Software Engineering Department
ORT Braude College

Capstone Project Phase A – 61998

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

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 DenseNet 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 DenseNet 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

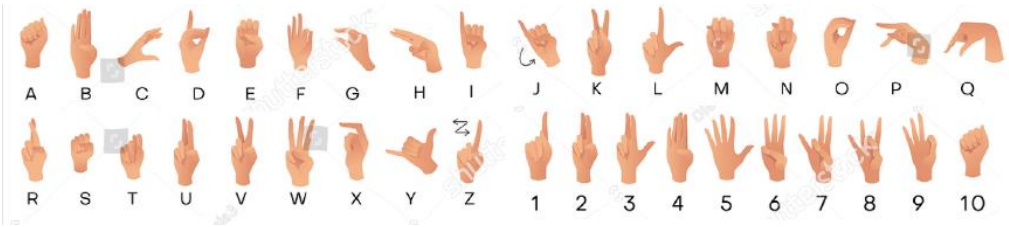
The World Health Organization (WHO) reports that the number of people with hearing or listening disabilities has increased, with 466 million affected in early 2018 and an estimated 400 million by 2050.
Sign language (SL) is a nonverbal communication language used primarily by those with hearing or listening 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.



Those gestures are only the basics of ASL.

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.

To address this challenge, 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.

The development of infrastructure that can convert signs into text and voice messages is proposed as a solution to the communication barriers faced by hearing-impaired individuals. The ongoing efforts to improve gesture recognition technology to support more effective communication is very important. Those efforts enhance the quality of life of those with hearing impairments.

Our proposed idea for solution to overcome this challenge is to develop an application that can recognize sign language gestures accurately and efficiently called hand gesture recognition (HGR) system.
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.

Our study will aim to identify the best neural network by conducting a thorough evaluation of several networks.
To evaluate the learning rate of each network, we will measure how quickly it can learn from the training dataset and how well it adapts to new data.
We will also assess the learning quality by evaluating the network's ability to generalize to new data and avoid overfitting.
In addition, we will analyze the loss function of each network (based on the epochs the learning part), which is used to quantify the difference between the network's predictions and the actual values. We will see the loss function of each network, compare the accuracy of each network, and determine which one has the best performance.
In the second part of our project, we will select the best one based on the above criteria. Once we have identified the best network, we will implement it in a real app to determine its performance in a real-world scenario.

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 its 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] the authors presented a client-server system that film the user via webcam and send the raw data and return the prediction of the gesture/s. Based on the footage from the webcam, the system sends single frame (static gestures) or 36 continues frames (dynamic gestures). In the UI of the website, the user needs to choose the option for what prediction type he want to have: static/ numbers /dynamic.

For the static gestures and numbers prediction - the webcam capture images have an interval of 1 second. In each interval the CNN (VGG16) determent the label by the highest probability class for the frame. If the predicted label of the gesture is the same for 5 continuous frames (5 seconds) then the predicted label is sent to the UI.

For the dynamic gestures prediction - the webcam capture video and send 36 continues frames, convert to grayscale frames to the 3DCNN-LSTM network the predicted label is sent to the UI.

There are three datasets for gesture recognition. The first dataset is for static gestures and contains 26 classes for letters A to Y (excluding J and Z) with 3000 images per class, resulting in a total of 78000 images. The second dataset is for number gestures and contains 10 classes for numbers 0 to 9, with 205 images per class, resulting in a total of 2050 images. The third dataset is for dynamic gestures and consists of 240 videos for each gesture, with 36 frames per video, for a total of 7 gestures performed by 4 different people in various background conditions.

The Accuracy on the test dataset of ASL letters is 97.50%, digits is 99.5%, and dynamic gestures is 98.81%. The accuracy tested against hand gestures for ASL letters captured by a webcam in real-time is 90%. The precision of the framework can be improved by appropriate enlightenment, and the accuracy of the framework may differ concerning intricate backgrounds.

Other use of 3D-CNN is the system that Shikhar Sharma and Krishan Kumar used in ASL-3DCNN:American sign language recognition technique article [4].
In this article, to overcome the problem of dynamic hand recognition, they used a four 3-D CNN fully connected layers cascaded for 4 different viewpoints to analyze multimodal information more accurately.
In each 3-D CNN layer they focused on particular viewpoint, trained each CNN and by that, they are increasing the efficiency of training. Once trained for all different viewpoints, it can be used to classify words pertaining to any one of four viewpoints.

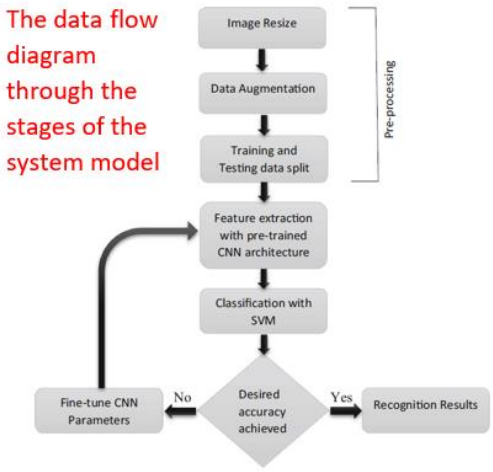
The raw date from the camera is pre-processed – they convert each video sequence into several frames (gray-scale image and filter noises) resize image and processing each frame individually.
Each CNN combined from multiple layers, stacking one layer on top of the previous layer, and where each layer recognizes the extended features provided the previous layer, with the final layer performing classification.

The Dataset used in this article is Boston ASL (Lexicon Video Dataset) LVD [5]
The CNN is trained for classification of 100 words on that dataset with more than 3300 English words signed by 6 different signers.

Model	Precision (%)	Recall (%)	F-measure (%)
Pentland	92.3	92.8	92.5
Cui	78.9	86.5	82.7
Uebersax	77.8	85.4	81.5
Koller	78.6	86.2	82.4
Proposed work	**96.0**	**97.1**	**96.4**

Table 1. shows the comparison of average precision, recall, and F-measure values obtained by previous known works and the article work different.

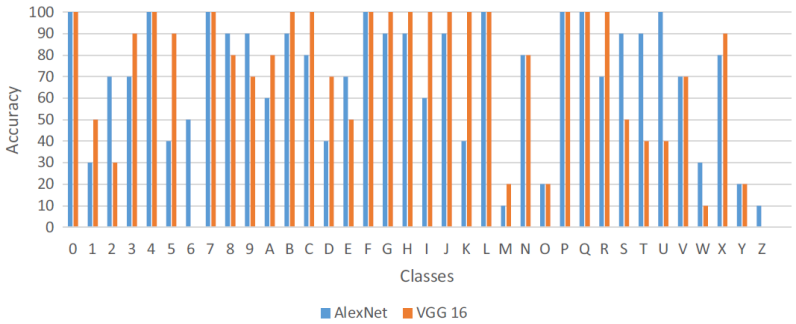
It can classify dynamic ASL hand signs into 100 different words.
The study showed the effectiveness of using convolution neural networks on both depths and intensity maps for ASL fingerspelling recognition.
The computing time of the proposed work is 0.19 seconds per frame, making it feasible for real-time applications.
The proposed work is the first to present the use of deep learning for ASL LVD recognition.
As we mentioned VGG16 is used to predict static images in the “CNN based feature extraction and classification for sign language” [6] the authors talk about 2 pre-trained CNN architecture VGG16 and AlexNet for transfer learning on a dataset.



In this work, the authors have used two CNN architectures, AlexNet and VGG16, for transfer learning on a dataset of American Sign Language (ASL) images. Initially, the pre-trained AlexNet and VGG16 models with weights from the ImageNet dataset were used for feature extraction, using the learned filters from the CNN architecture. The authors removed the final fully connected layers of the pre-trained networks, which contain information about the 1000 classes in the ImageNet dataset and added a new fully connected layer of 4096 neurons to collect the extracted features from the convolutional layers. Different fully connected layer features were investigated to obtain optimal recognition results. Finally, a multiclass support vector machine was applied on top of the extracted features for classification. The ASL dataset contains 36 classes of sign characters, with 70 images for each class, and the authors used both a random 70-30 and leave-one-subject-out validation approach to evaluate their models. Data augmentation was also used to increase the

number of images in the training data. Overall, the authors used transfer learning and fine-tuning techniques to adapt pre-trained CNN models for classification of ASL images.

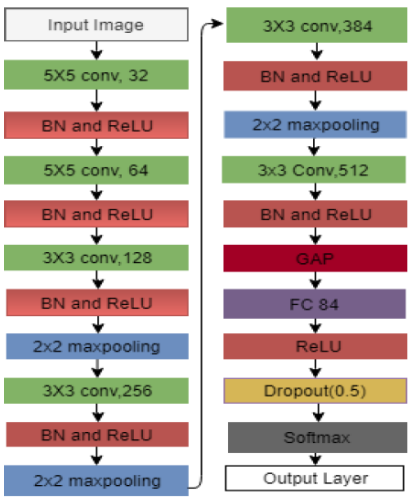
Both the modified pre-trained AlexNet and VGG16 showed promising recognition accuracy for most of the gesture classes except for a few misclassifications.



The characters that were misclassified in both models had high interclass similarities, and their recognition accuracy was lower when using a leave-one-subject-out validation method. However, the modified pre-trained VGG16 had a higher overall accuracy and fewer misclassifications compared to the modified pre-trained AlexNet, with only a few characters having incorrect predictions. The modified pre-trained VGG16 outperformed the state-of-the-art approaches and demonstrated robustness in recognizing 36 different static signs with lower error rates.

On the other hand, in the “A New Benchmark on American Sign Language Recognition using Convolutional Neural Network” article [7], 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 [8], the sign language digit dataset [9], the ASL finger spelling dataset [10], and the ASL Alphabet dataset [11]. 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.
SLRNet-8 consists of six convolution layers, three pooling layers and a fully connected layer besides the input output layers.

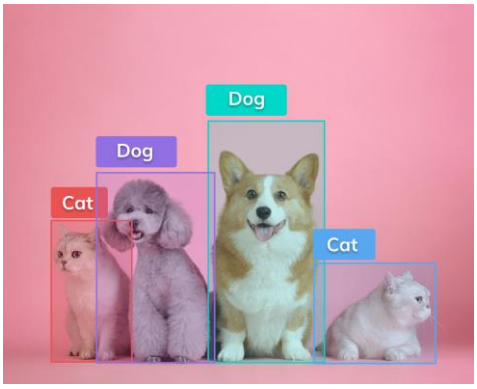
The study evaluated the proposed model's accuracy in recognizing ASL digits and alphabets using several datasets.

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

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.

4. Background

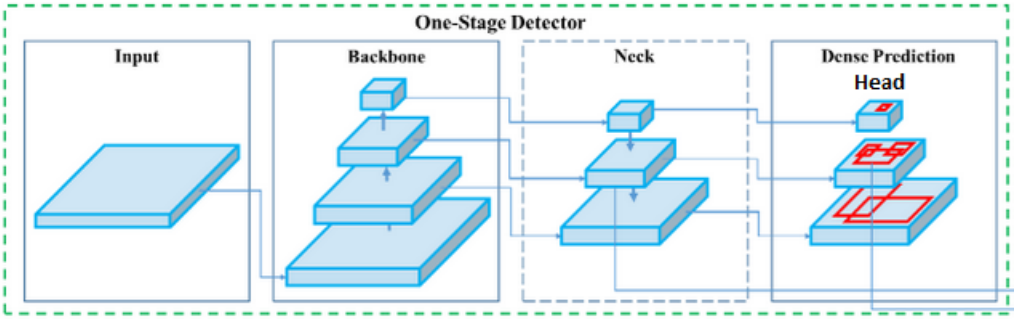
4.1. YOLO V5



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
One of the tasks in YOLO is Object Localization. Object localization is the task of determining the position of an object using a bounding box, while image segmentation involves partitioning an image into segments. Object detection combines classification, localization, and segmentation to correctly classify and localize objects in an image [12].

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.

Single-stage object detectors, like YOLO, architecture divided into three main components: the backbone, the neck, and the head (also called Dense prediction) [13]:



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.

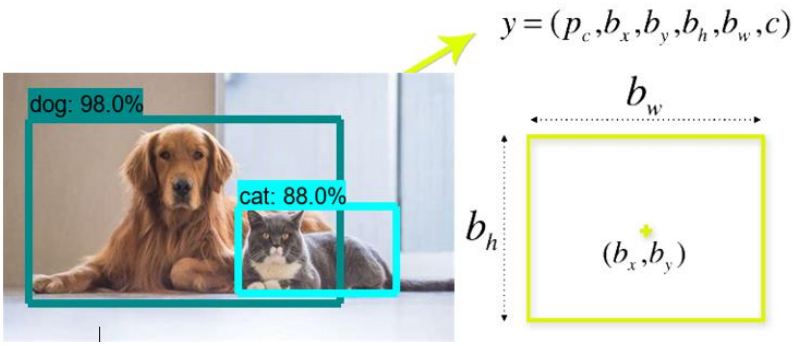
Neck:

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.

The YOLO algorithm output:

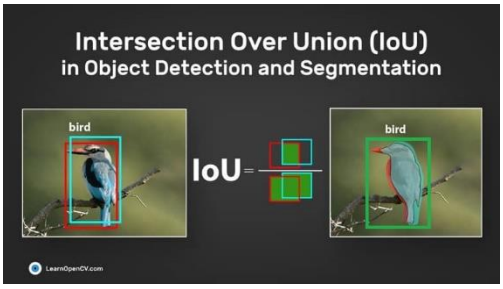


Width (**bw**)
Height (bh)

Class (for example, dog, cat, traffic light, etc.)- This is represented by the letter **c**.

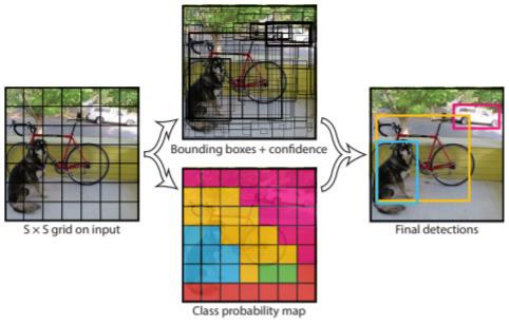
Bounding box center (**bx,by**)

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. YOLO utilizes IoU to precisely detect objects by producing an output bounding box that accurately encompasses the object.



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

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.

Implementation:

YOLOv5 can be implemented in a variety of programming languages and frameworks that support deep learning. Some popular choices include:

1. PyTorch: YOLOv5s is implemented in PyTorch, a popular deep learning framework that provides an easy-to-use interface for building and training deep learning models.

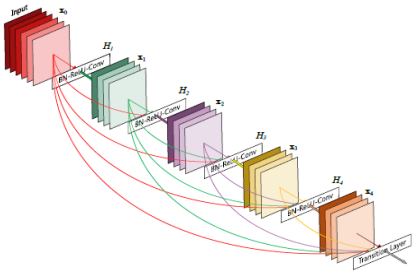
2. TensorFlow: TensorFlow is another popular deep learning framework that can be used to implement YOLOv5.

3. OpenCV: OpenCV is an open-source computer vision library that provides a variety of image processing functions and can be used to implement object detection models such as YOLOv5.

4. Darknet: YOLOv5 is also implemented in Darknet, an open-source neural network framework that provides support for object detection and other computer vision tasks.

In conclusion, The YOLOv5 algorithm aims to be highly accurate. It has been trained on multiple datasets across different domains such as COCO [14] and PASCAL VOC [15] resulting in improved accuracy. YOLOv5 is an impressive object detection tool that excels both in terms of speed and precision.

4.2. DenseNet



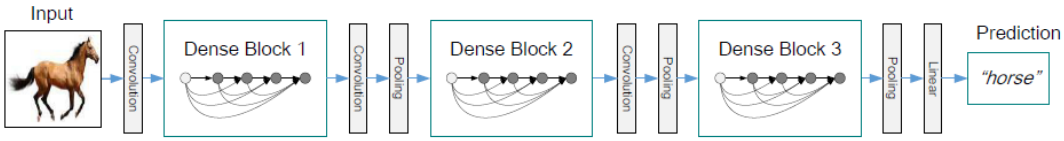
DenseNet is a deep neural network architecture in contrast to the other networks that connects each layer to the next layer the DenseNet connects each layer to every other layer in a feed-forward fashion. This connectivity pattern is called dense block, and it allows for more efficient feature reuse throughout the network. DenseNet has achieved state-of-the-art performance on several computer vision tasks, including image classification, object detection, and segmentation [16].

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. 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.



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.

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

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

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

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

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

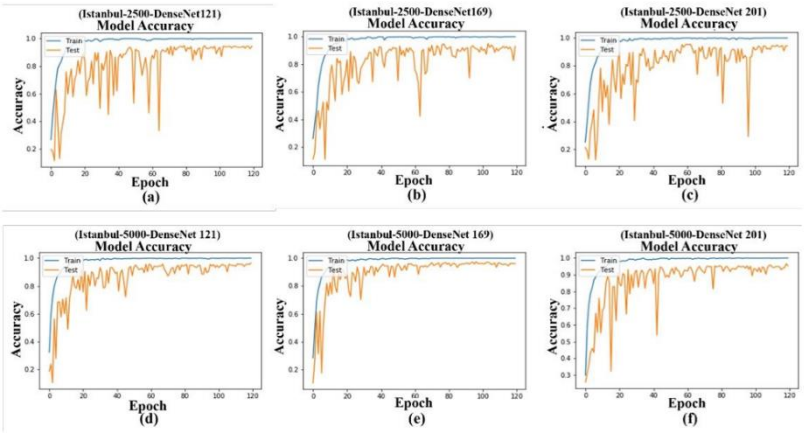
The architecture comes in different versions such as DenseNet-121, DenseNet-151, and DenseNet-159, where the number following "DenseNet-" indicates the number of layers in the network. The higher the number, the deeper the network, and typically the better the performance, but at the expense of increased computational resources and longer training times.
<u>One proposed Implementing of the architecture:</u>

- 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 224x224 input images is used.
- The initial convolution layer comprises 2k convolutions of size 7x7 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			

Benchmarking:

In the “A deep learning integrated mobile application for historic landmark recognition: A case study of Istanbul” article [17] ,they compare three types of DenseNet: as DenseNet-121, DenseNet-201, and DenseNet-169 on two datasets (Istanbul-2500 and Istanbul-5000)

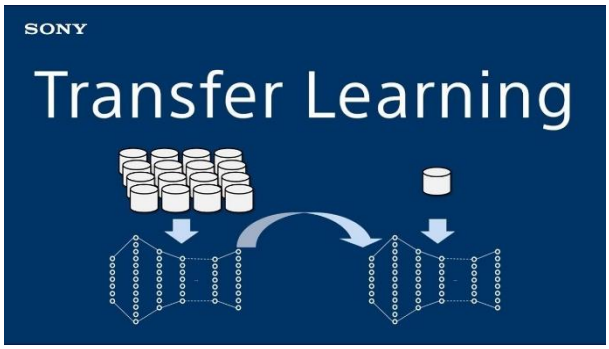


Increasing the number of layers in a neural network can improve its accuracy and efficiency. With more layers, the network can capture more complex features and patterns in the data, leading to better performance. Additionally, having more layers can enable the network to converge faster, requiring fewer training epochs to reach a satisfactory level of accuracy.

In conclusion, DenseNet is a neural network architecture that achieves state-of-the-art performance on various image classification tasks. Its key advantages include better accuracy with fewer parameters, efficient use of feature maps, and better feature propagation throughout the network.

4.3 Transfer learning

Transfer learning is a 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.
The difference types of transfer learning:

- Feature extraction transfer learning: In this approach, the pre-trained model is used as a fixed feature extractor, and only the weights of the new, fully connected layer(s) that are added on top of the pre-trained model are trained for the new task. The pre-trained model's convolutional layers are used to extract useful features from the input data, and those features are then fed into the new layers. This approach can be useful when the new task is like the task on which the pre-trained model was originally trained.

- Fine-tuning transfer learning: In this approach, the pre-trained model is adapted for the new task by fine-tuning the weights of the pre-trained model's layers in addition to training new layers on top of the pre-trained model. The pre-trained model's weights are not frozen during training of the new layers, but rather are updated along with the new layers. This approach can be useful when the new task is related to the task on which the pre-trained model was originally trained, but with some differences in the data or task requirements.

- Multi-task transfer learning: In this approach, the pre-trained model is used to learn features that are useful for multiple related tasks. The pre-trained model is trained on a large dataset that includes examples from multiple tasks, and the learned features are then used as a starting point for learning a new set of related tasks. This approach can be useful when there are multiple tasks that share some underlying structure or features.

- Domain adaptation transfer learning: In this approach, the pre-trained model is adapted for a new domain, where the distribution of the data is different from the distribution of the data on which the pre-trained model was trained. The pre-trained model is used to learn general features from the original domain, and those features are then adapted to the new domain through additional training on a smaller dataset of the new domain. This approach can be useful when the data from the new domain is limited, and there are differences in the data distribution between the two domains.

The common process of transfer steps:

- Pre-training: The pre-trained model is trained on a large dataset for a specific task. This training process usually involves using many examples to learn a set of features that can be used for the new task.

- Feature extraction: The learned features from the pre-trained model are extracted and used as inputs for the new model.

- Fine-tuning: The new model is trained on a smaller dataset for the new task. During this process, the weights of the pre-trained model are adjusted to better fit the new task.

- Evaluation: The new model is evaluated on a separate validation or test set to measure its performance.

By using transfer learning, a model can be trained with fewer examples, which can save time and resources. It can also improve the performance of the model on the new task, especially when the new task has a smaller dataset, and the pre-trained model has been trained on a much larger dataset.

5. Research Process

5.1. Process

Challenges:
1. Dataset : find the right data for our task and model – find enough images to train and test the DenseNet to achieve the most accurate result for the hand gestures classifier.

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.
we will try to train and execute our application on RTX-3090 (Invidia) GPU.

3. Lack of knowledge: We had no knowledge on sign language gestures. We studied ASL by ourselves to achieve some knowledge we read articles about ASL.

Our research process involves developing an application that can recognize sign language gestures. In the background section, we explain each subsystem individually. Now, we are going to integrate them into one system and describe their role in the overall system.

YOLOv5:
For implementing YOLOv5 we will be using PyTorch.
The original YOLOv5 repository, maintained by Ultralytics, is implemented in PyTorch. Using PyTorch for implementing YOLOv5 provides several advantages. PyTorch's intuitive Pythonic interface makes code development easier, and its strong community support offers extensive documentation and resources. PyTorch's flexibility and extensibility enable easy modifications to YOLOv5 models, while its efficient GPU utilization ensures faster training and inference times. Additionally, PyTorch hosts the official YOLOv5 implementation, guaranteeing compatibility and continuous updates, making it a preferred choice for YOLOv5 implementation.

To train the YOLOv5, we generate “American Sign Language Letters Dataset” [18] from roboflow (we refer to this Dataset as “ASL-DS”).

This dataset includes 1728 images of letters are annotated in YOLO v5 PyTorch format. The following pre-processing was applied to each image:
* Auto-orientation of pixel data (with EXIF-orientation stripping)
* Resize to 416x416 (Stretch)

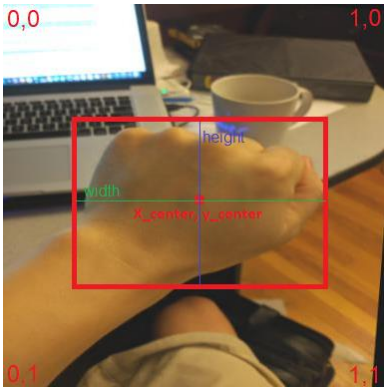
The following augmentation was applied to create 3 versions of each source image:
* 50% probability of horizontal flip
* Randomly crop between 0 and 20 percent of the image
* Random rotation of between -5 and +5 degrees
* Random shear of between -5° to +5° horizontally and -5° to +5° vertically
* Random brightness adjustment of between -25 and +25 percent
* Random Gaussian blur of between 0 and 1.25 pixels

in ASL-DL we divided the data to 3 parts: train, valid, test
1. Train – 1512 images and labels – 87.5% of the dataset
2. Valid – 144 images and labels – 8.33% of the dataset
3. Test – 72 images and labels – 4.166% of the dataset

in each part: the labels will look like that:

```
0 0.5268420439811305 0.5070755689714356 0.7004771178074257 0.44822516231696535
```

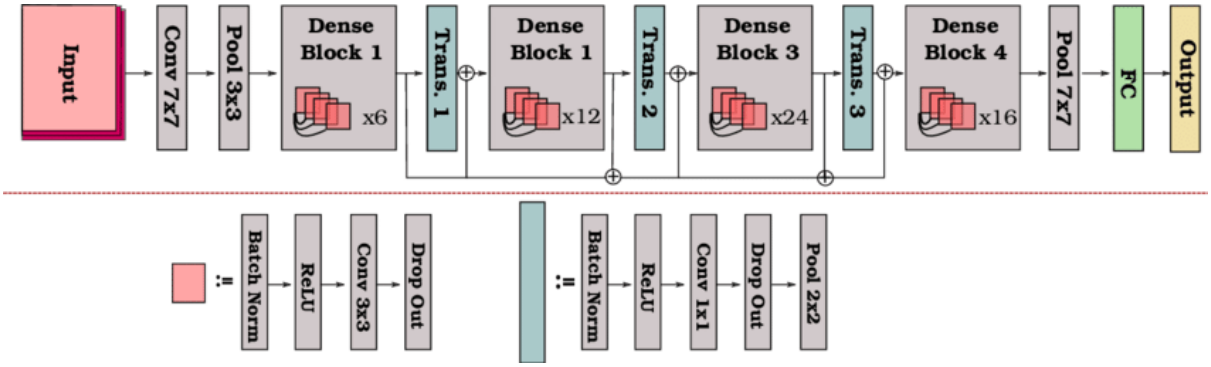
Each row is <class> <x_center> <y_center> <width> < height > format.



In our model, using the YOLOv5 will be to detect the hand gestures in the frame and then take the boundaries that the YOLOv5 was detected and crop this part and transfer it to the next sub-model of the DenseNet.

DenseNet:
We will use DenseNet-121 and for implementing DenseNet-121 we will be using PyTorch as well.

DenseNet-121 has four dense blocks with 6, 12, 24, and 16 layers, respectively. The growth rate is set to 32, meaning each layer within a dense block adds 32 feature maps. The total number of layers in DenseNet-121 is 121.



To implement DenseNet in Python, we will utilize deep learning frameworks in PyTorch, which provide pre-defined DenseNet architectures and modules.

Hyperparameters:
We are training our network with the following parameters:
1. Learning Rate(LR) - using the values: (0.000001, 0.00001, 0.0001)
2. Epochs - check sizes of 50 to 100.
3. Batch Size - check sizes of 8 to 32 .
4. Loss function – we will use cross-entropy function and Adam optimizer.
5. Dropouts – check probability of an element to be zeroed of 0.2 to 0.5

Training steps:

1. Dataset: we will use the ASL-DS that divided into three purposes

2. Define the Loss Function: we will select loss function that specified to detect hand gestures based on cross-entropy loss.

3. Optimizer: we will select an optimizer to update the model's parameters during training based on Adam optimizer

4. Training Loop: Implement the training loop, which consists of the following steps:
   - Iterate over the batches of data from the training function .
   - Pass the inputs through the model and obtain the predictions.
   - Compute the loss between the predictions and the ground truth labels.
   - Backpropagate the gradients and update the model's parameters using the optimizer.
   - Repeat these steps until we have processed all the batches from the training dataset.

5. Validation Loop: After each training epoch or a specified number of iterations, we will evaluate the model's performance on the validation dataset.
   Like the training loop, we will iterate over the validation data loader, and pass the inputs through the model, and calculate the validation loss.

6. Hyperparameter Tuning: Experiment with different hyperparameter values (learning rate, batch size, number of training epochs) to optimize the model's performance. Here we use the transfer learning technique to have initial values to the Hyperparameter weights instead of randomized them.

7. Save the Model: Save the trained model's parameters for future use or inference. we will use ".pth" file to save the Hyperparameter weights.

Implementation:
To develop our project, we have decided to leverage the power of PyCharm 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.

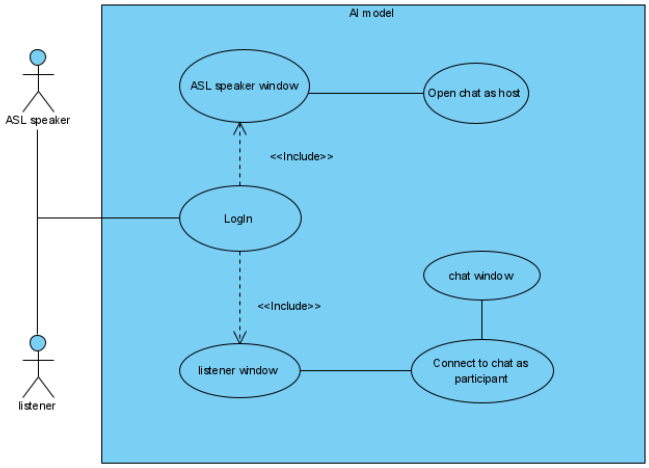
For our deep learning tasks, we have chosen to incorporate the Yolov5 and DenseNet121 architectures. To access the Yolov5 and DenseNet121 architectures, we will clone their repositories from Git. By doing so, we can benefit from the latest updates, bug fixes, and enhancements provided by the open-source community.

Moreover, to accelerate our model training process, we plan to utilize the CUDA drivers. CUDA is a parallel computing platform that allows us to leverage the immense computational power of NVIDIA GPUs. Specifically, we will take advantage of the NVIDIA RTX 3090 GPU. By utilizing CUDA and training our model on the RTX 3090 GPU, we can significantly reduce the training time and achieve faster convergence, enabling us to iterate and experiment more efficiently.

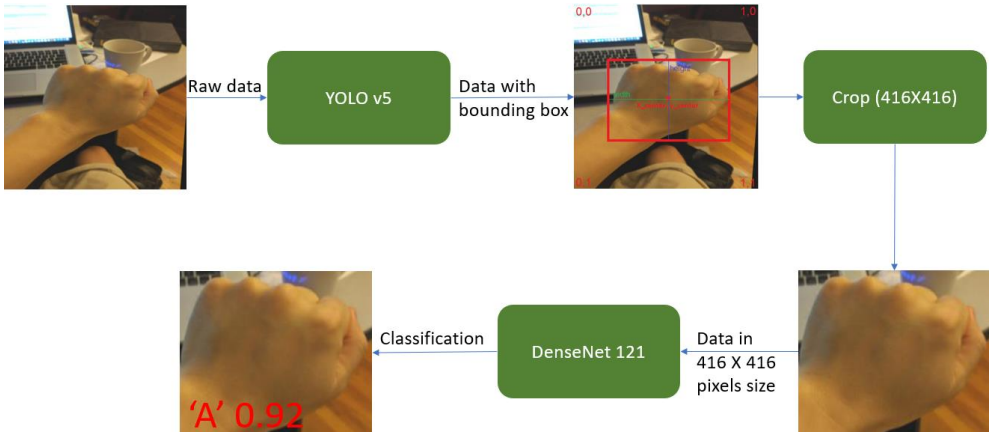
In summary, by combining the PyCharm IDE, pyTorch, Yolov5, DenseNet architectures, and harnessing the power of CUDA with the NVIDIA RTX 3090 GPU, we are poised to build a robust and efficient deep learning solution for our project.

5.2. Product

Use case:

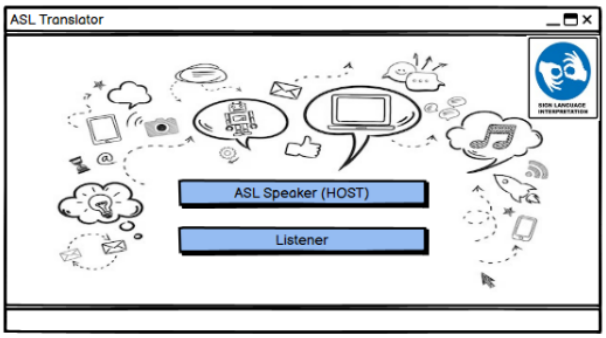


Flowchart:

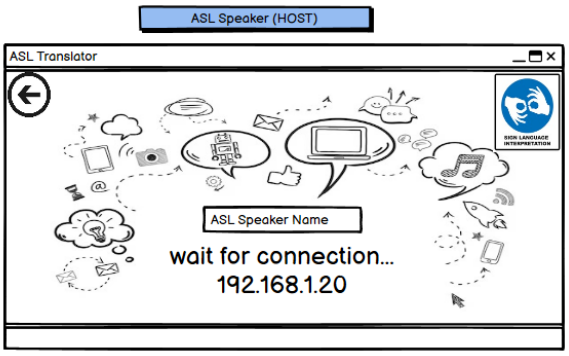


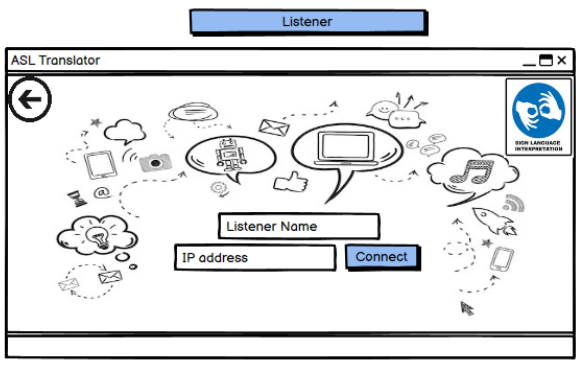
GUI:

ASL Translator app – Home Page:
1. ASL Speaker – Open the chat as host
2. Listener – Enter the chat as client.

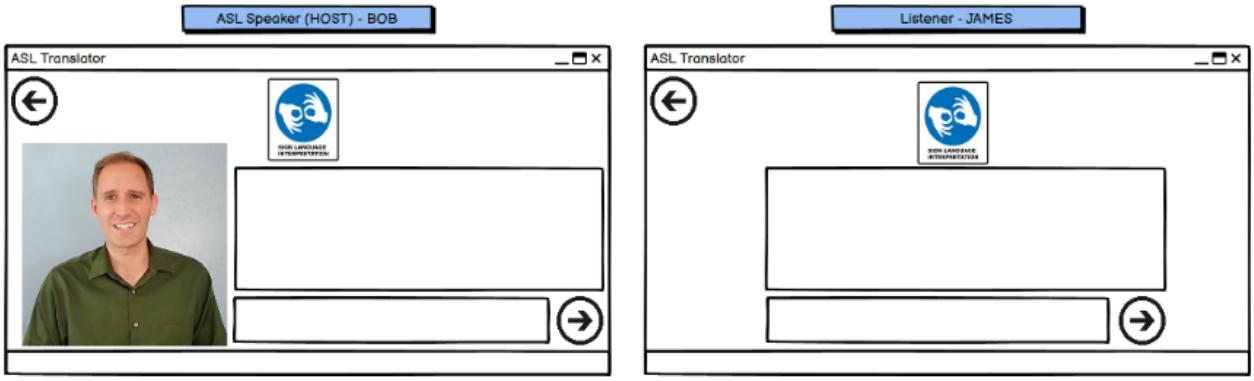


The login screens for each option:



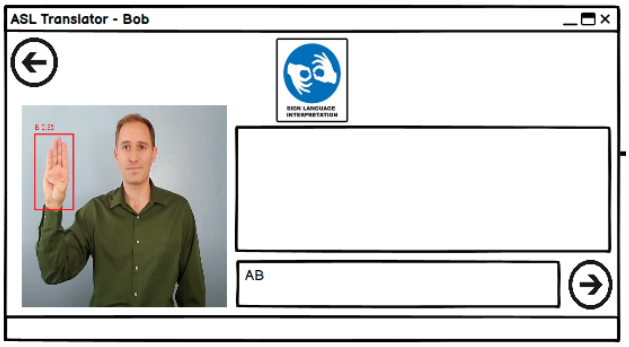


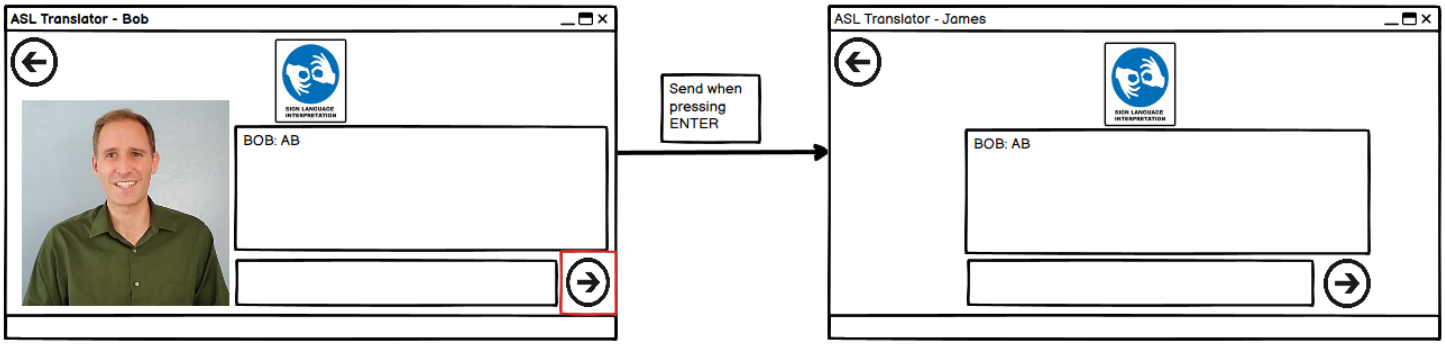
After filling names and connect to the IP address shown to the host, new chat windows will be opens to Bob (ASL speaker) and James (listener):



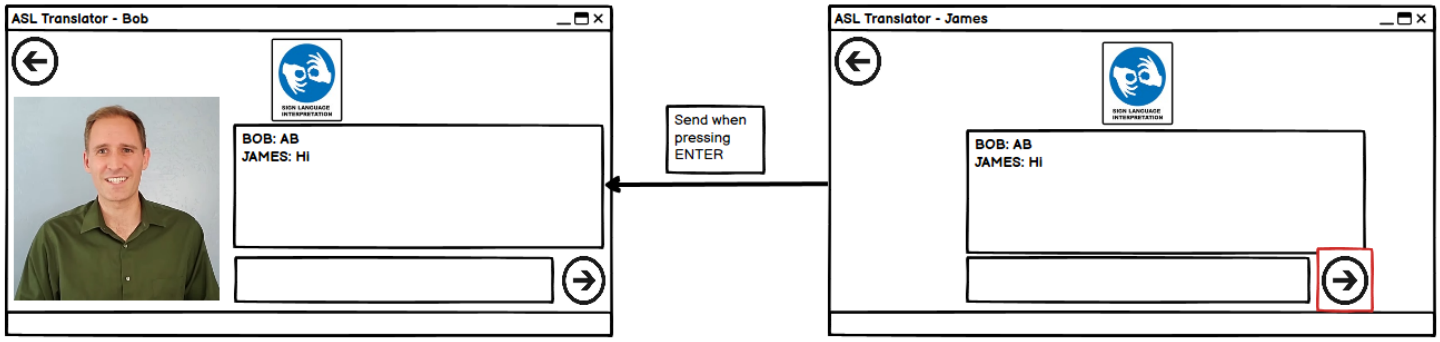
Sending message flow:

1. Form Bob(ASL speaker) to James(Listener):
   - Bob will do the signs for letters, for every letter after 3 sec identification the letter will be written in his personal chat box.
   - After writing what he wants, bob will the “send” button.





2. Form James to Bob: works as a regular chat – James will type text and will press the “send” button.



6. Evaluation/Verification Plan

CASE	Test explanation	Expected result
1.Home Page		
1.1	Press on “ASL SPEAKER” button	Open new window calls “waiting for connection”
1.2	Press on “LISTENER” button	Open new window calls “Connect via IP”
2.1. ASL Speaker Login		
2.1.1	Press “Return " button	Return to Home page
2.2. Listener Login		
2.2.1	Press “Return " button	Return to Home page
2.2.3	Press connected without filling IP textbox	Throws error: “Type Valid IP”
2.2.4	Press connected with not valid IP	Throws error: “IP not valid, try again”
2.2.5	Press connected with valid IP	For the listener opens that listener chat window For the ASL Speaker: opens that the ASL Speaker chat window
3. Chat window		
3.1.	Recognize letter from ASL speaker at least 3 seconds	The letter will be written in his personal chat box
3.2	Press “Send” button	Send the text in the personal section to the other side, and update the public chat box
3.3	Press “Return " button	Return to login window for both ASL speaker and listener
4. General		
4.1	Press “X" button from every window	Exit program

7. Expected Achievements

We are undertaking an ambitious project to develop a system that can detect ASL (American Sign Language) movements and translate them into written words. Our primary objective is to facilitate easier and more efficient communication for ASL speakers within their community.

To achieve our goals, we have outlined three key areas of focus: ASL speaker hand detection, ASL movement recognition, and a chat application for seamless communication. We plan to utilize YOLOv5 for hand detection and DenseNet for classification. By combining these techniques, we aim to create a high-resolution classification and translation system for ASL.

Our training dataset will consist of real ASL gestures, obtained from a reliable source such as the Kaggle dataset that contains ASL letters. By training our network on this dataset, we expect it to become proficient in recognizing and classifying ASL letter images into written words.

The indicators of success for our project include:

Building a network based on YOLOv5 and DenseNet: We will construct a robust network architecture that integrates both YOLOv5 for hand detection and DenseNet for classification.

Developing an accurate translation application: Our goal is to create an accessible application that achieves high accuracy in translating ASL movements into written words. This application will enable ASL speakers to communicate more effectively.

Optimizing hyperparameters: We will diligently experiment with different hyperparameter values to identify the most optimal configuration for our proposed network. Fine-tuning these parameters will enhance the performance and accuracy of our system.

Real-time chat application: Our aim is to design a real-time chat application that incorporates both image classification and normal text keyboard inputs. This application will enable seamless communication between ASL speakers and non-ASL users.

While developing the system, we anticipate challenges related to image classification, especially in differentiating between similar hand gestures. To address this issue, we plan to leverage the capabilities of YOLOv5, known for its ability to identify and classify similar objects. By marking the ASL letters using bounding boxes, we can then focus on classifying smaller sections of the images containing the hand gestures, thereby excluding irrelevant objects from the classification process.

By tackling these challenges and achieving our objectives, we believe our project will significantly contribute to improving communication for ASL speakers, making it easier and more efficient for them to connect with their community.

8.References

[1] Kumar, P., Gauba, H., Roy, P.P. and Dogra, D.P., 2017. A multimodal framework for sensor based sign language recognition. Neurocomputing, 259, pp.21-38.

[2] Munib, Q., Habeeb, M., Takruri, B. and Al-Malik, H.A., 2007. American sign language (ASL) recognition based on Hough transform and neural networks. Expert systems with Applications, 32(1), pp.24-37.

[3] Bendarkar, Dhanashree, et al. "Web based recognition and translation of American sign language with CNN and RNN." (2021): 34-50.

[4] Sharma, Shikhar, and Krishan Kumar. "ASL-3DCNN: American sign language recognition technique using 3-D convolutional neural networks." Multimedia Tools and Applications 80.17 (2021): 26319-26331.

[5] 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.
* https://www.bu.edu/asllrp/av/dai-asllvd.html

[6] Barbhuiya, Abul Abbas, Ram Kumar Karsh, and Rahul Jain. "CNN based feature extraction and classification for sign language." Multimedia Tools and Applications 80.2 (2021): 3051-3069.

[7] 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.

[8] A. Barczak, N. Reyes, M. Abastillas, A. Piccio, and T. Susnjak, “A new 2d static hand gesture colour image dataset for asl gestures,” 2011.

[9] F. Bes¸er, M. A. Kizrak, B. Bolat, and T. Yildirim, “Recognition of sign language using capsule networks,” in 2018 26th Signal Processing and Communications Applications Conference (SIU). IEEE, 2018, pp. 1–4.

[10] B. Garcia and S. A. Viesca, “Real-time american sign language recognition with convolutional neural networks,” Convolutional Neural Networks for Visual Recognition, vol. 2, 2016.

[11] A. Deza and D. Hasan, “Mie324 final report: Sign language recognition.”

[12] Thuan, Do. "Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detention algorithm." (2021).

[13] 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.

[14] https://www.kaggle.com/datasets/valentynsichkar/yolo-coco-data

[15] https://www.kaggle.com/datasets/huanghanchina/pascal-voc-2012

[16] Huang, Gao, et al. "Densely connected convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[17] Bayram, Bülent, et al. "A Deep learning integrated mobile application for historic landmark recognition: A case study of Istanbul." Mersin Photogrammetry Journal 2.2 (2020): 38-50.

[18] https://public.roboflow.com/object-detection/american-sign-language-letters/1