

Software Engineering Department
ORT Braude College

Capstone Project Phase B – 61999

**American sign language (ASL) to text conversion
using CNN**

23-2-D-4

Submitters:

Michael Ohayon – Michael.Ohayon@e.braude.ac.il

Ido Mialy – Ido.Mialy@e.braude.ac.il

Supervisor:

Miri Weiss Cohen

Contents

<u>1. Abstract</u>	<u>3</u>
<u>2. Introduction</u>	<u>3</u>
<u>3. Related work</u>	<u>5</u>
<u>4. Background</u>	<u>7</u>
<u>4.1 YOLOv5</u>	<u>7</u>
<u>4.2 DenseNet121</u>	<u>10</u>
<u>4.3 Transfer learning.....</u>	<u>12</u>
<u>5. Research Process</u>	<u>13</u>
<u>6. Results and conclusions.....</u>	<u>21</u>
<u>7. Documentation</u>	<u>23</u>
<u>7.1 User Manual</u>	<u>23</u>
<u>7.2. Maintenance Manual</u>	<u>26</u>
<u>8. References</u>	<u>28</u>

1. Abstract

American Sign Language (ASL) is a vital form of communication for the Deaf and hard-of-hearing community. However, there exists a significant communication barrier between individuals who use ASL as their primary means of communication and those who are hearing. This barrier prevents effective communication, limits social inclusion, and hampers equal participation in various aspects of life. To address this challenge, we propose an innovative solution using deep learning techniques to develop an ASL recognition chat application. Our application uses live video input from webcams to recognize and interpret ASL gestures. It converts these gestures into text (or voice) output, allowing hearing individuals to better understand and communicate with ASL users in chat. This promotes inclusive communication and equal opportunities for all.

In our study, we use YOLOv5 for object detection and DenseNet121 with transfer learning for robust ASL recognition. YOLOv5 efficiently detects hand gestures in video frames, allowing us to focus on sign language recognition. To improve accuracy and generalization, we employ DenseNet121 with transfer learning, leveraging pre-trained weights to capture complex spatial patterns from ASL sign frames. This approach reduces the need for a large, labeled dataset, resulting in superior performance with limited training data. We will train the all model based on “ASL-DS” dataset we created from roboflow and evaluate our network accuracy according to our experiment Hyperparameters.

2. Introduction

The World Health Organization (WHO) reports that the number of people with hearing or listening disabilities has increased, with 466 million affected in early 2018 and an estimated 400 million by 2050. Sign language (SL) is a nonverbal communication language used primarily by those with hearing or listening disabilities, and different SLs exist for different nations. It is a gesture-based communication and a type of nonverbal communication that involves the use of physical movements and gestures to convey meaning. One of the most well-known forms of gesture-based communication is American Sign Language (ASL), which is a complete language used by many members of the Deaf community in the United States and other parts of the world. It is an incredibly important and unique language used by over 500,000 deaf and hard-of-hearing individuals. ASL is a visual language that uses hand shapes, facial expressions, gestures, and body language to communicate. It is a language that is completely distinct from English and like spoken languages, ASL has its own grammar, syntax, and vocabulary. In ASL, the position of the hands and the direction of movement are important, as they can change the meaning of a sign [1].

Its use is particularly important in the home and the workplace, where individuals can communicate with each other without the need for interpreters or other assistive technology. ASL is an incredibly expressive language that allows for a great deal of creativity and personal expression. It is a language that allows individuals to express themselves in ways that are both unique and meaningful. It is a powerful tool that can be used to bridge the gap between people of different backgrounds, cultures, and abilities [2]. Gesture-based communication is also important tool for people who are

not deaf, as it can be used to communicate in noisy or crowded environments, or in situations where verbal communication is not possible or appropriate. American Sign Language (ASL) is composed of both static and dynamic gestures, much like the American alphabet: The definition of a static sign is based on the configuration of the hand, whereas the definition of a dynamic gesture is based on the sequence of movements and configurations of the hand. Sometimes dynamic gestures are accompanied by body language and facial expressions. ASL also includes a huge range of dynamic gestures and combinations of gestures that emphasize specific words or phrases. This complexity enables ASL to pass on meaning with precision and nuance, making it an incredibly rich and powerful language. In Figure 1 and Figure 2, we can observe the distinction between static and dynamic gestures, along with examples illustrating the richness of the ASL language.

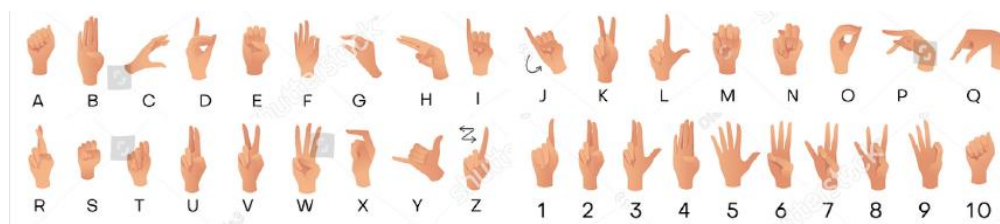


Figure 1: Static American Sign Language Hand Gestures (Letters and Numbers).



Figure 2: Dynamic American Sign Language Hand Gestures (Examples of Words).

To address these challenges, there are various solutions available today that aim to bridge the gap between those with hearing loss and those without. These solutions help to create connections and nurture understanding, and accessibility for all individuals regardless of their hearing abilities.

One of the many solutions was using a Third-party role that called Trained interpreter. The interpreter is the mediator between the hearing and non-hearing individuals, he translates the SL to speech and back by himself to each person. But this method is inefficient, upscale, and inconvenient as human attendance is mandatory. There are also hardware solutions like wearable devices that can recognize sign language such as hand gloves with flex sensor or accelerometer sensors, webcams, and Kinect as alternatives. On the other hand, these devices can be very expensive for some people and may not be accessible to everyone who needs them. Furthermore, the operation of these devices can be challenging for some people, adding to the complexity and expense of this solution. Therefore, we need a cost-effective solution that can help reduce the communication gap between individuals with hearing loss and with those without. A solution has been successfully developed to address the communication

barriers encountered by the hearing-impaired community. Through the creation of an innovative infrastructure capable of converting signs into both text messages, significant progress has been made in facilitating effective communication for this demographic. Additionally, ongoing advancements in gesture recognition technology have further bolstered these efforts, promising a higher quality of life for individuals with hearing impairments.

As a culmination of these endeavors, we proudly present our solution: the Hand Gesture Recognition (HGR) system chat application, an application designed to recognize sign language gestures accurately and efficiently. The system will focus on static HGR, which is used to recognize finger-spelled letters of American Sign Language. We will also want to achieve a recognition of dynamic HGR through videos or number of frames. In our study, we carefully evaluated multiple neural network models and identified YOLOv5 and DenseNet121 as the top performers. These two networks displayed superior performance across key metrics such as learning rate, adaptability, generalization, and resistance to overfitting. Now, we'll deploy them in real-world scenarios to assess their practical effectiveness. We carefully studied how fast each network learns from training data and adapts to new information. We also checked if they could generalize well and avoid overfitting. By analyzing each network's loss function over time, we measured the gap between predictions and real values. Then, we compared their accuracy and performance. Finally, we put the best model into a real-world app to see how well it works in practical situations.

3. Related work

Over the past few decades, lots of research is going on in gesture recognition as it can be used in various application domains. There is a major problem in every application and is to distinguish between the raw data into 2 main fields. The first one is static data, on standalone picture or frame that describe a letter or number in the SL. The second one is dynamic data that contain several continuous frames and together have a meaning (word or phrase). The system has no idea which type of data it should handle.

In the article Web Based Recognition and Translation of American Sign Language [3] there is a system for recognizing and translating American Sign Language (ASL) gestures using webcam footage. It offers predictions for static and dynamic gestures, allowing users to select the type of prediction via a website interface. For static gestures and numbers, webcam images are analyzed every second using CNN (VGG16), while dynamic gestures are processed through a 3DCNN-LSTM network on grayscale frames. Three datasets are used, covering static gestures, numbers, and dynamic gestures, with accuracy ranging from 97.50% to 99.5%. However, real-time accuracy for ASL letters captured by webcam is 90%, indicating room for improvement.

On the other hand, in the “A New Benchmark on American Sign Language Recognition using Convolutional Neural Network” article [4], the authors proposed newly convolutional neural network (CNN) model called SLRNet-8 to recognize sign language static gestures. They have been performed on the alphabet and numerals of four publicly available ASL datasets, the four datasets include the Massey University Gesture dataset [5], the sign language digit dataset [6], the ASL finger spelling dataset [7], and the ASL Alphabet dataset [8]. and the proposed CNN model significantly improves the recognition accuracy of ASL reported by some existing prominent methods. Previous methods for ASL recognition have reported their study on specific datasets, which makes it difficult to compare their effectiveness. To address this problem, the author considers four publicly available ASL datasets and studies the performance of the proposed CNN model on each dataset using either a separate train and test set or 10-fold cross-validation. The performance of the proposed method is also compared with previous methods on the same dataset and justified using cross-dataset testing. The study proposed an architecture of a CNN that called SLRNet-8 which maximizes their cognition accuracy. In Figure 3 we can see the SLRNet-8 architecture. SLRNet-8 consists of six convolution layers, three pooling layers and a fully connected layer besides the input output layers.

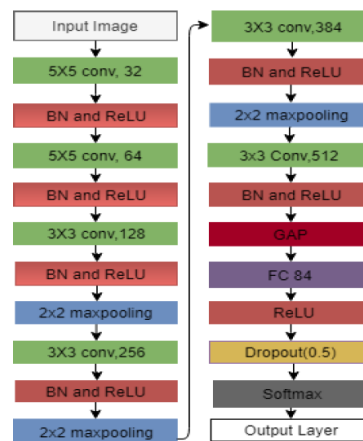


Figure 3: SLRNet-8 model architecture.

The study evaluated the proposed model's accuracy in recognizing ASL digits and alphabets using several datasets. Figure 4 presents an overview of SLRNet-8 performance, showcasing its accuracy across multiple datasets. The model achieved almost 100% accuracy in recognizing digits and alphabets of every dataset, except for 99.9% accuracy in recognizing the sign language digits dataset. The study also evaluated the model's performance on mixed datasets, where digits and alphabets were combined, and found a slight reduction in accuracy compared to individual recognition. The proposed model outperformed previous CNN-based models in terms of accuracy, achieving a 9% improvement on the same datasets.

Dataset	Category	Accuracy(%)
MU HandImages ASL	Alphanumeric	99.92
Sign Language Digits and ASL Alphabet	Alphanumeric	99.90
Sign Language Digits and Finger Spelling	Alphanumeric	99.90

Figure 4: SLRNet-8 Accuracy Results on Different Datasets for Digits and Spelling.

4. Background

4.1. YOLOv5

YOLO - “YOU ONLY LOOK ONCE” is an algorithm that uses neural networks to provide real-time object detection. This algorithm is popular because of its speed and accuracy. It has been used in various applications to detect objects quickly in both images and videos. In YOLO, one of the crucial tasks is Object Localization, exemplified in Figure 5. Object localization is the task of determining the position of an object using a bounding box (shown in Figure 6), while image segmentation involves partitioning an image into segments. Object detection combines classification, localization, and segmentation to correctly classify and localize objects in image [9].

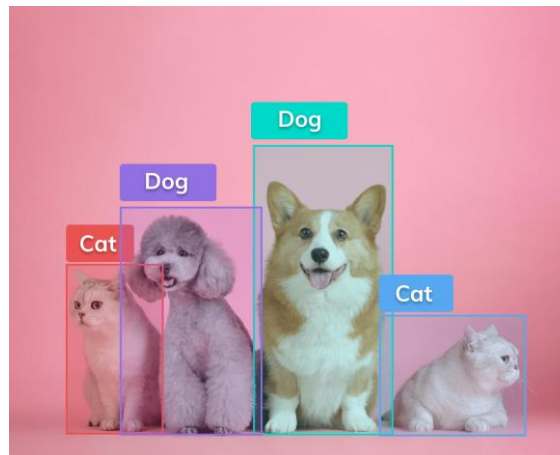


Figure 5: YOLOv5 Object Classification of Various Objects

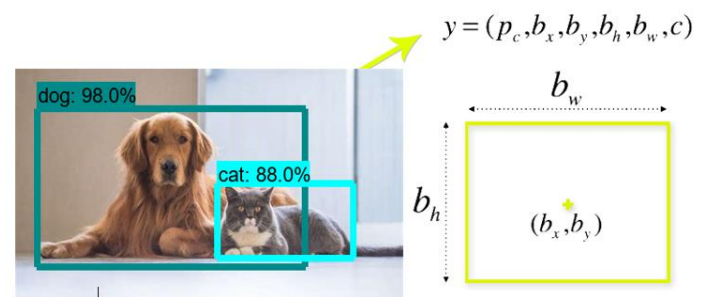


Figure 6: YOLOv5 output, where "bw" and "bh" denotes the width and height, "bx" and "by" indicate the coordinates of the center, and "c" signifies the predicted class.

There are two types of object detection models: two-stage object detectors and single-stage object detectors. Single-stage detectors predict objects directly in one pass, while two-stage detectors first generate proposals and then classify objects. Single-stage is faster but less accurate, while two-stage is more accurate but slower and more complex. The choice depends on the specific requirements.

The architecture of single-stage object detectors, such as YOLO, is typically divided into three main components: the backbone, the neck, and the head, as illustrated in Figure 7 (also referred to as Dense Prediction [10]):

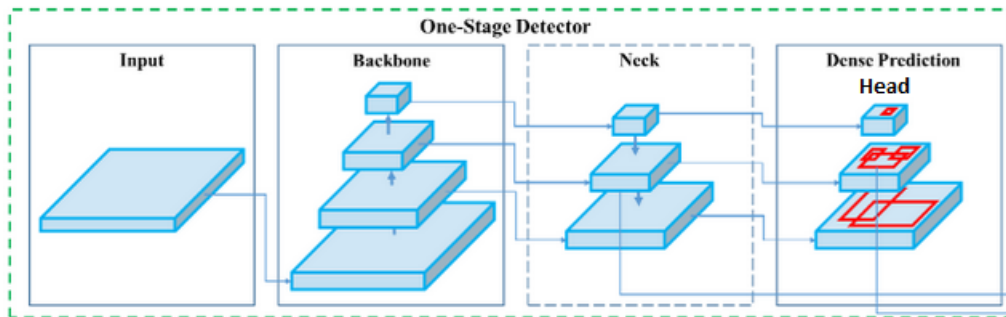


Figure 7: YOLOv5 Architecture of Single-Stage Detector.

Backbone: In object detection, the backbone network is responsible for processing the input image and extracting features that are relevant to object detection. The backbone network typically consists of multiple layers of convolutional, pooling, and activation functions that transform the input image into a set of feature maps. These feature maps represent the image at different levels of abstraction, with lower-level features such as edges and corners represented in earlier layers, and higher-level features such as object parts and shapes represented in later layers.

Neck: The neck in YOLOv5 connects the backbone to the head. It uses the Spatial Pyramid Pooling (SPP) module to perform multi-scale pooling operations on the feature maps and capture features at different scales. In the end, the neck refines the features extracted from the backbone for the next part of the object detection.

Head: The head is the final part of the YOLOv5 architecture, and it's responsible for predicting bounding boxes and class probabilities and adding them to the image or frame. YOLOv5 uses a fully convolutional head that predicts bounding boxes and class probabilities in a single pass. The head consists of several convolutional layers that output a tensor containing information about the predicted objects.

In addition, **Intersection over Union (IoU)** is a crucial metric in object detection that measures the degree of overlap between a predicted bounding box and the ground truth bounding box of an object shown in Figure 8. YOLOv5 utilizes IoU to precisely detect objects by producing an output bounding box that accurately encompasses the object.

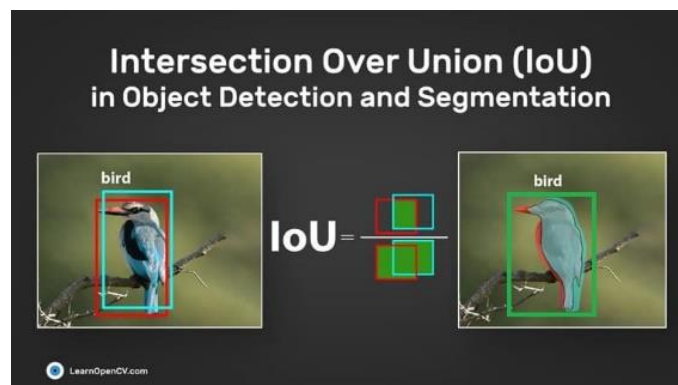


Figure 8: Intersection over Union illustration.

To demonstrate how the above-mentioned architecture works together, we will provide an example (Figure 9) that summarizes all the explanations.

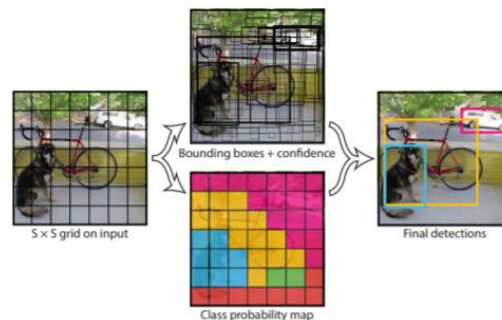


Figure 9: Intersection over Union (IoU) Divided by Several Classes

The image is divided into grid cells which forecast B bounding boxes and their confidence scores, along with predicting the class probabilities for each object. Intersection over union ensures that only bounding boxes that fit the objects perfectly are kept, resulting in a final detection consisting of unique bounding boxes. For example, a car is surrounded by a pink bounding box, a bicycle by a yellow bounding box, and a dog by a blue bounding box.

YOLOv5 in our project:

We utilized the YOLOv5 git repository, customizing it to suit our specific objectives.

1. PyTorch: Employed PyTorch to implement YOLOv5 and we use YOLOv5l6, leveraging its simplicity and flexibility for deep learning model development and training.
2. OpenCV: Utilized OpenCV, an open-source computer vision library, to integrate YOLOv5 for object detection tasks, benefiting from its extensive image processing capabilities.

In conclusion, The YOLOv5 algorithm aims to be highly accurate. It has been trained on dataset from roboflow and our own custom-made dataset resulting in improved accuracy. YOLOv5 is an impressive object detection tool that excels both in terms of speed and precision.

4.2. DenseNet

DenseNet is a deep neural network architecture in contrast to the other networks that connects each layer to the next layer the DenseNet connects each layer to every other layer in a feed-forward fashion as you can see in Figure 10. This connectivity pattern is called dense block, and it allows for more efficient feature reuse throughout the network. DenseNet has achieved state-of-the-art performance on several computer vision tasks, including image classification, object detection, and segmentation [11].

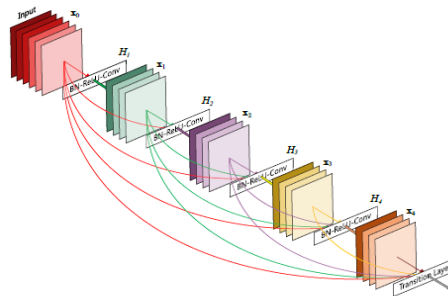


Figure 10: DenseNet basic-level architecture.

DenseNet architecture:

DenseNet is a unique neural network architecture that uses a different approach to data transformation compared to other standard networks. It achieves this by utilizing dense connections between layers. In a DenseNet, each layer receives the feature maps of all the preceding layers as input. This means that the output of each layer is passed as input to all the subsequent layers, creating a "dense block" of interconnected layers.

Dense block:

In DenseNet, a dense block is a set of layers where each layer is connected to every other layer in a feed-forward manner. In figure 11, The dense block takes the output of its preceding block as input and passes its output to the subsequent block, each layer in the dense block typically consists of a batch normalization layer, a rectified linear unit (ReLU) activation function, and a convolutional layer. The number of filters in the convolutional layer is usually kept constant throughout the block to ensure that the number of parameters in the network does not increase dramatically.

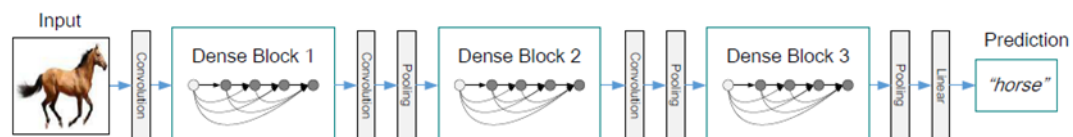


Figure 11: DenseNet low-level architecture with Dense blocks.

To reduce the number of hyperparameters in each layer of the dense block, composite functions are used for that purpose. , which consists of a batch normalization layer, a ReLU activation layer, a 3x3 convolutional layer, and a dropout layer.

Batch Normalization

In order to make this process more efficient and effective, DenseNet uses batch normalization to normalize the inputs of each layer by adjusting and scaling the activations, resulting in faster and more stable training. This normalization helps to mitigate the effect of covariate shift, which can cause the network to learn unstable or inaccurate features. The result is a highly connected network with a smaller number of parameters and state-of-the-art performance on various image classification tasks. The goal of this approach is to normalize the features (the output of each layer after passing activation) to a zero-mean state with a standard deviation of 1. In DenseNet we will use Mini-batch normalization that is a technique that performs batch normalization on smaller subsets of the training data, known as mini-batches, rather than the entire training dataset at once. By performing mini-batch normalization, the network can learn from the data more efficiently, and the training process can be faster and more effective.

$$\text{Mini - batch mean: } \mu = \frac{1}{m} \sum_{i=1}^m z^{(i)} \quad (1)$$

$$\text{Mini - batch variance: } \sigma^2 = \frac{1}{m} \sum_{i=1}^m (z^{(i)} - \mu)^2 \quad (2)$$

$$\text{Normalize: } z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad (3)$$

$$\text{Scale and shift: } \tilde{z}^{(i)} = \gamma z_{norm}^{(i)} + \beta \quad (4)$$

Equations (1)-(4): Batch Normalization calculations.

During mini-batch normalization, each mini-batch is normalized independently, and the mean (Equation 1) and standard deviation (Equation 2) of the mini-batch are used to normalize (Equation 3) the inputs. The parameters for normalization are updated for each mini batch, allowing the network to adapt to the changing statistics (Equation 4) of the input data.

The architecture comes in different versions such as DenseNet-121, DenseNet-151, and DenseNet-159, where the number following "DenseNet-" indicates the number of layers in the network (detailed in Figure 12). The higher the number, the deeper the network, and typically the better the performance, but at the expense of increased computational resources and longer training times.

DenseNet121 in our project:

- The DenseNet used in experiments has 3 dense blocks with an equal number of layers.
- A convolution with 16 output channels is performed before entering the first dense block.
- 1x1 convolution followed by 2x2 average pooling is used as transition layers between contiguous dense blocks.
- At the end of the last dense block, a global average pooling is performed and then a SoftMax classifier is attached.
- The feature-map sizes in the three dense blocks are 32x32, 16x16, and 8x8, respectively.

- DenseNet configurations tested are $\{L=40, k=12\}$, $\{L=100, k=12\}$, and $\{L=100, k=24\}$ for the basic structure and $\{L=100, k=12\}$, $\{L=250, k=24\}$, and $\{L=190, k=40\}$ for DenseNet-BC.
- In ImageNet experiments, a DenseNet-BC structure with 4 dense blocks on 224×224 input images is used.
- The initial convolution layer comprises $2k$ convolutions of size 7×7 with stride 2.

Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 36$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Figure 12: DenseNet Architecture Models Across Various Versions

4.3 Transfer learning

Machine learning technique where a pre-trained model is used as a starting point for solving a new task or problem. The pre-trained model has already been trained on a large dataset for a specific task, such as image classification or natural language processing, and has learned useful features that can be transferred to the new task. In transfer learning, the pre-trained model is typically fine-tuned on a smaller dataset for the new task, rather than being trained from scratch. This approach can significantly reduce the amount of data and time required to train a model from scratch, as well as improve the performance of the model on the new task.

In our project, we employed the fine-tuning transfer learning technique. This method involves adapting a pre-trained model to a new task by fine-tuning the weights of its layers while also training new layers on top of it. Unlike freezing the pre-trained model's weights, in fine-tuning, these weights are adjusted alongside the new layers during training. This approach proves particularly beneficial when the new task shares similarities with the original task for which the pre-trained model was developed, albeit with variations in data or task requirements.

Completed Transfer Learning Process with YOLOv5l6 and DenseNet121:

1. Pre-trained Weights Loading: We download pre-trained weights for YOLOv5l6.pt and DenseNet121.pt and initialize the YOLOv5 and DenseNet121 models with these pre-trained weights.
2. Model Setup: We initialize models with pre-trained weights and modify output layers. We modify the output layer(s) of both models to match the requirements of our specific task. we used torch.load_state_dict() to load the pre-trained weights.
3. Training: We fine-tune YOLOv5 for object detection using our dataset. We fine-tune the model by training on our dataset. During training, we update only the parameters of the modified layers, keeping the rest of the network frozen.
4. Optimization: We experiment with different learning rates (0.001-0.0001), optimizers(different resizes and Normalizations), and other hyperparameters (batch sizes, epochs) to further improve performance.

By employing transfer learning with YOLOv5l6 and DenseNet121 pre-trained weights, our model achieved improved performance on the new task while requiring fewer examples, thereby saving time and resources. This approach is particularly beneficial when dealing with smaller datasets for the new task, given the extensive pre-training on much larger datasets.

5. Research Process

Datasets:

- YOLOv5 Datasets:
This dataset is optimized for training the YOLOv5 network.
 - We used Roboflow dataset [12] that contain 792 images of hands gestures to help the model detect hands in images.
 - We supplemented our training with a custom dataset of 40 images. This addressed limitations in YOLOv5's performance on the Roboflow dataset, particularly in hand detection from varied angles. Integrating our dataset allowed for effective fine-tuning, improving the model's ability to detect hands accurately from any angles and images.
- DenseNet121 Datasets:
This dataset is curated to suit the requirements of DenseNet121 for our project.
 - We incorporated our custom dataset containing approximately 580 images of ASL letters to enhance the model's performance. We opted to create this dataset to address specific challenges we encountered with existing datasets, ensuring the model's proficiency in accurately recognizing American Sign Language gestures.

- We introduced three unique hand gestures, in Figure 13, that do not present in any other dataset, a deliberate choice to add functionally the model's training data and to our all chat applications:
 - SPACE: Adding a space between letters.
 - DELETE: Delete the last letters.
 - ENTER: Works as Enter to send the message.



Figure 13: Three unique hand gestures: Space, Delete and Enter.

"labellmg" [13] - Dataset creator tool:

To build a sizable dataset, we first memorized the ASL letters. Then, we captured photos of ourselves performing the gestures using a Python script. Next, we labeled these images appropriately. To streamline this labeling process, we utilized the "labellmg" tool (Figure 14), which provided a user-friendly interface for classification.

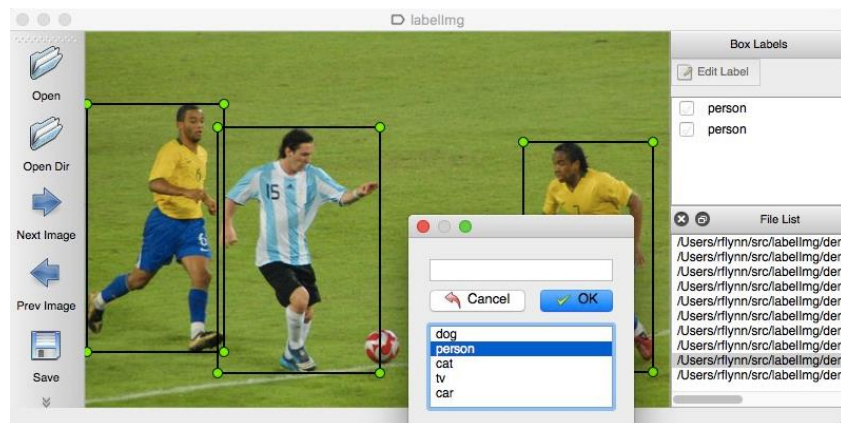


Figure 14: labellmg tool screenshot.

Training and Testing Approach:

For both YOLOv5 and DenseNet121 architectures, we employ a 70-30 split ratio for training and testing, utilizing both types of datasets to ensure comprehensive model evaluation.

Hyperparameters:

YOLOv5:

We are training our network with the following parameters:

1. Learning Rate(LR) - using the initial values: (0.01-0.001).
2. Epochs - 10.
3. Batch Size – 16.
4. Loss function – Adam with smooth L1 function.
5. Dropouts – The YOLOv5 does not use it.

The hyperparameters for training YOLOv5 were selected based on common ranges [14]. We chose a learning rate ranging from 0.01 to 0.001 to balance learning speed and stability. The number of epochs was set to 10 to allow sufficient iterations for convergence without overfitting. A batch size of 16 was chosen to balance computational efficiency and model stability during training. We utilized the Adam optimizer with a smooth L1 loss function, which is commonly used for object detection tasks. Overall, these hyperparameters were chosen through iterative testing to optimize performance and robustness of the model.

DenseNet121:

1. Learning Rate(LR) - using the value: (0.0001).
2. Epochs - 200.
3. Batch Size – 16.
4. Loss function – Adam with default Cross Entropy Loss function.
5. Dropouts – 0.2.

The hyperparameters for training DenseNet121 were carefully chosen from common ranges [15]. After testing LR values of 0.001 and 0.01, we settled on 0.0001 for stability. Training spanned 200 epochs to capture data patterns without overfitting. We choose a batch size of 16 rather than 32, we efficiently utilized resources. Employing the Adam optimizer with default Cross Entropy Loss ensured effective parameter optimization. Additionally, a dropout rate of 0.2 enhanced model generalization during training. These adjustments aimed to maximize performance on our task, ensuring accuracy and robustness in classification.

Detailed description of the project:

The project is a system that helps people who use American Sign Language (ASL) communicate more easily online. It will take videos of ASL and turn them into written text, making it simpler for ASL users to chat and connect online. This system could make a big difference in how ASL speakers engage in digital conversations, making online communication more accessible and inclusive for everyone. Our proposed method involves creating a server-client application that serves as a messaging platform. This application will offer the standard text messaging functionality while also providing the option for users to communicate using video to translate into text messages. The process involves capturing a frame from the webcam feed, identifying the hand using YOLOv5, and cropping the resulting image. This cropped image is then inputted into DenseNet121 for letter classification, determining the hand gesture. Finally, the identified letter is displayed in the message chat box area.

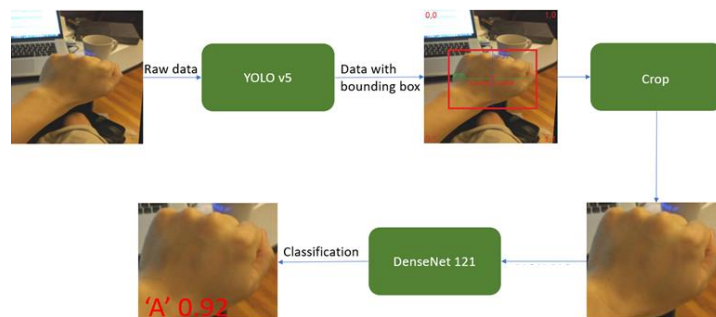


Figure 15: Flow chart of the proposed approach.

Figure 15 illustrates a process of our hand detection and ASL gesture classification using the YOLOv5 and DenseNet121 architectures:

1. **Raw Data:** The process begins with the raw input image, which in this case shows a person's hand holding an object above a keyboard.
2. **YOLOv5 hand detection:** The raw image is then processed by the YOLOv5 model, which stands for "You Only Look Once". This is a powerful machine learning model used for real-time object detection. YOLOv5 identifies and locates objects within the image and draws bounding boxes around them, which is depicted in the image as a red box around the hand.
3. **Crop:** The section of the image within the bounding box is then cropped out for further processing. This isolates the object of interest (the hand) from the rest of the image.
4. **DenseNet121 ASL gesture Classification:** The cropped image of the hand is then processed by DenseNet 121, which is a convolutional neural network used for image classification tasks. DenseNet 121 analyzes the cropped image to classify the object.
5. **Result:** The classification result is shown beneath the final image of the hand, indicating that the model has classified it with a label 'A' and a confidence score of 0.92, suggesting that the model is 92% confident in its classification.

To develop our project, we leveraged the power of Visual Code IDE that provides a user-friendly and feature-rich environment that enhances our productivity. We are also going to use the deep learning framework pyTorch. PyTorch Written in Python, it's relatively easy for most machine learning developers to learn and use. Moreover, to accelerate our model training process, we utilized the CUDA drivers. CUDA is a parallel computing platform that allows us to leverage the immense computational power of NVIDIA GPUs. Specifically, we took advantage of the NVIDIA RTX 3090 GPU. By utilizing CUDA and training our model on the RTX 3090 GPU, we significantly reduce the training time and achieve faster convergence, enabling us to iterate and experiment more efficiently.

In summary, by combining the Visual Code IDE, pyTorch, YOLOv5, DenseNet121 architectures, and harnessing the power of CUDA with the NVIDIA RTX 3090 GPU, we are poised to build a robust and efficient deep learning solution for our project.

Training process evaluations:

The webcam video feed will be processed by the YOLOv5 neural network, which will identify and isolate the user's hand movements. First, we trained the YOLOv5 on Roboflow dataset.

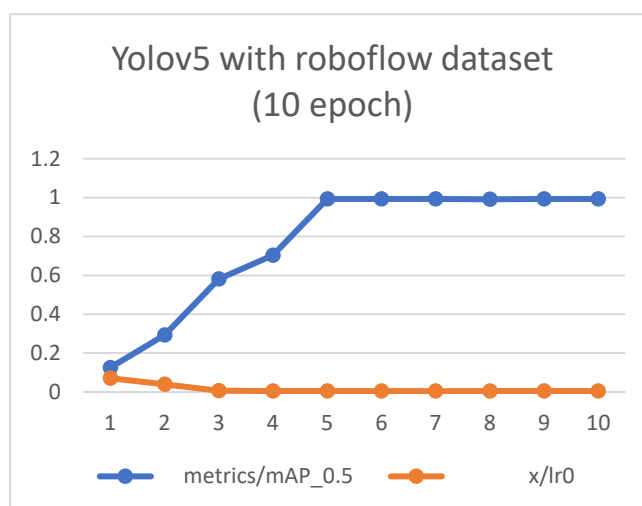


Figure 16: Yolov5 with roboflow dataset accuracy and loss of the yolov5 at 10 epochs.

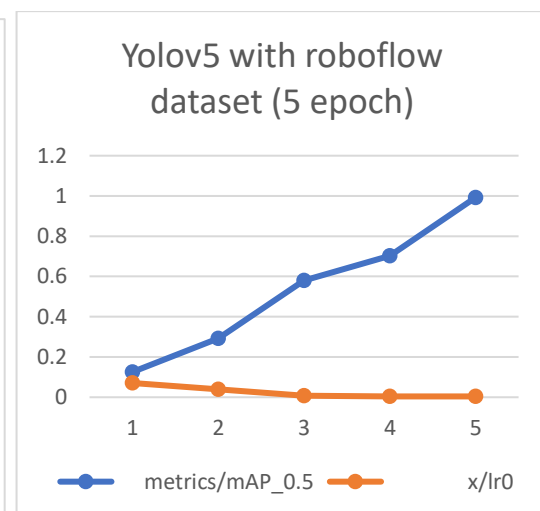


Figure 17: Yolov5 with roboflow dataset accuracy and loss of the yolov5 at 5 epochs.

In Figure 16, we initially trained YOLOv5 for 10 epochs and achieved commendable accuracy. However, upon closer inspection, we observed that the accuracy and loss plateaued during the last 5 epochs (Figure 17). Consequently, we opted to use the model trained for only 5 epochs, as it demonstrated comparable performance while requiring less computational resources. However, upon deploying the model on our webcam feed, we observed limitations in detecting certain movements and hand angles. To address this, we iterated on our training process by augmenting our dataset with an additional 40 custom-made images.

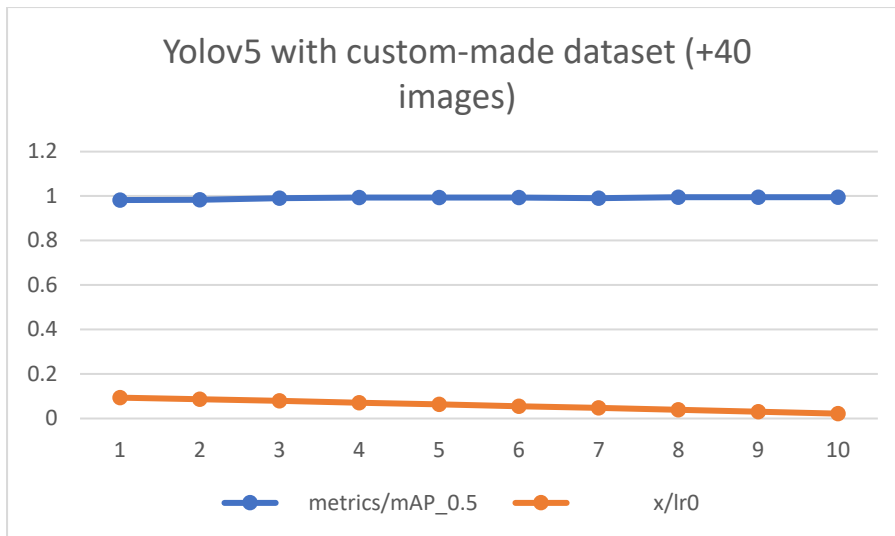


Figure 18: Yolov5 with custom-made dataset (+40 images) - accuracy and loss of the yolov5 at each epoch.

The second iteration of transfer learning, known as cascading, involves fine-tuning a pre-trained model obtained from the first iteration on a new, related task or dataset. This approach leverages the knowledge already captured in the pre-trained model to improve performance on the specific target task, requiring less computational resources compared to training from scratch. Figure 18 illustrates our retraining efforts. Despite our expectations, the overall accuracy remained unchanged or marginally improved, while the loss decreased notably for our custom dataset. Undeterred by the initial challenges, we kept at it and trained the YOLOv5 model again to make it better at recognizing different movements and hand positions. Subsequently, our enhanced YOLOv5 model now performs optimally for the subsequent stages of our project. After tracking hand movements, we want to use a DenseNet121 model to analyze them and classify each image to letter. But there was a problem: the sizes of the cropped images we are working with are not always the same. This causes problems because our DenseNet121 model needs all the images to be 224x224 pixels size. We ran into a problem because the sizes of the cut-out images were varied, as you can see in Figure 19.

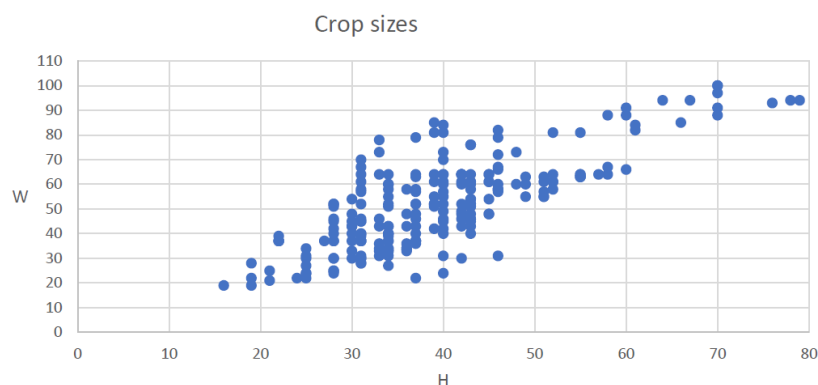


Figure 19: Yolov5 crop size values (width and height).

To fix this, we turned to some library in the PyTorch called "transforms." These are basically just ways to tweak or enhance our data before feeding it into the model. It's a common trick used to get our data ready for training or analyzing with neural networks. We specifically did this because if not, the DenseNet121 model would crop

or add zero-padding to fit its input layer, which is not ideal as it introduces many zeros into the image, potentially degrading the quality. By doing this, we make sure everything's the right size and shape, helping our model to get the right input size and work better.

To optimize the training process of our DenseNet121, we leveraged transfer learning by initializing with a pre-trained DenseNet121 model from PyTorch, rather than starting from scratch. This approach not only expedited the learning process but also significantly enhanced the accuracy of our model. By harnessing the knowledge learned from the pre-trained model, we were able to achieve superior performance while minimizing computational resources and training time. We proceeded by training our DenseNet121 model on our custom datasets, comprising approximately 580 images of American Sign Language (ASL) letters, over 200 epochs. Our focus was on optimizing hyperparameters such as learning rate and batch size to attain the highest accuracy and minimize loss.

#	lr	batch size	accuracy	loss
1	0.001	16	0.9137931034482759	0.0189
2	0.001	32	0.9827586206896552	0.046771
3	0.0001	16	1	0.0023
4	0.0001	32	0.993329845168556	0.046594664450

Table 1: DenseNet121 training with different parameters.

In Table 1, we present the results of various iterations, each representing a different combination of hyperparameters, showcasing the corresponding accuracy achieved. Notably, the optimal accuracy, as evidenced by categorical accuracy and loss metrics, was observed in line #3, where the learning rate (lr) was set at 0.0001 with a batch size of 16 images. Furthermore, we conducted an in-depth analysis of the loss fluctuations between the two most promising options: line #3 and line #4, during the initial 60 epochs, as illustrated in Figure 20.

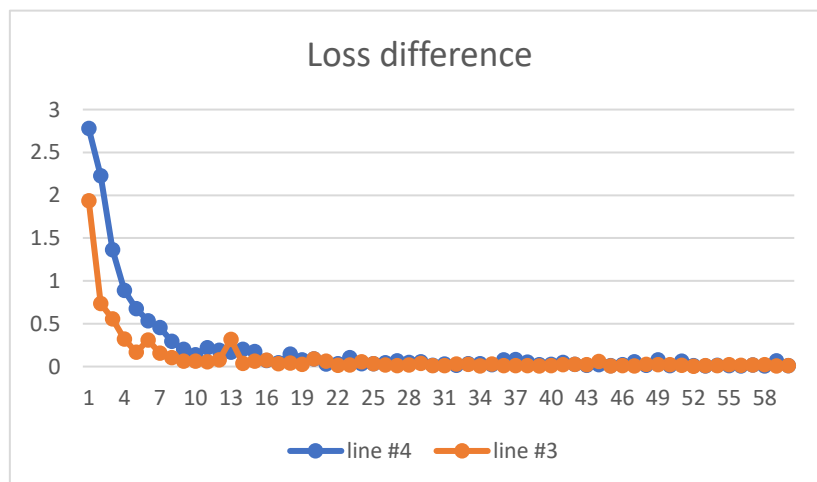


Figure 20: DenseNet121 Training Loss with custom-made dataset (approximately 580 images) with different parameters.

The DenseNet121 model has demonstrated remarkable performance, achieving a perfect 100% accuracy on test data and approximately 99% accuracy in real-time predictions from webcam feeds. Its proficiency lies in accurately predicting individual letters from input images, showcasing its exceptional capability in image recognition tasks. Once the DenseNet121 predicts the letters, by usage of our “predicted-letters-

history” algorithm, which continuously analyzes the last 10 frames of input to identify the most common letter with an accuracy ranging from 70% to 100%. The outputs of the algorithm are converted into text and displayed in the ASL user's text box. Finally, the user can send the message to the intended recipient. This innovative approach not only enhances communication for ASL users but also demonstrates the potential of combining cutting-edge technology with real-world applications.

Expected Challenges:

1. Dataset - Find enough images to train and test the DenseNet121 to achieve the most accurate result for the hand gestures classifier. Initially, we experimented with various datasets sourced from the internet. However, we encountered a significant issue where our model failed to detect the hand gestures accurately. To address this, we embarked on creating our custom dataset tailored to enhance the predictive accuracy of our model. While our custom dataset initially yielded promising results, we soon encountered another obstacle when users reported difficulties with the system due to overfitting. To overcome this challenge, we devised a strategy to create a mixed dataset that combines the initial dataset with our custom dataset, offering a more diverse and comprehensive range of training examples. This approach proved effective in mitigating overfitting while improving the overall performance of our model.

2. Real-Time Inference - In our model we want to achieve Real-time classify. Achieving low-latency predictions while maintaining accuracy can be challenging, requiring efficient model architectures, optimization techniques, and hardware acceleration. Our system is designed to operate in real-time, providing instantaneous letter detection with no visual latency for the user. This means that users can interact fluidly with the system without experiencing any delays in the detection of hand gestures and subsequent letter recognition. Our goal is to ensure a seamless and responsive user experience, allowing individuals to communicate effortlessly through our platform. We have successfully leveraged the power of the NVIDIA RTX 3090 GPU for both training and executing our system.

Unexpected Challenges:

1. Working and sharing platforms: After finishing part, A, we started building our system, which needed a strong setup. So, we looked for solutions that didn't rely on the user's specific setup. In our search, we found two possibilities. The first one was using Google Colab. But we decided against it because it required knowledge of different user interface components like PyQt and OpenCV, which weren't easy to work with. The second option was Jupyter Notebook, inspired by articles that we found online. But we realized it ran locally and depended on the user's hardware, so it wasn't ideal.

2. Integration: Even though we investigated each network separately (YOLOv5 and DenseNet121) and got them running on their own, we struggled to fit them into our project because of extra factors we hadn't thought about, like how long they took to run, the kinds of inputs they needed, and how they were trained. We could have used networks that work like black boxes, just taking input and giving output based on their training. But instead, we decided to tweak these models to better match what we needed for our project.

6. Results and conclusions

Our new ASL recognition tool has shown some promising results! We've been working hard to make it easier for ASL users and hearing people to communicate, and it looks like our efforts are paying off. By using advanced tech like YOLOv5 and DenseNet121, we're able to accurately detect ASL gestures in live video and translate them into text. This means that people who use ASL can chat with others more easily, breaking down barriers and promoting inclusion. Overall, we're excited about the potential of our system to make a real difference in people's lives, helping ASL users feel more connected and included in conversations and activities. Deciding on the right YOLOv5 and DenseNet versions was key for us, especially when balancing speed and accuracy in real-time. We had to find a sweet spot where our model could recognize ASL gestures quickly without sacrificing accuracy.

YOLOv5:

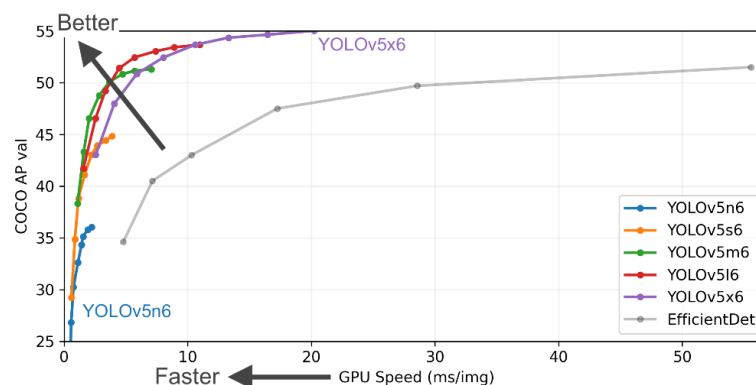


Figure 21: YOLOv5 Training Time on GPU for Various Model Sizes on COCO Dataset.

After trying out different options (Figure 21), we settled on YOLOv5l6. It's got a lot of parameters, but it runs super-fast, which was exactly what we needed for our ASL recognition app. This choice meant we could keep up with live video input while still getting accurate results, making communication smoother for everyone involved.

DenseNet:

DenseNet	Parameters	RunTime (ms)	Loss (avg)	Accuracy
DenseNet-121	8,000,000	20	0.2	0.96
DenseNet-169	14,000,000	28	0.189	0.965
DenseNet-201	20,000,000	34	0.162	0.971
DenseNet-264	25,000,000	42	0.128	0.976

Table 2: Real-Time Execution Time of DenseNet121 on GPU for Various Model Sizes

In Table 2, We found that as we moved up to higher versions like DenseNet-264, we gained more parameters, which usually means better performance potential. However, this also meant slightly longer run times. Ultimately, we chose DenseNet-121 as it struck a good balance between parameter count, runtime, and accuracy for our needs.

Table 3 below presents the system's results on various ASL gestures:






ASL Gestures	Results						
ASL letters	     						
3 Special Gestures DELETE, ENTER, SPACE	  						
LOOK-Alike Letters	<table> <tr> <td>FIST</td><td>    </td></tr> <tr> <td></td><td>   </td></tr> <tr> <td>2-finger</td><td>    </td></tr> </table>	FIST	  		 	2-finger	  
FIST	  						
	 						
2-finger	  						



Table 3: Real-Time System Performance Results

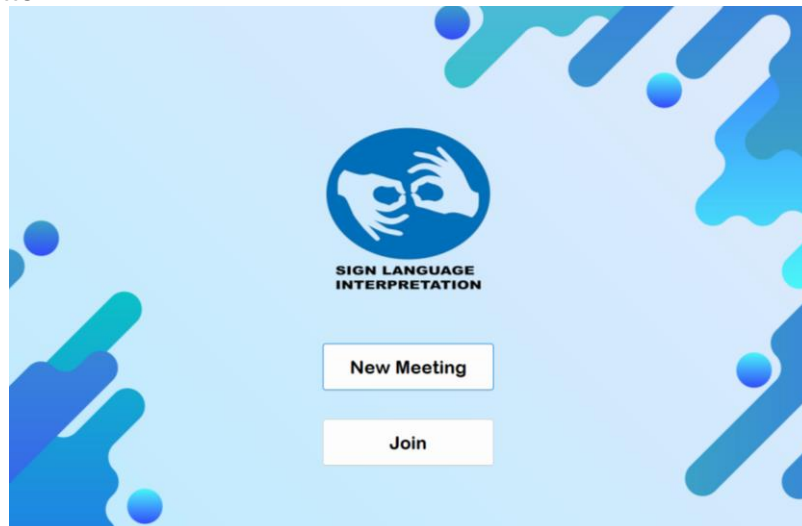
To tackle our challenges effectively, we broke down the main goal into several manageable sub-tasks. This approach allowed us to address each problem by focusing on smaller, more manageable tasks. Throughout the project, we made decisions carefully, ensuring that they complemented the progress we had already made and did not undermine our previous efforts. We successfully achieved the project goals by developing a system utilizing two deep networks. This system takes input from a webcam video, accurately identifies ASL gestures, and classifies them into corresponding letters.

7. Documentation

7.1. User Manual

1. Home Screen(Screenshot 1): Upon opening the application, users are presented with two primary options:

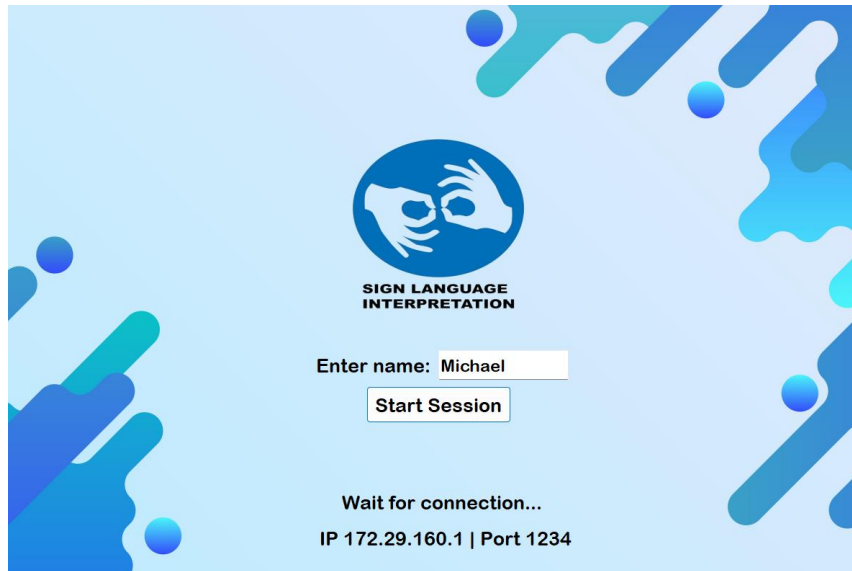
- “New Meeting”: This button allows users to initiate a new meeting session.
- “Join Meeting”: Users can opt to join an existing meeting session by clicking this button.



Screenshot 1: Home Screen.

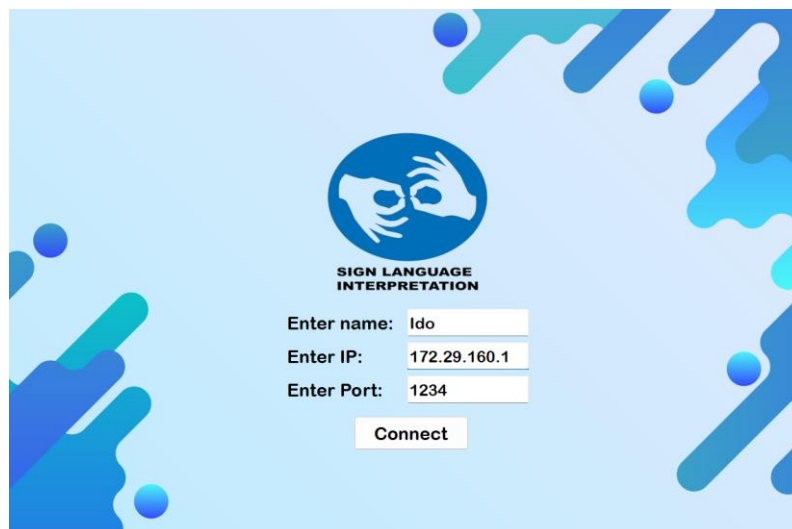
2. Connection screen(Screenshot 2):

After clicking "New Meeting": User will be prompted to enter their desired username. After entering the username, then press "Start Session" Button and a new session will be created, This IP and port can be shared with others to join the meeting.



Screenshot 2: Connection screen – New meeting.

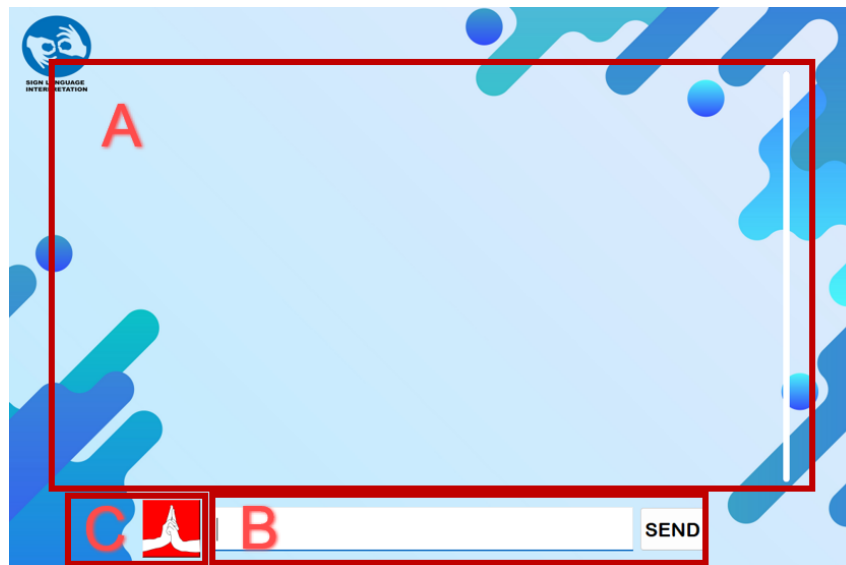
After clicking "Join": User will be prompted to enter their desired username and will be required to input the IP address and port number provided by the host of the meeting (Screenshot 3).



Screenshot 3: Connection screen – Join.

- When users select "New Meeting," a window opens for session creation. This window stays open until a participant presses the "Connect" button in the "Join" window.

3. Chat screen(Screenshot 4). After connection, both users' screens will transition to the chat screen automatically.



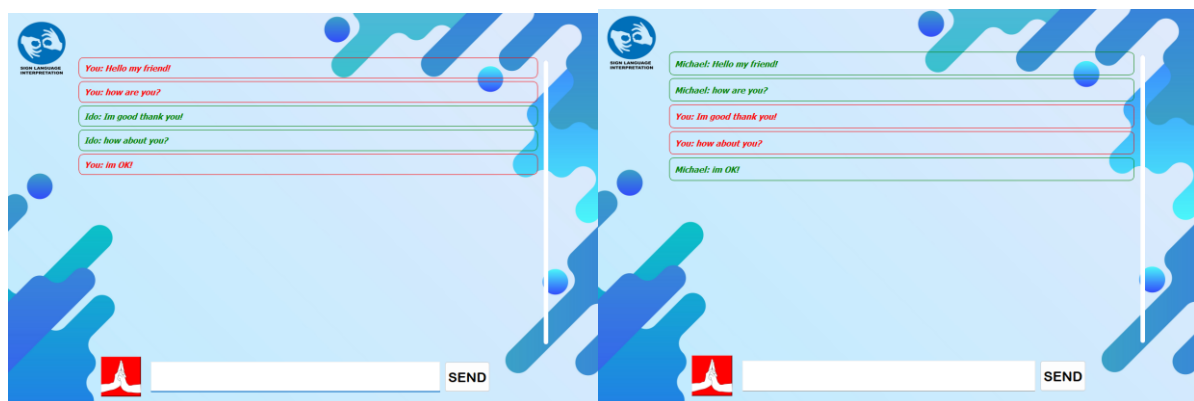
Screenshot 4: Chat screen.

Section A. Chat box - All messages exchanged between users will be displayed here. Also, the webcam feed will display in this area.

Section B .Personal text area - Users can input their messages here and send them by pressing the "Send" button or by pressing the ENTER key.

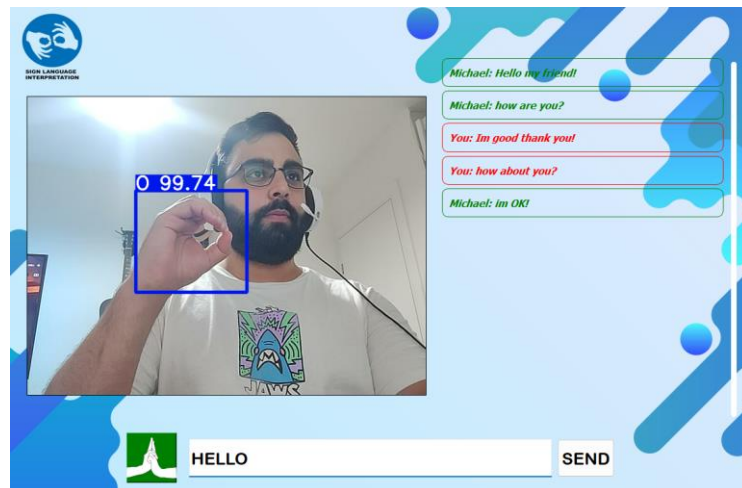
Section C. Activate ASL Translator Button - By activating this button, users can enable(green background)/disable(red background) the system to translate American Sign Language (ASL) gestures into text.

Demo session (ASL translator is disable) (Screenshot 5):



Screenshot 5: Chat screen with messages.

Demo session (ASL translator is enable) (Screenshot 6):

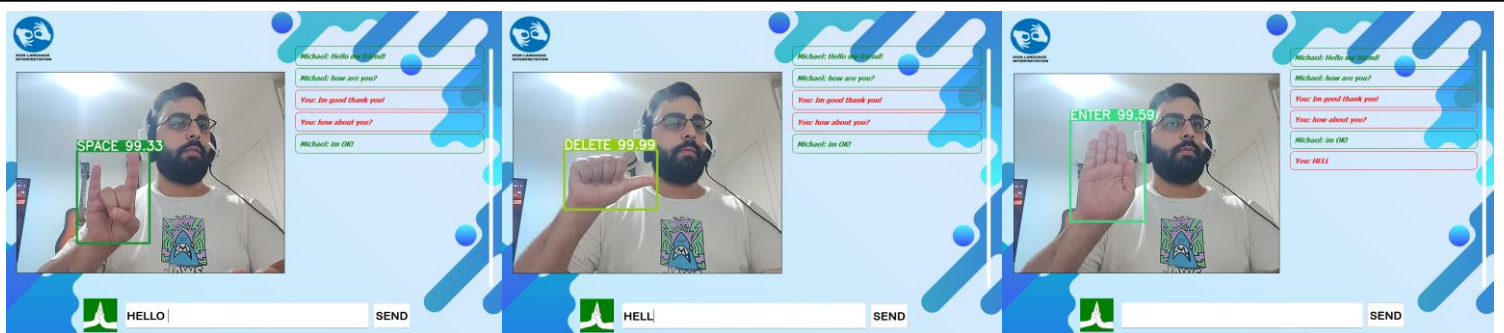


Screenshot 6: Chat screen with ASL translator.

Our system can accurately recognize all ASL letters (A-Z) along with three additional gestures (Screenshot 7):

1. SPACE: Adding a space between letters in the personal text area.
2. DELETE: This gesture allows users to delete the last letter in the personal text area.
3. ENTER: Users can send the text in the personal text area.

With these capabilities, our system ensures efficient communication through ASL translation.



Screenshot 6: Chat screen with three additional gestures.

7.2. Maintenance Manual

Maintaining YOLOv5 involves regularly updating the framework to ensure optimal performance and compatibility with evolving requirements. It's essential to stay informed about the latest developments and improvements in YOLOv5 architecture to enhance hand gesture detection and identification capabilities. Ensuring DenseNet121 efficiency requires monitoring the model's performance and accuracy over time. Keeping abreast of updates or advancements in DenseNet121 architecture is crucial to improve letter prediction accuracy. Fine-tuning the DenseNet121 model based on new research findings can enhance its effectiveness in converting hand movements into text.

For both YOLOv5 and DenseNet121, maintaining flexibility by training models with diverse datasets is essential. This ensures accommodation of different environmental conditions and user scenarios, allowing the system to adapt to various webcam resolutions, colors, and types of hands encountered in real-world scenarios. PyTorch updates and optimization are necessary to leverage new features and performance improvements. Continuously optimizing PyTorch code for training and inference tasks maximizes efficiency, particularly when using deep learning models like DenseNet121. Experimenting with PyTorch's capabilities can lead to potential enhancements in ASL translation accuracy and speed. Utilizing the NVIDIA RTX 3090 GPU involves regularly updating NVIDIA drivers for compatibility and performance optimization with PyTorch and CUDA. Monitoring GPU utilization and performance metrics helps identify potential bottlenecks or optimization opportunities. Leveraging the latest CUDA optimizations and features supported by the NVIDIA RTX 3090 GPU accelerates model training and inference. Continuous improvement and monitoring are essential aspects of maintaining an efficient ASL communication system. Regularly evaluating system performance metrics, soliciting feedback from users, and staying informed about advancements in ASL recognition and deep learning techniques ensure that the system remains up to date with the latest innovations in technology and research.

8.References

-
- [1] Kumar, Pradeep, et al. "A multimodal framework for sensor based sign language recognition." *Neurocomputing* 259 (2017): 21-38.
- [2] Munib, Qutaishat, et al. "American sign language (ASL) recognition based on Hough transform and neural networks." *Expert systems with Applications* 32.1 (2007): 24-37.
- [3] Bendarkar, Dhanashree, et al. "Web based recognition and translation of American sign language with CNN and RNN." (2021): 34-50.
- [4] Rahman, Md Moklesur, et al. "A new benchmark on american sign language recognition using convolutional neural network." *2019 International Conference on Sustainable Technologies for Industry 4.0 (STI)*. IEEE, 2019.
- [5] Barczak, Andre LC, et al. "A new 2D static hand gesture colour image dataset for ASL gestures." (2011).
- [6] Beşer, Fuat, et al. "Recognition of sign language using capsule networks." *2018 26th Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2018.
- [7] Garcia, Brandon, and Sigberto Alarcon Viesca. "Real-time American sign language recognition with convolutional neural networks." *Convolutional Neural Networks for Visual Recognition* 2.225-232 (2016)
- [8] Deza, Anna, and Danial Hasan. "MIE 324 final report: sign language recognition." *The computer engineering research group (the university of Totonto)* (2023).
- [9] Thuan, Do. "Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detention algorithm." (2021).
- [10] Diwan, Tausif, G. Anirudh, and Jitendra V. Tembhurne. "Object detection using YOLO: Challenges, architectural successors, datasets and applications." *multimedia Tools and Applications* 82.6 (2023): 9243-9275.
- [11] Huang, Gao, et al. "Densely connected convolutional networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.
- [12] <https://universe.roboflow.com/ncue-jcser/yolo-l2z3w>
- [13] <https://github.com/HumanSignal/labellmg>
- [14] https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/#dataset
- [15] Huang, Gao, et al. "Densely connected convolutional networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.