Vision-based Sign Language Translator

Project report submitted to
Visvesvaraya National Institute of Technology, Nagpur in partial fulfilment of the requirements for the award of the degree

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY

Abhishek Singh Dhadwal	BT17CSE003
Kopal Bhatnagar	BT17CSE038
Saurabh Pujari	BT17CSE068
Yash Kumar	BT17CSE092

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISVESVARAYA NATIONAL INSTITUTE OF

TECHNOLOGY NAGPUR (INDIA) – 440010

2020 -2021

ACKNOWLEDGMENT

We would like to express our utmost gratitude towards our project guide, Dr Poonam Sharma, Assistant Professor, Department of Computer Science and Engineering, VNIT Nagpur for their invaluable guidance, continuous support, and encouragement.

We would also like to sincerely thank Dr P. S. Deshpande, Head of Department, Department of Computer Science and Engineering, VNIT Nagpur for their support.

We are also sincerely thankful to the entire faculty of the Department of Computer Science and Engineering, VNIT Nagpur.

ABSTRACT

Sign language is the primary medium of communication between individuals suffering from hearing and vocal impairments. However, a large majority of the general population can’t communicate using sign language. To confront this problem, we have proposed a near real-time solution that implements the concepts of Image Processing and Deep Learning, where the symbols are converted to the corresponding alphabets in plain text.

This can further be improved and be developed into a general-purpose symbol-to-text converter, where a single hand sign can be represented as a particular tag or message instead of just an alphabet. This extends its utility from just healthcare of the deaf and dumb community to a lot of possibilities.

Contents

1. INTRODUCTION

1.1 Motivation

Communication is one of the most basic, if not the most basic requirement for survival in today's world, and this is something that individuals who are deaf and dumb struggle with regularly. While they are freely able to communicate amongst one another using the sign language conventions, a regular individual without these impairments may not be well-versed with them. This makes it extremely difficult for the deaf and dumb community to communicate with others without a dedicated special interpreter, whether human or machine who is comfortable with both sign language and any vocal language the regular individual speaks.

The Government of India has enacted the Rights of Persons with Disabilities Act 2016 (RPwD Act 2016), recognizing the Indian Sign Language (ISL) as an important communication medium, insisting on the need for sign language interpreters in all Government organizations and public sector undertakings.

In such a situation, a system that converts symbols in sign languages to plain text may enable real-time communication between the two parties involved. This can be extremely effective in the field of healthcare, where specialists who can converse with patients in sign language regarding their needs are always required. This will be helpful to the medical staff as well as those who work and take care of the elderly at old age homes. The work we have performed here is done to provide an interface to bridge this gap in communication between the two parties.

1.2 Problem Statement

Our primary aim is to translate Indian Sign Language (ISL) to plain text in a near-real-time scenario and make it partially independent of the background and illumination conditions.

1.3 About Sign Language

There are two main forms of Sign Languages:
1. Alphabet notational hand gestures
2. Ideographic notational hand gestures

The first kind focuses on representing each alphabet as a singular hand sign and then spelling the entire word or sentence one alphabet at a time.

On the other hand, the second kind expresses each meaningful word with a specific hand gesture. This is the form of sign language in use in today's world, but it differs across the world.

While we have taken alphabet notional hand gestures as our training dataset for simplification, we are using those signs to represent a word or a phrase effectively, therefore implementing ideographic notational hand gestures.

There is no universal sign language, and different countries use different symbols to denote different alphabets. There are more than 100 sign language systems across the world. Some countries like Belgium, the USA, India, etc. even have more than one sign language standard.

2. LITERATURE REVIEW

2.1 Literature Survey

Sign language is not a new computer vision problem. Over the past two decades, researchers have addressed it in various ways. These can be broadly classified into glove-based or sensor-based methods and appearance-based or vision-based methods.

Glove or sensor-based approaches provide good accuracy as these specific devices collect data or features directly from the signer but, it will have an overhead of carrying and signing with the external device.

Vision-based methods, on the other hand, do the recognition task from images or videos based on the features calculated using various image or video processing techniques. This can be again categorized into 2-D Vision-based techniques and 3-D Vision-based techniques. Researchers have used classifiers from a variety of categories that we can group roughly into linear classifiers, neural networks and Bayesian networks.

These two main approaches are used for acquiring gestures from the user. The first approach is vision-based, which provides more freedom to the user to perform natural actions and the other is a sensor-based approach that has simpler data acquisition and processing requirements.

In paper [1], a real-time sign language recognition system is implemented that can overcome a complex background scene using a hierarchical clustering classification method.

It uses 26 hand gestures in the training phase to develop a robust hand segmentation method to generate contour information. Three types of features are then extracted to represent the gesture and a dimension reduction step is applied to find the most discriminative features for the final

classification step. These features are combined to describe the contours and the salient points of hand gestures. Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), and Support Vector Machine (SVM) are integrated to construct a novel hierarchical classification scheme.

The procedures in the testing phase are similar to those in the training phase; however, the input in the testing phase is a video sequence composed of continuous hand gestures. Considering the real-time issue, a keyframe selection method filters out the redundant frames in the video, and only important/stable frames are kept to be recognized. To extract hand features with less noisy information even though the background is complex, the hand region is first located by skin colour detection. A method to adaptively update the skin colour model for different users and various lighting conditions is also designed.

This is evaluated over the CSL and public ASL datasets. The method achieves the accuracies of 99.8% and 94%, respectively, which outperforms the existing works.

In paper [2], a keyframing approach is proposed to reduce the motion data by extracting keyframes using a motion analysis approach in sampling windows. Motion changes in sampling windows for original motion without frame skipping and with frame skipping are computed. The difference in the motion changes is the main aspect in deciding whether the frames in sampling windows are possible candidates for keyframe selection.

Significant motion changes shall be defined as motion activity that holds above a threshold parameter. Five types of motion sequence were tested based on different dynamics of the motion, with low dynamic motion examples such as walking, jumping and stretching motion; and high dynamic motion examples such as dancing and playing basketball motion.

The proposed motion analysis method was tested to evaluate the performance of keyframe extraction. The proposed method which uses

different sampling window size at the same threshold value to extract keyframes performs better than the curve simplification method when comparing the same number of keyframes extracted.

For low dynamic motion such as the walking sequence, 58 frames were extracted out of a total of 316 frames and showed an improvement of 52.2% in performance. For high dynamic motion such as basketball sequence, the proposed method can reduce 70% of the total frames with an MSE error of 0.0463.

In paper [3], Indian sign language static alphabet recognition problems with a vision-based approach are addressed.

A Convolutional Neural Network (CNN) which is a deep learning technology is used to create a model named signet, which can recognize signs, based on supervised learning on data. The whole process can be divided into CNN training and model testing.

In this work, a dataset containing binary hand region silhouettes of the signer images is used. These images were extracted and saved after the preprocessing stage. This preprocessing includes Viola-Jones face detection algorithm to detect the face of the signer and then it is eliminated by replacing it with black pixels. This image is then processed with a skin colour segmentation algorithm followed by the largest connected component algorithm for hand region segmentation.

These images are used in this work to train and test the signet architecture. Images along with their class labels are given to the developed CNN architecture to learn the classification model. The learned classification model can be tested and then saved for recognizing ISL static alphabets.

The proposed method produced a remarkable result compared to current state of art methods. Training accuracy of 99.93% and validation accuracy of 98.64% was achieved.

In paper [4], an extensible system is discussed which uses one colour camera to track hands in real-time and interprets American Sign Language (ASL) using Hidden Markov Models (HMM's).

In this implementation, the tracking process produces only a coarse description of handshape, orientation, and trajectory. The hands are tracked by their colour: in the first experiment via solidly coloured gloves and the second, via their natural skin tone. In both cases, the resultant shape, orientation, and trajectory information is input to an HMM for recognition of the signed words.

By combining the relative motion of the hand between frames and the absolute position of the hand with the angle, eccentricity, area, and length of the major eigenvector, the highest fair test accuracy, 91.9%, was reached for natural skin tracking

In paper [5], an algorithm of Hand Gesture Recognition by using Dynamic Time Warping methodology is presented. The system consists of three modules: real-time detection of face region and two hand regions, tracking the hands' trajectory both in terms of direction among consecutive frames as well as the distance from the centre of the frame and gesture recognition based on analyzing variations in the hand locations along with the centre of the face.

The proposed technique is used to perform similarity measurement efficiently and increases the adaptability of a hand gesture recognition system. The technique works well under different degrees of scene background complexity and illumination conditions. Experimental results showed the recognition rate to be around 90%.

2.2 Previous Work

This problem statement of Vision-based Indian Sign Language Translator[6] was initially worked on by our seniors. They trained an Inception V3 Convolutional Neural Network model with a dataset of 7800 images divided into 26 classes, one for each alphabet. The training process ran for 2000 epochs. The frames get extracted from the input video (from the webcam) which is fed to the newly trained model, which further classifies the input sign in the image. A label is extracted corresponding to this classified image, and if it is an alphabet, it is printed to the terminal. It also incorporates signs for delete and space operations on the output sequence. This process is continued frame by frame till an escape command is hit. It gave a 99% training accuracy and a 98% test accuracy. Different letters gave different accuracies overall (Eg. O gave 99.1%, B gave 77%, etc.).

However, there were a few limitations reported regarding this program:
1. It required the input frames to be free of noise (eg. shadow, poor illumination, or any other object).
2. It was highly dependent on a white and clear background (especially for a live video).
3. The dataset consisted purely of symbols made by the right hand, so it didn't work with the left-hand symbols.
4. For video input, if the frame rate was slow, the symbols would get repeated.
5. For a live video, there was a fixed box on the screen, and the signer was restricted within its boundaries to show its symbols.

We later discovered that there were a couple more specific technical limitations and concerns for this project:
1. The dataset of 7800 images for 26 classes (essentially 300 images per class) was too small for such a large classification problem. As transfer learning was used, it magnified this problem of the small dataset as it didn't satisfy the minimum requirements to apply this method.

2. As 2000 iterations were performed over this limited dataset, it potentially could have skewed the performance of this model by overfitting to this dataset, which might also be the cause for the earlier limitations.
3. The code was difficult to interpret and improve upon directly.

The flowchart for the process is given below:

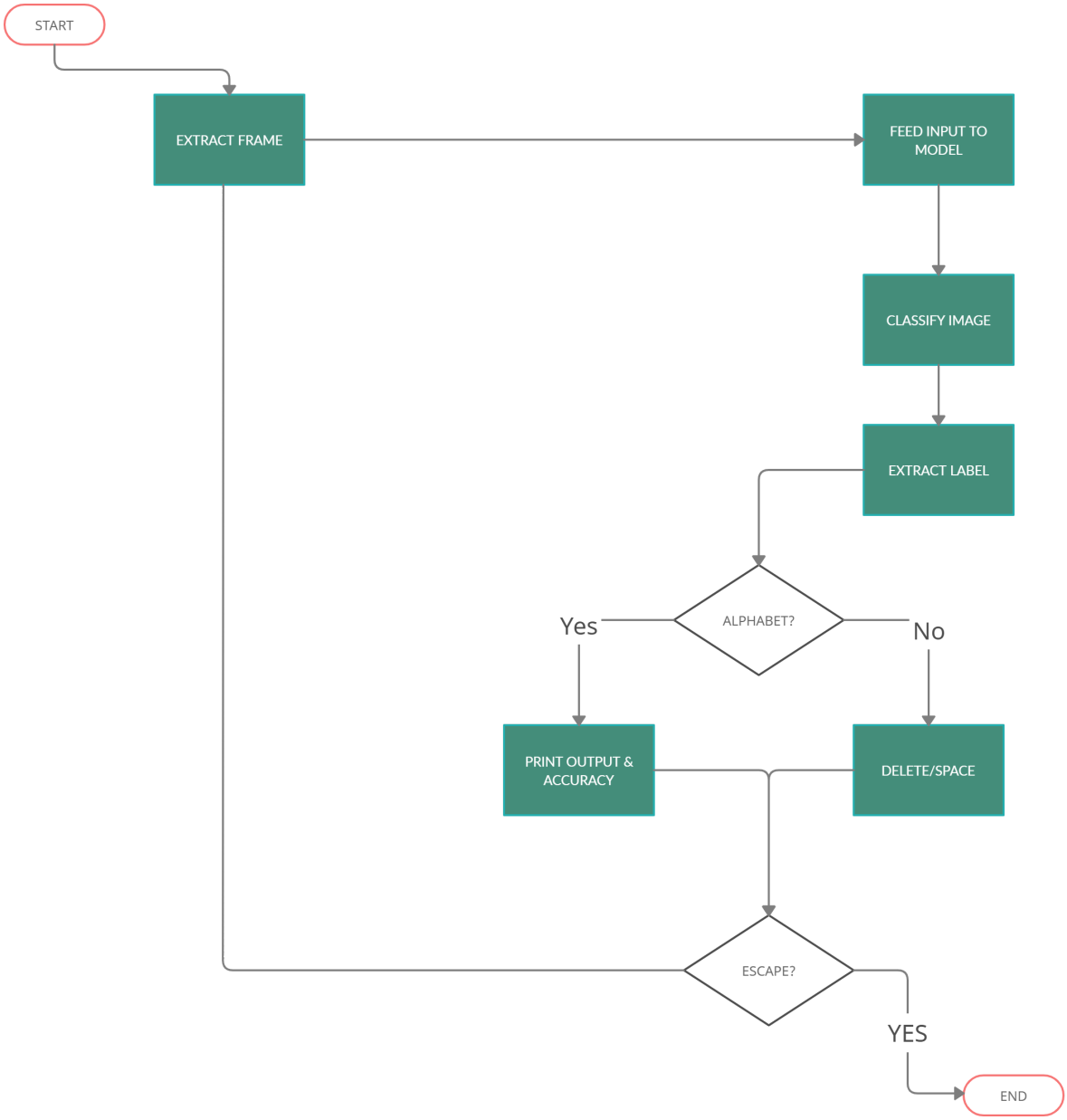


Fig 2.2.1: Flowchart diagram for the previous workflow

3. Dataset

3.1 About Our Dataset

There were two major datasets that we have used in our project:
1. Indian Sign Language dataset:
   - 26 alphabet classes
   - 1200 images per class

2. Manually created dataset:
   - A single sign for the letter ‘O’
   - 600 images
   - 6 types of images

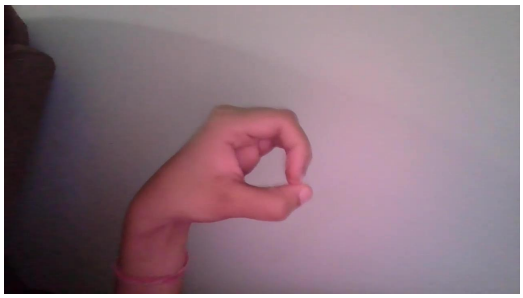
The first dataset, which was our primary dataset, was used for training the model to learn the hand signs.
Some examples of the dataset are given below:

Fig 3.1.1: Sample images from this dataset for the letters ‘G’, ‘K’, and ‘O’ respectively

On the other hand, the second dataset was created by us manually. It consists of six types of images:
   - Base condition (regular image)
   - Blurred
   - Containing background interference (other objects)
   - Containing other body parts

- Skin-coloured background
- Zoomed out images

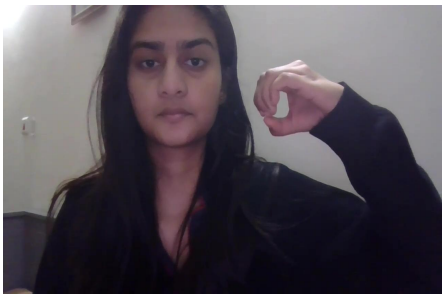
This dataset was used to observe the model's performance when subjected to images containing different types of interferences or noises.
Some examples of these six variants within the dataset are given below:



A. Base condition

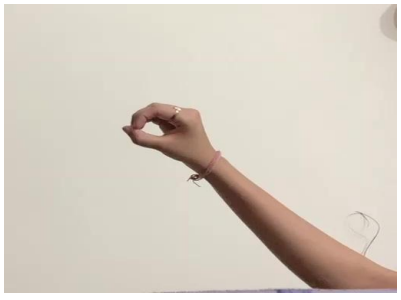
B. Blurred

C. Containing background interference



D. Containing other body parts

E. Skin-coloured background



F.. Zoomed out images

Fig 3.1.2: Sample images from this dataset for the letter 'O' of 6 types

To obtain this dataset, we created 8-10 second clips for each class and extracted nearly 100 frames from each of them individually. All these

images represent the same sign for the alphabet ‘O’, but they all represent different conditions that the model might face in practice.

3.2 Dataset Synthesis

The first ISL dataset, as mentioned earlier, contains 1200 images for each of the 26 classes. However, for the task of training a model to recognize and differentiate between 26 distinct classes, the size of the dataset is lower than ideal. Additionally, most of the images represent a base condition for the images - clear images with ideal lighting and a black background. Moreover, the images look quite similar and even identical in most cases.

In a real-life scenario, we are trying to solve all these problems as they will likely be prevalent in practical situations.

To do so, we performed dataset synthesis over the current dataset. A script was run over the original dataset, which generated 18 new images per image of the original dataset, which modified two factors:

- Illumination
- Focus

The resultant dataset is much larger than the original dataset and contains sufficient variation between its images.
For modifying the illumination, we have used the following ranges of the multiplicative factors:

- 0.4 - 0.9 for darkening
- 1.1 - 2 for brightening

For modifying the focus, we have used the following ranges of blur radius:

- Lower blur radius = 5
- Upper blur radius = 20

These manually chosen ranges can be modified easily and the exact values for modifying a single image are selected randomly within the given ranges.

This helps us to mimic real-world scenarios such as in a hospital where it's possible that the lighting may be dim or bright during different hours of the day. It also takes into account the hand motion which might result in a slightly blurred image being captured by the camera.

An example of the output is given below:

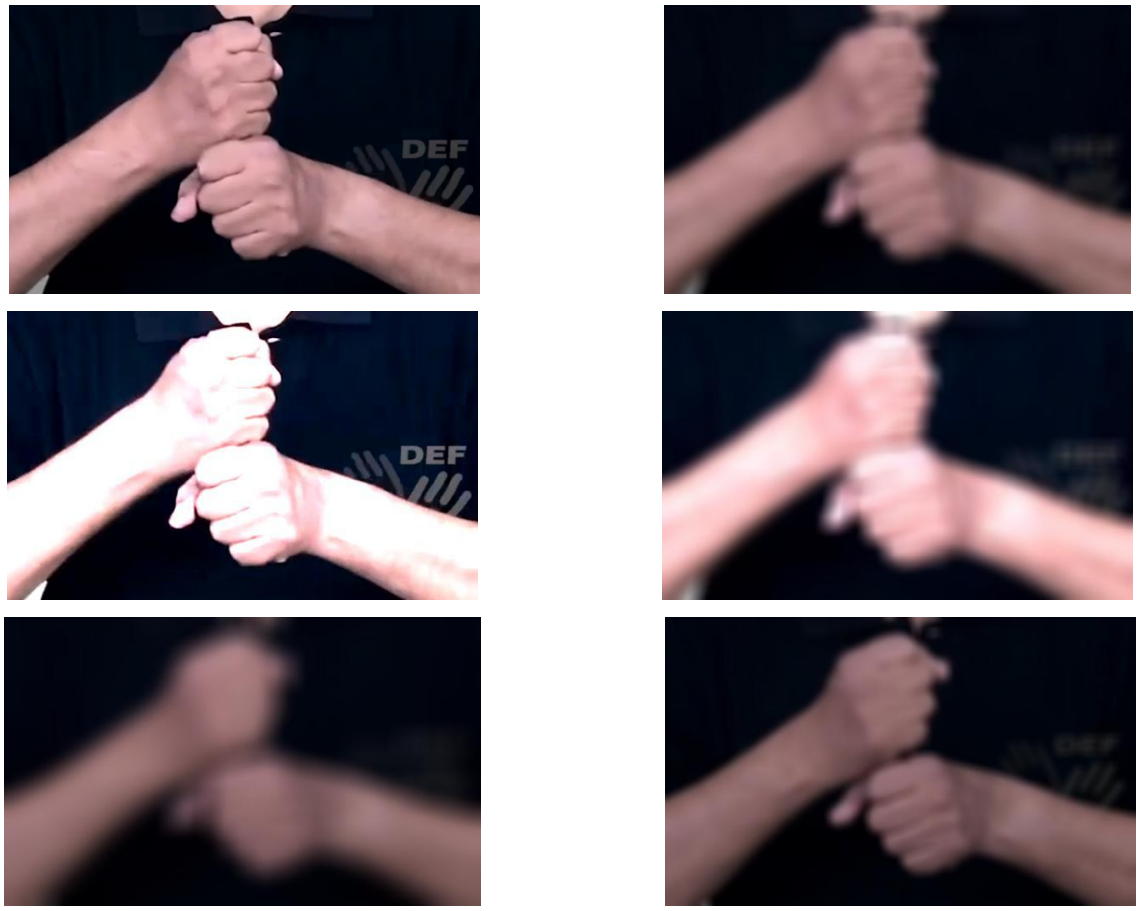


Fig 3.2.1: The top-left image is the original image, representing the symbol for the alphabet 'G'. The rest are generated from the script. As we can see, there are samples of brightened, darkened and blurred images.

This new dataset contains 20 classes (G - Z), each containing 21,600 images.

As mentioned earlier, the second dataset which was created manually was used mainly to test the model's performance over different physical conditions. Furthermore, the dataset was created keeping in mind the variation required while testing. Hence, there were no necessary actions that needed to be taken to increase its size or modify its images.

3.3 Dataset Cleaning

Having obtained the new 21,600 images per class dataset, we now had more than sufficient images to train the model. However, a new issue arose with this newly synthesized dataset. A significant proportion of the images were either too bright, too dark, or too blurred.
These images then act as noise in the model and hinder its performance. Thus, the results obtained from this model are not representative of its true performance, and hence, dataset cleaning must be done.

To counter this issue, we decided to perform hand detection on the dataset. This was performed firstly on the manually-created dataset to observe what kinds of images we were facing issues with. We chose pre-trained YOLO-v3 and its variant networks for this.

YOLO-v3 is a very common and fast approach for object detection, even in a real-time setting. We also tested its variants, YOLO-v3-Tiny and YOLO-v3-Tiny-PRN (Partial Residue Networks) over their tendencies to detect hands in the images, and then compared their results to select the one best suited to our problem.

For 0.75 confidence, the three algorithms were run over this dataset. We got the following results as shown in Table 3.3.1, which represents the percentage of images where hands were detected by the given algorithms:

Table 3.3.1:

Model	1. Base Condition		2. Blurry		3. Skin Colour Background		4. Background Interference		5. Zoomed Out		6. Front Face	
	Count (89)	Conf	Count (100)	Conf	Count (100)	Conf	Count (100)	Conf	Count (100)	Conf	Count (100)	Conf
YOLO-v3	89	1	0	N/A	100	0.96	100	0.97	100	0.99	77	0.85
YOLO-v3-Tiny	52	0.84	0	N/A	1	0.79	7	0.77	0	N/A	7	0.79
YOLO-v3-Tiny-PRN	77	0.91	0	N/A	0	N/A	96	0.9	19	0.79	67	0.86

From the given data, we can observe that YOLO-v3 performs the best over the six types of images. It's also worth noting that not a single blurred image had a hand detected by any of the algorithms. This is a major concern for the project.

We now pass the 432,000 images across 20 classes of the modified version of the first dataset through the YOLO-v3 pre-trained neural network. Certain alphabets have a single hand making the sign, whereas some have two. Accordingly, it checks whether the image contains the required number of hands in the image and if so, the image is stored for future usage [PASS]. Each image has been assigned certain confidence values with which the algorithm has detected the hands, and they are then accordingly assigned to three categories based on the detection confidence:

- Confidence ≥ 0.5
- Confidence ≥ 0.75
- Confidence ≥ 0.9

If it doesn’t at least have a minimum of 0.5 confidence, the image is discarded.

The following flowchart represents how the algorithm works:

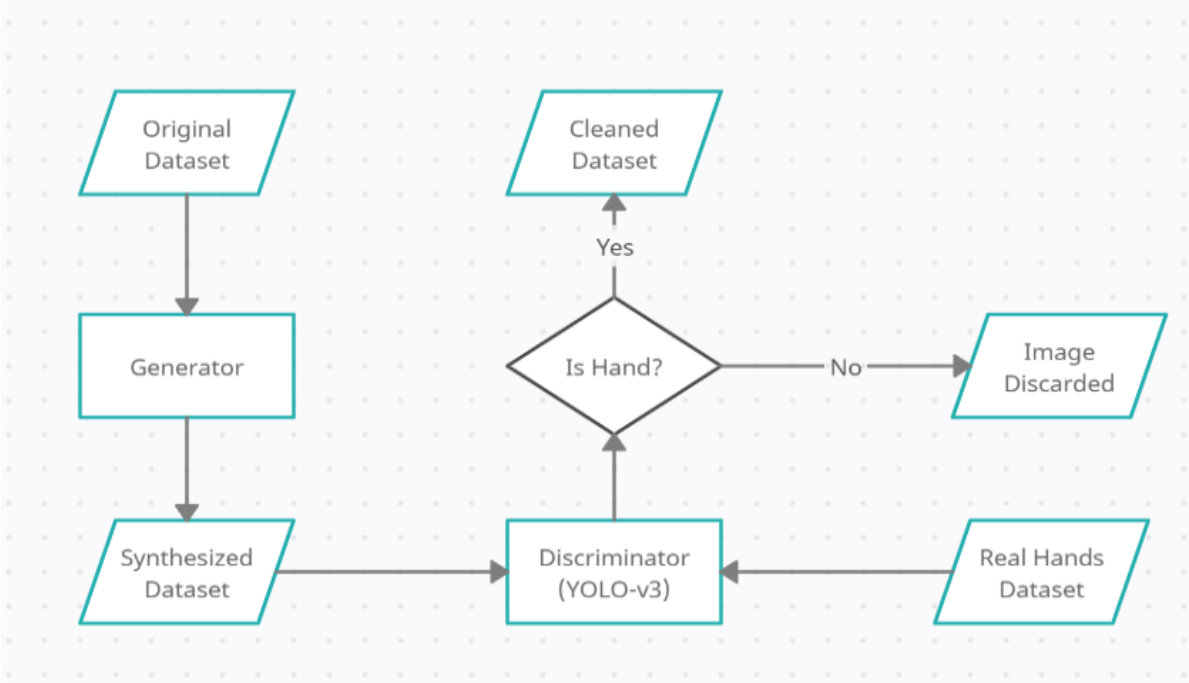


Fig 3.3.1: Flowchart diagram for the dataset synthesis & cleaning workflow

As we can see, we had the original ISL dataset, which we expanded and diversified using the Generator script. These new images formed the Synthesized Dataset, which was then passed to the discriminator, which is the YOLO-v3 algorithm. It detects whether a sufficient number of hands is present or not in the image, and if so, it is added to the Cleaned Dataset and if not, it is discarded. Also worth noting is that this YOLO-v3 object detection network was pre-trained to detect hands using the Real Hands dataset.

After doing so, we got the following results as shown in Table 3.3.2, showing the percentage of images across this generated dataset that had a sufficient number of hands detected when passed through the YOLO-v3 algorithm:

Table 3.3.2:

Confidence	% of Selected Images
0.5	12.16%
0.75	7.39%
0.9	2.20%

As our goal is not to detect all 26 hand signs but to detect some of them accurately which will then act as symbols for certain requirements, we shortlisted the top 10 alphabets which had the highest number of images with a 0.5 confidence value or more. Those alphabets and their number of images are as follows in Table 3.3.3:

Table 3.3.3:

Alphabet	Number of Images Detected with 0.5 Confidence
O	12078
P	6157
J	5680
Y	4927
V	4787
G	3750
X	3664
U	2974
S	2728
K	2326

By doing so, we have now removed the extremely noisy images from the dataset using our network. We have obtained a cleaner dataset by focusing on the top 10 classes. To maintain data uniformity, we selected 2326 images for each class. We also checked whether some alphabets had more than 2326 images in the 0.75 or 0.9 confidence lists, and if so, we have selected images from those lists instead of the 0.5 confidence list to optimize the quality of the images as much as possible.

Random selection is performed to select 2326 images per alphabet class and we finally obtain our final training dataset with 10 classes each containing exactly 2326 images.

These images go through the same preprocessing techniques as the frames that were extracted from the webcam feed go through, before they are finally passed as input to the model. These techniques are elaborated upon in detail in the paper's Architecture, Preprocessing section.

4. Proposed Architecture

4.1 Architecture

The proposed architecture can be divided into two major sections, represented by the flowchart diagram given below:

1. Training implementation:

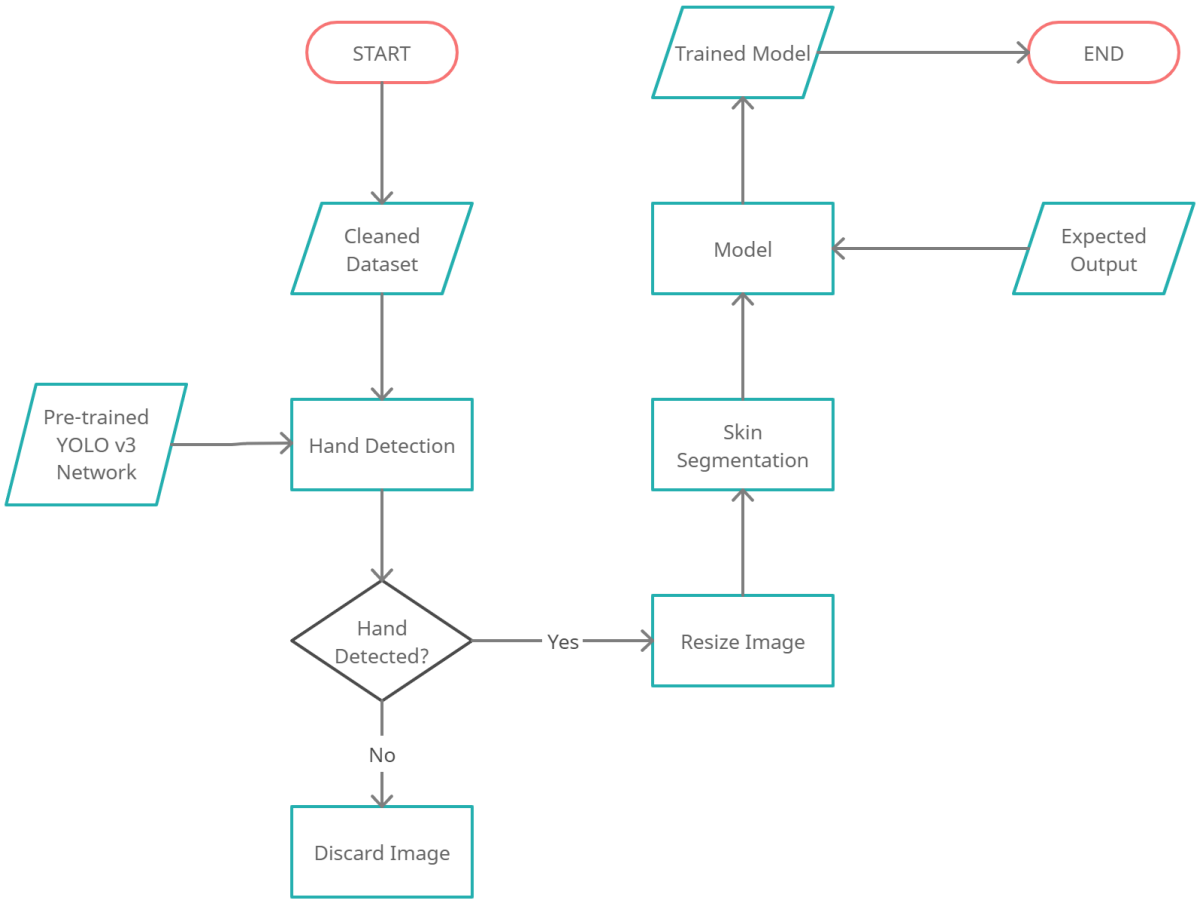


Fig 4.1.1: Flowchart diagram for the training implementation's workflow

This implementation, as the name suggests, is used to train the model. The images from the dataset that we had obtained after synthesis and cleaning in the previous phase are read from a folder. These images are preprocessed before they are sent to the model. Firstly, they are passed through the hand detection algorithm, the same as the one which we used

in the dataset cleaning step. If the network can detect the correct number of hands with sufficient confidence, it is moved on to the next part; else, it is discarded. The images that remain are resized to (224, 224) for the sake of maintaining the consistency of the images that are passed through the model. Finally, it goes through skin segmentation, which completes the preprocessing phase. The results from these are passed to the model along with the expected output of the image.

2. Webcam implementation:

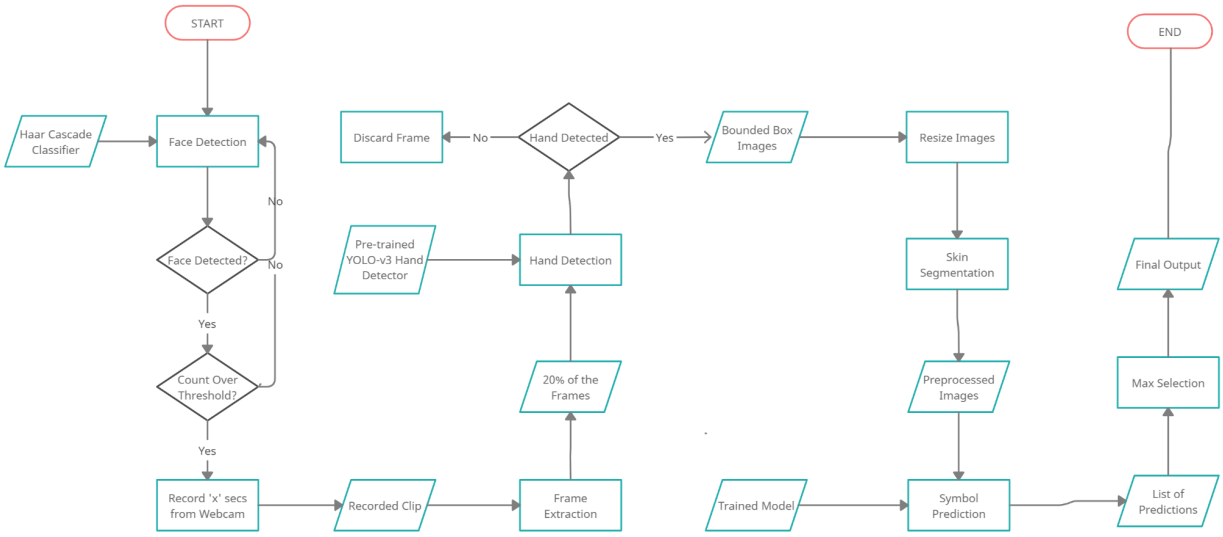


Fig 4.1.2: Flowchart diagram for the webcam implementation's workflow

The webcam implementation is divided into five phases. The first phase involves Face Detection. The intuition behind this is that the user will look at the camera for a couple of seconds straight when they wish to send a message, and doing so will then activate the next phase of the pipeline, Video Recording. For the next 'x' seconds, the user will hold up the symbol that corresponds to their message. This video clip is saved and a random sample of the frames, equivalent to 20% of the total number of frames, are extracted from it. These frames go through the same preprocessing steps as the previous implementation, i.e., hand detection, image resizing, and skin segmentation. The preprocessed images obtained are then passed through the model we had trained in the first implementation, which predicts an output for the given frame. The output which occurs the maximum number of times is selected as the final output.
The steps mentioned in the two implementations are explained in detail in the below subsections.

4.2 Face Detection & Webcam Feed Recording

The first two phases of the implementation were Face Detection and Webcam Feed Recording respectively. The first, Face Detection, is based on the idea that the user will look at the camera when they wish to make the hand gestures. When they do so for a couple of seconds at a stretch, the number of frames that contain a face looking at the camera directly will satisfy the set threshold value, and that will signify the user's intent to send a message using a hand gesture.

This was done by using a Haar Cascade Classifier, a machine learning object detection program that identifies objects in an image and video[7].

This approach has 3 main sub-components :

- Introduction of a new image representation called the "Integral Image" which allows the features used by our detector to be computed very quickly.
- A learning algorithm, based on AdaBoost, which selects a small number of critical visual features from a larger set and yields extremely efficient classifiers.
- A method for combining increasingly more complex classifiers in a "cascade" which allows background regions of the image to be quickly discarded while spending more computation on promising object-like regions

The OpenCV library contains pre-trained Haar Cascade Classifiers models to detect body parts in an image, and one such model is a full-frontal-face model, which has been used to detect the faces looking straight at the camera in our implementation.

When a face is detected, the program monitors the next few consecutive frames for a face, until a threshold value is reached and we can say with confidence that a face has been detected and thus we can trigger the sign language detection system.

An example of what the face detection algorithm looks while running like is given below:

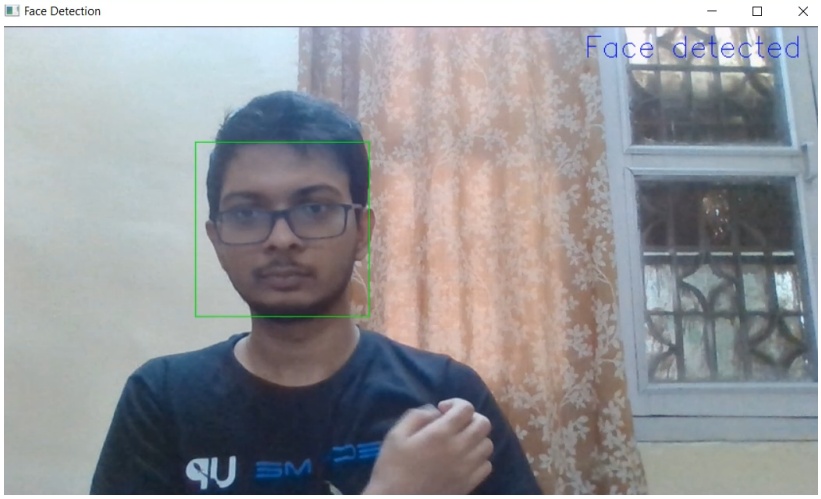


Fig 4.2.1: Face Detection algorithm running

The next phase is that of Webcam Feed Recording. As the name suggests, the user after completing the Face Detection phase, will now get a fixed amount of seconds where the camera will record the video of the user during execution. The user will make the appropriate hand gestures within that duration, which will then be saved and passed on to the next phase.

4.3 Frame Extraction

In the webcam implementation, the first step is to extract the frames from the video. For this, we clip and save the webcam feed every three seconds. These 3-second clips are stored continuously from the point that the webcam starts reading, and this whole process continues till the user performs the exit command.

Frame extraction is performed on these 3-second clips using three different approaches:

1. Using all frames
2. Keyframe Selection
3. Random sampling

The first approach is the most basic approach and self-explanatory. From a given clip, all the frames are extracted from it for further computation.

On the other hand, the second approach is a bit more intuitive. Key Frame Selection is the process of extracting a frame or a set of frames that have a good representation of a shot. This essentially means that these fewer frames can preserve the salient features of the shot while removing most of the repeated frames.

This was implemented using an open-source tool called Katna[8], a tool that automates the error-prone task of video key/best frames extraction, video compression and manual time-consuming task of image cropping.

The Frame Extraction and Selection Criteria for Katna is as follows:

1. Frames that are sufficiently different from previous ones using absolute differences in LUV colour space
2. Brightness score filtering of extracted frames
3. Entropy/contrast score filtering of extracted frames
4. K-Means Clustering of frames using image histogram
5. Selection of best frame from clusters based on and variance of laplacian (image blur detection)

The third and final approach was to use random sampling. A sample is meant to be a representation of the total population. Random sampling gives each frame an equal probability of being selected, and the sample chosen here will be an unbiased representation of the total population. In our case, $1/5^{th}$ of the frames are selected randomly.

These three approaches were compared over their relative accuracies and execution times. The results observed were as follows, as shown in Table 4.3.1:

Table 4.3.1:

Approach	Avg. Time (secs)	Avg. Relative Accuracy
All Frames	55.18	-
Key Frame	165.65	81.25%
Random Sampling	11.91	95.00%

The first approach was taken as the base case as it observes all frames of the video, and is most likely to be accurate. From the given data, we can see that the third approach, Random Sampling gives a comparable result with the base case while being significantly faster. Hence, Random Sampling is selected as the frame extraction method implemented in our architecture.

It is also worth noting that the Key Frame Selection approach, while expected to be quite efficient, is too slow to be used in a real-time scenario. Its performance however may have been hindered by the lack of movement in these clips which had only a small duration of 3 seconds, roughly amounting to ~45 frames per clip. It may not have detected significant change between the frames to consider them, and hence, the results are not completely accurate to their potential.

4.4 Image Preprocessing

The next phase of execution required the images to be preprocessed before they were fed to the model. The input images are passed through various detection and segmentation filters, which is followed by resizing. These tasks are performed in order to ensure our input images resemble the samples in our training dataset both in hand placement along with size. The filters utilized are provided below and are described in order of occurrence :

1. Hand detection based region cropping
2. Resizing

3. Skin Segmentation

In the first step of preprocessing, the input samples are passed through a YOLO-v3 neural network trained on the Real Hands dataset, through which we detect the presence of hands in the image.

YOLO-v3[9] is a real-time, single-stage object detection model that builds on YOLO-v2 with several improvements. Improvements include the use of a new backbone network, Darknet-53 that utilises residual connections as well as some improvements to the bounding box prediction step, and use of three different scales from which to extract features (similar to an FPN).

The cross-hands dataset consists of the CMU Hand DB[10], the Egohands dataset and trained images (mainly from marathon runners).

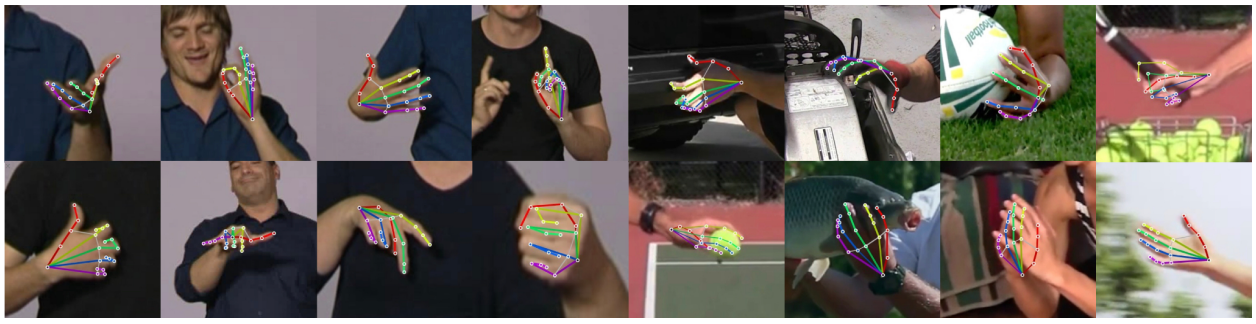


Fig 4.4.1: Sample images from Cross-hands dataset

For every frame selected from the clip, the network attempts to find a hand in it, and if any hands are found in the frame, an extended bounding box is created around the hand(s). An extended bounding box implies that if more than a single hand is detected, a bigger box combining the coordinates of the two is created. These images are then cropped to contain only the contents of the box, and are passed onto the next step of preprocessing. Should the network not detect even a single hand, the frame is discarded entirely from the next steps of the algorithm.

The next step in Preprocessing is that of Resizing. As mentioned earlier, the images that passed the hand detection step have been cropped to

contain only the bounding box content. These bounding box images are all of different sizes, and this essentially causes an issue of non-uniformity when it is passed to the model, which has been trained on images of the same size. To maintain the consistency, all images are resized to the size of 224x224, which was experimentally found to give the most consistent results.

The final step in Preprocessing is to perform Skin Segmentation. In this, skin-coloured pixels and regions are found in an image or a video. This process is typically used as a preprocessing step to find regions that potentially have human faces and limbs in images. Skin image recognition is used in a wide range of image processing applications like face recognition, skin dis-ease detection, gesture tracking and human-computer interaction and is useful for our requirements. There are many skin colour spaces like RGB, HSV, YCbCr, YIQ, YUV, etc. that are used for skin colour segmentation.

The following factors are considered for determining the threshold range for the above spaces:
1. Effect of illumination depending on the surroundings.
2. Individual characteristics such as age, sex and body parts.
3. Varying skin tone with respect to different races.
4. Other factors such as background colours, shadows and motion blur.

Skin detection is influenced by the parameters like Brightness, Contrast, Transparency, Illumination and Saturation. The detection is normally optimized by taking into consideration combinations of the mentioned parameters in their ideal ranges.

We have used a region growing algorithm (Watershed) and a combination of HSV (Hue, Saturation, Value) and YCbCr (Luminance, Chrominance) colour segmentations to work together to produce the output[13].

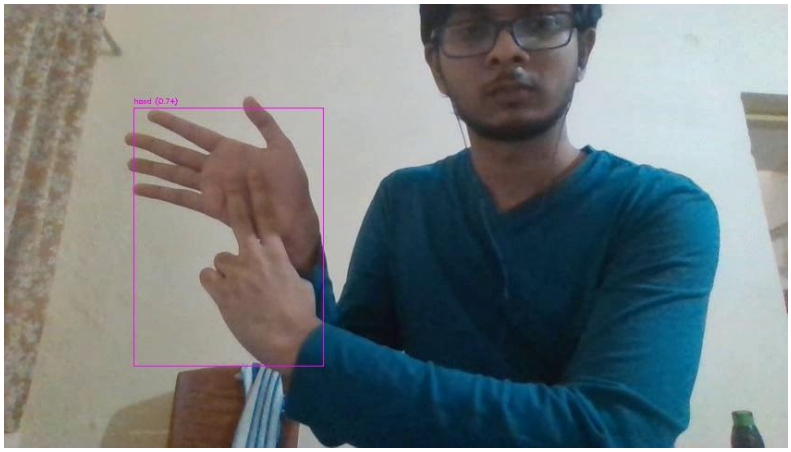


Fig 4.4.2: Webcam input with hand detection achieved

Fig 4.4.3: Cropped, Resized, and Skin segmented output of the image corresponding to Fig 4.4.2

We also attempted to use background subtraction/segmentation, a widely-used approach for detecting moving objects in videos from static cameras. The rationale in this approach is that of detecting the moving objects from the difference between the current frame and a reference frame, often called "background-image", or "background model". Background subtraction is mostly done if the image in question is a part of a video stream. Background subtraction provides important cues for numerous applications in computer vision, for example, surveillance tracking or human pose estimation was aimed to be useful for our purposes.

We utilised Google's Deeplab[12] for the given task. DeepLab is a state-of-the-art semantic segmentation model designed and open-sourced by Google. We have used the DeepLab v3 model for this experiment.

The DeepLab model is broadly composed of two steps:

- Encoding phase: This phase aims to extract essential information from the image. This is done using a pre-trained Convolutional Neural Network.
- Decoding phase: The information extracted in the encoding phase is used here to reconstruct the output of appropriate dimensions

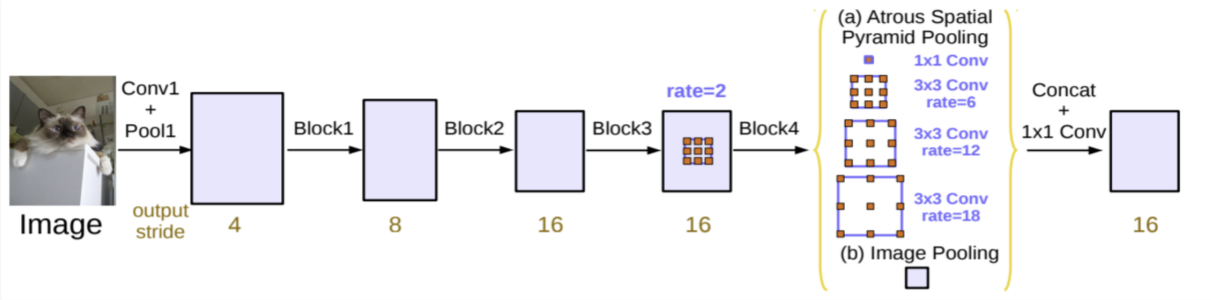


Fig 4.4.4: The Deeplab V3 architecture

In our original planned implementation, background subtraction was going to be in the place of the hand detection step initially. However, when we implemented hand detection and obtained the cropped images, we could eliminate the issue of having body parts present in the frame. The cropped image would only focus on the hand, and accordingly, skin segmentation would remove the rest of the background to a satisfiable extent. Thus, background segmentation was dropped and replaced by the hand detection step.

4.5 Model

The drawbacks in the previous approach were primarily attributed to overfitting, and the implemented model was chosen as a factor for improvement potentially. Hence, we have performed a model compatibility

study to identify a better fitting model while considering accuracy and training time.

The following models are selected for compatibility studies due to their proximity to the date of creation of the original model and also their speed of processing since we have time constraints:

1. DenseNet
2. Inception V3
3. SqueezeNet

The first of the three, DenseNet is a network architecture where each layer is directly connected to every other layer in a feed-forward fashion (within each dense block). On the large scale ILSVRC 2012 (ImageNet) dataset, DenseNet achieves similar accuracy as ResNet but using less than half the amount of parameters and roughly half the number of FLOPs.

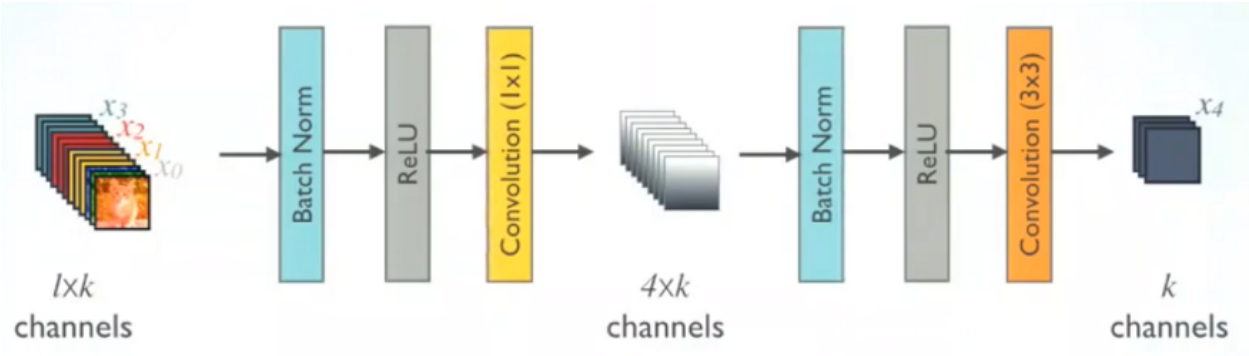


Fig 4.5.1: DenseNet B architecture

According to the authors of the network's paper[11], the inception of this network came from the given observation that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output.

The second model, the Inception V3 model by Google is the 3rd version in a series of Deep Learning Convolutional Architectures. Inception V3 was

trained for the ImageNet Large Visual Recognition Challenge where it was the first runner up.

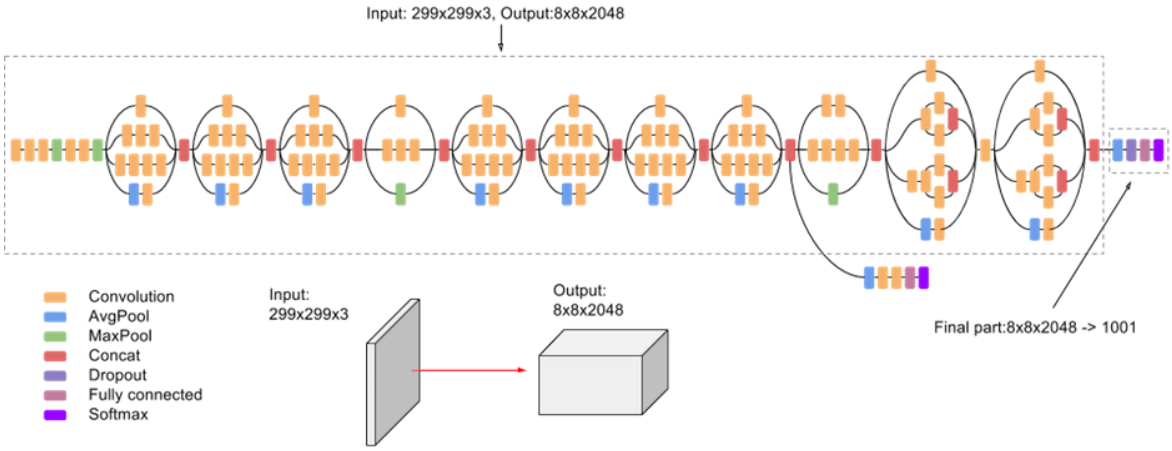


Fig 4.5.2: The Inception v3 architecture

Inception v3 mainly focuses on burning less computational power by modifying the previous Inception architectures. This idea was proposed in the paper *Rethinking the Inception Architecture for Computer Vision*[14], published in 2015. It was co-authored by Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, and Jonathon Shlens.

In comparison to VGGNet, Inception Networks (GoogLeNet/Inception v1) have proved to be more computationally efficient, both in terms of the number of parameters generated by the network and the economical cost incurred (memory and other resources).

If any changes are to be made to an Inception Network, care needs to be taken to make sure that the computational advantages aren't lost. Thus, the adaptation of an Inception network for different use cases turns out to be a problem due to the uncertainty of the new network's efficiency.

The third model, SqueezeNet is a smaller network that was designed as a more compact replacement for AlexNet. It has almost 50x fewer parameters than AlexNet, yet it performs 3x faster. This architecture was proposed by researchers at DeepScale, The University of California,

Berkeley, and Stanford University in the year 2016. It was first published in their paper[15].

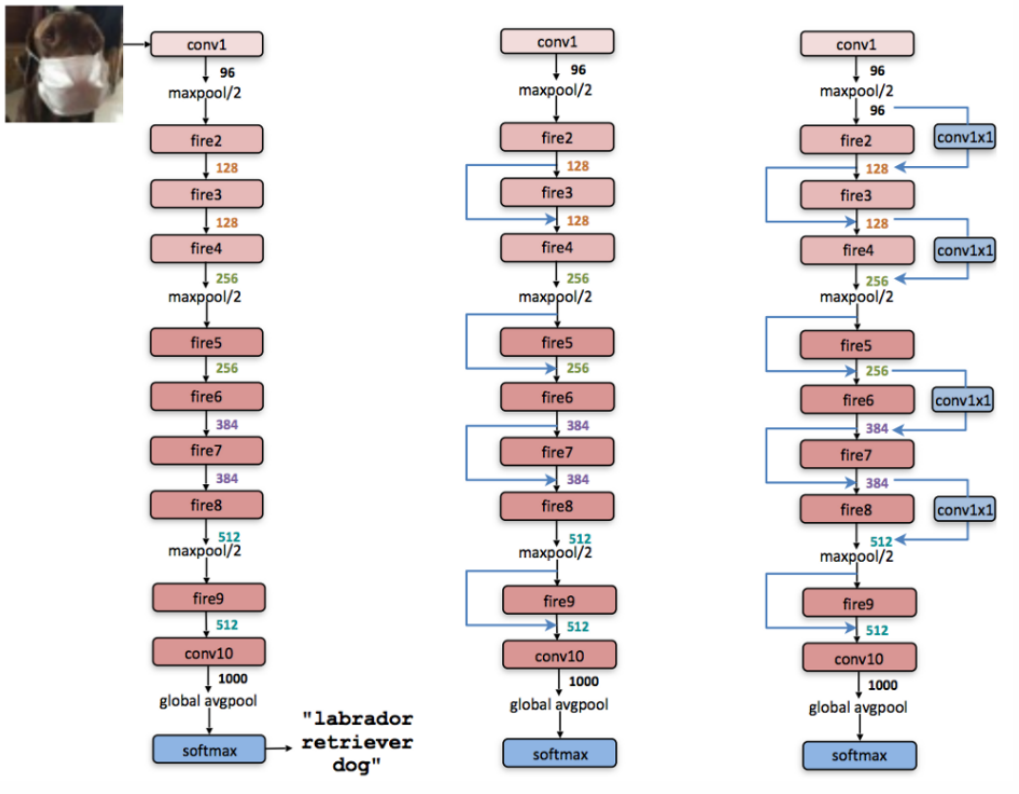


Fig 4.5.3 (from left to right): SqueezeNet, SqueezeNet with simple bypass, and SqueezeNet with complex bypass

Below are the key ideas behind SqueezeNet:

1. **Strategy One:** use 1 × 1 filters instead of 3 × 3
2. **Strategy Two:** decrease the number of input channels to 3 × 3 filters
3. **Strategy Three:** Downsample late in the network so that convolution layers have large activation maps

For comparing the models, two versions of each model were used - Plain models, and models with data augmentation on input datasets. Keras' data augmentation library was used for the aforementioned purpose. (6 models)

Dataset: ISL20C1200I was used. It is a subset of the ISL25C1200I dataset and contains 20 alphabets for the Indian Sign Language, along with 1200 images for each class. (80-20) split.
Number of epochs: 100 (standard for Transfer Learning models)
Batch size: 8
Checkpoints: performed if test accuracy increased with respect to previous epochs.

After performing this test, we observed the following results as shown below in Table 4.5.1:

Table 4.5.1:

Model	**Time for training (in hours)**	**Validation accuracy**
DenseNet	4	0.908
Inception v3	1.72	0.574
SqueezeNet	1.22	0.99
DenseNet (with dataset augmentation)	8.04	0.91
Inception v3 (with dataset augmentation)	6.67	0.54
SqueezeNet (with dataset augmentation)	5.83	0.98

We can say that the SqueezeNet architecture strongly outperforms its peers in both aspects -

- The accuracy provided (~ 0.98)
- Time for training (minimum)

Thus, it has been selected for utilisation.

4.6 Selection Method

For each 3-second clip, we have a certain number of frames. These frames may or may not give the same output when passed through our model since they may not have the same sign within the 3 seconds. In such a scenario, the alphabet must ultimately be selected given each frame's predicted output.
Each frame's output is computed through our model and stored collectively for the set of frames for a given video clip. This data now contains which alphabet was detected by our model in how many frames for this set. Given this data, we can apply a selection algorithm to choose the final output.

Two methods for this were proposed:
1. Max Occurrence Selection
2. Roulette Selection

The first approach, Max Occurrence Selection, is a basic method where the alphabet which has occurred the most number of times in the given set of frames is chosen as the final output.

The second approach, Roulette Selection weighs each alphabet's chances at being selected. Each alphabet is assigned a probability proportional to the number of frames that predicted that particular alphabet as its output when passed through the model. A random roulette selection is performed to get the result in this clip using these weighted probabilities assigned. This method weighs in favour of the heavily occurring alphabets while still giving the outlier alphabets a chance.

Both of these approaches combined with the three frame extraction were combined and observed with regards to their average execution time and average relative accuracies with respect to the base case, which is the method to extract all frames with maximum occurrence selection. The results obtained are shown in Table 4.6.1 as follows on the next page:

Table 4.6.1:

Approach	Selection	Avg. Time (secs)	Avg. Relative Accuracy
All Frames	Max	55.18	-
	Roulette	56.17	90.63%
Key Frame	Max	165.65	81.25%
	Roulette	165.97	76.88%
Random Sampling	Max	11.91	95.00%
	Roulette	12.01	83.75%

As observed earlier as well, random sampling was significantly faster and had a comparatively decent performance. Moreover, in most cases, max occurrence selection performed better than roulette selection with respect to the base case.

5. SOFTWARE & HARDWARE REQUIREMENTS

Concerning the Software Requirements of this project, it was done in its entirety using the programming language Python 3.8.6 and with the help of the following libraries:

1. **TensorFlow**: TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. Tensorflow is a symbolic math library based on dataflow and differentiable programming.

2. **Keras**: Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.

3. **Pillow**: Python Imaging Library is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.

4. **OpenCV**: OpenCV (Open Source Computer Vision Library) is an open-source library that includes several hundreds of computer vision algorithms. It is a library of programming functions mainly aimed at real-time computer vision.

5. **NumPy**: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Concerning the Hardware Requirements of this project, the training and testing processes of the model were done on the device with the following hardware configurations and hence can be treated as the minimum standard:

1. Intel i5 8250u @ 1.6 GHz
2. 20 GB DDR3 RAM
3. Intel inbuilt UHD 630 Graphics

6. RESULTS

A working implementation of the proposed architecture was created and tested a number of times. One such instance is presented here through some images.

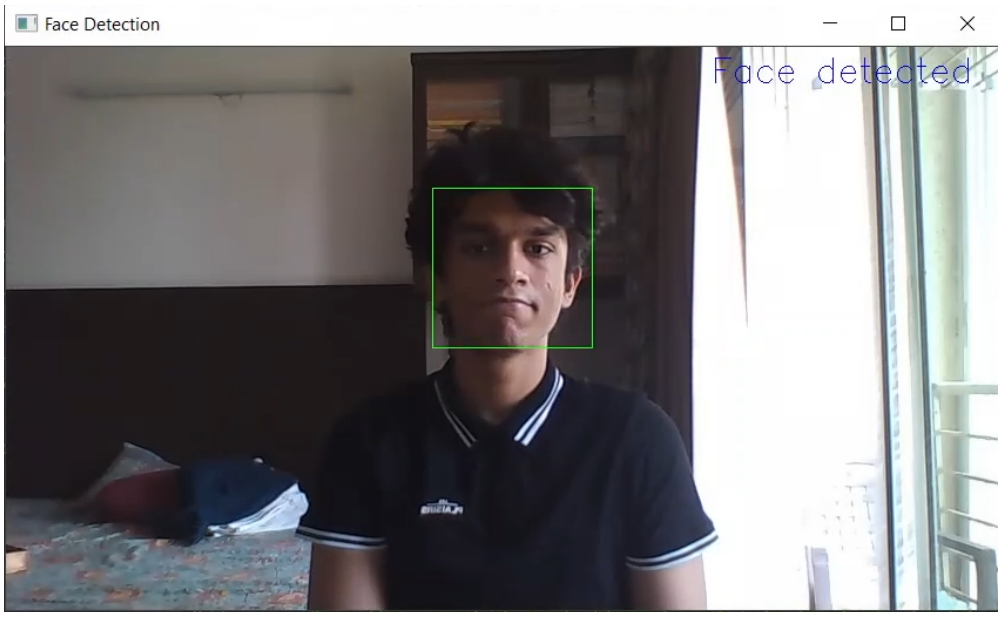


Fig 6.1: A screenshot of face detection phase, where a green bounding box is created over the face, and top right corner contains the text that a Face was detected

Once the face detection phase was completed, the webcam feed was recorded for a fixed number of seconds (10 seconds in this example). In this duration, the user made the symbols corresponding to the alphabet they wanted to sign. In this instance, the user is making the gesture for the letter ‘O’.

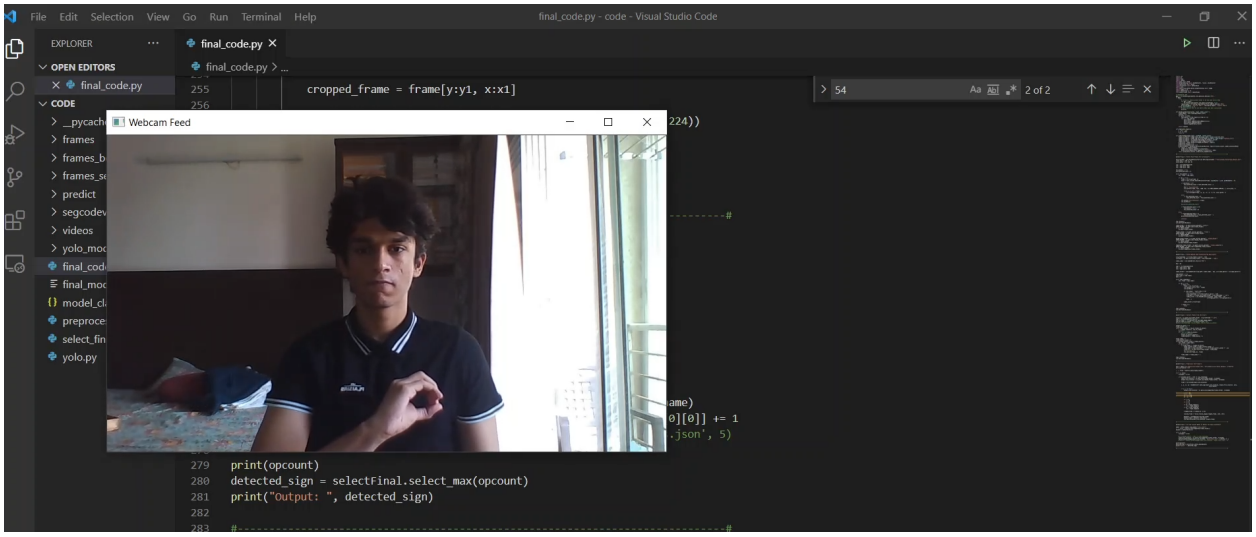


Fig 6.2: A screenshot of webcam recording phase, where the user is gesturing the letter ‘O’

After this, the frames were extracted from this recorded clip which went through the preprocessing techniques. The following three images represent the extracted frame, the bounded box frame, and the skin segmented frame corresponding to the same moment.

Fig 6.3: An extracted frame from the saved clip

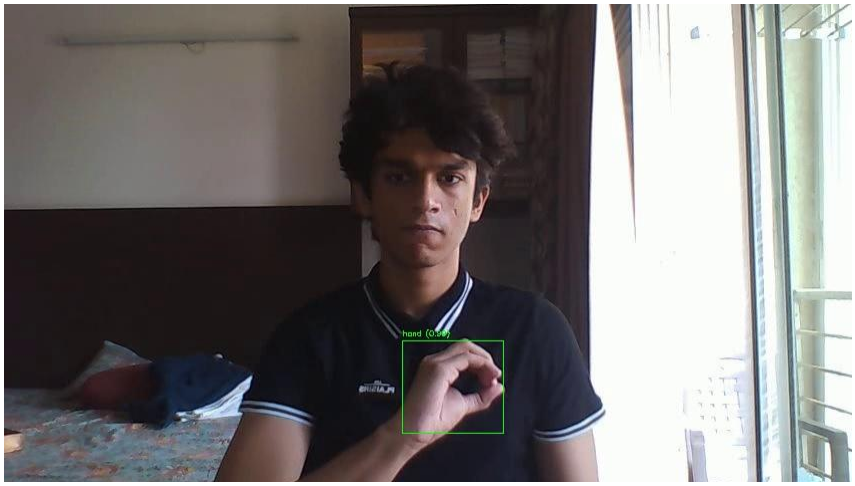


Fig 6.4: The frame with a bounding box around the detected hand

Fig 6.5: The frame after skin segmentation

37 frames in total were extracted from the 7 second clip. During preprocessing, hands were detected in 34 frames out of 37, and hence, these frames were qualified for resizing and skin segmentation.

All 34 frames gave the output as ‘O’ at the end of the execution, which was the correct output.

```
service.cc:176]   StreamExecutor device (0): Host, Default Version
2021-05-04 10:11:05.153366: I tensorflow/core/common_runtime/gpu/
gpu_device.cc:1257] Device interconnect StreamExecutor with strength 1
edge matrix:
2021-05-04 10:11:05.153378: I tensorflow/core/common_runtime/gpu/
gpu_device.cc:1263]
defaultdict(<class 'int'>, {'0': 34})
Output:  0
```

Fig 6.6: The final output for the given data

Furthermore, the execution time per phase was also recorded for this run.

Phase	Time Taken
Face Detection	5 seconds (setup); 3 seconds (runtime)
Webcam Feed Recording	6 seconds (setup); 7 seconds (runtime)
Frame Extraction	1 second
Preprocessing	16 seconds
Output Calculation	3 seconds

Furthermore, it is also worth noting that the model roughly had a 99% validation accuracy.

7. CURRENT LIMITATIONS

The first and the most crucial drawback of this implementation is that the execution time of a single run is higher than ideal. It took approximately 41 seconds from start till end and a response time of 20 seconds (i.e., after the user has finished recording the video), as given in the example run. This is ideally greater than we would have liked. The two primary concerns with this is in the setup part of face detection and webcam feed recording phases, and in the preprocessing phase.

Additionally, the current implementation is a single run process, which means it runs once from start to end and then stops. However, in a practical scenario, this program should run continuously. That would also create the need for a cleanup process to manage the storage resources of the system running this program.

8. FUTURE SCOPE

Improvements to the poor execution time can be performed by making the setup phases of the program a one-time thing to be done at the start of its execution. As for the Preprocessing phase, this can be solved by using libraries like pillow-simd, which can make the preprocessing steps to be executed parallely for multiple images, thereby significantly reducing the execution time and achieving a real-time or near-real-time execution.

A continuously running implementation with a proper cleanup process will be an easy addition to make.

Moreover, the whole project can also be made angle-independent. This would open the potential of implementing this algorithm with a CCTV camera.

Finally, as mentioned earlier, the idea behind this project is to use these symbols as not singular letters, but as gestures signifying phrases, terms, or messages. Hence, the current dataset, which contains Alphabet notational hand gestures can be converted seamlessly to Ideographic notational hand gestures, thereby achieving one of the goals of this project.

9. CONCLUSION

In this project, we primarily worked towards improving the previous implementation of the Vision-based Sign Language Translator, and solved the majority of the issues present in the earlier version. The proposed algorithm, albeit a bit uneconomical, increases the adaptability of a hand gesture recognition system in near real-time.

By implementing hand/skin segmentation and face detection, the technique aims to reduce the time complexity and also provide better results under different degrees of scene background complexity and illumination conditions.

This project can be used as a blueprint for achieving a more efficient algorithm which is implemented in real-time expeditiously and optimistically, helping the community of the hearing and speaking impaired.

10. BIBLIOGRAPHY

1. T. Pan, L. Lo, C. Yeh, J. Li, H. Liu and M. Hu, "Real-Time Sign Language Recognition in Complex Background Scene Based on a Hierarchical Clustering Classification Method," 2016 IEEE Second International Conference on Multimedia Big Data (BigMM), Taipei, Taiwan, 2016, pp. 64-67, DOI: 10.1109/BigMM.2016.44.
2. M. Kim, L. Chau and W. Siu, "Keyframe selection for motion capture using motion activity analysis," 2012 IEEE International Symposium on Circuits and Systems (ISCAS), Seoul, Korea (South), 2012, pp. 612-615, DOI: 10.1109/ISCAS.2012.6272106.
3. S. C.J. and L. A., "Signet: A Deep Learning based Indian Sign Language Recognition System," 2019 International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 2019, pp. 0596-0600, DOI: 10.1109/ICCSP.2019.8698006.
4. Thad Starner and Alex Pentland, “Real-Time American Sign Language Recognition from Video using Hidden Markov Models”, 1995.
5. Washef Ahmed, Kunal Chanda, Soma Mitra, “Vision-based Hand Gesture Recognition using Dynamic Time Warping for Indian Sign Language”, 2016.
6. J. Vishnu, Phanesh R., “Vision-Based Sign Language Translator,” VNIT, Nagpur, 2019.
7. P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 2001, pp. I-I, doi: 10.1109/CVPR.2001.990517.
8. Katna tool documentation, PyPI.
9. Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." arXiv preprint arXiv:1804.02767 (2018).
10. Hand Database - CMU Panoptic Dataset
11. Seema Kolkur, et al, “Human Skin Detection Using RGB, HSV and YCbCr Color Models,” 2017.
12. Beeren Sahu, “The evolution of Deeplab for Semantic Segmentation”, Towards Data Science.

13. Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, “Densely Connected Convolutional Networks,” 2016 (revised 2018).
14. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, “Rethinking the Inception Architecture for Computer Vision,” 2015.
15. Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size,” 2016.