NORTH SOUTH UNIVERSITY

Department of Electrical And Computer Engineering

CSE499
Senior Project Design

Sign Language Translation Using Machine Learning And TensorFlow

Submitted By
Atif Karim 1921398042
Farida Yeasmin 1911653042
Rezowan Khan Rahat 1922050042
Malisha Rahman Tonni 1921946042

Supervised By
Dr. Shafin Rahman
PhD, Assistant Professor,
Electrical and Computer Engineering,
North South University.

NORTH SOUTH UNIVERSITY

Sign Language Translation Using Machine Learning And TensorFlow

A DISSERTION
SUBMITTED TO THE DEPARTMENT OF
ELECTRICAL AND COMPUTER ENGINEERING
OF NORTH SOUTH UNIVERSITY
IN THE PARTIAL FULFILMENT OT THE REQUIREMENTS
FOR THE DEGREE OF BACHELOR OF SCIENCE IN
COMPUTER SCIENCE AND ENGINEERING

Date
25th January 2023, Wednesday

Declaration

It is hereby acknowledged that:

- No illegitimate procedure has been practiced during the preparation of this document.
- This document does not contain any previously published material without proper citation.
- This document represents our own accomplishments while being Undergraduate Students in the **North South University**

Sincerely,

Student 1: Farida Yeasmin 1911653042, Signature	**Student 2:** Atif Karim 1921398042, Signature
Student 3: Rezowan Khan Rahat 1922050042, Signature	**Student 4:** Malisha Rahman Tonni 1921946042, Signature

Approval

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation.

DR. Shafin Rahman

PhD, Assistant Professor,
Electrical and Computer Engineering North South University, Dhaka, Bangladesh

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation.

DR. Rajesh Palit

Ph.D Professor Chair,
Electrical and Computer Engineering North South University, Dhaka, Bangladesh

Abstract

Sign Language play a crucial role in nonverbal communication and essential to daily living. Sign Language Recognition, which has been researched for many years, is a breakthrough for assisting deaf-mute people. Unfortunately, each study has its own limitations and cannot be used commercially. Some studies have proven to be successful in recognizing sign language, but commercialization is prohibitively expensive.With the aid of a hand gesture detection system, we have access to a novel, comfortable, and user-friendly method of interacting with computers that is more suited to human needs. We intend to redefine the Sign Language Translator using deep learning. Our aim is to make an improved model that will come close to or even improve on current models. Our first task was to understand existing projects and models, which is why we were comparing our own findings with the findings of current research papers. This will inspire us to take the best parts of the existing models and put our own spin on them in the future. Even though we had a very rocky start, getting very poor results compared to our target, we managed to increase our accuracy by learning more and evolving our findings and understanding of the topic. We concluded with improved results and are currently at half way point in our project. In the next half, we intend to make the model using CNN and also make a recognizer to recognize the signs.

Keywords: Deep learning, Convolutional Neural Network (CNN), Artificial Neural Network (ANN), Python, dataset, data augmentation, InceptionV3, MobileNet, DenseNet, VGG16, transfer learning, hand gesture recognition, Sign Language Recognition (SLR).

Contents

List of Figures

List of Tables

Acknowledgements

We would like to express my sincere gratitude to all those who have contributed to this dissertation titled “Sign Language Translator Using Deep Learning”.

First and foremost, I would like to thank my supervisor, Dr Shafin Rahman, for their guidance, support, and encouragement throughout this journey. Their invaluable insights and constructive feedback have greatly enhanced the quality of my work.

Our sincere thanks also go to North South University, Dhaka, Bangladesh for providing an opportunity in our curriculum which enabled us to have an industrial level experience as part of our academics.

Finally, we would like to thank our family and loved ones for their unconditional love and support. Without their constant encouragement and motivation, we would not have been able to complete this dissertation.

Thank you all.

1 Introduction

Deep Learning is one of the most exciting fields of study today. It is also one of the most discussed in the tech community. Many people know what deep learning is, but few understand how it works or what its applications are. By talking about these things, we can let everyone know about this cutting-edge technology and all the ways it can be used.

Deep learning is a method of artificial intelligence that allows computers to learn and adapt through the use of neural networks, without requiring explicit programming for each task. Essentially, it allows computers to make decisions and take action on their own without being told to do so. This area of research is exciting because it could lead to computers that can act on their own, do tasks like humans, and talk to the outside world. In the past decade, deep learning has become more accessible and practical, allowing for rapid advancement in this emerging field.

Transfer learning is a machine learning technique in which a model that has been trained on one task is repurposed for a second, related task. It means using what you learned while solving one problem to solve another problem that is different but related. There are several ways to apply transfer learning, including fine-tuning, where the pre-trained model is adjusted to the specific requirements of the new task, and feature extraction, where the pre-trained model is used as a fixed feature extractor, and a new classifier is trained on top of the extracted features. Transfer learning has been used a lot in many different fields, such as natural language processing, computer vision, and speech recognition. It works best when there isn't enough labeled data for the task at hand.

Sign language is a rich and expressive way for deaf and hard-of-hearing individuals to communicate, but it can be difficult for those who are not proficient in sign language to understand. This is where a sign language translator can be helpful. Using neural networks and machine learning, it is possible to build a system that can recognize and translate sign language gestures into text or speech, making it easier for hearing individuals to communicate with deaf individuals.

For CSE499A and B, our goal was to design and build a sign language translator that can accurately recognize and translate a wide range of sign language gestures into text. We will use a combination of neural networks and machine learning techniques to train the system to identify and classify different gestures and then translate them into a form that everyone can understand. Our goal is to create a tool that can improve communication, facilitate greater understanding between deaf and hearing individuals,

and shed light on a minority of people who do not have their own voice.

Implementation: Many people think that deep learning is only used by tech companies like Google or Facebook. However, consumers are actually using deep learning every day. The most common way deep learning shows up in our lives is in apps and digital services that help us organize and manage our lives. For example, you may use your calendar every day, which is a great use of deep learning. Other common uses include text prediction and promoting product sales by recommending related items to customers browsing in retail stores. The possibilities are limitless when a field as complex as deep learning can address such a wide range of needs and issues. As exciting as Deep Learning is, developing ethical solutions for its uses is difficult work on that needs to happen now more than ever before. Consumers are already using deep learning every day, but no laws currently govern how developers use this technology responsibly. That needs to change so we can effectively manage the many ethical concerns associated with this exciting area of research.

1.1 Background Studies

Using Deep Learning for image recognition has gone to the next level already. Many types of solutions are available to solve this problem. ZSL is the most successful approach for recognizing SL. The best feature of Zero-Shot Learning is that it can work for both seen and unseen cases, whereas the traditional deep learning approach works mostly for seen cases. Recognizing gestures from different hand sizes, skin tones, camera angles, lighting, and backgrounds is hard. A simple yet efficient algorithm is key to getting the most accurate results. Nevertheless, nothing has been decided yet. [1]

According to the study, there were two methods used to train the model on the temporal and spatial data, and both varied in how they fed the RNN's temporal feature training inputs. The spatial and temporal information for each frame was extracted using the inception model (CNN) and RNN. CNN made predictions for each frame of each video, and those predictions were used to show what each video was about. The RNN received this as input. The background parts of the body other than the hand were taken out of each video frame that was matched to a gesture so that a grayscale image of

the hands could be made. This kept the model from learning information about colors. [2]

We have also found a study on Ethiopian Sign Language (ESL). Principal Component Analysis (PCA) and the Gabor Filter (GF) have been used to extract features from digital images of hand gestures. The Artificial Neural Network (ANN) is used to recognize English as a Second Language (ESL) from the extracted features and translate them into Amharic voice. The experimental results show that the system has produced a recognition rate of 98.53% [1]. It proves that ANN can be a very good model for our system if we use it efficiently. In another paper, they made a visual hierarchy transcription network to capture the spatial appearance and temporal motion cues of sign video on multiple levels. In the meantime, they use a lexical prediction network to get useful information about context from the output predictions. We have gone through these studies thoroughly. [3]

There are different data-reading techniques. A sensor called a Leap Motion controller (Figure 1) transforms signals from hand movements into commands for computers. Three infrared LEDs and two IR cameras make up the device. The reflected data of 300 frames is produced by the camera every second, and the IR light signal is produced by the LED. Through a USB cable, these signals are transmitted to the computer for additional processing. In transforming, Indian Sign Language to text, research by P. Karthick et al. used this device for data reading.

As seen in Figure 1.1, Microsoft's Kinect is a motion sensor for the Xbox 360 gaming system [4]. It is made up of a multi-array microphone, a depth sensor, and an RGB camera. Speech and facial expressions are recognized. [5] This technique recognizes hand gesture signals using a separate sensor. The hand motion signal is an analog signal. The analog signal is transformed into digital form using an ADC. It is made up of an accelerometer and a flex sensor. Bend signals are discovered using a flex sensor. [6] Images are captured using a web camera in this technique. Following that, picture segmentation is complete. Extracted features from the input image include palms and fingers. Different hand motions—half-open, closed, and semi-closed—were picked up. Data is recorded as a vector, which is then utilized to recognize alphabets. [7]

2 Literature Review

Using machine learning for image recognition has gone to the next level already. Many types of solutions are available to solve this problem. ZSL is the most successful approach for recognizing SL. The best feature of Zero-Shot Learning is that it can work for both seen and unseen cases, whereas the traditional machine learning approach works mostly for seen cases. Recognizing gestures from various sizes of hands, skin tones, camera angles, lighting, and backgrounds is a tough challenge. In order to get the most accurate results, a simple yet efficient algorithm is key. Though it has not been decided yet. One significant feature of this paper is that they have separated their dataset into two different sections and named those seen and unseen classes. Then they used their algorithm and model for both seen and unseen cases. However, they were not able to achieve the same level of accuracy in unseen cases as they did in seen cases. Yet, using zero-shot learning for sign language recognition was such a novel step. [1]

The publication "Better Sign Language Translation with STMC-Transformer" used a novel STMC-Transformer model for video-to-text translation that outperformed GT glosses translation. The first successful application of Transformers to SLT achieved state-of-the-art results in both gloss-to-text and video-to-text translation on the PHOENIX-Weather 2014T and ASLG- PC12 datasets. In "SIGN LANGUAGE RECOGNITION USING NEURAL NETWORK," they use the CNN approach to translate the gestures from the video into simple English words and then construct sentences out of each term. [2]

Bangla Sign	**Precision**	**Recall**	**True Negative**	**Accuracy**
0	100%	100%	0%	***100%***
1	100%	100%	0%	***100%***
2	100%	100%	0%	***100%***
3	100%	100%	0%	***98%***
4	100%	100%	0%	***100%***
5	100%	100%	0%	***100%***
6	100%	100%	0%	***100%***
7	100%	100%	0%	***100%***
8	100%	100%	0%	***100%***
9	100%	100%	0%	***100%***

Table 2.1: Bangla Sign Language Accuracy with different datasets

2.0.1 Leap Motion Controller

A sensor called a Leap Motion controller detects hand movements and translates those signals into computer commands. Three infrared LEDs and two IR cameras make up the device. 300 frames of reflected data are produced by the camera every second and the IR light signal is by the LED. Through a USB cable, these signals are transmitted to the computer for additional processing. Leigh Leap Motion Controller was used by Ellen Potter et al.to recognize Australian sign language. To detect hand motion and translate it into computer commands, a leap motion controller is employed. The training of symbols is done using an artificial neural network. That system's drawback was its lack of fidelity and accuracy.

Figure 2.1: Leap Sensor: Used to interpret motion made by users

Human action recognition has drawn more and more attention in the pattern recognition and computer vision areas as a result of the growth of several interactive applications in human-computer interaction. An essential component of recognizing human behavior is dynamic hand gesture recognition. Recent advancements in depth sensors, such as the Leap Motion Controller (LMC), which offers three-dimensional (3-D) depth data of the scene, have greatly aided in the identification of 3-D hand gestures and object segmentation.The depth data, which includes palm direction, fingertips, palm center position, and other pertinent points, is the LMC's output. Therefore, obtaining this information doesn't need any additional computing labor.Additionally, compared to other depth sensors, LMC has a higher localization precision (which is about 0.2 mm). Recently, researchers used LMC to recognize hand gestures. [8] For example, Marin et al. used

an LMC and Kinect sensor to recognize American Sign Language (ASL), and Xu et al. used an LMC to recognize ten simple dynamic hand gestures. The LMC outputs depth data frames that consist of finger position, hand position, scaling data, frame timestamp, rotation, and so on. [3]

Leap Motion has been widely used by researchers in recent years for hand gesture recognition, including the recognition of American Sign Language (ASL), Indian sign language (ISL), and Arabic sign language. All of them achieved high gesture recognition accuracy, with one study achieving a recognition accuracy of 90% from a data set of 45 gestures.The use of the Leap Motion Controller and the introduction of machine learning have raised the bar for human-machine interaction.The Leap Motion controller's design is actually fairly straightforward. [9] Three infrared LEDs and two cameras make up the device's main components. They follow infrared light at 850 nanometers in wavelength, which is not part of the visible spectrum. The Leap Motion Application Programming Interface (API) enables developers to gain access to, integrate Leap Motion's capabilities with, and create new applications that perform functions including obtaining user hand location data and gesture recognition.This study includes 10,000 steps of training and the final 400 training results. The average accuracy rate can basically reach more than 98% at the end of the training. [10]

2.0.2 Kinect Sensor

Microsoft's Kinect motion sensor is used with the Xbox 360 gaming system. It consists of a multi-array microphone, a depth sensor, and an RGB camera. Speech and facial expression are recognized. Microsoft Kinect was utilized by Cao Dong et al.to identify American Sign Language. To detect the ASL alphabet, a depth camera is employed as a Kinect sensor. The feature extraction process uses a distance adaptive technique. For classification purposes, support vector machines and RF classifier algorithms are used. An ANN network was used to train the data. The system had a 90% accuracy rate. Kinect was utilized by Uan Yao et al.to recognize hand gestures. It first recognizes hand movement before matching the counter model. The second assignment involved finding a multicolored glove and identifying various color zones.A theirGaussian color model was used to train the data, and a per-pixel classifier was used to classify the data. One flaw in this system is its lack of accuracy.[11]

Recent improvements to 3D depth cameras and Microsoft Kinect sensors have given multimedia computing a lot of new possibilities. Kinect was created to fundamentally alter how people play video games and enjoy entertainment.The Kinect sensor features a variety of cutting-edge sensing components. Because they are able to monitor

the entire body in three dimensions, recognize faces, and hear conversations, its depth sensor, color camera, and four-microphone array stand out. The Kinect sensor has countless uses in both established and emerging sectors.

KINECT is an RGB-D sensor that produces images with synchronized color and depth. A 3-D human motion capture algorithm lets people play games without touching a controller. Recent research from the Computer Vision Society suggests that the Kinect's ability to measure depth may now be used for much more than just games and at a much lower cost than traditional 3-D cameras.An IR projector, an IR camera, and a color

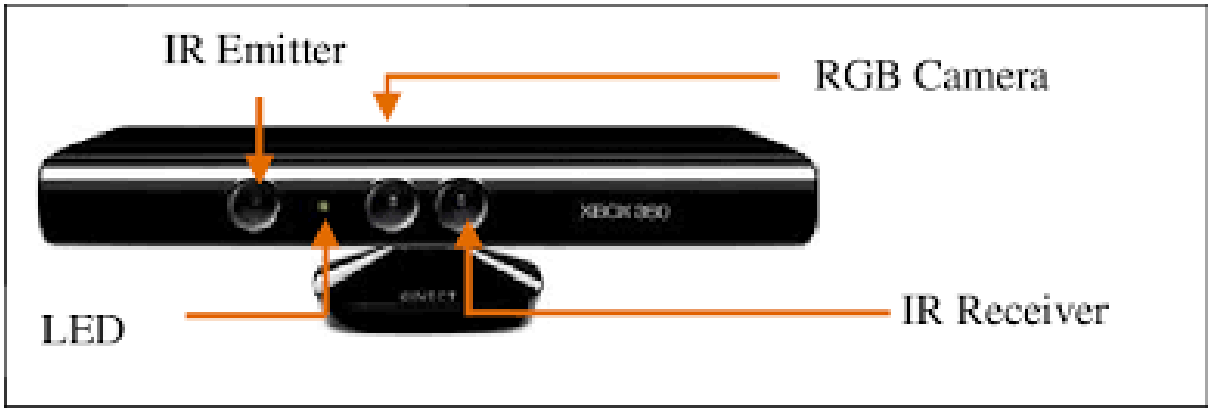

Figure 2.2: Kinect Sensor: Most commercial and widely used sensor to interpret motion

cameras make up a Kinect sensor. The IR projector and IR camera make up the depth sensor. The IR camera records the IR speckles that are reflected, while the IR projector puts an IR speckle dot pattern into the 3D scene. Therefore, Kinect is an example of a structured light depth sensor. [4]

Different sensors are used by data gloves to detect hand gesture signals. An analog signal is used for hand gestures. Analog-to-digital signal conversion is accomplished using an ADC. It has an accelerometer and a flex sensor. Bend signal detection uses a flex sensor. Anarbasi Rajamohan et al. used a data glove-based method to figure out how to understand American Sign Language. Flex sensors, accelerometers, and tactile sensors make up the system. This sensor converts hand gestures into code and is used to detect them. That system had a 90% accuracy rate.

The data glove has many sensor units that can tell the computer where the user's hand and fingers are and how they are moving. Also, some gloves mimic the feeling of touch by being able to pick up on very precise finger bends or even by giving the user feedback through touch. Data gloves are accessible for all degrees of hand freedom, which is their finest perk.[12]Over the past few years, there has been a noticeable rise in the number of academics who place a higher value on data protection. But there are still some problems with this technology, such as the high cost of most data gloves (like CyberGlove,

HumanGlove, and 5DT Data Glove), the limited accuracy of calculations, and the short time this technology can be used.

A data glove is a sort of hardware that measures the bending angles of figure joints and hand posture and transmits the information to other systems. The main purpose of data gloves is to gauge hand posture. Data gloves can be broadly categorized. [13]

Figure 2.3: Data Gloves: another way to interpreter motion in 3D Space.

into three categories: visual graphics-based gloves, mechanical gloves, and gloves made of optical fiber. When it comes to gloves with visual graphics, the key figure positions can have LEDs or color graphics. The graphics algorithm identifies hand posture. [14] The algorithm is more demanding, with poor accuracy and real-time performance for the technique. In a mechanical glove, sensors are placed at each joint to track movement. This makes the design complicated and makes it hard to use. Since light sources and receiving devices are put at both ends of the optical fiber in the glove, optical energy loss when the fingers bend will be calculated.The design of data gloves is based on ways to fix bad hand postures, such as measuring the angle of the posture in real time, updating the posture, and resolving the posture. [5]

2.0.3 Vision-Based

Images are captured using a webcam in the vision-based method. Following that, picture segmentation is complete. Extracted features from the input image include the palm and finger. Different hand motions—half-open, closed, and semi-closed—were picked up.

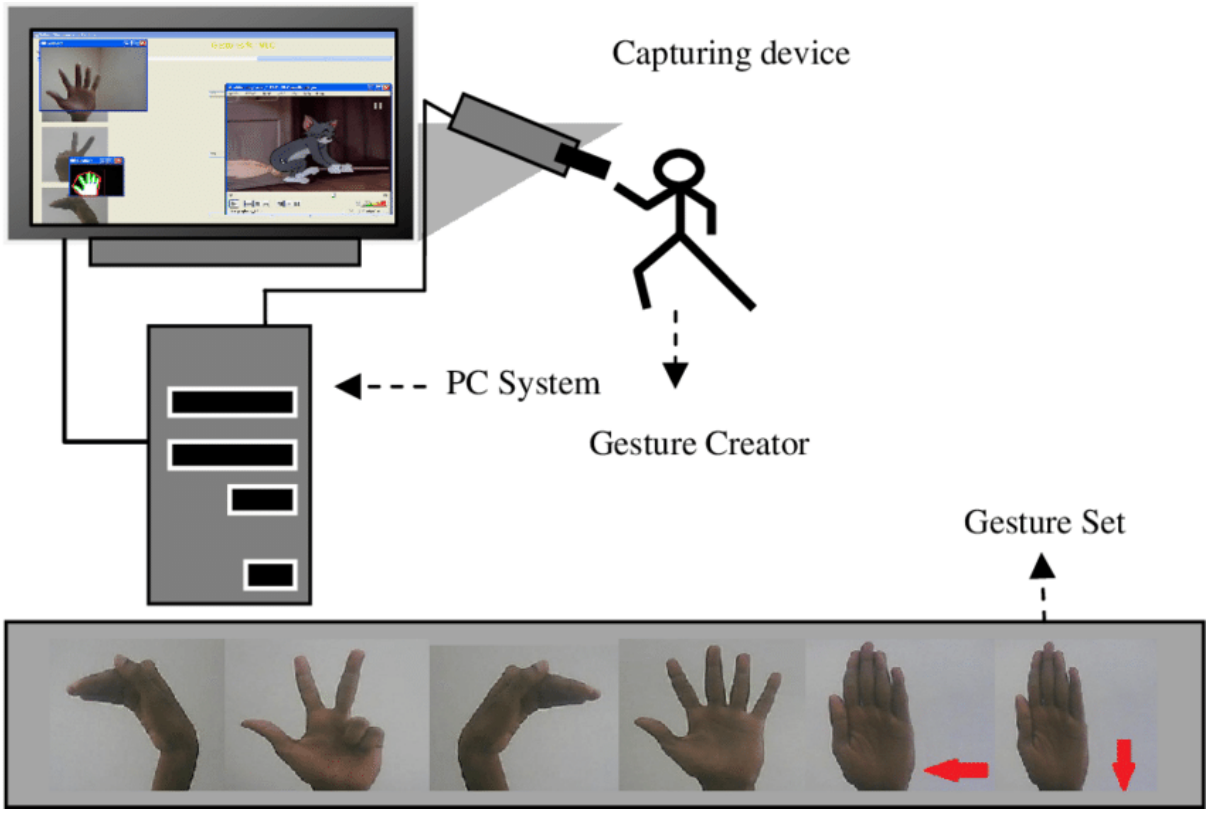

Figure 2.4: Vision Based: Illustration showing how vision based interpretation works

Data is recorded in a vector, which is then utilized to recognize alphabets.Portuguese language recognition was done using a vision-based approach by Paulo Trigueiros et al.Hand gestures were recorded in real-time for their deployment. [15] The SVM algorithm is employed for classification. Vowel recognition accuracy in this system is 99.4%, while consonant recognition accuracy is 99.6%.Usually, when an image is taken for an experiment, head movement is added to images of the hands. Cameras are positioned above signers to eliminate this overlap between hand and head movements. However, this body and face gesture was lost. A smaller hand gesture was employed by Nilsen et al. to facilitate quick recognition. [6] We go over our Gantt chart and explain how we intend to follow the timeline we have set and how we are doing in terms of the chart. We will also elaborate on how we achieved these phases incrementally. We will also be talking about our documentation process and how we are managing task distribution on a week-by-week basis. We will also go over how we are documenting all our progress, which will play a big role in our final project.

2.0.4 Gantt Chart

Our Gantt chart is shown above. We made it, have been following it, and will continue to do so for the rest of the semester. We are currently in week 4 and going over to week 5, where our team member Atif has already started prototyping the technology we intend to showcase. Farida, a member of our group, is working on the prototype and doing research at the same time. She will be the link between the research team and the development team. Rahat and Malisha on our team are in charge of spending all of their time researching and making improvements to our prototype. They learn from what they read and look at. Since our semester is 12 weeks, we have decided to divide them in a way so that we can give importance to the topics that need it the most, which is how we ended up with this. Even though it shows that we will be spending weeks 11 and 12 on our last and main thesis paper, we are already making an informal compilation of notes and findings that will guide us in constructing our thesis paper to the max.

2.0.5 Weekly Reports and Findings

As per our curriculum, we have to submit a weekly report every week to show the progress that we have made. These are also very helpful in constructing our thesis paper when the term comes to an end. We also have once-a-week face-to-face sessions with sir to convey what we have been up to for the past week. Our supervisor quizzes us to see the authenticity of our work and our knowledge. This is also the opportunity to get a lot of our own problems settled. But if the problem does seem too large to tackle in that time, we are also encouraged to reach out to him during his allotted office hours.

3 Models and Algorithms

At first, we analyzed our project. We mainly tried to find the answer to the following queries- what are we going to do, how will we do it, what do we need to complete it, and who will complete what task? Ask if it is a group project and when we will meet for all the work. We believe that if we find answers to these questions, we can organize our project and work accordingly.

Research Papers: Our project analysis begins with reading papers, journals, and articles on related projects. We have read a lot of documents from famous sites like Research Gate, IEEE, Springer, and many others. These papers not only provide us with proper knowledge but also introduce us to advanced and updated technologies. It also helped to understand the writing method of a report.

Existing Solutions: We are not the first to work on this project. Few other scholars have worked on this in previous years and developed exciting and practical solutions. These existing solutions helped us know their solutions' pros and cons. We also find what we should do and what we should not do. What drawbacks do these solutions contain, and how will we overcome them?

Design: We followed a sequential approach to design and build the project. Finding the perfect and effective Neural Network was our first and hardest challenge. Then we start looking for datasets. After seeing the datasets, we started building the system's models. We have worked with four different datasets of 2 other languages. InceptionV3, DenseNet, MobileNet, and VGG16 are four models we have constructed.

Build: Building the system was a layer-based approach. We designed the first layer, then another, and another until we got a significant success rate. We got stuck many times. Since we have worked with large datasets and complex architectural models, we were often caught in loops that were very hard-to-solve loops. Building the prototype was even more challenging. With proper patience and hard work, we solved every problem.

Test: After building the model, we tested our models. We get different accuracy results based on other models and diverse datasets. Our models provided 15% to 80% accuracy after training with the trained generator, validation generator, and 10-20 epochs. Our first dataset got the highest validation accuracy at 80%; the second dataset had an accuracy of 40%; the third dataset had a 58% accuracy, and the fourth dataset had a 75% testing accuracy.

3.1 Models

3.1.1 K-nearest neighbor (KNN)

The k-nearest neighbor's algorithm, also called KNN or k-NN, is a supervised learning classifier that uses proximity to classify or make predictions about how a single data point should be grouped. Based on the supervised learning method, it is one of the simplest machine learning algorithms. It can be used for both classification and regression problems, but it is most often used for classification because it is based on the idea that similar points can be found close to each other. A KNN model calculates similarity using a graph's distance between two points. The greater the distance between the points, the less similar they are.

The K-NN algorithm assumes that the new case and the existing case are similar. It then puts the new case into the category that is most like the existing categories, stores all the existing data, and adds a new data point based on how similar the new case is to the existing cases.[16] This means that the K-NN method can be used to quickly and accurately put new data into the right category. It's also important to note that the KNN algorithm belongs to a family of models called "lazy learning," which means that it doesn't go through a training stage but instead just stores a training dataset. This also means that all the math is done when a prediction or classification is made. It is also called an "instance-based" or "memory-based" learning method because it stores most of its training data in memory. The goal of the k-nearest neighbor algorithm is to find the query point's closest neighbors so that we can give that point a class label. KNN only requires a k-value, which defines how many neighbors will be checked to determine the classification of a specific query point, and a distance metric, which is low when compared to other machine learning algorithms. It is one of the first classifiers that a new data scientist will learn, and as new training samples are added, the algorithm adjusts to account for any new data since all training data is stored in memory. On the other hand, KNN is a lazy algorithm; in comparison to other classifiers, it uses more memory and data storage. Both in terms of time and money, this can be expensive. With more memory and storage, business costs will go up, and processing more data may take longer. K-Nearest Neighbors is one of the simplest machine learning algorithms. Despite how straightforward KNN is conceptual, it is a strong algorithm that provides a reasonable amount of accuracy for most situations.

3.1.2 Convolutional Neural Network (CNN)

For processing organized arrays of data, such as portrayals, a deep learning neural network called a "convolutional neural network," or CNN was developed. It can take in an input image, assign importance (learnable weights and biases) to various aspects and objects in the image, and be able to distinguish one from the other. [17] This does a great job of finding things like lines, gradients, circles, and even eyes and faces in the image. It doesn't require any pre-processing and may be used immediately for an underexposed image. A feed-forward neural network with 20 or fewer layers is what it is. It can tell the difference between human faces with 25 layers and recognize handwritten numbers with three or four convolutional layers. CNN has an input layer consisting of a matrix of images. Weights connect to the input layer, and the output layer connects to weights. In order to perform a number of tasks, including media reconstruction, picture and video identification, image inspection and classification, recommendation systems, natural language processing, and other things, this discipline aims to provide robots the ability to see the environment similarly to humans. [18] CNN models come in a variety of varieties, including LeNet, AlexNet, ResNet, GoogleNet, MobileNet, and VGG. CNN has the following applications:

- Decoding Facial Recognition.
- Understanding Climate.
- Collecting Historic and Environmental Elements

CNN is extremely effective at picture recognition tasks. We can develop a model that can accurately recognize various hand gestures by training a CNN on a sizable collection of photos. CNN may be used in a variety of ways to recognize hand gestures. One approach is to use a model that has previously been trained on a huge image dataset. This model can be improved so that it works well with a new set of hand movements. The training of a CNN using a dataset of hand gestures is also possible. This method can produce accurate results, but it is frequently more challenging to implement. That's why we have used CNN as our main architectural design. We have used multiple models from CNN. [19]

CNN has been widely used for sign language detection. Incher used CNN and the SIFT (Scale-Invariant Feature Transform) to detect 51 Bengali letters. They have implemented ROI (regions of interest) for better detection. Features are extracted by SIFT (Scale-Invariant Feature Transform). They have also used K-means clustering and the Bag of Features (Bof) [7]. The accuracy of CNN changes drastically when shifting is used and when it is not. After splitting the dataset, it needs to be trained and tested for better accuracy. For unexpected poor result, both the algorithm and dataset needs to be inspected. More training gives more accuracy. The accuracy of detecting Bengali sign language is shown in the table. [8]

Recognition Rates						
Classifier	Testing With Same Dataset	Testing With Dif-ferent Dataset	Testing With Same Dataset	Testing With Dif-ferent Dataset	Testing With Same Dataset	Testing With Same Dataset
MDC	57.71	52.46	58.22	54.81	59.95	55.49
Adabost	65.68	60.36	66.47	61.19	67.81	62.91
ANN	75.77	66.88	76.54	68.8	78.45	73.63
Deep ANN	83.98	74.89	84.75	77.01	85.74	81.84
Prposed CNN	91.12	82.03	91.89	84.15	92.88	88.98

Table 3.1: Recognition rate of different classifiers with same and different datasets

3.1.3 Convolution

Convolution filters are the initial layers to receive an input signal. During the convolution step, the network tries to classify the signal coming in by using what it has learned so far. The "cat" reference signal will be added to the input signal if the input signal looks like cat photos that have been seen before. The subsequent layer receives the generated output signal. [20]

Translational invariance is a convenient characteristic of convolution. This means that each convolution filter represents an important feature, like whiskers or fur, and that the CNN algorithm learns which features make up the reference image (i.e., a cat) that comes out of the whole process. The presence of the characteristics doesn't change the output signal strength; instead, their presence determines it. So, a cat could sit in a variety of positions and the CNN algorithm would still recognize it. Consider the example of convolving a 5x5x3 filter with a 32x32x3 image (32x32 image with 3 channels, R, G, and B, respectively). We use the 5*5*3 filter on the whole image, taking the dot product between the filter and different parts of the original image.

The convolutional layer is the core component of a convolutional neural network. In the convolution layer, there are several different filters (6 in the example shown). Each filter is independently convolved with the image, and we end up with 6 feature maps of shape 28*28*1.

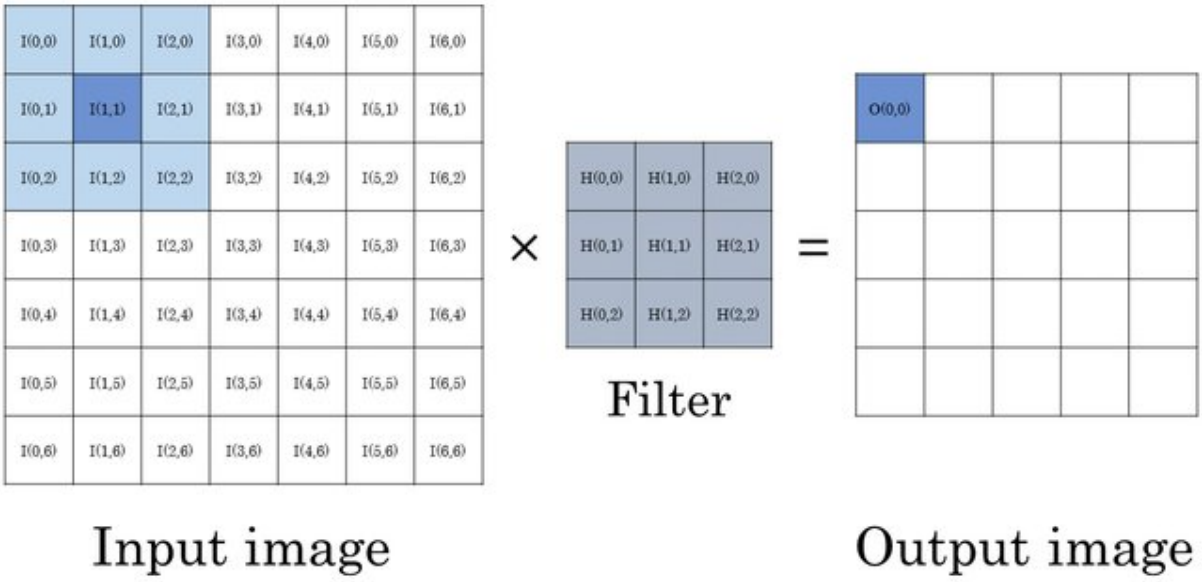

Figure 3.1: CNN: How two different layers each have 6 and 10 filters

The CNN may have a number of convolutional layers, each with a distinct number or similar number of independent filters. For example, the next diagram shows what happens when two convolutional layers each have 6 and 10 filters.

All of these filters are set up at random, and they will be used as the parameters that the network will learn.

3.1.4 Subsampling

Convolution layer inputs can be "smoothed" to lessen their susceptibility to noise and fluctuations. Subsampling is the process of smoothing out the signal. This is done by taking the average of a sample of the signal or the maximum of that sample. Reducing the size of the image or the color contrast across the red, green, and blue (RGB) channels are two examples of subsampling techniques (for image signals).

3.1.5 Pooling Layer

Another element of a CNN is a pooling layer.

Its job is to gradually shrink the representation's space in order to reduce the amount of processing and parameters required by the network. Each feature map is treated separately by the pooling layer. Max pooling, which picks the region's maximum as its representative, is the most common way to pool. For instance, the greatest value

in the following diagram replaces a 2x2 rectangle.

3.1.6 Activation Function

The activation layer controls how the signal moves through the layers, which is a simulation of how neurons in the brain turn on. Strongly correlated output signals with references from the past would turn on more neurons, which would make it easier for the signals to spread and be used for identification. The most common activation function for modeling signal propagation that can be used with CNN is the Rectified Linear Unit (ReLU), which is preferred because it can be trained more quickly.

3.1.7 Fully Connected Layer

The last layers of the network are fully linked, which means that every neuron in the layers before them is connected to every neuron in the layers after them. This is a simulation of higher-level reasoning, in which all possible paths from the input to the output are taken into account.

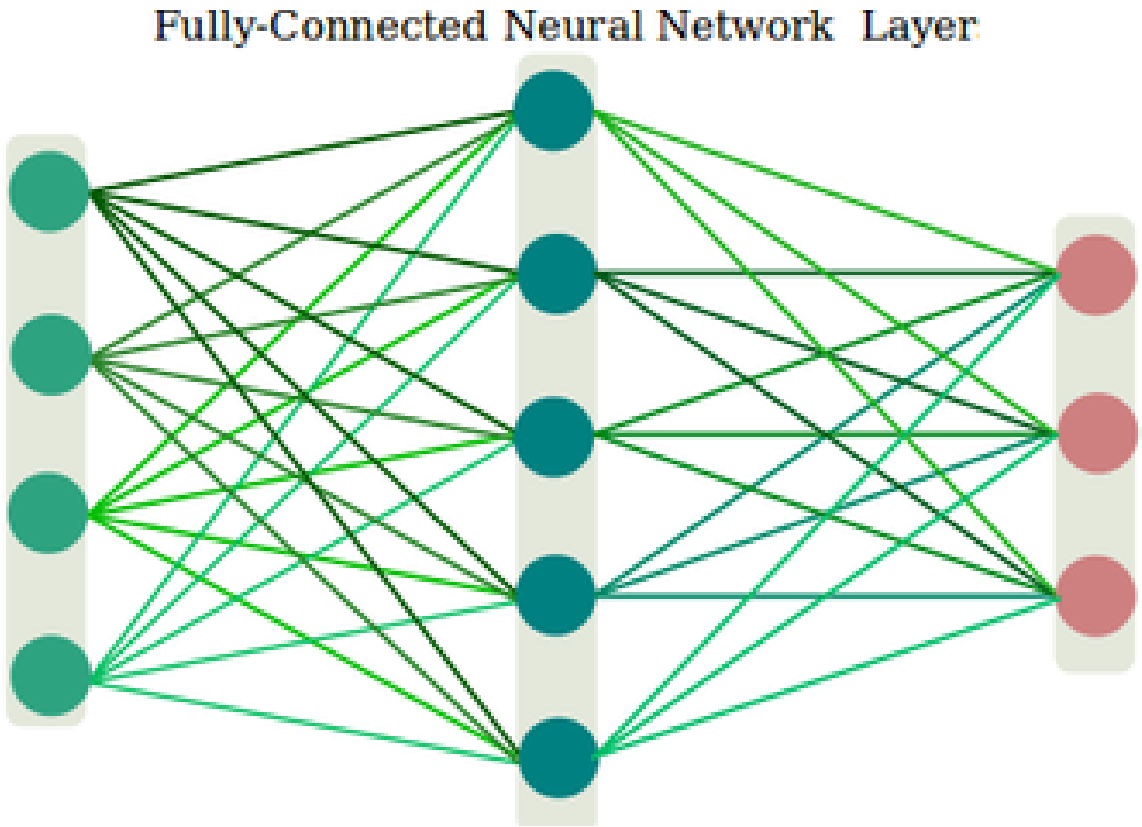

Figure 3.2: Fully Connected Layer: a simulation of a fully connected layer

3.1.8 (During Training) Loss

The loss layer is an extra layer that is used during neural network training. This layer tells the neural network if it got the inputs right and, if it didn't, how wrong its guesses were. This aids in directing the neural network's reinforcement of the proper notions during training. During training, this is always the final layer.

3.2 Related Research

3.2.1 InceptionV3

A convolutional neural network, InceptionV3, was developed as a GoogLeNet module to aid with image processing and object detection. It is the third version of Google's Inception convolutional neural network, which was first shown during the ImageNet Recognition Challenge. 42 layers make up the Inception V3 model in total, which is a little more than the Inception V1 and V2 models. However, this model's efficiency is quite outstanding. It makes a number of improvements, such as using label smoothing, factorized 7x7 convolutions, and an auxiliary classifier to pass label information down the network (along with batch normalization for layers in the side head).

3.2.2 VGG16

VGG16 turned out to be a major turning point in the quest to give computers the ability to "see" the world. The discipline of computer vision (CV) has achieved tremendous advancements in this area several decades ago.[21] VGG16 is one of the major innovations that paved the way for other breakthroughs in this industry. Karen Simonyan and Andrew Zisserman at the University of Oxford came up with the convolutional neural network (CNN) model. The model achieves 92.7% top-5 test accuracy in ImageNet, a collection of over 14 million photos divided into 1000 classes and organized according to the WordNet hierarchy (currently only the nouns), with tens of thousands of images representing each node of the hierarchy. It outperforms AlexNet by sequentially substituting several 33-kernel-sized filters for AlexNet's large kernel-sized filters (11 and 5, respectively, in the first and second convolutional layers).[22] NVIDIA Titan Black GPUs

were used to train VGG16 over several weeks. The VGGNet-16 contains 16 layers and can categorize photos into 1,000 distinct kinds of things, including mice, keyboards, and other devices. The VGGNet16 architecture's simplicity makes the network more appealing. Its uniformity may be demonstrated by its architecture alone. A pooling layer that follows a few convolution layers reduces the height and width. There are approximately 64 filters available, and we can multiply those by two to produce approximately 128 filters, and so on up to 256 filters. Thirteen convolutional layers and three fully connected layers make up the VGG16.[23]

3.2.3 MobileNet

A CNN architecture model for mobile vision and image classification is called MobileNet. There are other models as well, but MobileNet stands out since it can operate or apply transfer learning with very few processing resources. [24] This makes it the perfect option for portable devices, embedded systems, and PCs with low processing power or no GPU, all without significantly impacting the accuracy of the results. It is also most appropriate for web browsers because of their computing, graphics processing, and storage restrictions. It is a straightforward but efficient convolutional neural network that can be applied to mobile vision applications and requires few computing resources. Among MobileNet's practical applications are object detection, fine-grained classifications, face features, and localization. [25] MobileNet employs depth-wise separable convolutions. It significantly lowers the number of parameters compared to a network with traditional convolutions of the same depth. The result is a class of deep neural networks that are lightweight. 27 convolutional layers make up the MobileNet model, including 13 depth-wise convolutions, an average pool layer, a fully connected layer, and a softmax layer.[26]

In terms of Convolution layers, there are

- 13 3x3 Depthwise Convolution
- 1 3x3 Convolution
- 13 1x1 Convolution

In MobileNet, 95% of the time is spent on 1x1 convolution. Andrew G. Howard and other Google researchers created this model.

MobileNet models take advantage of the fact that many of the model parameters in trained models are redundant and capture the same things in the images. Hence, there is significant room for pruning and reducing the number of parameters to produce much more time-efficient models that can be used on lightweight and computationally restrained devices, such as mobile phones. This is done precisely via depth-wise-separable filters.

In comparison to other comparable models, such as the datasets from the Inception model, MobileNet performs better in terms of latency, size, and accuracy. With a fully developed model, the output performance isn't as good as it could be. The trade-off is fine when the model can be used on a mobile device for offline detection in real-time.

3.2.4 DenseNet

A DenseNet is a type of convolutional neural network that uses Dense Blocks to directly connect all layers (with matching feature-map sizes) with one another in order to create dense connections between the layers. To keep the feed-forward nature of the system, each layer receives additional inputs from all earlier layers and broadcasts its own feature maps to all later layers. Convolutional networks with a lot of connections are known as dense nets. It is quite similar to a ResNet, with a few key distinctions. In contrast to ResNet, which only uses the most recent result as an input, DenseNet employs an additive method, using all of the prior output as an input for a subsequent layer. DenseNet was developed primarily to improve accuracy in high-level neural networks due to the enormous distance between the input and output layers and the fact that information vanishes before it reaches the target. [27] Some advantages of the dense net-

- **Parameter efficiency:** Only a small number of parameters are added by each layer; for instance, only roughly 12 kernels are learned per layer.
- **Implicit deep supervision:** Increased gradient flow through the network thanks to direct access to the gradient and loss function for all layer's feature maps.

3.3 Methodology

The approach we used during the entire journey can be explained in four distinct ways. First, we gathered a variety of study articles on this topic that would be helpful during the entire process. In order to determine the path and determine where we needed to start, we gathered 30 to 40 documents and read over them. Second, we looked for various models and computation methods. Thirdly, we employed multiple models and used the dataset we gathered to train them. Four datasets in total, including Bangla and American

sign language, have been gathered. We graded the models based on their performance after training. Finally, in order to display correct results, we integrated the best results model into our prototype.[28]

3.4 Dataset Description

- **ASL-1:** Our first dataset is ASL (American Sign Language). It has a total of 6,500 images. The dataset was divided into 5,200 photos for training and 1,300 for testing, with an 80:20 proportion between the two. This dataset was obtained from Kaggle, an open source.
- **ASL-2:** This dataset is our second dataset. It has a total of 87,000 Images where 69,600 is for training and 17,400 is for testing. It was also taken from Kaggle.
- **BSL:** BSL (Bangla Sign Language) is our third dataset. The total image size, in this case, is 12,581, with 8,068 for training and 4,513 for testing. We got this dataset from Kaggle.
- **BdSL:** Our final dataset is Bangla Sign Language using zero-shot learning and transfer learning. It contains a total of 34,151 images, of which 21,338 are used for training and 12,813 are used for testing. There are 37 Bangla number classes in this dataset, ranging from 0 to 36. This dataset was taken from a renowned research paper.

Figure 3.3: a collage of Dataset of individual ASL characters

3.5 Block Diagram

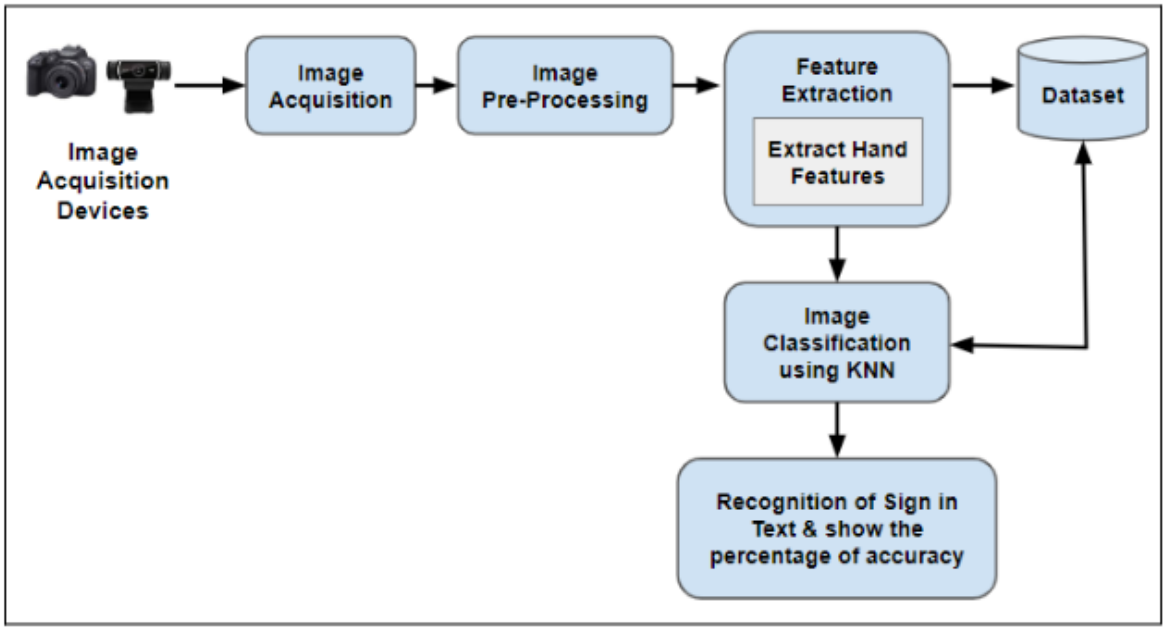

Figure 3.4: Block Diagram of how our Demo will be working

4 Result and Analysis

We checked how accurate our dataset was using different models, like CNN, ANN, and KNN. After training with the trained generator, the validation generator, and 10–20 epochs, our CNN model provided 15–79% accuracy for four separate datasets. Then, our ANN model offered 50–70% accuracy. Finally, the KNN model delivered the best overall value, with 95% accuracy. Therefore, we decided to use the KNN classifier to create our final prototype.

The histories of the testing accuracy of four CNN architectures are given below:

Architectures	ASL1	ASL2	BSL	BdSL
InceptionV3	15%	80%	58%	78%
VGG16	30%	78%	46%	75%
MobileNet	28%	75%	35%	56%
DenseNet	20%	40%	30%	13%

Table 4.1: Results using Diffrent Architectures using ASL1, ASL2, BSL, BDSL

In comparison to other architectures, the accuracy of the InceptionV3 architecture was good, and the training loss was lower right away. It achieved improved accuracy and less training loss after a few epochs.

4.1 Inception V3

Among the various model architectures, Inception V3 delivered the best outcome. In this case, we got the lowest accuracy (15%) in the ASL1 dataset and the highest accuracy (80%) in the ASL2 dataset. We used stochastic gradient descent (SGD) and the Adam optimizer for different architectures. Additionally, we used the learning rate 0.001, validation split 0.2 and batch size 80 in our models. The result of the ASL2 dataset is shown below-

```
[ ]  initial_learning_rate = 0.001
     lr_schedule = keras.optimizers.schedules.ExponentialDecay(
         initial_learning_rate,
         decay_steps=2000,
         decay_rate=0.16,
         staircase=True)

     # tell the model what cost and optimization method to use
     model.compile(
       loss='categorical_crossentropy',
       optimizer=keras.optimizers.SGD(learning_rate=lr_schedule),
       metrics=['accuracy']
     )
```

```
[ ]  # Use the Image Data Generator to import the images from the dataset
     from tensorflow.keras.preprocessing.image import ImageDataGenerator

     # create generator
     train_datagen = ImageDataGenerator(rescale = 1./255,
                                        validation_split=0.2,
                                        shear_range = 0.2,
                                        zoom_range = 0.2,
                                        horizontal_flip = True)

     test_datagen = ImageDataGenerator(rescale = 1./255)
```

Figure 4.1: Code for Inception V3 part 1

```
[ ]  # fit the model
     # Run the cell. It will take some time to execute
     r = model.fit(
       training_set,
       validation_data=test_set,
       epochs=10,
       steps_per_epoch=len(training_set),
       validation_steps=len(test_set)
     )
```

```
Epoch 1/10
267/267 [==============================] - 8884s 33s/step - loss: 2.4781 - accuracy: 0.3319 - val_loss: 1.7357 - val_accuracy: 0.5142
Epoch 2/10
267/267 [==============================] - 579s 2s/step - loss: 1.4024 - accuracy: 0.5817 - val_loss: 1.3120 - val_accuracy: 0.6305
Epoch 3/10
267/267 [==============================] - 582s 2s/step - loss: 1.1944 - accuracy: 0.6665 - val_loss: 1.1270 - val_accuracy: 0.6809
Epoch 4/10
267/267 [==============================] - 578s 2s/step - loss: 1.0357 - accuracy: 0.7103 - val_loss: 1.0216 - val_accuracy: 0.7069
Epoch 5/10
267/267 [==============================] - 574s 2s/step - loss: 0.9154 - accuracy: 0.7471 - val_loss: 0.9127 - val_accuracy: 0.7431
Epoch 6/10
267/267 [==============================] - 578s 2s/step - loss: 0.8425 - accuracy: 0.7661 - val_loss: 0.8664 - val_accuracy: 0.7563
Epoch 7/10
267/267 [==============================] - 578s 2s/step - loss: 0.7782 - accuracy: 0.7817 - val_loss: 0.8076 - val_accuracy: 0.7723
Epoch 8/10
267/267 [==============================] - 576s 2s/step - loss: 0.7231 - accuracy: 0.8003 - val_loss: 0.7814 - val_accuracy: 0.7795
Epoch 9/10
267/267 [==============================] - 575s 2s/step - loss: 0.6831 - accuracy: 0.8151 - val_loss: 0.7746 - val_accuracy: 0.7817
Epoch 10/10
267/267 [==============================] - 574s 2s/step - loss: 0.6732 - accuracy: 0.8185 - val_loss: 0.7702 - val_accuracy: 0.7826
```

Figure 4.2: Code for Inception V3 part 2

4.2 VGG16

For VGG16, we got the lowest accuracy (30%) in the ASL1 dataset and the highest accuracy (78%) in the ASL2 dataset. The result of the ASL2 dataset is shown below-

```
[17] r = model.fit(
       training_set,
       validation_data=test_set,
       epochs=10,
       steps_per_epoch=len(training_set),
       validation_steps=len(test_set)
     )

Epoch 1/10
163/163 [==============================] - 2337s 14s/step - loss: 8.0148 - accuracy: 0.6315 - val_loss: 8.9141 - val_accuracy: 0.6062
Epoch 2/10
163/163 [==============================] - 979s 6s/step - loss: 2.1428 - accuracy: 0.8654 - val_loss: 8.8721 - val_accuracy: 0.6469
Epoch 3/10
163/163 [==============================] - 979s 6s/step - loss: 1.7067 - accuracy: 0.8950 - val_loss: 8.0984 - val_accuracy: 0.7200
Epoch 4/10
163/163 [==============================] - 981s 6s/step - loss: 1.6418 - accuracy: 0.9098 - val_loss: 8.2484 - val_accuracy: 0.7238
Epoch 5/10
163/163 [==============================] - 983s 6s/step - loss: 1.4013 - accuracy: 0.9252 - val_loss: 9.9437 - val_accuracy: 0.7369
Epoch 6/10
163/163 [==============================] - 977s 6s/step - loss: 1.3156 - accuracy: 0.9335 - val_loss: 7.8798 - val_accuracy: 0.7777
Epoch 7/10
163/163 [==============================] - 979s 6s/step - loss: 1.0648 - accuracy: 0.9427 - val_loss: 12.0651 - val_accuracy: 0.7331
Epoch 8/10
163/163 [==============================] - 983s 6s/step - loss: 0.8540 - accuracy: 0.9523 - val_loss: 9.8043 - val_accuracy: 0.7446
Epoch 9/10
163/163 [==============================] - 981s 6s/step - loss: 0.9707 - accuracy: 0.9523 - val_loss: 8.5016 - val_accuracy: 0.7800
Epoch 10/10
163/163 [==============================] - 980s 6s/step - loss: 0.8072 - accuracy: 0.9588 - val_loss: 9.8847 - val_accuracy: 0.7546
```

Figure 4.3: Code for VGG16

4.3 MobileNet

For MobileNet, we got the lowest accuracy (28%) in the ASL1 dataset and the highest accuracy (75%) in the ASL2 dataset. The BdSL dataset provided 56% accuracy in this model. The result of the BdSL dataset is shown below-

```
# fit the model
# Run the cell. It will take some time to execute
r = model.fit(
  training_set,
  validation_data=test_set,
  epochs=10,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

```
Epoch 1/10
346/346 [==============================] - 3529s 10s/step - loss: 4.2923 - accuracy: 0.3725 - val_loss: 4.0161 - val_accuracy: 0.4296
Epoch 2/10
346/346 [==============================] - 74s 214ms/step - loss: 3.2872 - accuracy: 0.5109 - val_loss: 3.9026 - val_accuracy: 0.4883
Epoch 3/10
346/346 [==============================] - 73s 212ms/step - loss: 3.2029 - accuracy: 0.5584 - val_loss: 3.8586 - val_accuracy: 0.4908
Epoch 4/10
346/346 [==============================] - 73s 211ms/step - loss: 3.0630 - accuracy: 0.5828 - val_loss: 4.3720 - val_accuracy: 0.5046
Epoch 5/10
346/346 [==============================] - 72s 207ms/step - loss: 2.9362 - accuracy: 0.6065 - val_loss: 5.1353 - val_accuracy: 0.4776
Epoch 6/10
346/346 [==============================] - 73s 210ms/step - loss: 3.0089 - accuracy: 0.6180 - val_loss: 4.0326 - val_accuracy: 0.5428
Epoch 7/10
346/346 [==============================] - 73s 210ms/step - loss: 2.8918 - accuracy: 0.6374 - val_loss: 3.8787 - val_accuracy: 0.5520
Epoch 8/10
346/346 [==============================] - 73s 210ms/step - loss: 2.7828 - accuracy: 0.6473 - val_loss: 4.7123 - val_accuracy: 0.5283
Epoch 9/10
346/346 [==============================] - 72s 209ms/step - loss: 2.8974 - accuracy: 0.6573 - val_loss: 5.0549 - val_accuracy: 0.5079
Epoch 10/10
346/346 [==============================] - 72s 209ms/step - loss: 2.7327 - accuracy: 0.6686 - val_loss: 4.6315 - val_accuracy: 0.5605
```

Figure 4.4: Code for MobilrNEt

4.4 DenseNet

Our accuracy wasn't sufficient in DenseNet. We got the lowest accuracy (13%) in the BdSL dataset and the highest accuracy (40%) in the ASL2 dataset. The result of the BdSL dataset is shown below-

```
# Run the cell. It will take some time to execute
r = model.fit(
  training_set,
  validation_data=test_set,
  epochs=10,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

```
Epoch 1/10
370/370 [==============================] - 12272s 33s/step - loss: 0.6990 - accuracy: 0.7782 - val_loss: 62.2349 - val_accuracy: 0.1337
Epoch 2/10
370/370 [==============================] - 146s 395ms/step - loss: 0.3365 - accuracy: 0.8892 - val_loss: 62.5194 - val_accuracy: 0.1351
Epoch 3/10
370/370 [==============================] - 140s 378ms/step - loss: 0.3281 - accuracy: 0.9002 - val_loss: 63.1786 - val_accuracy: 0.1333
Epoch 4/10
370/370 [==============================] - 140s 379ms/step - loss: 0.2554 - accuracy: 0.9206 - val_loss: 63.9425 - val_accuracy: 0.1319
Epoch 5/10
370/370 [==============================] - 139s 376ms/step - loss: 0.2987 - accuracy: 0.9176 - val_loss: 63.8428 - val_accuracy: 0.1360
Epoch 6/10
370/370 [==============================] - 139s 376ms/step - loss: 0.3342 - accuracy: 0.9159 - val_loss: 65.0999 - val_accuracy: 0.1319
Epoch 7/10
370/370 [==============================] - 139s 376ms/step - loss: 0.2760 - accuracy: 0.9296 - val_loss: 65.0775 - val_accuracy: 0.1343
Epoch 8/10
370/370 [==============================] - 141s 381ms/step - loss: 0.2611 - accuracy: 0.9321 - val_loss: 65.0828 - val_accuracy: 0.1346
Epoch 9/10
370/370 [==============================] - 139s 375ms/step - loss: 0.2954 - accuracy: 0.9328 - val_loss: 66.8840 - val_accuracy: 0.1347
Epoch 10/10
370/370 [==============================] - 139s 376ms/step - loss: 0.2417 - accuracy: 0.9405 - val_loss: 65.8502 - val_accuracy: 0.1353
```

Figure 4.5: Code for DenseNet

4.5 Model Comparison

Out of the four datasets, ASL-2 and BdSL provided us with a higher level of accuracy. For both datasets, the accuracy for the CNN model is 75% and 79%, respectively. It is 70% and 65% for ANN. KNN produced better results, with 95% and 92% accuracy, respectively.

Models	ASL-2	BdSL
CNN	57.71	52.46
ANN	65.68	60.36
KNN	75.77	66.88

Table 4.2: Results using Different Models using ASL2, BDSL

CNN incurs a substantial computational cost. A significant amount of training data is required to achieve good accuracy. Aside from being computationally costly, ANN has difficulty determining the appropriate network structure. On the other hand, KNN implementation is straightforward. The algorithm is fairly simple to grasp.

4.5.1 Usability

We are making a system to understand sign language and turn it into text. It can be an alphabetical term, a word, or a specific sentence. Typically, those who are deaf or mute communicate with others via sign language. There are only a few people who understand sign language. Other people, who do not understand, can never feel what they are trying to say. And in a situation like that, our system can come in handy. Using it, regular people can understand sign language.

- **People:** As we know, there are 700,000 to 900,000 dumb and mute people worldwide. They can use our system to convert their sign language to text. It will make things easy to work.
- **Devices:** We will make a short application as the working module of our system. We will have a mobile application for both IOS and ANDROID and software for computers. We are also planning to build an entire website for this system. Then it won't depend on devices anymore. Our system works flawlessly on any device that can take pictures.[29]
- **Language:** As our system can learn the gesture of any language, it is a language impediment system. There are too many languages available right now. We do not even have enough datasets to train our model for every language. However, we are currently trying to work with at least five most used languages like English, Chinese, Hindi, and Bangla. Later on, we plan to work with other languages. If we can make the datasets, then training the system is not very hard.
- **Age:** The customer will get an effortless application to use the sign language recognition system. They won't see the layers of millions of complex neural networks. They won't have many choices to make. The font name will automatically turn on when the app is launched, and it will begin looking for the gesture. After finding the correct motion, it will convert it to our everyday language. That's why any person of any age can use it.

4.5.2 Manufacturing

Since it's a software-based solution, we do not need to go to the industry to build it. It can be created by writing codes, connecting databases, using neural network layers, and other things. As our system does not require any extra hardware except the built-in front camera that every phone has, it won't take up much space or consume much energy. That way, we can say that we manufactured this system without any additional trouble.

4.5.3 Sustainability

Nowadays, every piece of hardware is being replaced by software. There are so many reasons behind it. First of all, copying the code of the software is easy. That gives software reusability. Another thing is that hardware gets damaged more easily than software. With proper inspection and updates, any software can survive in the long run.

Software that we developed with the aid of a convolutional neural network is at the heart of the solution we are working on. It is extremely sustainable and safe. Our production team will look at the data we get and release a new version with new features. That makes our system sustainable in the long run.

5 Prototype

5.1 Prototype

The technology that was used can be divided into hardware and software:

- **Webcam, Any computer or laptop with a decent internet connection:** A webcam is a digital camera that is connected to a computer and used for video communication or recording. In the context of a sign language translator, a webcam can be used to capture video of a person signing in real-time and send it to the server for processing.
  There are various approaches to building a sign language translator using a webcam and a computer. One approach is to use a pre-trained model that has been trained on a large dataset of sign language videos and is able to recognize and classify different signs and gestures. The model can then be fine-tuned to improve its accuracy for the specific task of translating sign language.

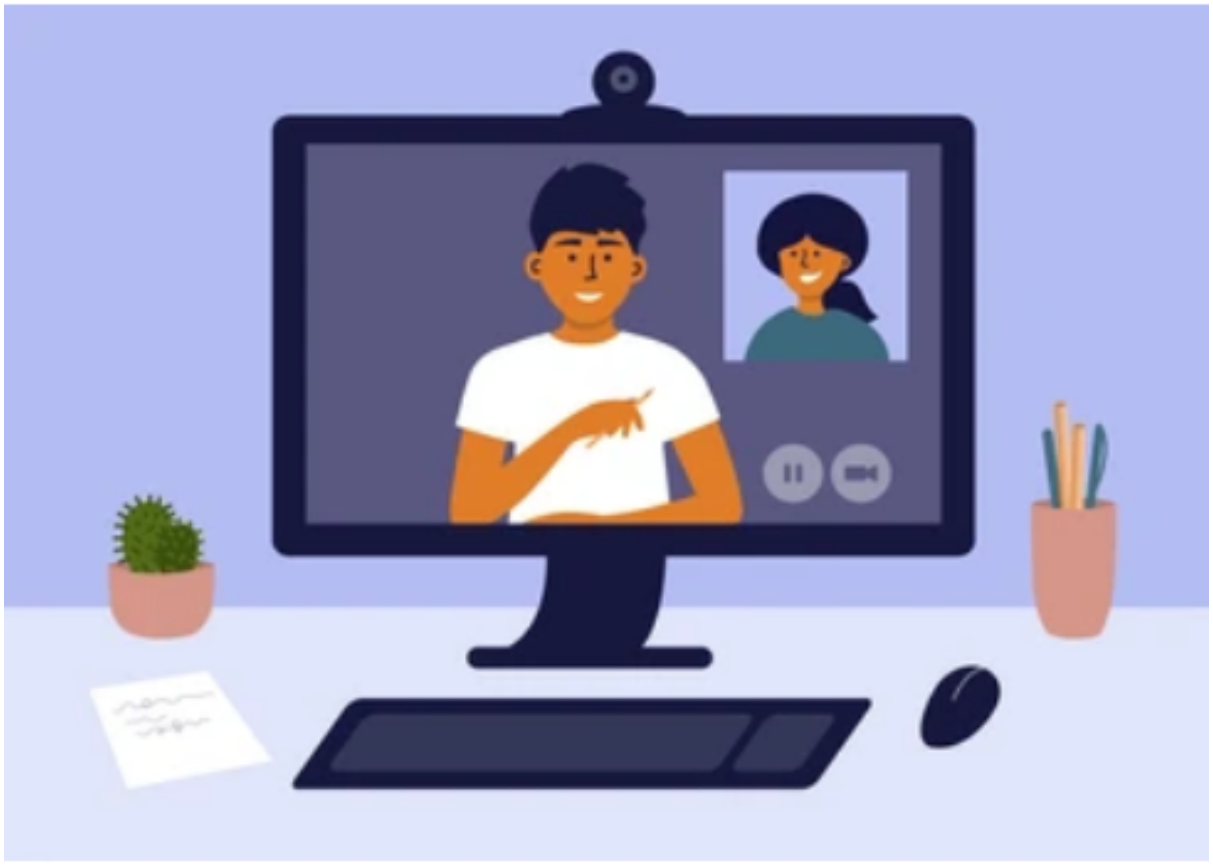

Figure 5.1: The Usual Setup for our Prototype Illustrated

- **HTML, Bootstrap, CSS, JavaScript, React, Google collab:** In 499A, we used Google Collab to train models and compare accuracy with other classifiers. For the prototype, we found a classifier that would work best with our web application. We intended to build the web application using React JS and Bootstrap as our code base.

Figure 5.2: Technologies Used for the Prototype

Our prototype can be found by visiting our website: https://atifkarim.xyz/Sign-Language-Translator/index.html

The histories of the testing accuracy of four CNN architectures are given below:

5.1.1 Learn New Sign Language Alphabets (ASL) and phrases

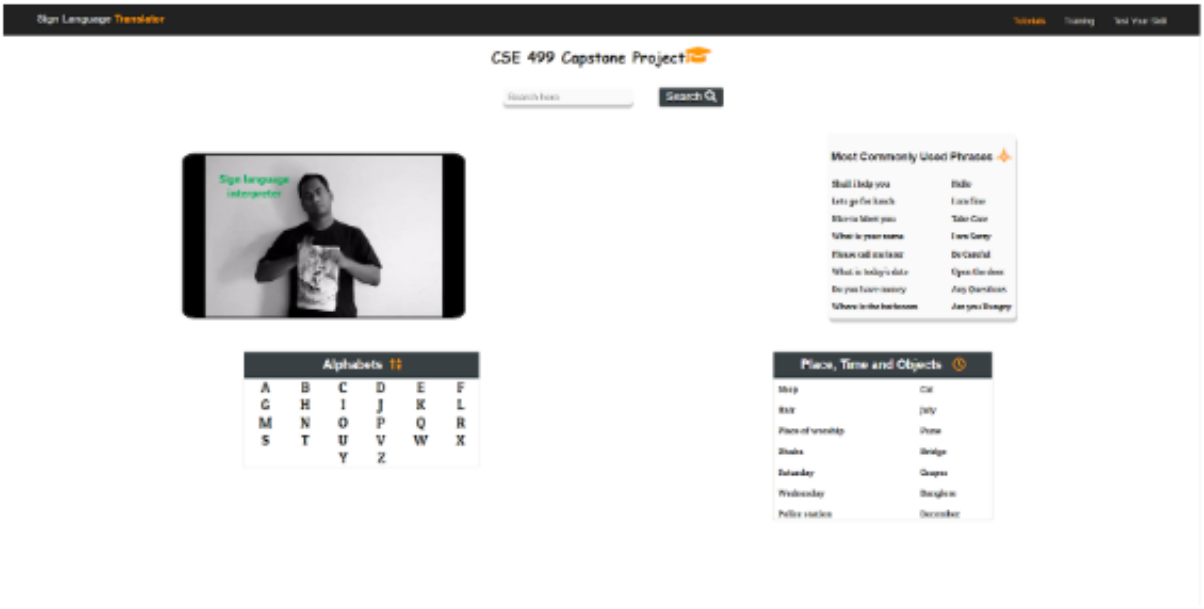

Figure 5.3: Landing Page of Our Prototype

Select any phrases or alphabets from the "Most Commonly Used Phrases" section, alphabets from the "Alphabets" section, or any phrase from the "Place, Time, and Objects" section, and learn through high-quality videos and images.

5.1.2 Make Your Own Sign Language

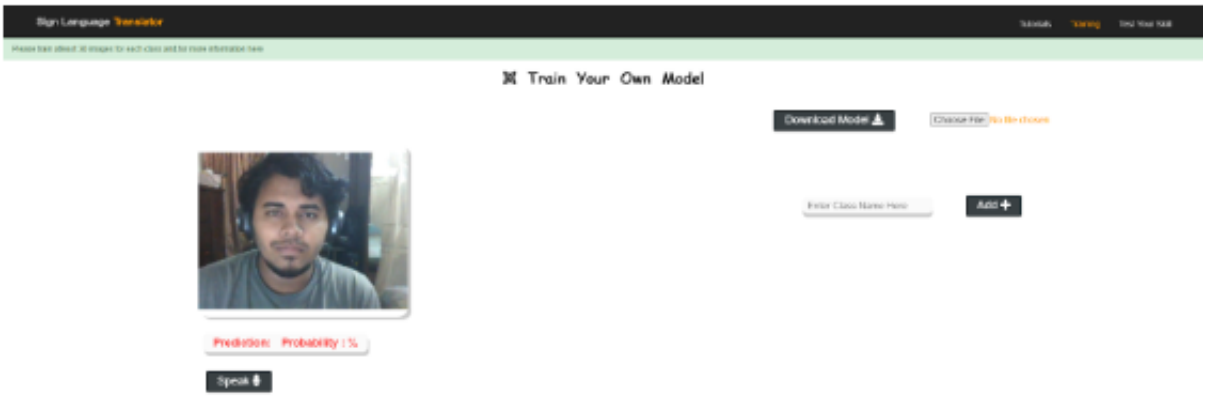

Figure 5.4: Making Your Own Sign Language Page

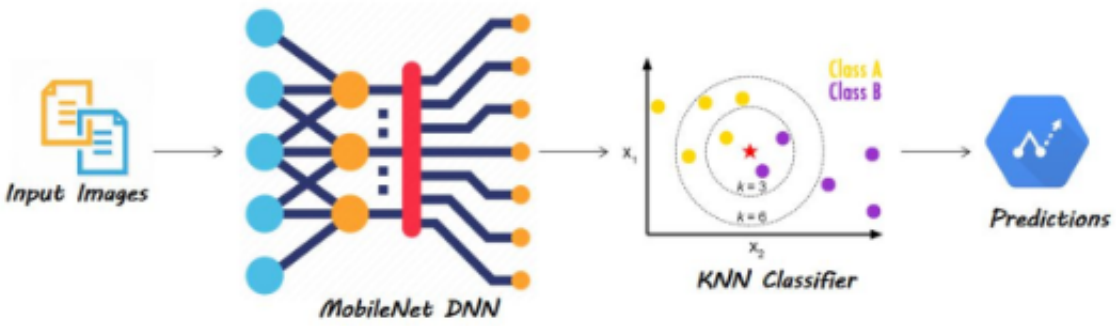

Figure 5.5: Pipeline for This Page

Write the text that is your label for that particular class and click on the "Add" button; after that, using a webcam as input, click on the “Add New Images” button and add 10-40 images per class. You will be able to observe the real-time predictions below the video.

The Sign Language Training Model lets anyone build their own image classification model with no coding required. All you need is a webcam. The approach we’re going to take is called transfer learning. This technique starts with an already trained model and specializes in the task at hand. This lets you train far more quickly and with less data than if you were to train from scratch. We bootstrap our model from a pre-trained model called MobileNet. Our system will learn to make predictions using our own classes that were never seen by MobileNet. We do this by using the activations made by this model that has already been trained. These activations are a rough way to represent high-level semantic features of the image that the model has learned. Training is so effective that we don’t have to do anything fancy like train another neural network; instead, we just use the nearest-neighbor approach. What we do is feed an image through MobileNet and find other examples in the dataset that have similar activations to this image. In practice, this is too noisy, so instead, we choose the k-nearest neighbors and the class with the most representation. By bootstrapping our model with MobileNet and using k-nearest neighbors, we can train a realistic classifier in a short amount of time, with very little data, all in the browser. Doing this fully end-to-end, from pixels to prediction, won’t require too much time and data for an interactive application. [30]

5.1.3 Practice Alphabets and Phrases

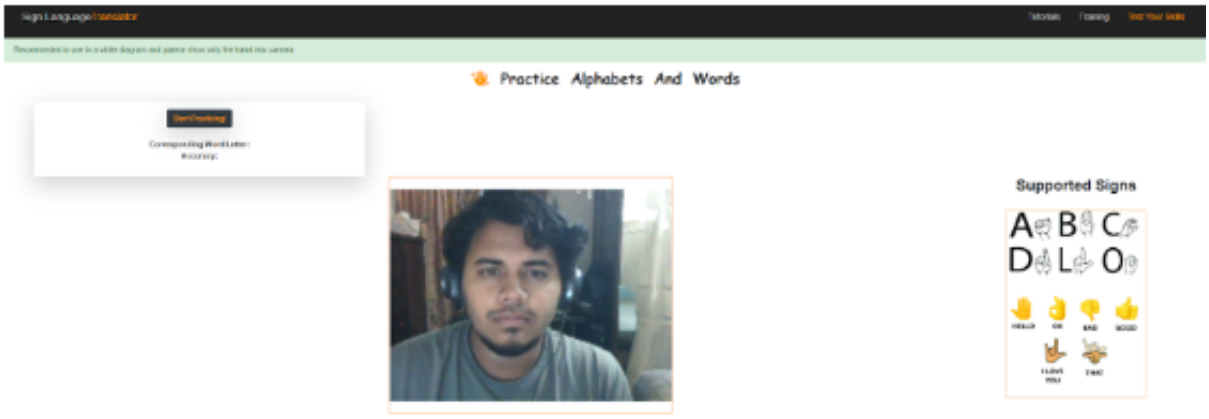

Figure 5.6: Practicing On Preset Datsets page

Click on the "Start Practicing" button, place your hand in front of the camera against a white background, and start practicing your signs (alphabets). You will be able to see the prediction below the button.

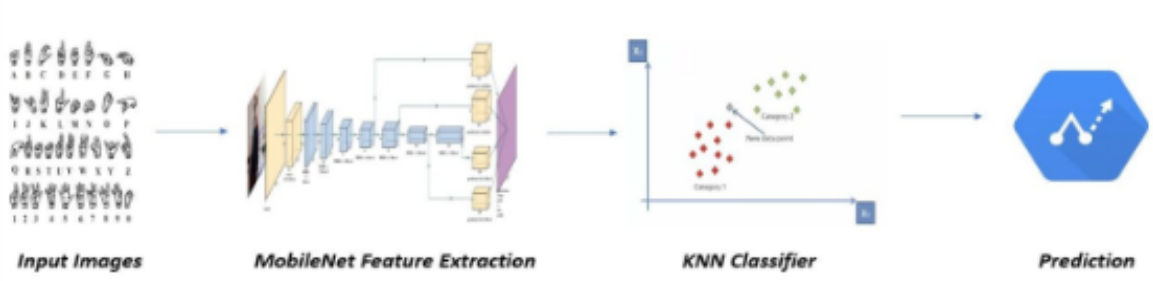

Figure 5.7: Pipeline for this Page

5.2 How it works

The Practice Sign Model lets you practice your sign language skills, which you learned on our tutorial page

The above model is mainly divided into three parts:

- Dataset.
- MobileNet Feature Extraction.
- Knn Classifier

Starting with the dataset, which is made up of more than 1000 images from 21 different classes of sign languages. Each image is 64 x 64 pixels in size. The approach we're going to take is called transfer learning. This technique starts with an already trained model and specializes in the task at hand. This lets you train far more quickly and with less data than if you were to train from scratch. We bootstrap our model from a pre-trained model called MobileNet. Our system will learn to make predictions using our own classes that were never seen by MobileNet. We do this by using the feature extraction technique. A sort of dimensionality reduction known as feature extraction divides a set of raw data into more manageable, smaller groups. These enormous data sets share the feature of having a lot of variables, which requires a lot of processing power. We can simply apply the nearest-neighbor method because the pretraining is so successful; there is no need to perform any elaborate training of additional neural networks.

5.3 Code

```
const start = async() => {
    const trainingCards = document.getElementById("training-cards")
    const predictions = document.getElementById("predictions")
    const confidence = document.getElementById("confidence")

    const createKNNClassifier = async() => {
        console.log('Loading KNN Classifier');
        return await knnClassifier.create();
    };
    const createMobileNetModel = async() => {
        console.log('Loading Mobilenet Model');
        return await mobilenet.load();
    };
    const createWebcamInput = async() => {
        console.log('Loading Webcam Input');
        const webcamElement = await document.getElementById('webcam');
        return await tf.data.webcam(webcamElement);
    };

    const mobilenetModel = await createMobileNetModel();
    const knnClassifierModel = await createKNNClassifier();
    const webcamInput = await createWebcamInput();
    var preloader = document.getElementById("loading");
```

Figure 5.8: Code for KNN Classifier used in the prototype

Above, we have the code that initializes the KNN classifier.

```
  // Extract the already learned features from MobileNet
  featureExtractor = ml5.featureExtractor('MobileNet', modelReady);
  const knnClassifier = ml5.KNNClassifier();


  // Create a new classifier using those features and give the video we want to
      use
  const options = { numLabels: 21 };
  classifier = featureExtractor.classification(video, options);
  // Set up the UI buttons
  setupButtons();
}

// A function to be called when the model has been loaded
async  function modelReady() {
  select('#modelStatus').html('MobileNet Loaded!');
  // If you want to load a pre-trained model at the start

  await classifier.load('./model.json', function() {
     select('#modelStatus').html('Custom Model Loaded!');
   });
}

// Classify the current frame.
function classify() {
  classifier.classify(gotResults);
}
```

Figure 5.9: Code for Mobile Net used in the prototype

Above, we have the MobileNet architecture, using which we can load the models we had previously trained and which will be used by the KNN classifier.

```
const imageClassificationWithTransferLearningOnWebcam = async() => {
    console.log("Machine Learning on the web is ready");
    while (true) {
        if (knnClassifierModel.getNumClasses() > 0) {
            const img = await webcamInput.capture();

            // Get the activation from mobilenet from the webcam.
            const activation = mobilenetModel.infer(img, 'conv_preds');
            // Get the most likely class and confidences from the classifier
                module.
            const result = await knnClassifierModel.predictClass(activation
                );

            if (uploadedModel) {
                predictions.innerHTML = result.label
                confidence.innerHTML = Math.floor(result.confidences[result
                    .label] * 100)

            } else {

                try {
                    predictions.innerHTML = classes[result.label - 1].name
                    confidence.innerHTML = Math.floor(result
                        .confidences[result.label] * 100)
                } catch (err) {
                    predictions.innerHTML = result.label - 1
                    confidence.innerHTML = Math.floor(result
                        .confidences[result.label] * 100)
                }
            }


            // Dispose the tensor to release the memory.
            img.dispose();
        }
        await tf.nextFrame();
    }
};
```

Figure 5.10: Code that initializes transfer learning by taking pictures from the webcam used in the prototype

Above we have the code that initializes transfer learning by taking pictures from the webcam.

```
const addDatasetClass = async(classId) => {

    // Capture an image from the web camera.
    const img = await webcamInput.capture();

    // Get the intermediate activation of MobileNet 'conv_preds' and pass
        that
    // to the KNN classifier.
    const activation = mobilenetModel.infer(img, 'conv_preds');

    // Pass the intermediate activation to the classifier.
    knnClassifierModel.addExample(activation, classId);

    let classIndex = classes.findIndex(el => el.id === classId)
    currentCount = classes[classIndex].count
    currentCount += 1
    classes[classIndex].count = currentCount

    var temp_id = 'images-' + classId.toString()
    document.getElementById(temp_id).innerHTML = currentCount;

    // Dispose the tensor to release the memory.
    img.dispose();
};
```

Figure 5.11: Code to initialize the datasets used in the prototype

Above, we have the datasets that are stored on our server and how they are used by the KNN classifier.

5.4 Environmental Considerations

After the whole project is done, we will make a simple mobile app to do the recognition. The complex neural network architecture won't be shown to the user. They will find a simple, small, portable application. We plan to launch an entire website that can work the same as the application. Each calculation will be done on the server, so it won't change how the application works.

- **Effect to Environment:** Since it would be a small app, it won't occupy much space. Otherwise, if this application needs ample space, it could increase the use of semiconductors and other storage materials. And these semiconductor devices can harm the environment as we know that the company uses energy and water-intensive

materials to make these semiconductors, which is terrible for the environment. Also, these semiconductors produce hazardous waste. So by making a small application, we are not causing any damage to the environment.

- **Harmful Radiation:** The system we are designing has two sides. We confess that it has a very complex architecture. For each sign recognition, many calculations will run in the background. And here comes the exciting part, all of these will happen on the server side. It will save the user from any harmful or hazardous radiation. Besides, it is a mobile app, so it does not need to release radiation or anything like that.
- **Heat & Energy:** Our application won't consume much energy. Since it is a small and straightforward app, it will use a small amount of charge on the phone. There will be no heat issue because of the server-side calculation technique. And our production team will monitor regular activity for any heat issues or other problems.
- **Energy Wasting:** Software wastes a considerable amount of energy for many reasons. One of the main reasons is that software runs in the background all the time. It not only requires and occupies extra space on the ram but also uses and wastes a lot of energy. Fortunately, this application does not need to run in the background. The client will use it only when they need to understand sign language. Once it is closed, it won't use any space of the ram, let alone use the electrical energy.
- **Carbon Footprint:** This Sign Language to Text application produces no carbon emissions. Since this is a software-based application, it contains thousands of lines of code. The other complex calculation will also happen in the microprocessor, which does not produce any carbon or carbon-related material. That is why we can proudly say that it has no carbon footprint.

5.5 Environmental Sustainability

Every consideration we have discussed earlier is 100 percent sustainable. We will keep updating the software according to our customer's needs, but the size won't increase much. One of our main focuses is to ensure that there is no data duplication. It will remain the same, as we are now using the server side for complicated calculations. So we can save our customers from harmful or hazardous radiation.

As our design method dictates, we won't need this application to run in the background. So it is inevitable that we will be able to save energy and reduce waste. None of these fixes are limited to the current problem. It will continue its environment-friendly journey into the future, too.

6 Conclusion

The main goal of this project was to build an easy solution for people who don't understand sign language. Our solution can show the corresponding alphabet by recognizing the hand gestures of sign language. We have used four data sets in two different languages, Bangla and English. InceptionV3, VGG16, MobileNet, and DenseNet are the model architectures that we have used for this project. We have improved the accuracy of the validation test by tuning parameters. Finally, the highest accuracy we have gotten is 80% on the InceptionV3 model trained with the American Sign Language data set.

We intend to further this research and include our very own modified model, which will match or even exceed current models. This will have huge social effects as it will help us be much closer to people with special needs, which will help us communicate far better with our loved ones who are facing such adversities.

Our goal is to get better at correctly recognizing continuous sign language movements. The method utilized for individual gestures can be used for word and sentence-level sign language. Additionally, the current procedure employs two distinct models: KNN and CNN in training. In the future, we will be able to focus on more models and combine different models into one.

We also intend to reuse our prototype and use vertical scaling to make the website into an API to be used in websites and applications.

We had to use many internal and external tools to complete the project. We were already familiar with some of these tools, but we had to learn others. Before designing and validating the system, we had to go through previous reports and papers. We made a list of all the tools and techniques we will use in the project. Then we shortlist the new things that we need to learn. We use both online and offline platforms to know them. One of the top websites where we directly learned a lot of things was YouTube. We also read many papers, journals, and articles. Some websites, like GeeksforGeeks and Stack Overflow, also helped us when we got stuck. And our respected professor helped us whenever we needed both online and offline meetings.

Designing and Validating Tools that we used are some frameworks, working environments, cloud storage, and others. Some of these had free versions to use, and for others, we had to pay. Yet, by using them together, we have finished getting the system to recognize sign language from hand gestures and turn it into the correct text.

- **Pytorch:** An open-source machine learning (ML) framework called PyTorch is built using the Torch library and the Python programming language. For deep

learning research, it is one of the most popular platforms. This framework was acquired to hasten the transition from research prototyping to deployment.

- **Tensorflow:** We mainly used it for its wide range of software libraries. TensorFlow offers a software library that can be used for various tasks, as is well known. It is primarily focused on deep neural network training and interface. The CNN architecture was used, and we used multiple layers.
- **Google Colab Pro:** Our proposed system is software-based, so we needed a friendly interface and environment to work on. People who have worked on related projects have suggested the Jupyter Notebook environment but it would need a vast space on the hard disk to run for 24 to 48 hours straight. Because we needed to distribute the work among the group members, we used Google Colab. Colab is an entirely cloud-based environment that is free to use. Most significantly, no setup is needed, and team members can update the notes that create concurrently.
- **Cloud Storage:** We had to work with various data sets for training and testing purposes. Some data sets were so significant that we had to use multiple PCs to download them quickly. And colab requires the dataset to be stored in the cloud, which is why we have used Google Drive as our cloud storage.
- **Architectural Design:** According to our professor, we strictly followed architectural design. CNN was our first choice as the prominent architecture. We have used many convolution layers, pooling layers, softmax algorithms, and other essential components.
- **GitHub:** We must use GitHub to combine our codes from each teammate. Gits repository helped us to secure our code and everything. It also allowed us to debug the code. And for tracking teammates' activity, GitHub is incomparable. Because of it, we could see who was changing what portion of the code.
- **Latex:** All the documents for this research were written using the typesetting system latex. Almost any type of publishing can use it, but it is most frequently used for medium- to large-sized technical or scientific documents. Here, it's simple for everyone in the group to write and edit simultaneously.

References

[1] Ragib Amin Nihal, Sejuti Rahman, Nawara Mahmood Broti, and Shamim Ahmed Deowan. Bangla sign alphabet recognition with zero-shot and transfer learning. *Pattern Recognition Letters*, 150:84–93, 2021.

[2] Sarfaraz Masood, Adhyan Srivastava, Harish Chandra Thuwal, and Musheer Ahmad. Real-time sign language gesture (word) recognition from video sequences using cnn and rnn. In *Intelligent Engineering Informatics*, pages 623–632. Springer, 2018.

[3] Yonas Fantahun Admasu and Kumudha Raimond. Ethiopian sign language recognition using artificial neural network. In *2010 10th International Conference on Intelligent Systems Design and Applications*, pages 995–1000. IEEE, 2010.

[4] P Karthick, N Prathiba, VB Rekha, and S Thanalaxmi. Transforming indian sign language into text using leap motion. *International Journal of Innovative Research in Science, Engineering and Technology*, 3(4):5, 2014.

[5] Cao Dong, Ming C Leu, and Zhaozheng Yin. American sign language alphabet recognition using microsoft kinect. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 44–52, 2015.

[6] Anbarasi Rajamohan, R Hemavathy, and M Dhanalakshmi. Deaf-mute communication interpreter. *International Journal of Scientific Engineering and Technology*, 2(5):336–341, 2013.

[7] Helene Brashear, Thad Starner, Paul Lukowicz, and Holger Junker. Using multiple sensors for mobile sign language recognition. Georgia Institute of Technology, 2003.

[8] Leandro A Silva and Emilio Del-Moral-Hernandez. A som combined with knn for classification task. In *The 2011 International Joint Conference on Neural Networks*, pages 2368–2373. IEEE, 2011.

[9] Shadman Shahriar, Ashraf Siddiquee, Tanveerul Islam, Abesh Ghosh, Rajat

Chakraborty, Asir Intisar Khan, Celia Shahnaz, and Shaikh Anowarul Fattah. Real-time american sign language recognition using skin segmentation and image category classification with convolutional neural network and deep learning. In *TENCON 2018-2018 IEEE Region 10 Conference*, pages 1168–1171. IEEE, 2018.

[10] Arun Singh, Ankita Wadhawan, Manik Rakhra, Usha Mittal, Ahmed Al Ahdal, and Shambhu Kumar Jha. Indian sign language recognition system for dynamic signs. In *2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pages 1–6. IEEE, 2022.

[11] Shruti Chavan, Xinrui Yu, and Jafar Saniie. Convolutional neural network hand gesture recognition for american sign language. In *2021 IEEE International Conference on Electro Information Technology (EIT)*, pages 188–192. IEEE, 2021.

[12] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[13] Fran Jurišić, Ivan Filković, and Zoran Kalafatić. Multiple-dataset traffic sign classification with onecnn. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 614–618. IEEE, 2015.

[14] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[15] Shirin Sultana Shanta, Saif Taifur Anwar, and Md Rayhanul Kabir. Bangla sign language detection using sift and cnn. In *2018 9th international conference on computing, communication and networking technologies (ICCCNT)*, pages 1–6. IEEE, 2018.

[16] Sang-Ki Ko, Jae Gi Son, and Hyedong Jung. Sign language recognition with recurrent neural network using human keypoint detection. In *Proceedings of the 2018 conference on research in adaptive and convergent systems*, pages 326–328, 2018.

[17] Wei Lu, Zheng Tong, and Jinghui Chu. Dynamic hand gesture recognition with leap motion controller. *IEEE Signal Processing Letters*, 23(9):1188–1192, 2016.

[18] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7291–7299, 2017.

[19] Fitri Utaminingrum, Putra Pandu Adikara, Yuita Arum Sari, Dahnial Syauqy, and Anggi Gustiningsih Hapsani. Left-right head movement for controlling smart wheelchair by using centroid coordinates distance. *Journal of Theoretical & Applied Information Technology*, 96(10), 2018.

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Swapna Johnny and S Jaya Nirmala. Sign language translator using machine learning. *SN Computer Science*, 3(1):1–6, 2022.

[22] Mr S Shiva Prakash, Manasa Bandlamudi, and Ragitha Radhakrishnan. Educating and communicating with deaf learner's using cnn based sign language prediction system.

[23] Jian Wu, Lu Sun, and Roozbeh Jafari. A wearable system for recognizing american sign language in real-time using imu and surface emg sensors. *IEEE journal of biomedical and health informatics*, 20(5):1281–1290, 2016.

[24] Kshitij Bantupalli and Ying Xie. American sign language recognition using deep learning and computer vision. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4896–4899. IEEE, 2018.

[25] CJ Sruthi and A Lijiya. Signet: A deep learning based indian sign language recognition system. In *2019 International conference on communication and signal processing (ICCSP)*, pages 0596–0600. IEEE, 2019.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[27] Ahmed Kasapbaşi, Ahmed Eltayeb AHMED ELBUSHRA, AL-HARDANEE Omar,

and Arif Yilmaz. Deepaslr: A cnn based human computer interface for american sign language recognition for hearing-impaired individuals. *Computer Methods and Programs in Biomedicine Update*, 2:100048, 2022.

[28] G Anantha Rao, K Syamala, PVV Kishore, and ASCS Sastry. Deep convolutional neural networks for sign language recognition. In *2018 Conference on Signal Processing And Communication Engineering Systems (SPACES)*, pages 194–197. IEEE, 2018.

[29] PC Thirumal, K Sruthi, R Dhivya, and V Santhosh Sivan. A survey on sign language tanslator.

[30] Kayo Yin and Jesse Read. Better sign language translation with stmc-transformer. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 5975–5989, 2020.