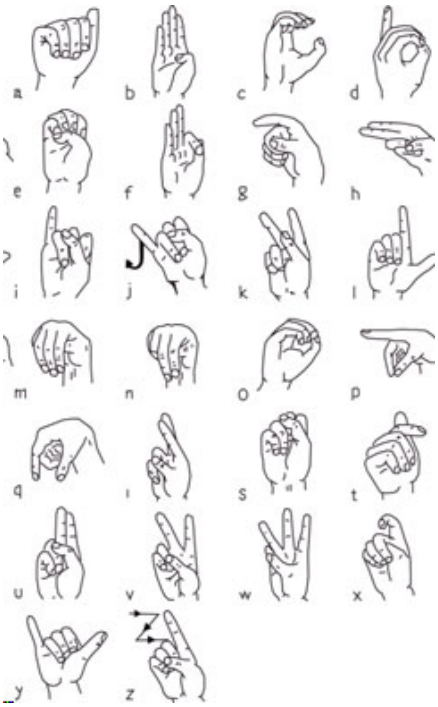

Figure 3: ASL Alphabets

An example for A is shown below

Figure 4: Alphabet 'A' in ASL

4. Papers Implemented

We have taken 2 recent papers similarly related to our work. The papers are:

4.1. Bangla Sign Language Detection Using SIFT and CNN

This paper was published by IEEE in 2018. This paper was presented in 2018 *on 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Bangalore [8]. The authors of this paper are S. S. Shanta, S. T. Anwar and M. R. Kabir. This work done in this paper was closely related to the work we were trying to do. The paper was recently published as well. The only difference was the dataset. The authors in this paper were classifying Bangla sign language where as were doing the work for American Sign language. The authors in this paper have implemented CNN but we have implemented MLP for the classification due to lack of time for the implementation. Also, we have used SVM as well to compare the results with the other paper we have implemented.

4.2. Hand sign language recognition for Bangla alphabet using Support Vector Machine

This paper was published by IEEE in 2016. This paper was presented in on *2016 International Conference on Innovations in Science, Engineering and Technology (ICISET)*, Dhaka [9]. The authors of this paper are M. A. Uddin and S. A. Chowdhury. This work done in this paper was closely related to the work we were trying to do. The paper was recently published as well. The only difference was the dataset. The authors in this paper were classifying Bangla sign language whereas were doing the work for American Sign language. The authors in this paper have implemented SVM but we have implemented MLP as well along with SVM for the classification to compare the results with the other paper we have implemented.

5. Objectives

The main objective of our project was to achieve a better recognition system for ASL alphabets using different classifier and feature extraction methods. After researching many papers and journals we found out that SVM, CNN and MLP could give us a better result. Also SIFT and Gabor features would give us better features for our dataset. So, our main objective was to find the combination of best classifier and the best feature extraction techniques for the classification of ASL alphabets. Thus, we tried to implement two of the papers with SVM, MLP and CNN in order to find the best possible combination to overcome the problem of ASL alphabet classification problem.

6. Methodologies

6.1. Classification using Gabor Feature

6.1.1. RGB to HSV Conversion

The ASL dataset is from kaggle is taken, which are separated into 29 different folders which represents the 29 different classes. The 29 different classes include the 26 English alphabet, delete which is used to delete the alphabets for classification, nothing which represents the background and space to give spaces between the alphabets We take only 26 classes, which are the English alphabets. The images are taken in different lightning conditions from different angle. The size of every image is 200 x 200

A python script is written to iterate through all the folder and all the images inside them.

```
#Get List of sub directories
subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if os.path.isdir(os.path.join(dirpath,o))]
i=0
while(i<len(subdirs)):
    res=subdirs[i][subdirs[i].rindex('/')+1:]
    newDest = destDirectory+res
    oldDir = dirpath+'/'+res
    os.mkdir(newDest)
    filenames = os.listdir(oldDir)
    for fname in filenames:
        srcpath = os.path.join(oldDir, fname)
        preprocess_for_gabor(srcpath,fname,newDest)
    i+=1
print(len(subdirs))
# Run the gaborextract python file next
```

Figure 5: Iterating through all sub folders and files

Then RGB to HSV conversion is done for all the images and copied to a different folder

```
img=cv2.imread(filePath)
img_HSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
#skin color range for hsv color space
HSV_mask = cv2.inRange(img_HSV, (0, 15, 0), (17,170,255))
```

Figure 6: RGB to HSV conversion

The RGB model is made up of three different colors which are red, green and blue [10]. We are converting from RGB to HSV color space because HSV is less sensitive to illumination than the RGB. The HSV color space is made up of Hue, Saturation and value [11]. The hue ranges from 0 to 360 degrees where different range of the angle represent different color. Value gives the brightness of color and ranges from 0 to 100% and the saturation represents the amount of gray in the color which also ranges from 0 to 100%. After we provide the threshold value of HSV, the skin region of the hand can be segmented from the image.

```
11      #skin color range for hsv color space
12      HSV_mask = cv2.inRange(img_HSV, (0, 15, 0), (17,170,255))
```

Figure 7: Skin segmentation using HSV thresholding

6.1.2. HSV to Binary Conversion

The binary images only have only two-pixel values that is 0 and 255 which are two potential colors black and white. This helps us to carry out morphological operation and also decreases time for further computation and operation.

A python script is written to convert the image in HSV color space to binary image.

```
15      #Binary conversion
16      HSV_result = cv2.bitwise_not(HSV_mask)
```

Figure 8: HSV to Binary Conversion

6.1.3. Morphological operation

Morphological image processing consists of non-linear operation, that are related to the image's feature's shape. They only rely the relative ordering of the pixel values and thus they are suited to the processing of binary images. The fundamental morphological operations include dilation and erosion.

The dilation operation adds the pixels to the boundaries of objects in an image and the erosion removes the pixels on the boundaries of an object. The pixels added and removed depends upon the shape and size of the structuring element used in the image processing.

We have implemented a variant of erosion and dilation which is opening. The process involved in opening is erosion followed by dilation. This is useful in removing noise.

```
#Morphological Operations
HSV_mask = cv2.morphologyEx(HSV_mask, cv2.MORPH_OPEN, np.ones((3,3), np.uint8))
```

Figure 9: Morphological operations

6.1.4. Feature Extraction Using gabor filter

Gabor filter is a linear filter which is used in image processing applications for edge detection, image analysis, feature extraction etc. Gabor filter when it is set with a given direction gives a strong response for location in the target image in that given direction. Gabor filter extracts features from a image from different scales and orientation [12]. A gabor filter of 'n' scale and 'm' orientation gives 'n x m' number of different representations which is also called response matrices. The response matrices when convolve with the target image gives the feature vectors. The feature vector may consist of local energy, mean amplitude phase, amplitude or orientation whose local has maximum energy etc. Gabor filter is invariance to different lighting conditions, translation, scaling, and rotation which is one of its most significant advantage.

Each and every image in different folder is iterated using a python script

```
# To get all images from the path
while (i < len(subdirs)):
    allVectors = []
    allLabels = []
    print(i)
    res = subdirs[i][subdirs[i].rindex('/') + 1:]
    oldDir = dirpath + res
    filenames = os.listdir(oldDir)
    for fname in filenames:
        if (fname != '.DS_Store'):
            print(j)
            srcpath = os.path.join(oldDir, fname)
            print(srcpath)
            # read each image and apply gabor extraction function
            img = cv2.imread(srcpath, 0)
            get_gabor_feature(img, res)
            j += 1

        # Save the values and labels
        np.save('gabor_features_now/gabor'+res+'.npy', np.asarray(allVectors))
        np.save('gabor_features_now/label'+res+'.npy', np.asarray(allLabels))

    i += 1
```

Figure 10: Iterating images for gabor feature extraction

Gabor filter with 5 scale and 8 orientation is taken in order to extract gabor features from the image. 40 features are extracted from each images. A python script is run to save gabor feature of all images of a particular class in a single files. We obtain 26 different files with gabor features of different image of that particular class

```
# Function to extract Gabor features
def get_gabor_feature(image, name):
    classify = np.array([])
    label = np.array([])
    # For 5 scales
    for i in range(0, 5, 1):
        # 8 orientations
        for j in range(0, 8, 1):
            # Get the real values from gabor function
            real_val = gabor(image, frequency=(i + 1) / 10.0, theta=j * pi / 8)[0]
            # Get the imaginary values from gabor function
            img_val = gabor(image, frequency=(i + 1) / 10.0, theta=j * pi / 8)[1]
            # Get the square of both values and add them up to get a complete result
            result = real_val * real_val + img_val * img_val
            res_mean = np.mean(result)
            classify = np.append(classify, res_mean)
            label = np.append(label, name)
    print('Gabor Features:')
    print(classify)
    print('Length Gabor Features:')
    print(len(classify))
    allVectors.append(classify)
    allLabels.append(name)
```

Figure 11: Gabor feature extraction

6.1.5. Sign recognition using Support Vector Machine (SVM)

A Support Vector Machine is a supervised learning model used for classification problem. It can be used alongside of a kernel to transform the data to find an optimal boundary between the classes. SVM is considered as one of the most robust and accurate algorithm amongst the classic pattern recognition algorithms [13].

The equation of a line can be given as y = mx + c. In most traditional classification algorithms, we define a decision boundary which will separate classes. In case of a linear separation, this boundary is a line. For SVM we will rewrite the equation of the line as $y = \beta x + \beta_0$. Where y is the target class, x are the data points and β and β_0 are parameters which will determine the decision boundary. This is the equation of a line however, it can be extended to higher dimensions as well. In higher dimension, the line changes to a hyperplane. Now the goal here is to obtain an optimal boundary such that most of the data points can be easily separated. Which means the distance between the margin needs to be minimized. This means we need to minimize β. However, there is a constraint, this constraint is that the data points which will affect the boundary will only be certain points which satisfy a condition. These points are called support vectors and the condition is $y\ (\beta x + \beta_0) \geq 1$.

So, we have
$\min \frac{1}{2} |\beta|^2$ such that $y\ (\beta x + \beta_0) \geq 1$.
We can define a Lagrangian so solve this,
$L(\beta, \beta_0, \alpha_i) = \frac{1}{2}\beta^T\beta - \sum_{i=1}^{n} \alpha_i [y_i\ (\beta x_i + \beta_0) - 1]$
where n is the total number of samples (so we do this for all of the dataset) and α_i is the lagrangian multiplier.
This problem does not have a closed solution so we write the maximum form of this.
We derive the Lagrangian and set it to zero we get,

$$\frac{\delta L}{\delta \beta} = \beta - \sum_{i=1}^{n} \alpha_i y_i x_i = 0$$

$$\beta = \sum_{i=1}^{n} \alpha_i y_i x_i$$

Next, we have,

$$\frac{\delta L}{\delta \beta_0} = -\sum_{i=1}^{n} \alpha_i y_i = 0$$

and,

$$\frac{\delta L}{\delta \alpha_i} = \sum_{i=1}^{n} [y_i \, (\beta x_i + \beta_0) - 1] = 0$$

we put this in the original and get the maximum as

$$L(\beta, \beta_0, \alpha_i) = \max \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \, \alpha_j y_i y_j x_i x_j \right.$$

such that

$$\alpha_i \geq 0$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

This problem is a quadratic problem and can be solved by SMO optimization. SMO optimization is used to solve the quadratic problem in SVM.

There are 26 classes with each class having 3000 samples, the classes are linearly separable due to which a clear boundary could be established.

A python script is written to obtain a model using SVM. All the images of 26 classes are taken as training data for the SVM.

```
# Function with path given from UI_design class
def svm(path):
    i=0
    feature_array=[]
    labels_array=[]
    for f in path:
            # Get all gabor feature files
       if re.match('g', f):
            features = np.load('./gabor_features/'+f).tolist()
            feature_array += features
            i+=1

    feature_array = np.array(feature_array)

    for f in path:
        # Get all the labels
       if re.match('l', f):
            labels = np.load('./gabor_features/'+f).tolist()
            labels_array += labels
            i+=1

    data = np.array(feature_array)
    labels=np.array(labels_array)
    # Train the model usig MLP
    model = SVC(gamma='auto').fit(data, labels)


    #Computing the SVM for the complete dataset and saving the model
    dump(model, './now_output/finalsvmgabor.joblib')
```

Figure 12: SVM for gabor features

The trained model is used to classify the testing image which is taken as a snapshot using the webcam using a python script.

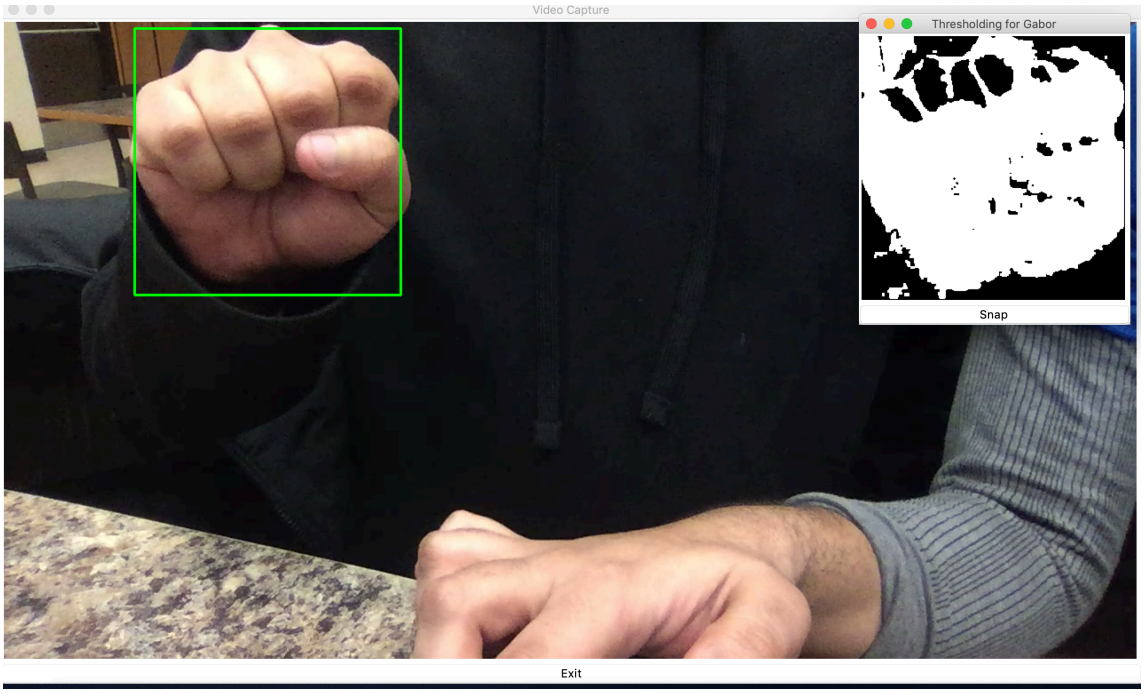

Figure 13: Taking Snapshot for classification

Figure 14: Classification Result

6.1.6. Sign recognition using Multi-Layer Perceptron (MLP)

MLP is a logistic regression classifier where the input gets transformed using a learnt non-linear transformation. It then projects the input data to a space where the data can be linearly separable. This layer is also called a hidden layer [14].

We also use MLP as the other classifier for the sign recognition. Here the extracted gabor features are used as the training data to create a model

```
# Function with path given from UI_design class
def mlp(path):
    i=0
    feature_array=[]
    labels_array=[]
    for f in path:
        # Get all gabor feature files
       if re.match('g', f):
            features = np.load('./gabor_features/'+f).tolist()
            feature_array += features
            i+=1

    feature_array = np.array(feature_array)

    for f in path:
        # Get all the labels
       if re.match('l', f):
            labels = np.load('./gabor_features/'+f).tolist()
            labels_array += labels
            i+=1

    data = np.array(feature_array)
    labels=np.array(labels_array)
    # Train the model usig MLP
    model = MLPClassifier(hidden_layer_sizes=(40,40,40), max_iter=40000).fit(data, labels)


    #Computing the MLP for the complete dataset and saving the model
    dump(model, './now_output/finalmlpgabor.joblib')
```

Figure 15: Creating model using gabor features and mlp

The trained model is used to classify the testing image which is taken as a snapshot using the webcam using a python script.

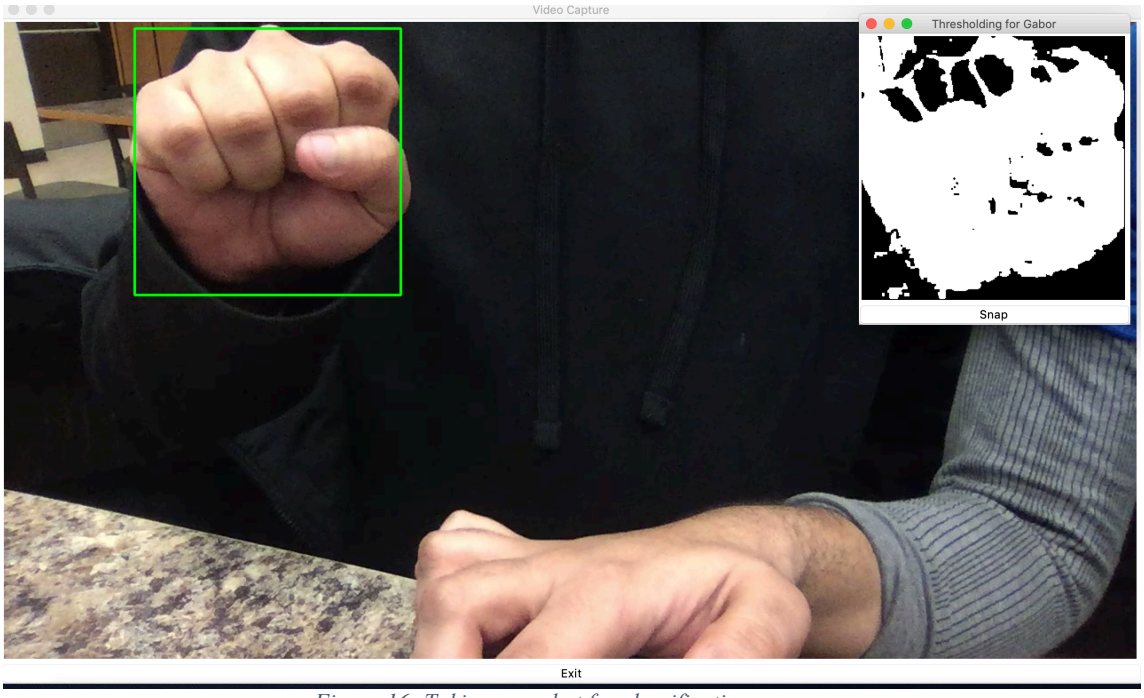

Figure 16: Taking snapshot for classification

Figure 17: Classification result

6.2. Classification using SIFT features

6.2.1. RGB to gray scale

The RGB model is made up of three different colors which are red, green and blue. For the preprocessing to train the model we convert the images in RGB color space to gray scale. A grayscale image just needs intensity information of that pixel which represents how bright or how dark is that pixel. A grayscale image need a single byte for each pixel, which can store a value from 0 to 255. These values can cover all shades of gray.

A python script is run to iterate through each and every image in all the folders.

```
#Set source directory and destination directories
dirpath = './asl_alphabet_train'
destDirectory = './randomTrainGray/'

#Get List of sub directories
subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if os.path.isdir(os.path.join(dirpath,o))]
i=0
while(i<len(subdirs)):
    res=subdirs[i][subdirs[i].rindex('/')+1:]
    newDest = destDirectory+res
    oldDir = dirpath+'/'+res

    os.mkdir(newDest)
    filenames = os.listdir(oldDir)
    for fname in filenames:
        srcpath = os.path.join(oldDir, fname)

        pre_process(srcpath,fname,newDest)

    i+=1
print(len(subdirs))
# Call the sift python file for further process
import siftIt
```

Figure 18: Iterating through all the folders and imagees

Then all the images iterated are converted to grayscale and saved into the folders respective of their classes

```
# Function to convert from RGB to Gray for sift
def pre_process(filePath,name,dest):

    img=cv2.imread(filePath)
    img_HSV = cv2.cvtColor(img, cv2.cv2.COLOR_BGR2GRAY)
    fullpath= dest+'/'+name
    cv2.imwrite(fullpath,img_HSV)
```

Figure 19: Grayscale conversion

6.2.2. Feature Extraction using Scale Invariant Feature Transformation (SIFT)

Corners detectors like Harris corner detector are rotation invariant. SIFT uses difference of Gaussians i.e. approximation of LoG. The difference of gaussians (DoG) can be obtained by the difference of gaussian blur a different scale. After the DoG is found, the local extrema over the scale and space is looked up in the image [15]. For e.g. one pixel in an image is compared with its 10 neighbours as well as 11 pixel in next scale and 11 pixels in the previous image. If that pixel is a local extrema that is regarded as a potential keypoint. The keypoints found are then refined for more accurate results. Taylor series of expansion is used to get more accurate location of that extrema and when the intensity is found to be less than the threshold value it is discarded. DoG also responds more to the edges so edges are removed as well. Then an orientation is assigned to each keypoint. This helps to get invariance to the image rotation. A particular neighbourhood is taken around the keypoint with respect to the scale and the gradient magnitude after which the direction is calculated in that region. After the keypoints are extracted keypoint descriptors are created.

A python script is written to iterate through all the images in each sub folders

```
dirpath = './randomTrainGray/'
subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if os.path.isdir(os.path.join(dirpath, o))]
i = 0

j = 0

while (i < len(subdirs)):
    allKeypoints = []
    allVectors = []
    allLabels = []
    print(i)
    res = subdirs[i][subdirs[i].rindex('/') + 1:]
    oldDir = dirpath + res
    filenames = os.listdir(oldDir)
    for fname in filenames:
        if (fname != '.DS_Store'):
            print(j)
            srcpath = os.path.join(oldDir, fname)
            print(srcpath)

            extract_features(srcpath,res)
            j += 1

        np.save('sift_features_now/sift_'+res+'.npy', np.asarray(allVectors))
        np.save('sift_features_now/label_'+res+'.npy', np.asarray(allLabels))

    i += 1
```

Figure 20: Iteration through all images and sub folders

Then the sift feature is extracted from each and every image. Then a single file is made for the all the sift features of images of that particular class.

```
# The cluster should be about number of classes (26) * 10
def read_and_clusterize(num_cluster=260):

    dirpath = './randomTrainGray/'
    subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if os.path.isdir(os.path.join(dirpath, o))]
    i = 0
    j = 0

    sift_keypoints = []

    #looping through all the images
    while (i < len(subdirs)):
        print(i)
        res = subdirs[i][subdirs[i].rindex('/') + 1:]
        oldDir = dirpath + res
        filenames = os.listdir(oldDir)
        for fname in filenames:
            if (fname != '.DS_Store'):
                print(j)
                srcpath = os.path.join(oldDir, fname)
                print(srcpath)


                #
                #read image
                image = cv2.imread(srcpath,1)
                # grayscale conversionn
                image =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
                # SIFT feature extraction
                sift = cv2.xfeatures2d.SIFT_create()
                kp, descriptors = sift.detectAndCompute(image,None)
                sift_keypoints.append(descriptors)


                j += 1
        i += 1

    sift_keypoints=np.asarray(sift_keypoints)
    sift_keypoints=np.concatenate(sift_keypoints, axis=0)
    #kmeans clustering
    print("Training kmeans")
    kmeans = MiniBatchKMeans(n_clusters=num_cluster, random_state=0).fit(sift_keypoints)
    #return the learned model
    return kmeans

a = read_and_clusterize()
dump(a, 'kmeans.joblib')
```

Figure 21: SIFT feature extraction

6.2.3. Sign recognition using SVM

SVM is already explained in the previous section (1.1.5.) of this report.
A python script is written to create a model using SVM. The obtained SIFT features is used as training data. A model is obtained which is saved to use for further classification of image

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
model = SVC(gamma='auto').fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print("Training: ", model.score(x_train, y_train))
print("Testing: ", model.score(x_test, y_test))

dump(model, './now_output/sift_svm.joblib')
```

Figure 22: SVM model training

The trained model is used to classify the testing image which is taken as a snapshot using the webcam using a python script.

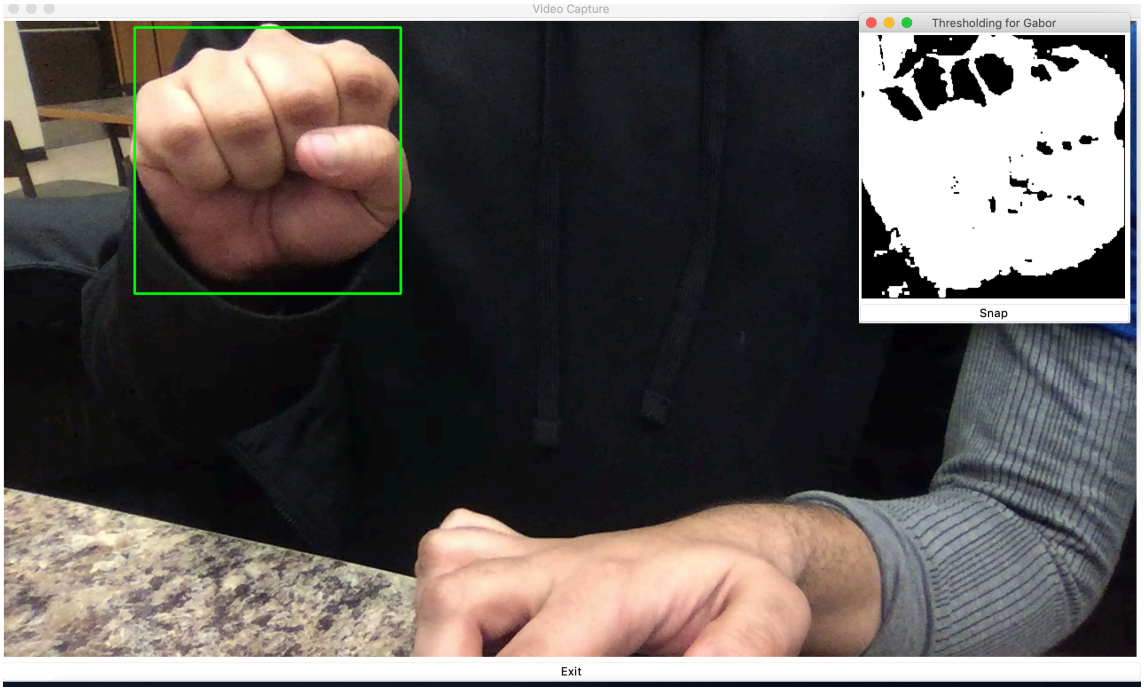

Figure 23: Taking snapshot for classification

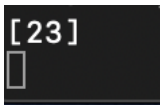

Figure 24: Classification result where 23 corresponds to x

6.2.4. Sign recognition using MLP

MLP is already explained in the previous section (1.1.6.) of this report.
A python script is written to create a model using SVM. The obtained SIFT features is used as training data. A model is obtained which is saved to use for further classification of image

```
#generating the feature vector after the kmeans clustering model
#creating the histogram of classified keypoints from kmeans
def calculate_centroids_histogram(model):

    dirpath = './randomTrainGray/'
    subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if os.path.isdir(os.path.join(dirpath, o))]
    i = 0
    j = 0

    feature_vectors=[]
    class_vectors=[]

    while (i < len(subdirs)):
        print(i)
        res = subdirs[i][subdirs[i].rindex('/') + 1:]
        oldDir = dirpath + res
        filenames = os.listdir(oldDir)
        for fname in filenames:
            if (fname != '.DS_Store'):
                print(j)
                srcpath = os.path.join(oldDir, fname)
                print(srcpath)
                #read image
                image = cv2.imread(srcpath,1)
                #grayscale conversion
                image =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
                #SIFT extraction
                sift = cv2.xfeatures2d.SIFT_create()
                kp, descriptors = sift.detectAndCompute(image,None)
                predict_kmeans=model.predict(descriptors)
                #histogram calculation
                hist, bin_edges=np.histogram(predict_kmeans, bins=260)
                feature_vectors.append(hist)
                class_sample=define_class(res)
                class_vectors.append(class_sample)

                j += 1
        i += 1

    feature_vectors=np.asarray(feature_vectors)
    class_vectors=np.asarray(class_vectors)
    #vectors and classes are returned for classification
    return class_vectors, feature_vectors

y, x = calculate_centroids_histogram(a)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
model = MLPClassifier(hidden_layer_sizes=(40,40,40), max_iter=40000).fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print("Training: ", model.score(x_train, y_train))
print("Testing: ", model.score(x_test, y_test))

dump(model, './now_output/sift_mlp.joblib')
```

Figure 25: Training the model using MLP

The trained model is used to classify the testing image which is taken as a snapshot using the webcam using a python script.

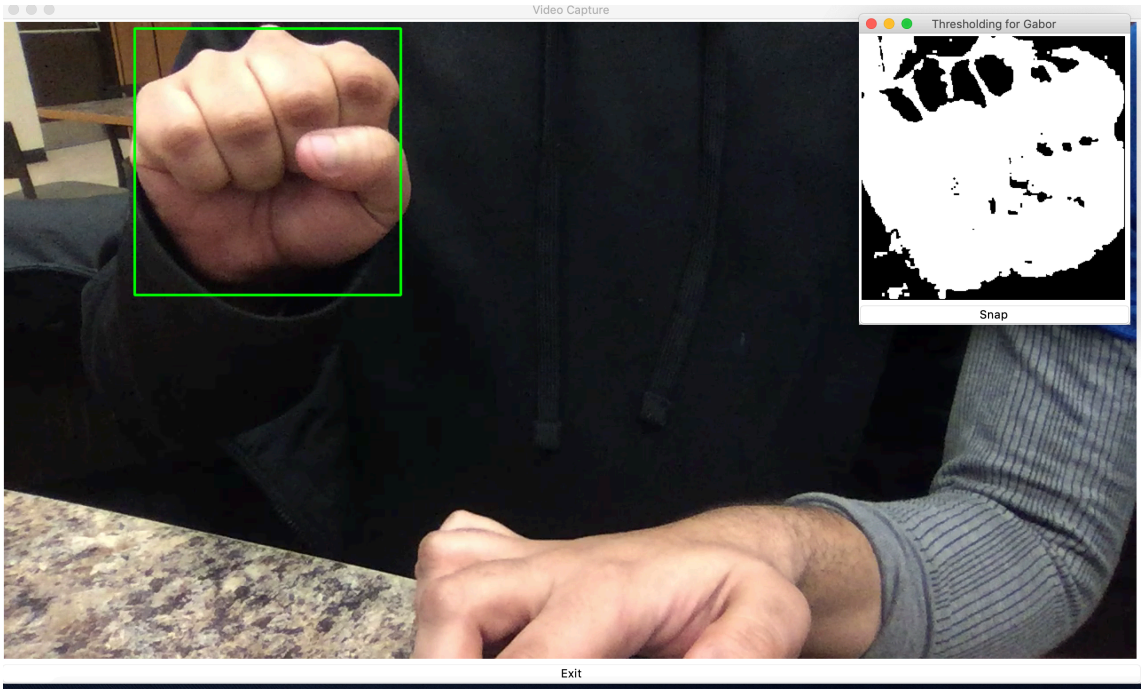

Figure 26: Taking snapshot

Figure 27: Classification result

7. Results

7.1. Using Gabor features

Initially the experimentation was done using 10 samples from each class and both SVM and MLP was done. The following results was obtained

Number of Samples per class	SVM		MLP	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
10	100%	0.2%	100%	0.1%

Table 1: Accuracy comparison between MLP and SVM on 10 samples per class for Gabor features

It was found that the model was overfitted and the number of samples was again increased to 25 per class. The following results was obtained

Number of Samples per class	SVM		MLP	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
25	100%	0.3%	100%	0.23%

Table 2: Accuracy comparison between MLP and SVM on 25 samples per class for Gabor features

Still it was found that the model was overfitted and the number of samples was again increased to 100 per class. The following results was obtained:

Number of Samples per class	SVM		MLP	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
100	100%	0.42%	100%	0.41%

Table 3: Accuracy comparison between MLP and SVM on 100 samples per class for Gabor features

Still it was found that the model was overfitted and the number of samples was again increased to 1000 per class. The following results was obtained:

Number of Samples per class	SVM		MLP	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
1000	99%	4%	98%	3.7%

Table 4: Accuracy comparison between MLP and SVM on 1000 samples per class for Gabor features

Still it was found that the model was overfitted and the number of samples was again increased to 3000 per class. The following results was obtained:

Number of Samples per class	SVM		MLP	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
3000	86%	48%	87%	51%

Table 5: Accuracy comparison between MLP and SVM on 3000 samples per class for Gabor features

7.2. Using SIFT features

Initially the experimentation was done using 10 samples from each class and both SVM and MLP was done. The following results was obtained

Number of Samples per class	SVM		MLP	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
10	100%	0.3%	100%	0.18%

Table 6: Accuracy comparison between MLP and SVM on 10 samples per class for SIFT features

It was found that the model was overfitted and the number of samples was again increased to 25 per class. The following results was obtained

Number of Samples per class	SVM		MLP	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
25	100%	0.6%	100%	0.4%

Table 7: Accuracy comparison between MLP and SVM on 25 samples per class for SIFT features

Still it was found that the model was overfitted and the number of samples was again increased to 100 per class. The following results was obtained:

Number of Samples per class	SVM		MLP	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
100	100%	0.57%	100%	0.%

Table 8: Accuracy comparison between MLP and SVM on 100 samples per class for SIFT features

Still it was found that the model was overfitted and the number of samples was again increased to 1000 per class. The following results was obtained:

Number of Samples per class	SVM		MLP	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
1000	97%	5%	98%	4%

Table 9: Accuracy comparison between MLP and SVM on 1000 samples per class for SIFT features

Still it was found that the model was overfitted and the number of samples was again increased to 3000 per class. The following results was obtained:

Number of Samples per class	SVM		MLP	
	Training Accuracy	Testing Accuracy	Training Accuracy	Testing Accuracy
3000	84%	42%	88%	39%

Table 10: : Accuracy comparison between MLP and SVM on 3000 samples per class for SIFT features

8. Comparison

After the model was created for both SIFT and Gabor features using both MLP and SVM the following results were obtained and the following comparison was done:

8.1. Gabor Features

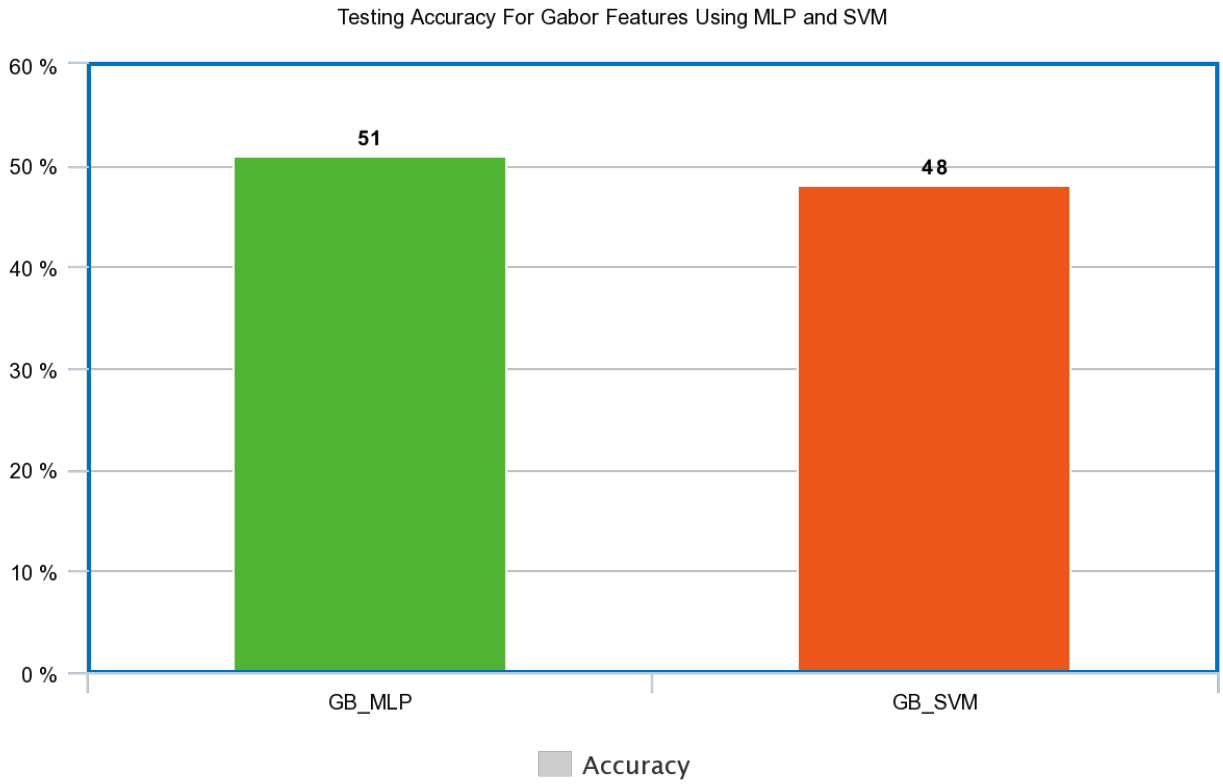

Figure 28: Accuracy comparison of Gabor Features on MLP and SVM

It was found that for classification using gabor features, MLP was found to be better than SVM for classification. The MLP gave higher accuracy (51%) compared to SVM (48%).

8.2. SIFT Features

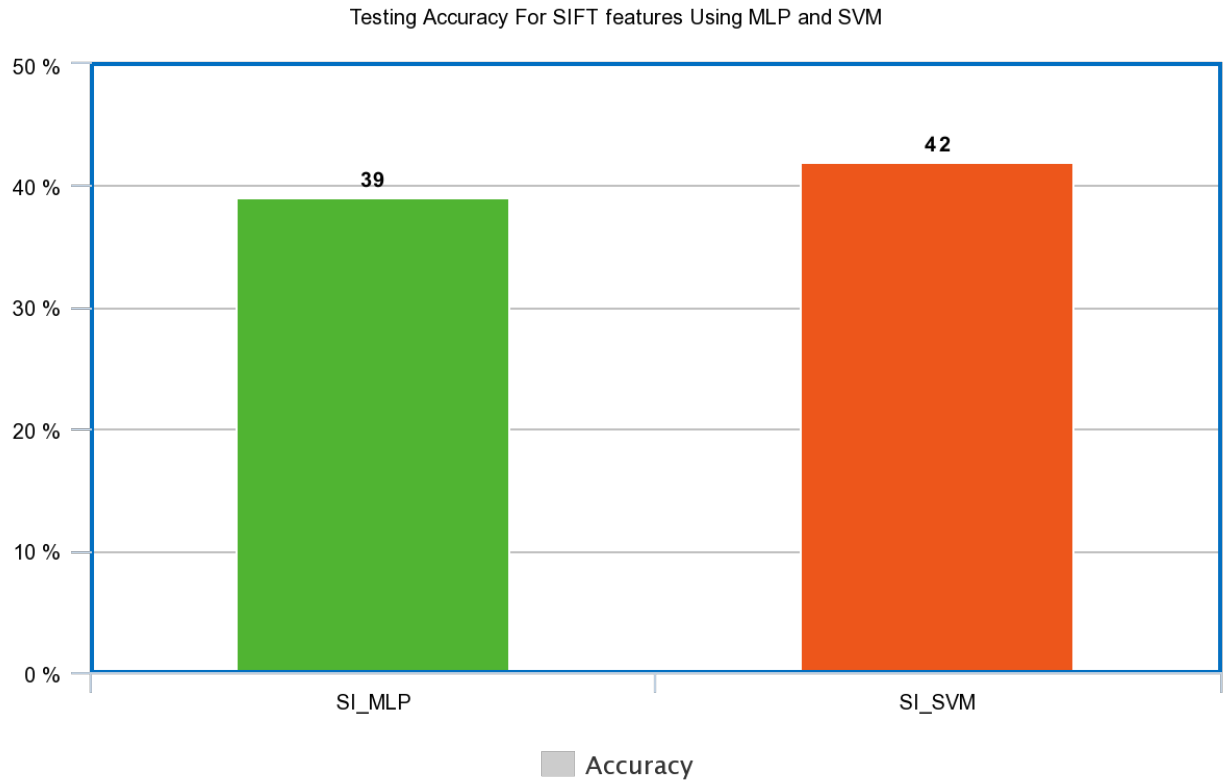

Figure 29: Accuracy comparison of Sift features using SVM and MLP

It was found that for classification using SIFT features, SVM was found to be better than MLP for classification. The SVM gave higher accuracy (42%) compared to SVM (39%)

8.3. SVM on Gabor Features and SIFT Features

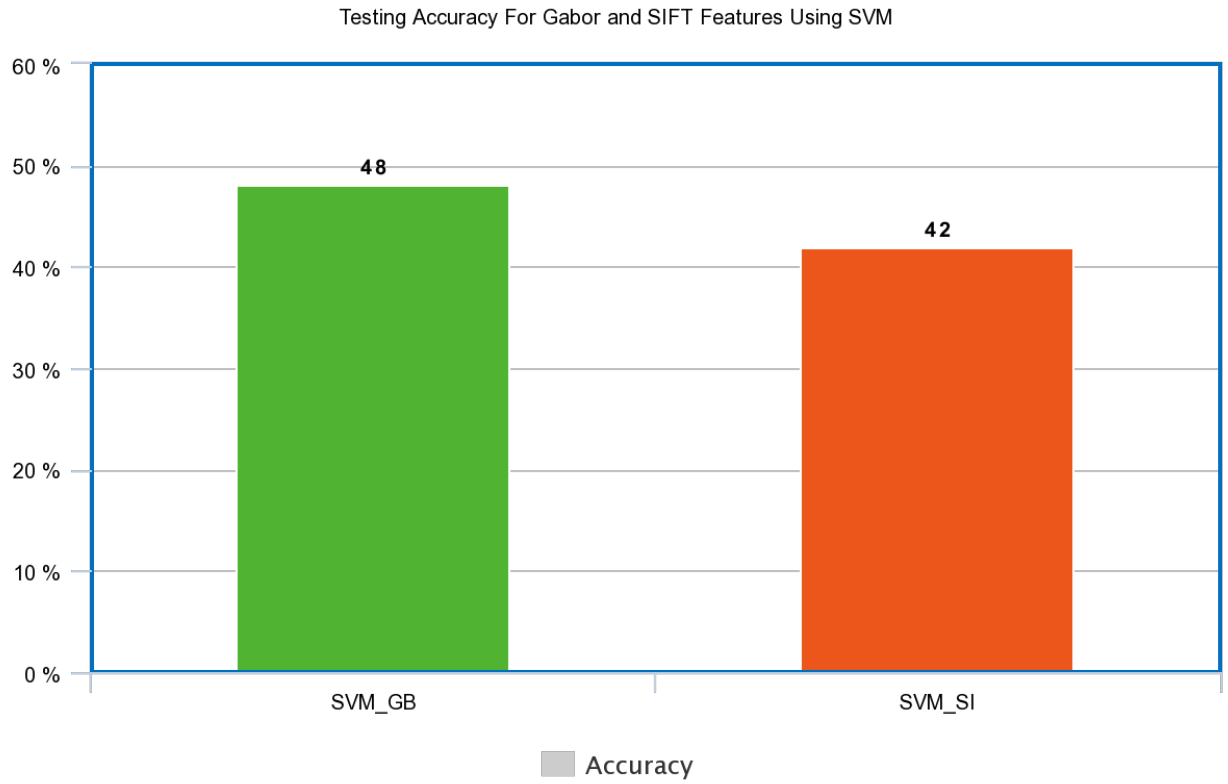

Figure 30: Accuracy comparison of SVM on gabor and sift features

It was found that SVM gave better accuracy for gabor features (48%) than the SIFT features (42%)

8.4. MLP on Gabor Features and SIFT Features

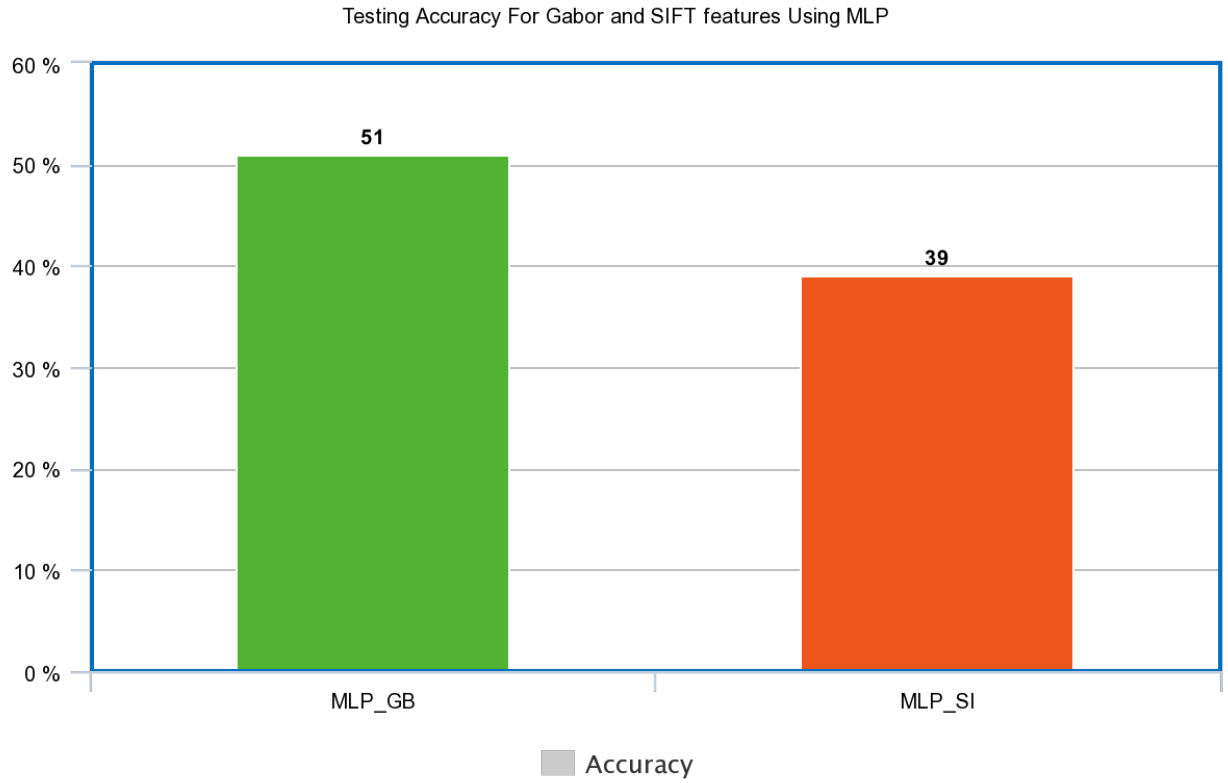

Figure 31: Accuracy comparison of MLP on Gabor and Sift features

It was found that MLP gave better accuracy for gabor features (51%) than the SIFT features (39%)

8.5. Overall

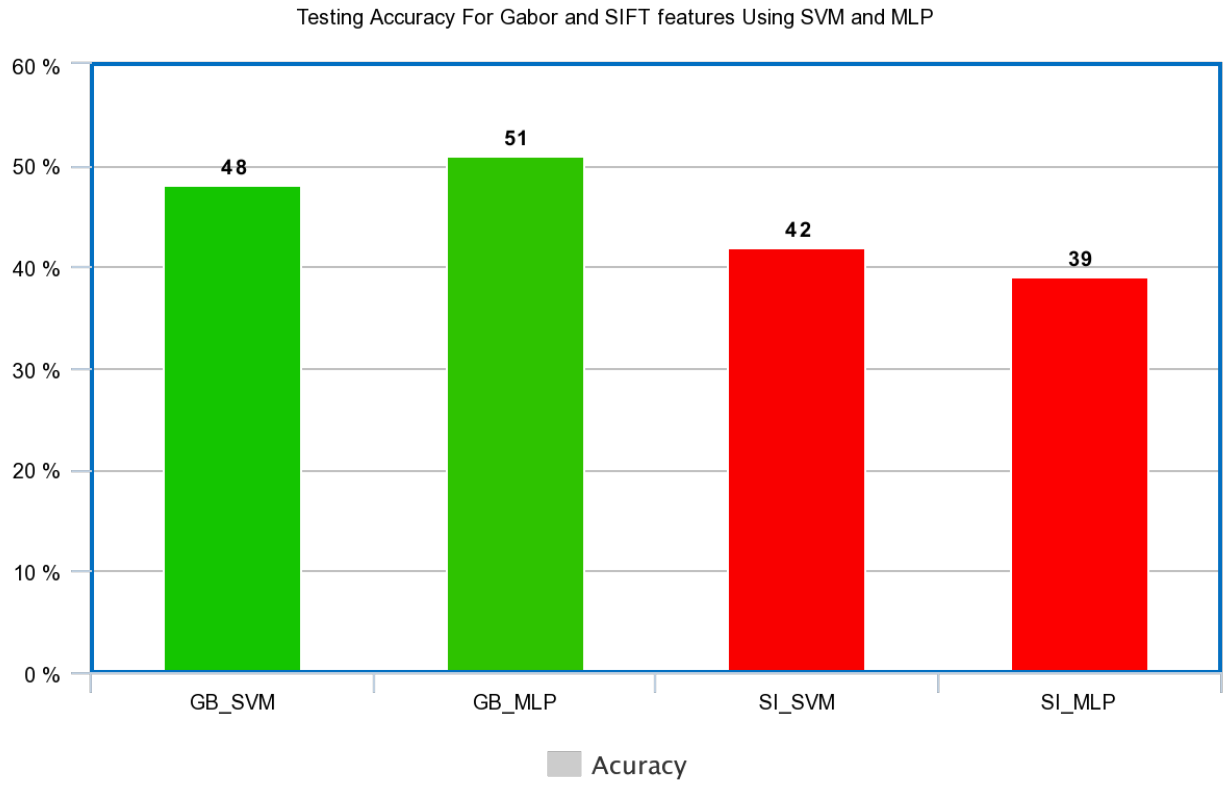

Figure 32: Overall Comparison

It was found that classification using Gabor features using both MLP and SVM gave better results (51% and 48% respectively) than classification using SIFT features using both MLP and SVM (39% and 42% respectively). Overall classification using Gabor Features with SVM gave the better accuracy (51%) than the rest.

9. Discussion

The training was done on 26 classes with 3000 images of each respective classes which resulted in total of 78000 images. Two different types of feature extraction method were used which was gabor filter and SIFT. The gabor filter resulted in 40 features from each image which further resulted in 120000 features per class. Whereas the SIFT resulted in 50 features from each image which further resulted in 150000 features. But the gabor feature having a big kernel and also it did convolution with a small window size throughout the image, took significantly longer time than the SIFT. The gabor feature extraction for all the 78000 images took 74 hours in total whereas the SIFT feature extraction took only about 15 minutes.

Total Number Of Images	Time Taken for SIFT	Time Taken for GABOR
78000	15 minutes	74 hours

Table 11: Time taken for gabor vs sift feature extraction

Lack of good computational power and good GPU made us difficult to train the whole images.

SVM is a robust model however when there are not enough data points it is easy for it to overfit. Same is the case with Multi-Layer Perceptron. This is why they are giving so high accuracy in the training model and such a low accuracy in the testing model. The time taken to train the whole model for SVM is higher than the time taken for training MLP. Ideally SVM should outperform MLP but this is not the case in our models. This might because we used 40 hidden layers in our MLP model. These models are also not optimized to the dataset, which means the models could be trained and the best values for hidden layers could be found out. In case of SVM we are using SVM without a kernel, using a kernel may give better result.

11. Conclusion

We implemented SIFT and Gabor Filter for the feature extraction. We implemented 2 classifiers to train our dataset i.e. SVM and MLP. Gabor filter with MLP gave us a better but it took more time. SIFT with SVM gave us a lower accuracy but it took less time. We also contacted the authors of the papers for their dataset for the comparison of accuracy with our dataset but did not receive any response. We could have achieved a better result if we had received their dataset. Also, due to lack of good computational power and GPU for faster training and better result we achieved lower accuracy compared to the paper.

12. Future Work

For the future work we plan to implement the following things:

- Use better optimization for MLP
- Implement CNN
- Real time gesture recognition for all classes

13. References

[1] N. P. Nagori and V. Malode, "Communication Interface for Deaf-Mute People using Microsoft Kinect," in *2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)*, Pune, 2016.

[2] S. Shivashankara and S. Srinath, "A review on vision based American sign language recognition, its techniques, and outcomes," in *2017 7th International Conference on Communication Systems and Network Technologies (CSNT)*, Nagpur, 2017.

[3] F. W. S. a. A. Z. M. C. Ariesta, "Sentence Level Indonesian Sign Language Recognition Using 3D Convolutional Neural Network and Bidirectional Recurrent Neural Network," in *2018 Indonesian Association for Pattern Recognition International Conference (INAPR)*, Jakarta, Indonesia, 2018.

[4] J. E. a. M. Joshi, "Indian sign language recognition using ANN and SVM classifiers," in *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, Coimbatore, 2017.

[5] K. Bantupalli and Y. Xie, "American Sign Language Recognition using Deep," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018.

[6] J. a. R. Pansare and M. Ingle, "Vision-Based Approach for American Sign Language Recognition Using Edge," in *2016 International Conference on Image, Vision and Computing*, 2016.

[7] Akash, "kaggle," 2017. [Online]. Available: https://www.kaggle.com/grassknoted/asl-alphabet.

[8] S. S. Shanta, S. T. Anwar and M. R. Kabir, "Bangla Sign Language Detection Using SIFT and CNN," in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Bangalore, 2018.

[9] M. A. Uddin and S. A. Chowdhury, "Hand sign language recognition for Bangla alphabet using Support Vector Machine," in *2016 International Conference on Innovations in Science, Engineering and Technology (ICISET)*, Dhaka, 2016.

[10] Z. Q. and K. S., "A novel color space based on RGB color barycenter," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP*, 2016.

[11] S. G., Y. G. and N. S., "Real time implementation of RGB to HSV/HSI/HSL and its reverse color space models," in *2016 International Conference on Communication and Signal Processing (ICCSP)*, Melmaruvathur, 2016.

[12] J. Kamarainen, "Gabor features in image analysis," in *012 3rd International Conference on Image Processing Theory, Tools and Applications (IPTA)*, Istanbul, 2012.

[13] M. E. Mavroforakis and S. Theodoridis, "Support Vector Machine (SVM) classification through geometry," in *2005 13th European Signal Processing Conference*, Antalya, 2005.

[14] G. Singh and M. Sachan, "Multi-layer perceptron (MLP) neural network technique for offline handwritten Gurmukhi character recognition," in *2014 IEEE International Conference on Computational Intelligence and Computing Research*, Coimbatore, 2014.

[15] H. R. K. a. V. K. Thakar, "Scale Invariant Feature Transform Based Image Matching and Registration," in *2014 Fifth International Conference on Signal and Image Processing*, Bangalore, 2014.

14. Appendix

UI DESIGN

```
# This is the python file which when run will generate the UI.
import tkinter as tk
import numpy as np
import PIL.Image, PIL.ImageTk
import cv2
from tkinter import messagebox
from sklearn.externals import joblib
from skimage.filters import gabor
from math import pi
import os

class main_controller_class:
   # Make the basic UI
   def generateUI(self):
      root = tk.Tk()

      # Creating Frames to hold similar type objects
      label_frame = tk.Frame(root)
      label_frame.pack()

      radioButton1_frame = tk.Frame(root)
      radioButton1_frame.pack(fill = tk.X)

      button1_frame = tk.Frame(root)
      button1_frame.pack(fill = tk.X)

      radioButton2_frame = tk.Frame(root)
      radioButton2_frame.pack(fill = tk.X)


      bottom_frame = tk.Frame(root)
      bottom_frame.pack(side = tk.BOTTOM,fill = tk.X)

      button2_frame = tk.Frame(root)
      button2_frame.pack(fill = tk.X)

      label_warning2 = tk.Label(label_frame, text="We have extracted features ourselves and
provided them. The models are also trained by us")
      label_warning2.pack()
      label_warning1 = tk.Label(label_frame,fg="Red", text="NOTE: SIFT cannnot be run on the
latest version of openCV. We are using OpenCV '3.4.2'")
      label_warning1.pack()
```

```
        # 4 Radio buttons for each type of Feature extraction process
        self.v = tk.IntVar()
        radiobutton1 = tk.Radiobutton(radioButton1_frame, text='Gabor Features', variable=self.v,
value=0)
        radiobutton1.pack(side = tk.LEFT)
        tk.Radiobutton(radioButton1_frame, text='SIFT Features', variable=self.v,
value=1).pack(side = tk.LEFT)
        tk.Radiobutton(radioButton1_frame, text='Our Gabor Features', variable=self.v,
value=2).pack(side = tk.LEFT)
        tk.Radiobutton(radioButton1_frame, text='Our SIFT Features', variable=self.v,
value=3).pack(side = tk.LEFT)

        # To start extracting features from images, a button
        f_extract_button = tk.Button(button1_frame, text = 'Extract Features',command= lambda:
self.extractFeatures_button_event())
        f_extract_button.pack(fill = tk.X)

        # 4 Radio buttons for each type of classifier process
        self.var = tk.IntVar()
        tk.Radiobutton(radioButton2_frame, text='SVM', variable=self.var, value=0).pack(side =
tk.LEFT)
        tk.Radiobutton(radioButton2_frame, text='MLP', variable=self.var, value=1).pack(side =
tk.LEFT)
        tk.Radiobutton(radioButton2_frame, text='Our SVM', variable=self.var, value=2).pack(side
= tk.LEFT)
        tk.Radiobutton(radioButton2_frame, text='Our MLP', variable=self.var, value=3).pack(side
= tk.LEFT)

        # To start classification
        classify_button = tk.Button(button2_frame, text = 'Classify',command= lambda:
self.classify_button_event())
        classify_button.pack(fill = tk.X)

        # Button to play video in a tkinter frame
        video_button = tk.Button(button2_frame, text = 'Start Video',command= lambda:
self.start_video(root))
        video_button.pack(fill = tk.X)

        #Button to classification of the snapshot clicked
        test_image_button = tk.Button(button2_frame, text = 'Classify the snap',command= lambda:
self.classify_snap())
        test_image_button.pack(fill = tk.X)

        # Exit button
```

```
        tk.Button(bottom_frame, text ='Exit',command=root.destroy).pack(side = tk.BOTTOM,fill
= tk.X)
        root.mainloop()

    # Function called when extract features button is clicked
    def extractFeatures_button_event(self):
        # Get the value of the first group of radio buttons
        selection = self.v.get()
        if selection == 0:
            print ("Gabor Features")
            if messagebox.askyesno("Warning","This will take a lot of time. Do you want to
continue?"):
                #Call the python file
                import preprocess

        elif selection == 1:
            if messagebox.askyesno("Warning","This will take a lot of time. Do you want to
continue?"):
                #Call the python file
                print ("SIFT")
                import pregray
        elif selection == 2:
            print ("Our Gabor Features")
            # Use features created by us. Do nothing

        elif selection == 3:
            print ("Our SIFT")
            # Use features created by us. Do nothing



    # Function called when classify button is clicked
    def classify_button_event(self):
        selection = self.var.get()
        select = self.v.get()
        if selection == 0:
            if messagebox.askyesno("Warning","This will take a lot of time. Do you want to
continue?"):
                if select == 0:

                    print("SVM + Gabor")
                    #Call the python file
                    import allSvmGabor
                    # Give the proper path
                    allSvmGabor.svm(sorted(os.listdir('./gabor_features_now')))
```

```
            elif select == 1:
                print("SVM + SIFT")
                #Call the python file
                import sipiup

            elif select == 2:
                print("SVM + Our Gabor")
                #Call the python file
                import allSvmGabor
                  # Give the proper path
                allSvmGabor.svm(sorted(os.listdir('./gabor_features')))
            elif select == 3:
                print("SVM + Our SIFT")
                #Call the python file
                import sipiup

        elif selection == 1:
            if messagebox.askyesno("Warning","This will take a lot of time. Do you want to
continue?"):
            if select == 0:
                print("MLP + Gabor")
                #Call the python file
                import allmlpGabor
                  # Give the proper path
                allmlpGabor.mlp(sorted(os.listdir('./gabor_features_now')))
            elif select == 1:
                print("MLP + SIFT")
                #Call the python file
                import mlpsift

            elif select == 2:
                print("MLP + Our Gabor")
                #Call the python file
                import allmlpGabor
                  # Give the proper path
                allmlpGabor.mlp(sorted(os.listdir('./gabor_features')))
            elif select == 3:
                print("MLP + Our SIFT")
                #Call the python file
                import mlpsift

        elif selection == 2:
            print("Our SVM")
            # Use features created by us. Do nothing

        elif selection == 3:
```

```
        print("Our MLP")
        # Use features created by us. Do nothing


# Button Event which corresponds to starting the video camera
def start_video(self,root):
    # The first window has frames related to Gabor filter (the thresholded values)
    self.window_threshold = tk.Toplevel()
    self.window_threshold.title("Thresholding for Gabor")
    self.canvas_2 = tk.Canvas(self.window_threshold, width = 300, height = 300)
    self.canvas_2.pack()
    self.snap_button=tk.Button(self.window_threshold, text="Snap", command=self.get_snap)
    self.snap_button.pack(anchor=tk.CENTER, expand=True,fill = tk.X)

    # The second window is just a regular Video Capture except its OpenCV played in a Tkinter
    self.window = tk.Toplevel()
    self.window.title("Video Capture")
    self.vid = video_capture_opencv_tkinter()
    self.canvas = tk.Canvas(self.window, width = self.vid.width, height = self.vid.height)
    self.canvas.pack()
    self.exit_button=tk.Button(self.window, text="Exit", command=self.destroy_everything)
    self.exit_button.pack(anchor=tk.CENTER, expand=True,fill = tk.X)

    # Add a delay so the processing does not take too much power
    self.delay = 15
    self.update()
    self.window_threshold.protocol("WM_DELETE_WINDOW", self.on_closing)
    self.window.protocol("WM_DELETE_WINDOW", self.on_closing)
    self.window.mainloop()

# Update, updates the Tkinter window every 15 milliseconds with the latest value
def update(self):
    ret, frame,_,frame_2,__ = self.vid.get_frame()
    if ret:
        self.photo = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(frame))
        self.canvas.create_image(0, 0, image = self.photo, anchor = tk.NW)

        self.photo_2 = PIL.ImageTk.PhotoImage(image = PIL.Image.fromarray(frame_2))
        self.canvas_2.create_image(0, 0, image = self.photo_2, anchor = tk.NW)

    self.window.after(self.delay, self.update)

# Saving the snapshot for classification
def get_snap(self):
    ret, frame,_,frame_2,roi = self.vid.get_frame()
    cv2.imwrite("./camera_image_output/test_image_gabor.png", frame_2)
```

```
        cv2.imwrite("./camera_image_output/test_image_sift.png", roi)

    # Destroy the object and all windows to avoid errors when navigating
    def destroy_everything(self):
        if messagebox.askyesno("Warning","Are you Sure you want to Exit?"):
            self.window.destroy()
            self.window_threshold.destroy()
            del self.vid

    # Similar to destroy_everything function but is to handle when user press the x button instead
of exit
    def on_closing(self):
        if messagebox.askyesno("Warning","Are you Sure you want to Exit?"):
            self.window.destroy()
            self.window_threshold.destroy()
            del self.vid

    # Get gabor features for the snap image
    def get_gabor_feature(self,image):
        classify = np.array([])
        # For 5 scales
        for i in range(0, 5, 1):
            # 8 orientations
            for j in range(0, 8, 1):
                # Get the real values from gabor function
                real_val = gabor(image, frequency=(i + 1) / 10.0, theta=j * pi / 8)[0]
                # Get the imaginary values from gabor function
                img_val = gabor(image, frequency=(i + 1) / 10.0, theta=j * pi / 8)[1]
                # Get the square of both values and add them up to get a complete result
                result = real_val * real_val + img_val * img_val
                res_mean = np.mean(result)
                classify = np.append(classify, res_mean)
        return classify

    # Classify snap image when the button is pressed
    def classify_snap(self):
        selection = self.var.get()
        select = self.v.get()
        if selection == 0:
            if select == 0 or select == 2:
                self.model = joblib.load("./now_output/finalsvmgabor.joblib")

                #Some pre processing
                img = cv2.imread("./camera_image_output/test_image_gabor.png")
                img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                gb = self.get_gabor_feature(img)
```

```
            gb_reshape = gb.reshape((1,40))
            print(self.model.predict(gb_reshape))

        elif select == 1 or select == 3:
            model = joblib.load("kmeans.joblib")
            #read image
            image = cv2.imread("./camera_image_output/test_image_sift.png",1)
            image = cv2.resize(image,(200,200))
            #Convert them to grayscale
            image =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
            #SIFT extraction
            sift = cv2.xfeatures2d.SIFT_create()
            kp, descriptors = sift.detectAndCompute(image,None)
            #classification of all descriptors in the model
            predict_kmeans=model.predict(descriptors)
            #calculates the histogram
            hist, bin_edges=np.histogram(predict_kmeans, bins=260)
            #histogram is the feature vector
            self.model = joblib.load("./now_output/sift_svm.joblib")

            hist = hist.reshape(1, -1)
            print(self.model.predict(hist))

    elif selection == 1:

        if select == 0 or select == 2:
            self.model = joblib.load("./now_output/finalmlpgabor.joblib")
            #Some pre processing
            img = cv2.imread("./camera_image_output/test_image_gabor.png")
            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            gb = self.get_gabor_feature(img)
            gb_reshape = gb.reshape((1,40))
            print(self.model.predict(gb_reshape))
        elif select == 1 or select == 3:
            model = joblib.load("kmeans.joblib")
            #read image
            image = cv2.imread("./camera_image_output/test_image_sift.png",1)
            image = cv2.resize(image,(200,200))
            #Convert them to grayscale
            image =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
            #SIFT extraction
            sift = cv2.xfeatures2d.SIFT_create()
            kp, descriptors = sift.detectAndCompute(image,None)
            #classification of all descriptors in the model
            predict_kmeans=model.predict(descriptors)
            #calculates the histogram
```

```
        hist, bin_edges=np.histogram(predict_kmeans, bins=260)
        #histogram is the feature vector
        self.model = joblib.load("./now_output/sift_mlp.joblib")

        hist = hist.reshape(1, -1)
        print(self.model.predict(hist))

elif selection == 2:
    if select == 0 or select == 2:
        self.model = joblib.load("finalsvmgabor.joblib")
        #Some pre processing
        img = cv2.imread("./camera_image_output/test_image_gabor.png")
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        gb = self.get_gabor_feature(img)
        gb_reshape = gb.reshape((1,40))
        print(self.model.predict(gb_reshape))

    elif select == 1 or select == 3:
        model = joblib.load("kmeans.joblib")
        #read image
        image = cv2.imread("./camera_image_output/test_image_sift.png",1)
        image = cv2.resize(image,(200,200))
        #Convert them to grayscale
        image =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
        #SIFT extraction
        sift = cv2.xfeatures2d.SIFT_create()
        kp, descriptors = sift.detectAndCompute(image,None)
        #classification of all descriptors in the model
        predict_kmeans=model.predict(descriptors)
        #calculates the histogram
        hist, bin_edges=np.histogram(predict_kmeans, bins=260)
        #histogram is the feature vector
        self.model = joblib.load("finalsvmsift.joblib")

        hist = hist.reshape(1, -1)
        print(self.model.predict(hist))



elif selection == 3:
    if select == 0 or select == 2:
        self.model = joblib.load("gabormlp1.joblib")
        #Some pre processing
        img = cv2.imread("./camera_image_output/test_image_gabor.png")
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        gb = self.get_gabor_feature(img)
```

```
            gb_reshape = gb.reshape((1,40))
            print(self.model.predict(gb_reshape))

        elif select == 1 or select == 3:
            model = joblib.load("kmeans.joblib")
            #read image
            image = cv2.imread("./camera_image_output/test_image_sift.png",1)
            image = cv2.resize(image,(200,200))
            #Convert them to grayscale
            image =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
            #SIFT extraction
            sift = cv2.xfeatures2d.SIFT_create()
            kp, descriptors = sift.detectAndCompute(image,None)
            #classification of all descriptors in the model
            predict_kmeans=model.predict(descriptors)
            #calculates the histogram
            hist, bin_edges=np.histogram(predict_kmeans, bins=260)
            #histogram is the feature vector
            self.model = joblib.load("sipiup.joblib")

            hist = hist.reshape(1, -1)
            print(self.model.predict(hist))


# This class has functions to help with opening an Opencv videocapture in a Tkinter window
class video_capture_opencv_tkinter:
    def __init__(self):
        # The Opencv function to start the webcam
        self.vid = cv2.VideoCapture(0)
        if not self.vid.isOpened():
            raise ValueError("Unable to start the video capture")
        # Get the height and width of the frame
        self.width = self.vid.get(cv2.CAP_PROP_FRAME_WIDTH)
        self.height = self.vid.get(cv2.CAP_PROP_FRAME_HEIGHT)
        # This is the ROI (Region of Intrest) which is shown on video capture
        self.top, self.right, self.bottom, self.left = 10, 150, 310, 450

    # Function to get each frame from the video
    def get_frame(self):
        if self.vid.isOpened():
            # Read the frame from the video and store it so now we deal with an image
            ret, frame = self.vid.read()

            frame = cv2.flip(frame, 1)
            # Two bounding boxes for the same region of intrest
            roi = frame[self.top:self.bottom, self.right:self.left]
```

```
            roi2 = frame[self.top:self.bottom, self.right:self.left]
            # Make a rectagle for the region of intrest ROI
            cv2.rectangle(frame, (self.left, self.top), (self.right, self.bottom), (0,255,0), 2)

            # Convert the image into HSV (RGB to HSV conversion)
            img_HSV = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
            # Adding a medianBlur filter to reduce the noise and smoothen the image
            img_HSV = cv2.medianBlur(img_HSV,3)
            # Set the values for skin color for HSV
            HSV_mask = cv2.inRange(img_HSV, (0, 15, 0), (17,170,255))
            # Use morphological operation (opening; erosion then dialation)
            HSV_mask = cv2.morphologyEx(HSV_mask, cv2.MORPH_OPEN, np.ones((3,3),
np.uint8))
            roi = cv2.bitwise_not(HSV_mask)
            # Convert the image into binary image
            et,thresh1 =
cv2.threshold(roi,127,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

            if ret:
                # Return two frames for two windows (Note: Third value is also returned however its
just a Grayscle value which is no longer used)
                return (ret, cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB),et,thresh1,cv2.cvtColor(roi2, cv2.COLOR_BGR2GRAY))
            else:
                return (ret, None,None,None)
        else:
            return (ret, None,None,None)
    # Rekease the video when this object is deleted
    def __del__(self):
        if self.vid.isOpened():
            self.vid.release()

if __name__ == "__main__":
    app = main_controller_class()
    app.generateUI()
```

MLP for gabor

```
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from joblib import dump, load
import os, re

# Function with path given from UI_design class
def mlp(path):
    i=0
    feature_array=[]
    labels_array=[]
    for f in path:
        # Get all gabor feature files
      if re.match('g', f):
          features = np.load('./gabor_features/'+f).tolist()
          feature_array += features
          i+=1

    feature_array = np.array(feature_array)

    for f in path:
        # Get all the labels
      if re.match('l', f):
          labels = np.load('./gabor_features/'+f).tolist()
          labels_array += labels
          i+=1

    data = np.array(feature_array)
    labels=np.array(labels_array)
    # Train the model usig MLP
    model = MLPClassifier(hidden_layer_sizes=(40,40,40), max_iter=40000).fit(data, labels)


    #Computing the MLP for the complete dataset and saving the model
    dump(model, './now_output/finalmlpgabor.joblib')
```

SVM GABOR

```
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from joblib import dump, load
import re

# Function with path given from UI_design class
def svm(path):
    i=0
    feature_array=[]
    labels_array=[]
    for f in path:
          # Get all gabor feature files
      if re.match('g', f):
          features = np.load('./gabor_features/'+f).tolist()
          feature_array += features
          i+=1

    feature_array = np.array(feature_array)

    for f in path:
       # Get all the labels
      if re.match('l', f):
          labels = np.load('./gabor_features/'+f).tolist()
          labels_array += labels
          i+=1

    data = np.array(feature_array)
    labels=np.array(labels_array)
    # Train the model usig MLP
    model = SVC(gamma='auto').fit(data, labels)


    #Computing the SVM for the complete dataset and saving the model
    dump(model, './now_output/finalsvmgabor.joblib')
```

Gabor Extraction

```
import numpy as np
import cv2
from skimage.filters import gabor
from math import pi
import os

# Function to extract Gabor features
def get_gabor_feature(image, name):
    classify = np.array([])
    label = np.array([])
    # For 5 scales
    for i in range(0, 5, 1):
        # 8 orientations
        for j in range(0, 8, 1):
            # Get the real values from gabor function
            real_val = gabor(image, frequency=(i + 1) / 10.0, theta=j * pi / 8)[0]
            # Get the imaginary values from gabor function
            img_val = gabor(image, frequency=(i + 1) / 10.0, theta=j * pi / 8)[1]
            # Get the square of both values and add them up to get a complete result
            result = real_val * real_val + img_val * img_val
            res_mean = np.mean(result)
            classify = np.append(classify, res_mean)
            label = np.append(label, name)
    print('Gabor Features:')
    print(classify)
    print('Length Gabor Features:')
    print(len(classify))
    allVectors.append(classify)
    allLabels.append(name)

# Get values from this path
dirpath = './randomTrain/'
subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if os.path.isdir(os.path.join(dirpath,
o))]
i = 0

j = 0

# To get all images from the path
while (i < len(subdirs)):
    allVectors = []
    allLabels = []
    print(i)
    res = subdirs[i][subdirs[i].rindex('/') + 1:]
```

```
oldDir = dirpath + res
filenames = os.listdir(oldDir)
for fname in filenames:
    if (fname != '.DS_Store'):
        print(j)
        srcpath = os.path.join(oldDir, fname)
        print(srcpath)
        # read each image and apply gabor extraction function
        img = cv2.imread(srcpath, 0)
        get_gabor_feature(img, res)
        j += 1

    # Save the values and labels
    np.save('gabor_features_now/gabor'+res+'.npy', np.asarray(allVectors))
    np.save('gabor_features_now/label'+res+'.npy', np.asarray(allLabels))

i += 1
```

MLP SIFT

```
import cv2
import numpy as np
import os
import pandas as pd
import csv
from skimage import io
from sklearn.cluster import MiniBatchKMeans
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from joblib import dump
from sklearn.svm import SVC

# The cluster should be about number of classes (26) * 10
def read_and_clusterize(num_cluster=260):

    dirpath = './randomTrainGray/'
    subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if
os.path.isdir(os.path.join(dirpath, o))]
    i = 0
    j = 0

    sift_keypoints = []

    while (i < len(subdirs)):
        print(i)
        res = subdirs[i][subdirs[i].rindex('/') + 1:]
        oldDir = dirpath + res
        filenames = os.listdir(oldDir)
        for fname in filenames:
            if (fname != '.DS_Store'):
                print(j)
                srcpath = os.path.join(oldDir, fname)
                print(srcpath)
                #read image
                image = cv2.imread(srcpath,1)
                # grayscale conversion
                image =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
                # SIFT extraction
                sift = cv2.xfeatures2d.SIFT_create()
                kp, descriptors = sift.detectAndCompute(image,None)
                sift_keypoints.append(descriptors)

                j += 1
```

```
        i += 1

    sift_keypoints=np.asarray(sift_keypoints)
    sift_keypoints=np.concatenate(sift_keypoints, axis=0)
    print("Training kmeans")
    kmeans = MiniBatchKMeans(n_clusters=num_cluster, random_state=0).fit(sift_keypoints)
    return kmeans

a = read_and_clusterize()
dump(a, 'kmeans.joblib')

#Changing the class name from alphabets to number
def define_class(img_patchname):

    if(img_patchname=="A"):
        class_image=0

    if(img_patchname=="B"):
        class_image=1

    if(img_patchname=="C"):
        class_image=2

    if(img_patchname=="D"):
        class_image=3

    if(img_patchname=="E"):
        class_image=4

    if(img_patchname=="F"):
        class_image=5

    if(img_patchname=="G"):
        class_image=6

    if(img_patchname=="H"):
        class_image=7

    if(img_patchname=="I"):
        class_image=8

    if(img_patchname=="J"):
        class_image=9

    if(img_patchname=="K"):
        class_image=10
```

```
if(img_patchname=="L"):
   class_image=11

if(img_patchname=="M"):
   class_image=12

if(img_patchname=="N"):
   class_image=13

if(img_patchname=="O"):
   class_image=14

if(img_patchname=="P"):
   class_image=15

if(img_patchname=="Q"):
   class_image=16

if(img_patchname=="R"):
   class_image=17

if(img_patchname=="S"):
   class_image=18

if(img_patchname=="T"):
   class_image=19

if(img_patchname=="U"):
   class_image=20

if(img_patchname=="V"):
   class_image=21

if(img_patchname=="W"):
   class_image=22

if(img_patchname=="X"):
   class_image=23

if(img_patchname=="Y"):
   class_image=24

if(img_patchname=="Z"):
   class_image=25
```

```
    return class_image


#generating the feature vector after the kmeans clustering model
#creating the histogram of classified keypoints from kmeans
def calculate_centroids_histogram(model):

    dirpath = './randomTrainGray/'
    subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if
os.path.isdir(os.path.join(dirpath, o))]
    i = 0
    j = 0

    feature_vectors=[]
    class_vectors=[]

    while (i < len(subdirs)):
        print(i)
        res = subdirs[i][subdirs[i].rindex('/') + 1:]
        oldDir = dirpath + res
        filenames = os.listdir(oldDir)
        for fname in filenames:
            if (fname != '.DS_Store'):
                print(j)
                srcpath = os.path.join(oldDir, fname)
                print(srcpath)
                #read image
                image = cv2.imread(srcpath,1)
                #grayscale conversion
                image =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
                #SIFT extraction
                sift = cv2.xfeatures2d.SIFT_create()
                kp, descriptors = sift.detectAndCompute(image,None)
                predict_kmeans=model.predict(descriptors)
                #histogram calculation
                hist, bin_edges=np.histogram(predict_kmeans, bins=260)
                feature_vectors.append(hist)
                class_sample=define_class(res)
                class_vectors.append(class_sample)

                j += 1
        i += 1

    feature_vectors=np.asarray(feature_vectors)
    class_vectors=np.asarray(class_vectors)
    #vectors and classes are returned for classification
```

```
    return class_vectors, feature_vectors

y, x = calculate_centroids_histogram(a)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
model = MLPClassifier(hidden_layer_sizes=(40,40,40), max_iter=40000).fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print("Training: ", model.score(x_train, y_train))
print("Testing: ", model.score(x_test, y_test))

dump(model, './now_output/sift_mlp.joblib')
```

RGB to gray

```
import cv2
import numpy as np
import os

# Function to convert from RGB to Gray for sift
def pre_process(filePath,name,dest):

    img=cv2.imread(filePath)
    img_HSV = cv2.cvtColor(img, cv2.cv2.COLOR_BGR2GRAY)
    fullpath= dest+'/'+name
    cv2.imwrite(fullpath,img_HSV)


#Set source directory and destination directories
dirpath = './asl_alphabet_train'
destDirectory = './randomTrainGray/'

#Get List of sub directories
subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if
os.path.isdir(os.path.join(dirpath,o))]
i=0
while(i<len(subdirs)):
    res=subdirs[i][subdirs[i].rindex('/')+1:]
    newDest = destDirectory+res
    oldDir = dirpath+'/'+res

    os.mkdir(newDest)
    filenames = os.listdir(oldDir)
    for fname in filenames:
        srcpath = os.path.join(oldDir, fname)

        pre_process(srcpath,fname,newDest)

    i+=1
print(len(subdirs))
# Call the sift python file for further process
import siftIt
```

RGB to HSV with morphology

```
import cv2
import numpy as np
import os


# Function to do preprocessing before doing gabor feature extraction
def preprocess_for_gabor(filePath,name,dest):

    img=cv2.imread(filePath)
    img_HSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    #skin color range for hsv color space
    HSV_mask = cv2.inRange(img_HSV, (0, 15, 0), (17,170,255))
    #Morphological Operations
    HSV_mask = cv2.morphologyEx(HSV_mask, cv2.MORPH_OPEN, np.ones((3,3), np.uint8))
    #Binary conversion
    HSV_result = cv2.bitwise_not(HSV_mask)
    fullpath= dest+'/'+name
    cv2.imwrite(fullpath,HSV_result)


#Set source directory and destination directories
dirpath = './asl_alphabet_train'
destDirectory = './randomTrain/'

#Get List of sub directories
subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if
os.path.isdir(os.path.join(dirpath,o))]
i=0
while(i<len(subdirs)):
    res=subdirs[i][subdirs[i].rindex('/')+1:]
    newDest = destDirectory+res
    oldDir = dirpath+'/'+res
    os.mkdir(newDest)
    filenames = os.listdir(oldDir)
    for fname in filenames:
        srcpath = os.path.join(oldDir, fname)
        preprocess_for_gabor(srcpath,fname,newDest)
    i+=1
print(len(subdirs))
# Run the gaborextract python file next
import gaborextract
```

SIFT Features

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 28 04:24:22 2019

@author: abhigya
"""

import cv2
import numpy as np
import os
import pandas as pd
import csv
from skimage import io
from sklearn.cluster import MiniBatchKMeans
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from joblib import dump
from sklearn.svm import SVC

# The cluster should be about number of classes (26) * 10
def read_and_clusterize(num_cluster=260):

    dirpath = './randomTrainGray/'
    subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if
os.path.isdir(os.path.join(dirpath, o))]
    i = 0
    j = 0

    sift_keypoints = []

    #looping through all the images
    while (i < len(subdirs)):
        print(i)
        res = subdirs[i][subdirs[i].rindex('/') + 1:]
        oldDir = dirpath + res
        filenames = os.listdir(oldDir)
        for fname in filenames:
            if (fname != '.DS_Store'):
                print(j)
                srcpath = os.path.join(oldDir, fname)
                print(srcpath)
```

```
            #
            #read image
            image = cv2.imread(srcpath,1)
            # grayscale conversionn
            image =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
            # SIFT feature extraction
            sift = cv2.xfeatures2d.SIFT_create()
            kp, descriptors = sift.detectAndCompute(image,None)
            sift_keypoints.append(descriptors)


            j += 1
        i += 1

    sift_keypoints=np.asarray(sift_keypoints)
    sift_keypoints=np.concatenate(sift_keypoints, axis=0)
    #kmeans clustering
    print("Training kmeans")
    kmeans = MiniBatchKMeans(n_clusters=num_cluster, random_state=0).fit(sift_keypoints)
    #return the learned model
    return kmeans

a = read_and_clusterize()
dump(a, 'kmeans.joblib')

#convertinng class alphabets to numbers
def define_class(img_patchname):

    if(img_patchname=="A"):
        class_image=0

    if(img_patchname=="B"):
        class_image=1

    if(img_patchname=="C"):
        class_image=2

    if(img_patchname=="D"):
        class_image=3

    if(img_patchname=="E"):
        class_image=4

    if(img_patchname=="F"):
        class_image=5
```

```
if(img_patchname=="G"):
   class_image=6

if(img_patchname=="H"):
   class_image=7

if(img_patchname=="I"):
   class_image=8

if(img_patchname=="J"):
   class_image=9

if(img_patchname=="K"):
   class_image=10

if(img_patchname=="L"):
   class_image=11

if(img_patchname=="M"):
   class_image=12

if(img_patchname=="N"):
   class_image=13

if(img_patchname=="O"):
   class_image=14

if(img_patchname=="P"):
   class_image=15

if(img_patchname=="Q"):
   class_image=16

if(img_patchname=="R"):
   class_image=17

if(img_patchname=="S"):
   class_image=18

if(img_patchname=="T"):
   class_image=19

if(img_patchname=="U"):
   class_image=20
```

```
    if(img_patchname=="V"):
        class_image=21

    if(img_patchname=="W"):
        class_image=22

    if(img_patchname=="X"):
        class_image=23

    if(img_patchname=="Y"):
        class_image=24

    if(img_patchname=="Z"):
        class_image=25

    return class_image

#generating the feature vector after the kmeans clustering model
#creating the histogram of classified keypoints from kmeans
def calculate_centroids_histogram(model):

    dirpath = './randomTrainGray/'
    subdirs = [os.path.join(dirpath, o) for o in os.listdir(dirpath) if
os.path.isdir(os.path.join(dirpath, o))]
    i = 0
    j = 0

    feature_vectors=[]
    class_vectors=[]

    while (i < len(subdirs)):
        print(i)
        res = subdirs[i][subdirs[i].rindex('/') + 1:]
        oldDir = dirpath + res
        filenames = os.listdir(oldDir)
        for fname in filenames:
            if (fname != '.DS_Store'):
                print(j)
                srcpath = os.path.join(oldDir, fname)
                print(srcpath)
                #read image
                image = cv2.imread(srcpath,1)
                #Convert them to grayscale
                image =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
                #SIFT extraction
                sift = cv2.xfeatures2d.SIFT_create()
```

```
            kp, descriptors = sift.detectAndCompute(image,None)
            #descriptors classification
            predict_kmeans=model.predict(descriptors)
            #calculates the histogram
            hist, bin_edges=np.histogram(predict_kmeans, bins=260)
            feature_vectors.append(hist)
            #defining the class
            class_sample=define_class(res)
            class_vectors.append(class_sample)

            j += 1
        i += 1

    feature_vectors=np.asarray(feature_vectors)
    class_vectors=np.asarray(class_vectors)
    #vectors and classes are returned for classification
    return class_vectors, feature_vectors

y, x = calculate_centroids_histogram(a)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
model = SVC(gamma='auto').fit(x_train, y_train)
y_pred = model.predict(x_test)

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print("Training: ", model.score(x_train, y_train))
print("Testing: ", model.score(x_test, y_test))

dump(model, './now_output/sift_svm.joblib')
```