**Methodology:**

**Data Set:**

We created a dataset of different types of hand gestures, such as the victory sign, thumbs up sign, palm, etc. rather than using a dataset available online. This was done in order to gain a better understanding of how to manually create a dataset from scratch, as well as to work with a dataset that was not tampered in any way. Initially we made use of our phone camera and created a subset of the planned dataset to be used in the initial single layer fully-connected neural network. But as the size of the dataset started to increase we decided to automate this.

We made use of the inbuilt camera in our laptops and developed a script to automate and simplify the process of creating the dataset. This accelerated the process of creating properly labelled images and categorized those images into different names.

**Preprocessing:**

With a proper dataset in hand, our next step was to work on pre-processing and normalization of the images of the dataset. This step is crucial, as providing pre-processed and normalized images to our neural networks as inputs would enhance its performance. Different steps done in order to preprocess the dataset are mentioned below.

Uniform size and aspect ratio:

One of the basic steps involved in creating a data set is to ensure that each image in the data set has the same size and aspect ratio. As the images that we capture through our laptop cameras are of varying dimensions, e.g. $1920 \times 1080$ and $1280 \times 720$, we decided to resize the images in a uniform rectangular form of dimensions $160 \times 90$. Through this step, all the images we collected through the creation of the data set are converted into uniform images of constant size and aspect ratio.

Another advantage of resizing the image to a smaller dimension is that it reduces the input size of the images (i.e. the number of pixels), hence lowering the number of variables that is to be calculated in our CNN. If the dimensions of the input images are high, then the number of variables utilized in the CNN would be larger, and the total time taken for one iteration of the CNN would be longer as well. Thus reducing the size of the image also aids in reducing the overall training time of the CNN.



Fig: Victory sign image in dimension 1920X1080

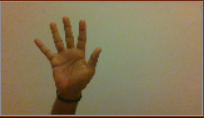


Fig: Palm sign resized to 160 X 90

Dimensionality Reduction:

The images captured by our laptops are of RGB-scale channel, and thus, to reduce the dimension of the images, we decided to convert them into a single gray-scale channel. As the RGB-scale channel has three layers, red, green and blue respectively, each image is represented as a three-dimensional matrix. This colored information individually is not particularly useful in identifying the important features (i.e. the shape of the hand) from the image. So with the help of some functions, we changed the RGB images to grayscale images without losing much information. Converting the images to grayscale helps us to simplify the inputs that are given to the CNN, as grayscale images utilize only a single channel of pixel intensities, rather than 3 different channels of pixel intensities in RGB images.



Fig: Grayscale version of victory hand sign

Image Segmentation:

As we are interested in only the hand signs, we should use only these features for training the neural network and try to mitigate all the other unnecessary information. The pre-processing done until this stage have resulted in a dataset consisting of images in grayscale. To further highlight the shape of the hand, and decrease the effect of the background of the image, we segment the image using an automatic image thresholding technique known as Otsu’s threshold.

Otsu’s threshold takes a grayscale image and returns a single intensity threshold that separates the pixels into a foreground and a background. In simpler terms this means that each pixel is checked with a threshold value and if the value of the pixel is more than threshold value, it is classified as black (with pixel value 0), and otherwise, it is classified as white (with pixel value 255). This produces an image consisting of only black and white pixels, hence forming a binary image. Thus converting the image to binary through a threshold value removes information that is not required in identifying the hand gesture depicted in the picture. However, one of the limitations of utilizing Otsu’s method for converting grayscale images to binary images is that this method only produces ideal results when image is uniformly illuminated.



Fig: Binary image of victory sign

**Results:**

In the flowchart below, we have a visual representation of what our pre-processing stage accomplishes thus far, from the initial RGB image to the final binary image. Initial images were converted to a smaller dimensions of $160 \times 90$, converted from RBG to grayscale, and finally converted to binary using Otsu’s thresholding algorithm. These final binary images are then provided as input to the CNN.



Fig: Images mentioned above shows the conversion from RGB image to binary image

Background Subtraction:

Initially we considered the technique of Background Subtraction where the algorithm extracts the image’s foreground for processing. The foreground mask is calculated by subtracting the background information from the original image. This method is usually used to detect a moving object in the videos which are captured by a stationary camera. The limitation of this technique is that it assumes that the camera does not move, and hence we are able to subtract the background, which is constant, and acquire the foreground.

But while implementing this we came to a standstill, as the algorithm performs best for objects which are moving, which would not be ideal for our dataset. Thus we have decided to move on with other algorithms, keeping this at bay.

**Plan for Completion**

Our current focus is on minimizing and simplifying the input provided to CNN, so that it would have less data to compute. We have tried several approaches for the pre-processing stage, due to its crucial role in improving the performance of the CNN. Listed below are some of the techniques that we have researched on, and currently considering for our final implementation.

Grab Cut:

As we require only the essential features of the image (i.e. the shape of the hand), we have decided to extract only the required part from the image without losing any edge information. One technique to achieve the above result is through grab cut. This is an image segmentation technique that is based on graph cuts. This technique makes use of the bounding box rectangle specified by the user and checks for a color distribution of the target image and background. We were able to successfully extract the essential part of the image without losing the edge information. The image below provides an example of the output of the grab cut algorithm.

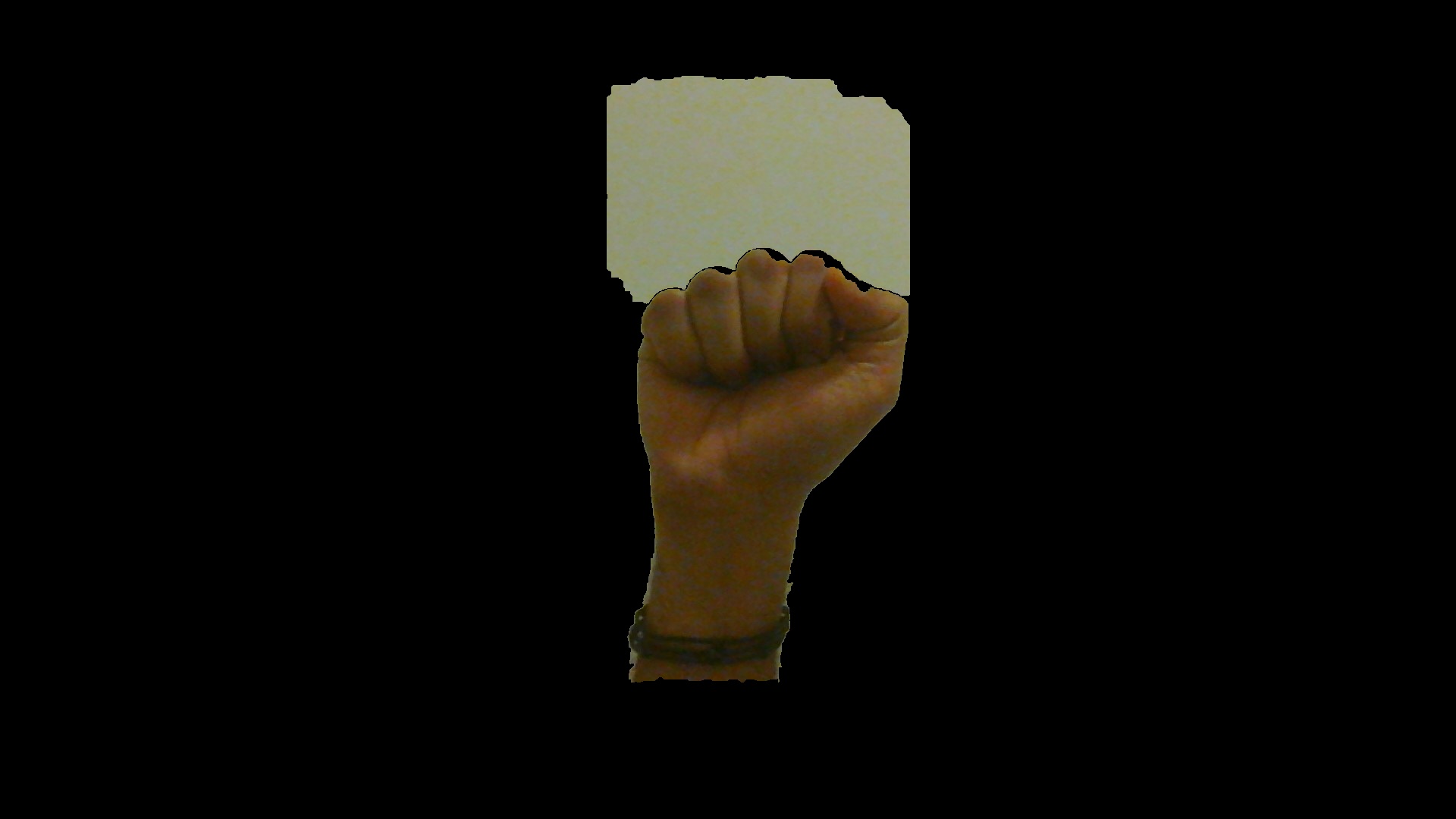


Fig: Grab cut technique on fist image

We are currently working on automating the user specified rectangle to reduce manual work, so that we can automate this process on the dataset, hence filtering the dataset to contain only the essential features and ignore the noise in other areas of background.

Region growing:

This is a simple region based image segmentation technique. It is more of a pixel based segmentation as it depends on an initial seed point. The idea of this approach is that it checks the nearby points and looks at its properties, if they are similar then that pixel is added to the region. It is more of a clustering method. We are currently looking at the implementation, and use cases that would contribute to our neural network.

Discussion,

Looking back the progress that we have achieved over the course of our project, we feel that we have covered a significant amount of ground, and gained a considerable amount of knowledge in topics relating to convolutional neural networks, i.e. creating a proper dataset, pre-processing the data, and actual implementation of the CNN. However, given the huge domain of our project, there are still some minor improvements that could be incorporated. We have discussed the additional work that could be implemented in the future, as well as ideas that did not make the cut, in the section below.

**Background Subtraction**

Initially we considered the technique of Background Subtraction where the algorithm extracts the image’s foreground for processing. The foreground mask is calculated by subtracting the background information from the original image. This method is usually used to detect a moving object in a video which is captured by a stationary camera. The limitation of this technique is that it assumes that the camera does not move, and hence we are able to subtract the background (which is constant) and acquire the foreground.

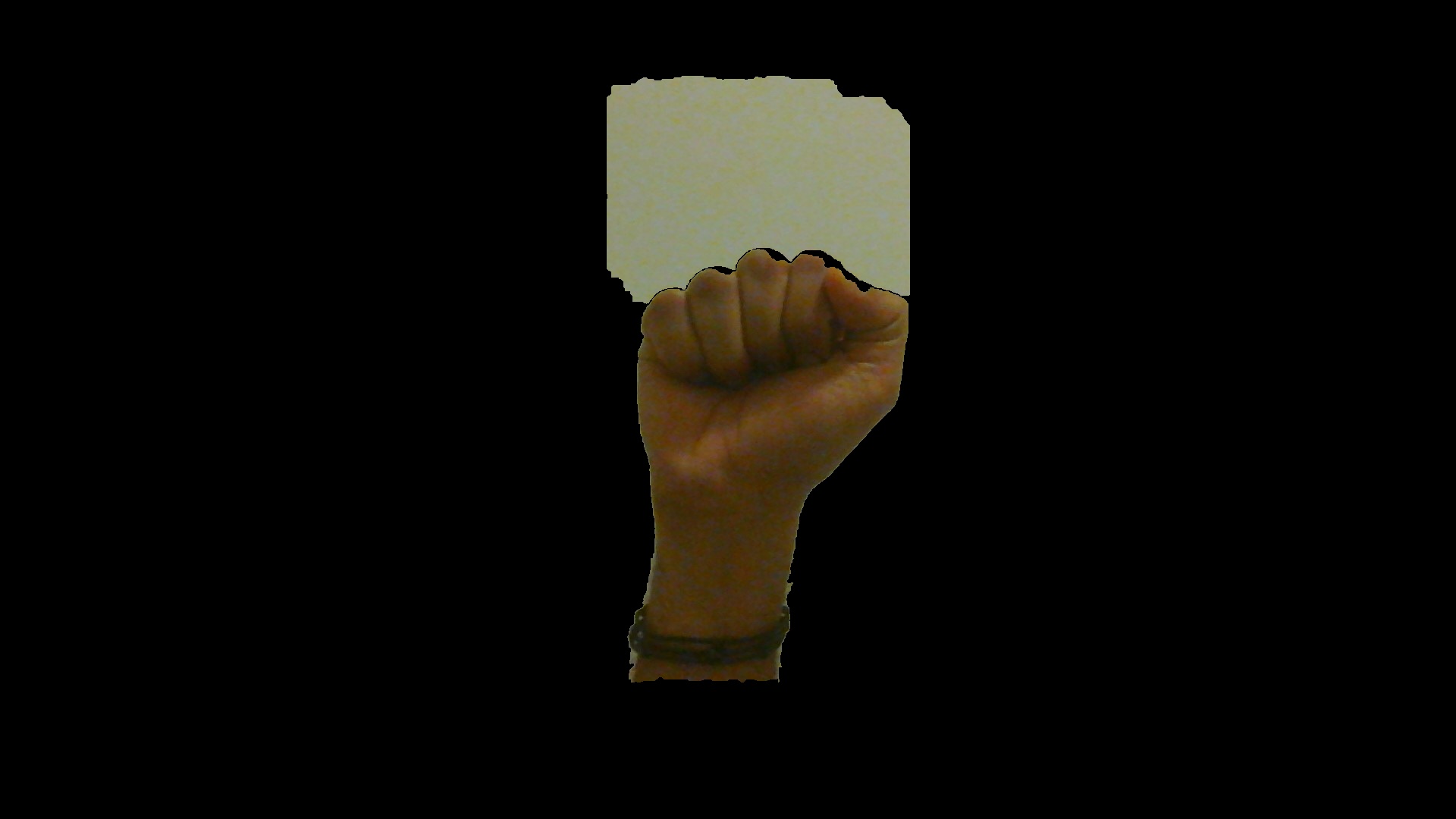
But while implementing this we came to a standstill, as the algorithm performs best for objects which are moving, which would not be ideal for our dataset. Thus we have decided to move on with other algorithms, keeping this at bay.

**Grab-Cut**

One of the main ideas that we considered for the pre-processing stage was the grab-cut algorithm. As the shape of the hand was our main region of interest, we theorized that we could use this algorithm to detect the contour of the hand in the image, and use this contour to significantly define the shape of the hand. This would enable us to get a clearly segmented image, with the shape of the hand being outlined perfectly, while also eliminating any noise present in the image.

The main difficulty we encountered with this algorithm was that it requires user interference to point to the region of interest in the image. This made it troublesome to specify the region of interest for each image of the dataset. In order to eliminate the user interference to specify the interested region, we created a function that would automate this process. This function creates a bounding box on the largest area in the image and then extracts that from the image. As our learning model has a training dataset of only hand signs, the background information gets eliminated.

However, we had some issues with the grab-cut algorithm itself, as it was unable to perfectly detect the contour of the hand in certain images. This limited the effectiveness of this grab-cut as a valid pre-process algorithm, and hence, this idea was not utilized in our final implementation. But, this algorithm did show some promising results on a few images, and given some additional time, we could have possibly ironed out the issues present in this algorithm. The image below shows an example of an image that we obtained by using the grab-cut algorithm.



**Region-Growing**

This is a simple region based image segmentation technique, that we considered earlier on in the project. It is more of a pixel based segmentation algorithm, as it depends on an initial seed point. The idea of this approach is that it checks the nearby points and looks at their properties, and the pixel is then added to the region with which it is considered to share the highest similarity. As a result, this technique is regarded to be more of a clustering method.

We managed to do some simple testing based on this algorithm, but due to the slight complexity of the algorithm in comparison to other segmentation algorithms, we did not have sufficient time to perfect the implementation of this algorithm. However, this is one algorithm that could definitely be considered in the process of segmenting the hand from the background.

**CNN**

The main issue that we had with our CNN was that the training time was too long, especially when convolution and max-pool layers were introduced into our model. This comes from the fact that we were using our CPU to train the CNN model, which is highly limited in parallelizing this process. To improve our model in this aspect, we were planning to train our model on the GPU, as the architecture of the GPU is such that it allows for tasks to be highly parallelized (in comparison to the CPU).

One possible method to achieve the above would be to utilize the \textbf{numba} library available in python, which utilizes CUDA-enabled Nvidia graphics cards to parallelize the execution of the code, resulting in a much lesser training time. This required significant alterations to be made to the python code, and considering the large size of the code, we were unable to complete this procedure for the final implementation. However, this is definitely something that we could incorporate in the future.

Methodology

Image Segmentation Using Global Thresholding:

As mentioned above

Image Segmentation Using Adaptive Mean and Adaptive Gaussian Thresholding:

From the above technique of image segmentation using global thresholding, we understood that it works best on uniformly illuminated images. As the image cannot be always taken in a perfect environment for testing, our developing model would fail with this technique. With an aim of using our developed CNN in real world scenarios, we started to look at other techniques. Another issue with global thresholding technique is that while converting the image, each pixel is taken and then compared with a global threshold value. This threshold value would decide if that pixel is a part of 0 or 255, that is black or white cluster. As shown in the figure below we can see this problem of illumination and the global thresholding value.

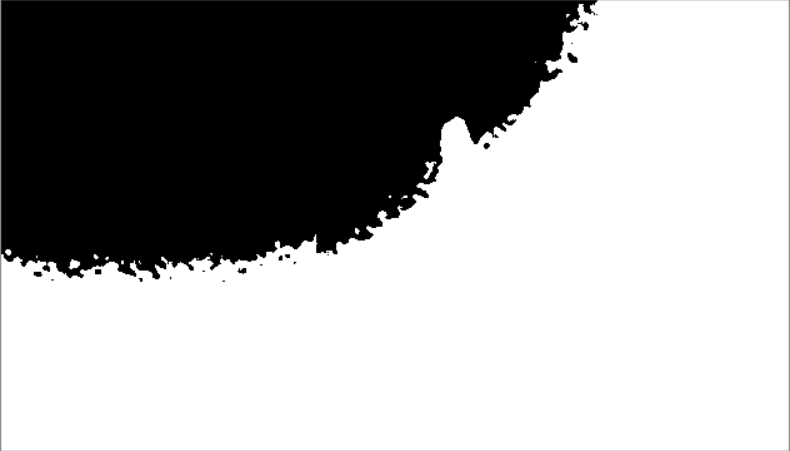


Fig: Shows Palm Sign under high illumination

From the above figure which shows the Palm sign, using Otsu’s global thresholding with uneven illumination we got to observe that further preprocessing is required for developing a model that could work better than the baselines. With this in our thought we went with adaptive mean thresholding and adaptive gaussian thresholding.

In adaptive mean thresholding, rather than using only one value thresholding just like global thresholding technique, this method uses a value that is the average or mean of the neighborhood area. This would provide a much better output for the same image using global thresholding. The below mentioned image shows the adaptive mean thresholding technique.



Fig: Adaptive Mean Thresholding

In adaptive gaussian thresholding technique, instead of using the mean of the neighborhood area this method makes use of the gaussian-weighted sum of the neighborhood values. The below mentioned image shows the adaptive gaussian threshold for the same image shown above.



Fig: Adaptive Gaussian Threshold

We can see from the above images a much clearer image reproduction can be seen with adaptive mean thresholding and adaptive gaussian thresholding compared with global thresholding technique mentioned earlier.

We took adaptive gaussian technique as our final thresholding technique to preprocess the data images as this would give us a positive edge to develop a better predictive model. As the illumination problem is not present here because we convert the image firstly to a grayscale and then use the technique, we eliminate the issue. Thus, with the dataset preprocessed, we move on to the next stage which is developing a convolutional neural network model for correctly predicting the hand sign image in the given image.

Conclusion

Through this project, we learnt about a myriad of topics, ranging from dealing with raw, non pre-processed data to developing a fully functional convolutional neural network. Starting off, we created our own dataset rather than using a pre-processed and ready-to-go dataset, to get an idea about what we should keep in mind when creating a dataset from scratch. We had a huge learning curve in this phase of pre-processing.

We looked at different possible steps for pre-processing, such as dimensionality reduction, where we reduced the dimensions of the image from a 3-channel RGB image to a single channel grayscale image. Additionally, maintaining a uniform aspect ratio for the entire image dataset was crucial. We also reduced the size of the images in the dataset, as it will enable the CNN to perform faster without much loss in accuracy.

We also learnt to use image segmentation techniques like Otsu’s method, global thresholding and adaptive thresholding. Upon experimentation, we understood that Otsu’s method works best for uniformly illuminated images that are captured and processed. We realized this would be a major issue and went with the method adaptive gaussian thresholding for image segmentation, which eliminated the need for uniform illumination in our dataset images. Through the above process, we recognized the effort that is required to clean the dataset, as well the importance of the pre-processing step in creating an ideal dataset, as it will eventually enhance the performance of our CNN.

In the CNN implementation portion of our project, we have gained an immense understanding of the inner workings of the convolutional neural network model, and what are the methods that are employed to optimize the CNN model. We learnt that the leaky ReLU function works best in the CNN model, while the Kaiming initialization method is the preferred technique to utilize for initializing weight vectors and biases. We also determined that the learning rate plays a crucial role in the CNN model, and based on our experimentation, observed that the exponential cyclic learning rate function gave us the best results.

Finally, through adjusting all the possible hyper-parameters on the CNN model, we obtained the best configuration possible, and were able to achieve a maximum test accuracy of 76.33\%. This was compared to an equivalent \textbf{Keras} CNN model, which was able to attain a test accuracy of 80.67%. The difference in test accuracies was observed to be marginal, and hence, we have concluded that we have managed to successfully achieve our goal of implementing a fully-fledged CNN for the purpose of hand gesture recognition.

Results

