RAAR: Real-Time ASL Alphabet Recognition Using Convolution and Computer Vision

Jose Chan
University of California, San Diego
jchanpineda@ucsd.edu

Diego Williams
University of California, San Diego
ddw004@ucsd.edu

Doanh Nguyen
University of California, San Diego
don012@ucsd.edu

Brandon Chiou
University of California, San Diego
djlakhani@ucsd.edu

Matthew Mizumoto
University of California, San Diego
mmizumoto@ucsd.edu

Ken Vandeventer
University of California, San Diego
kvandeventer@ucsd.edu

Abstract

American Sign Language (ASL) serves as an incredibly important form of communication for the Deaf and Hard-of-Hearing community. Recent advances in deep learning, particularly in computer vision, have allowed for real-time machine learning classification. We hope to be able to extend these advances to the ASL Alphabet. This project tries to perform live classification for ASL alphabet recognition. It aims to recognize and classify hand gestures in real time. Initially, we sought to use a single model for semantic segmentation and hand gesture classification using a UNET architecture, but experimentation with this approach made us revise our strategy. Our second and final model use a CNN architecture, while utilizing YOLOv8/OpenCV to draw boundary boxes around detected hands to reduce the search area, allowing the CNN to have some degree of translation invariance in a live, generalized real-world scenario. In whole, we evaluated various model architectures with the ultimate goal of designing a lightweight classification system that could be deployed in real-world scenarios like sign language translation or accessibility tools.

1 Introduction

Sign languages rely on combining dynamic hand gestures, facial expressions, and body movements, making them extremely complex for machines to learn and recognize patterns. However, ASL, one of the most widely used sign languages, has a defined alphabet where each letter is represented by a unique static hand gesture. Therefore, we decided to simplify our model goals and limit the recognition scope to just the ASL alphabet. The ability to automatically recognize and interpret these gestures in real time has significant implications for accessibility, education, and human-computer interaction. This project aims to develop a live semantic segmentation model capable of identifying and classifying ASL alphabet signs. Our method incorporates semantic segmentation to precisely localize the hand within an image frame and assign a corresponding letter label. By implementing deep learning architectures, specifically CNNs and UNETs, we can enhance the speed and accuracy of the system while preserving real-time performance suitable for deployment in assistive technologies and educational applications.

CNNs have become the dominant approach for visual recognition tasks due to their ability to learn spatial features. A CNN consists of multiple layers, including convolutional layers that extract meaningful patterns, pooling layers that reduce spatial dimensions, and fully connected layers that perform classification. For ASL recognition, CNNs enable the system to effectively learn the unique hand shapes associated with different letters. However, simple CNN-based classification models do not provide spatial localization of the hands within an image, which is essential for real-time applications.

To address this limitation, our project sought to use a U-Net architecture, a popular CNN-based model for semantic segmentation. A U-Net is an encoder-decoder model originally designed for biomedical image segmentation but has been widely applied to various vision tasks that require precise object localization. The encoder path extracts feature representations, while the decoder path uses transposed convolutions to reconstruct a high-resolution segmentation map. The U-Net's skip connections help retain fine-grained spatial information, making it particularly effective for segmenting hands in real-time ASL recognition. Specifically, we thought that this retention would allow it to learn more precise recognition based especially on shape that would be useful in dynamic environments with varying lighting conditions and backgrounds. We reasoned this would empower our model to detect, segment and classify hand gestures efficiently.

2 Related Work

The field of computer vision has been widely explored, so there are lots of places that we looked to to take inspiration from. Thanks to deep learning, computer vision has now become more efficient than ever. We looked at many different articles covering the use of CNNs with ASL, or just CNNs in general to see how lightweight we could make the model while maintaining accuracy.

The main source of inspiration for this project was the work by Kshitij Bantupalli and Ying Xie (2018), in which they translated ASL into text using video. However, this work did not have the end goal of being able to do this in real time, and instead took some time to process the data. Aside from using a CNN to extract features from the data, they also used an RNN to be able to convert the text into english text due to the differing grammar syntax between ASL and English. So while we took inspiration from this work, we also tried to simplify the CNN structure to make it more lightweight and be able to perform in real time, and we also took out the RNN since sentence translation was not our end goal. The article took inspiration from a different work that we also looked at, "Going deeper with convolutions" by Szegedy et al. (2015). This work was used to implement the CNN in the work by Bantupalli and Xie, which was named Inception.

Similarly, the work of Sincan and Keles (2020) explored the use of transfer learning with pre-trained models such as ResNet and VGG for sign language alphabet recognition, improving accuracy. However, this was done with Turkish sign language, and focused on whole-image classification rather than detecting and segmenting hand gestures in real-world scenarios.

When looking specifically at semantic segmentation, the work by Long et al. (2015) on Fully Convolutional Networks (FCNs) often showed up, as it laid the foundation for pixel-wise classification tasks. More recent advancements, such as DeepLabV3+ (Chen et al., 2018), have improved segmentation accuracy through convolutions and encoder-decoder architectures. These methods provide promising methods for enhancing ASL recognition by allowing for precise localization of hand gestures within an image frame. Additionally, real-time object detection models like YOLO (Redmon et al., 2016) and SSD (Liu et al., 2016) have been leveraged in gesture recognition tasks to achieve rapid inference speeds.

This project builds upon these prior works by integrating semantic segmentation techniques with ASL alphabet classification to improve real-time gesture recognition. By adopting a CNN-based approach optimized for semantic segmentation, our method enhances localization accuracy while maintaining efficiency for live applications. Future work in this domain could explore transformer-based architectures, such as Vision Transformers (ViTs), to further enhance feature representation and classification robustness in complex environments.

3 Methods

3.1 Dataset

Initially, our goal was to be able to both semantically segment images and classify them. We thought that by letting a single model learn to both identify compositional pixels of hands and classify the ASL letter, it would learn features focused on the hand, and its shape, which would help it remain robust to changes in the background or color/lighting conditions of the image. We thus set out to find data that would allow us to easily create masks for the purpose of semantic segmentation. We settled on a dataset created by LexSet's Seahavan synthetic data generation, containing 27,000 512x512 images of different letters of the ASL alphabet. We then used a pretrained deeplabv3 model, using resnet101 as its backbone to create masks of the hands in our image dataset. It's a state of the art model, but quite large, making it undesirable to try and fine tune. We used this model to create our target images.

As we'll go into more detail in the discussion section, the selected data has a few quirks. Firstly, because it's synthetically generated using 3d rendering to include only a hand against a background, and varies in many qualities of the image such as skin color of the hand, background setting, lighting condition, 3d orientation, there were not that many perfectly similar images for the model to learn off of. Additionally, the artifice of the conditions of images in our dataset suggested we might need to add more augmentation when we tried to implement the real-time component for increased robustness.

3.2 Semantic segmentation with U-Net

Our baseline model was training a U-Net model for the task of semantic segmentation and classifying the relevantly semantically segmented image as a letter of the ASL alphabet. We decided, initially, to have a single U-Net learn both of these tasks simultaneously so it would learn representations useful for both tasks, which we justified by recognizing that the silhouette of the hand shape is pertinent to the letter it corresponds to.

Our training pipeline for our baseline model was as follows: first, we group training images, their respective segmented masks, and their ASL letter label into minibatches. We then feed those minibatches into our training loop, which runs U-Net's convolutional "encoding" layers on the image, passes it through a bottleneck, and then subsequent "decoding" layers and produces both a classification of corresponding symbol and a semantically segmented image of the same size as the training image labeling each pixel as part of a hand or not. We use cross-entropy for the label classification and binary cross-entropy for the semantic segmentation. The losses are then combined to be used as the learning impulse fed to the weights, and the model continues training, saving the best-performing model weights (on the validation set) for evaluation on the test set. It should be mentioned that we do use data augmentation on the initial training images, though at the moment, we only use cropping and flipping.

Our initial model consisted of the following layers.

Table 1: Initial U-Net Architecture

#	Layer	in_channel	out_channel	Activation	Kernel	Stride	Padding
1	Encoding Block (Conv2d, Conv2d)	3	64	ReLU (after each convolution)	3x3	1	1
2	MaxPool	64	64	N/A	2x2	1	0
3	Encoding Block Conv2d Conv2d	64	128	ReLU (after each convolution)	3x3	1	1
4	MaxPool	128	128	N/A	2x2	1	0
5	Encoding Block Conv2d Conv2d	128	256	ReLU (after each convolution)	3x3	1	1
6	MaxPool	256	256	N/A	2x2	1	0
7	Encoding Block Conv2d Conv2d	256	512	ReLU (after each convolution)	3x3	1	1
8	ConvTranspose2d	512	256	NA	9 2	2	0
9	Decoding Block Conv2s Conv2d	512	256	ReLU	3	1	1
10	ConvTranspose2d	256	128	NA	9 2	2	0
11	Decoding Block Conv2s Conv2d	256	128	ReLU	3	1	1
12	ConvTranspose2d	128	64	NA	9 2	2	0
13	Decoding Block Conv2s Conv2d	128	64	ReLU	3	1	1
14	classifier AdaptiveAvgPool2d Flatten Linear	64	27	NA	NA	NA	NA

3.3 Classification with CNN (w/ YOLO)

After we scrapped the idea of letting the model learn features for both semantic segmentation and classification, we had no reason to keep the U-Net architecture. We thus opted for a similarly lightweight CNN that simply learns nonlinear features through successive layers of convolution and classifies the image using those nonlinear features as input to a final softmax layer. We used

cross-entropy loss, although with resizing of the image to 64x64 (in earlier iterations, 128x128 later for better robustness of distinction between letters) to save on computational requirements.

Table 2: Convolutional Neural Network Architecture

#	Layer	in_channel	out_channel	Activation	Kernel	Stride	Padding
1	Encoding Block (Conv2d, Conv2d)	3	32	ReLU (after each convolution)	3x3	1	1
2	Batch Normalization	32	32	N/A	N/A	N/A	N/A
3	Encoding Block Conv2d Conv2d	32	64	ReLU (after each convolution)	3x3	1	1
4	Batch Normalization	64	64	N/A	N/A	N/A	N/A
5	Encoding Block Conv2d Conv2d	64	128	ReLU (after each convolution)	3x3	1	1
6	Batch Normalization	128	128	N/A	N/A	N/A	N/A
7	Encoding Block Conv2d Conv2d	128	256	ReLU (after each convolution)	3x3	1	1
8	Batch Normalization	256	256	NA	N/A	N/A	N/A

3.4 Bi-LSTM Model

In an effort to recognize sequential hand gesture data (e.g., from short video clips or motion capture streams), we explored a model incorporating a Bidirectional LSTM (BiLSTM). The architecture used recurrent neural networks to process time-series data derived from MediaPipe trackers on the hands in each frame. The following code snippet illustrates the structure of our BiLSTM-based model:

```
model = Sequential([
    Input(shape=(max_seq_length, feature_dim)),
    Bidirectional(LSTM(256, return_sequences=True)),
    BatchNormalization(),
    Dropout(0.3),
    Bidirectional(LSTM(128, return_sequences=True)),
    BatchNormalization(),
    Dropout(0.3),
    # Optional attention mechanism or a Dense "context" layer
    Dense(128, activation="tanh"),
    Dropout(0.3),
    LSTM(64),
    Dense(256, activation='relu'),
    Dropout(0.3)
])
```

This design was intended to capture temporal patterns in the data: each Bidirectional(LSTM(...)) layer processes information both forward and backward in time, while batch normalization and dropout layers combat overfitting. The final dense output layer then classifies each sequence into one of the ASL letters.

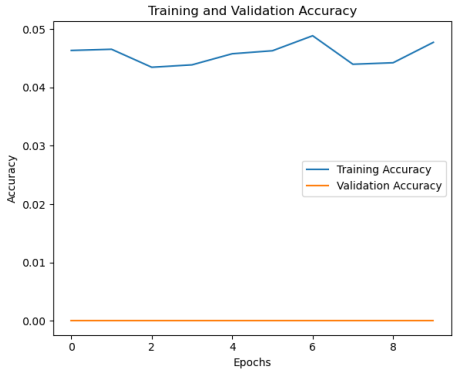
3.5 SuperLightWeight CNN

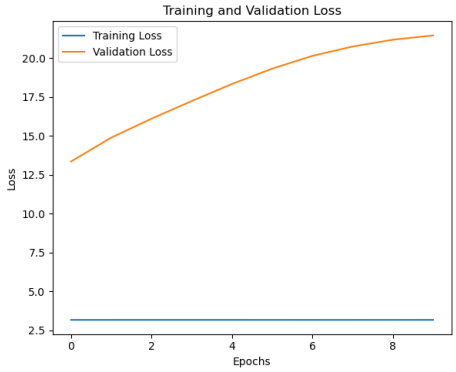
A lightweight CNN architecture is defined, consisting of two convolutional blocks with batch normalization and dropout, followed by a fully connected layer for classification. The model is trained for ten epochs using the Adam optimizer and categorical crossentropy loss. Upon completion, the trained model is saved as a .keras file.

For real-time inference, OpenCV accesses the system camera and defines a region of interest (ROI) for the user's hand instead of initial attempts of using YOLO. This region is segmented with adaptive thresholding, capturing the largest contour as the hand. The resulting binary image is saved temporarily and passed to the CNN, which predicts the corresponding ASL letter. The predicted class is then overlaid onto the video feed, allowing users to see real-time predictions of their signed letters. This approach demonstrates a straightforward but effective pipeline for end-to-end ASL letter recognition, from dataset collection to deployment in a live camera-based application.

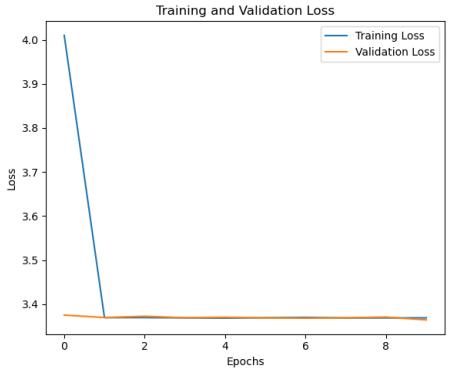
4 Results

Accuracy and loss plots for U-net, no shuffle





Accuracy and loss plots for U-net, shuffle



Accuracy and Loss plots for CNN

Example of model misclassification
Cropped from a YouTube video in attempt to see if model generalizes to simple, real-world letters with variation

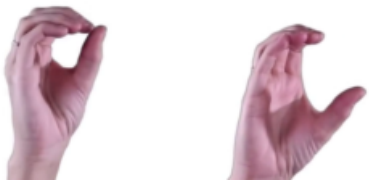
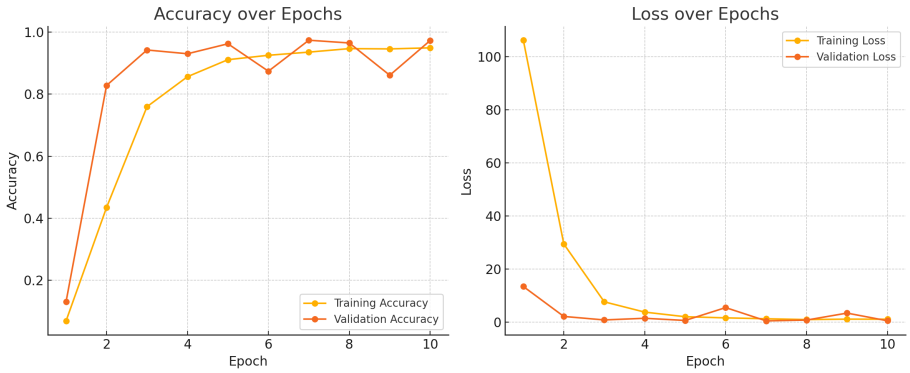
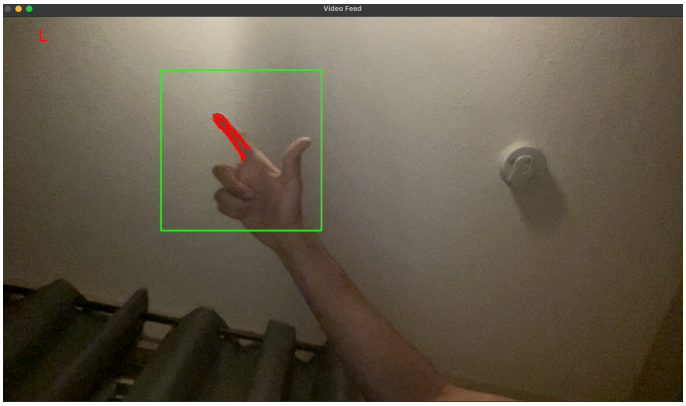


Figure 1: On the left is ASL letter O, on the right is ASL letter c, the model mistakenly categorized both as C (note that training data would indicate a thinner c than the video suggests)

Accuracy and Loss plots for 2nd CNN



Live demonstration



Diego, a team member, making the ASL letter "L" with his hand, and the model detecting and correctly predicting the character as L in the upper left-hand corner

5 Discussion

5.1 Baseline

5.1.1 Loss function

Though we initially used the standard loss function of cross entropy as our classification loss, and Binary Cross Entropy with Digits as our regression loss, this turned out to not be the greatest choice. The biggest problem with using BCEwithDigits as our regression loss was that BCE is a loss designed for classification loss, not regression loss. We initially thought that we would be able to use BCE since we would be able to frame our boundary box as a binary mask, and could be used to classify which pixels should belong to the boundary box. However, this requires transforming the boundary box problem into a classification problem. This isn't desirable, as it is much more efficient to retain the regression aspect of the problem. The boundary box coordinates are continuous values, as we are supposed to calculate the xmin, xmax, ymin, and ymax values of the boundary box. We found a better loss function to implement instead, Scyllan IoU. This loss function incorporates 3 different aspects necessary for efficient boundary box regressions problems. It is computed as $1 - Iou + \frac{\delta+\Omega}{2}$ IoU measures the intersection over union between the ground truth and predicted bounding boxes. It encourages maximum overlap between the two bounding boxes. δ represents the distance cost, which measures the Euclidean distance between the centers of the two bounding boxes, and encourages the bounding boxes centers to align. The last major metric was Ω, which represented the shape cost. The shape cost ensures that the predicted boundary box has the same shape as the ground truth boundary box. Unfortunately, due to how relatively new this loss function is, there are no straightforward implementations of this loss function. So we set out to implement this loss function on our own.

Unfortunately, after several iterations, we were still unable to get this to work correctly. Our attempts at this loss function would eventually devolve into returning NaN. We ended up turning back to cross entropy loss.

5.1.2 Dataset

Another issue we encountered, as mentioned above, was being unable to find a good dataset. This ended in creating our own datasets. We ran our synthetic ASL dataset through the pretrained deeplabv3+ network with ResNet101 as its backbone. However, this data sett ended up being too large for datahub, as it would exceed the storage limitations set by datahub. This data set was also too large to upload to GitHub, and instead we had to split it into parts to be able to share the dataset appropriately.Then, we ran into some issues with the actual masks created, as they were not always accurate. In fact, upon a closer inspection of the masks created, much of the data was extremely inaccurate. There were many masks that were created that were completely empty. Though we tried to clean this dataset, due to the sheer size of the repository, we were not able to completely clean this dataset. We were able to remove the empty masks when they were not the intended results but we still had plenty of masks that were missing huge chunks of the correct output. This poor quality in the dataset would not allow for us to properly train a model meant to draw boundary boxes as the incorrect masks would slow down, or completely disallow convergence of the model. Also, the initial images are synthetically created, and only contain a hand against a random background. There are no other objects in the foreground, not even people. We believed this to be a detrimental aspect of the data that would not result in a good model. We decided to turn away from having to rely on the the subpar masks created to train our model.

5.1.3 Baseline results

For the reasons mentioned above, we were not able to achieve a good baseline model. One of the worrying aspects of the graph we produced of the initial model as the constant 0 validation loss and accuracy. This was due to the way our data was structured. Our dataset was structure into an input and target folder. Within each folder, we had 2 sub folders, Training and Testing. Within each of these, we had a folder for each of the letters of the alphabet. So when our dataloader created our datasets, it always used the same letters to create the training data and different letters to create the validation data. It also trained on the data in alphabetical learning, preventing actual learning. Shuffling the data allowed model to actually attempt to learn, though for the reasons previously mentioned, the learning was still not good.

5.2 CNN + YOLOv8

5.2.1 Dataset

Despite being able to train the model to have great performance on the holdout and test sets of our original dataset, the model performed badly on real-world data. We attributed this to some flaws with our chosen dataset, the most crucial one being insufficient number of training images. This was only exacerbated by excessive variation of gesture's yaw (and insufficient pitch and roll) when in real-world situations, ASL gestures can be assumed to vary little in yaw, lighting conditions, color of hand conditions, background conditions that exacerbated the limited amount of data we could use to train the model. We also had some concerns about our model getting results that might come from it learning some pattern of how the synthetic data was generated, which might make it wholly ungeneralizable to real data. Unfortunately, given the time constraints of the project and compute constraints of DataHub and our members' GPUs, we didn't opt to find a new dataset for training our own classifier. With more time, we'd consider a different dataset that more closely resembles real-world sign language's usage, especially since we no longer have the restraint of needing to be able to easily create masks from the data.

5.2.2 YOLO Interface

Despite using a YOLOv8 model pre-trained to detect hands, we found that the model was struggling to detect only hands, often predicting a boundary box around the whole human. This resulted partially from difficulty in creating a discriminating boundary box around solely the hand– especially when other distracting objects like people's head (sharing skin tone) or other hands were in the picture. We

tried a few different pre-trained models, but none were working well, so we decided to constrain the "input area" to afford some regularity, but found where we mostly contained the gesture in the box and fed that into our CNN that our CNN struggled to produce meaningful predictions, often overpredicting certain getures and wholly underpredicting others (despite balanced, impressive performance on the synthetic dataset).

5.3 BiLSTM

The dataset used, WLASL, was incomplete as most of the videos could not be extracted from youtube so this left us with a complete ASL dictionary (not simply the alphabet that included over 2000 labels but did not include sufficient data to train the model only allowing the model to reach 40% accuracy on the train data and far worse on the test data. Given a complete dataset and a longer period of time, this accuracy could have been improved, however, this endeavor was left behind due to its high complexity.

5.4 Lightweight CNN + OpenCV

The CNN model had high accuracy 95% in the test and validation classification; however, in the live application, the prediction would intermittently change with noise, lighting, and motion preventing it from maintaining a prediction for longer than a second despite attempts to reduce this. This could have been improved with a longer delay and tweaking the values for threshold, but overall showed the most promised as a non-latent, live classifier.

6 Further Research

Given extra time, we would explore using a different dataset for training the CNN backbone, as well as possibly a custom-trained YOLO model suited to our dataset. We suffered quite a bit from difficulty with OpenCV/YOLO not correctly segmenting/creating boundary boxes that reduced the problem for our CNNs by creating a pseudo translation invariance. Specifically, we would want to find a dataset that includes a lot more stills of hand gestures, preferably taken from settings resembling our use case: people's offices, work environments with less variability in lighting condition, background and rotation along axes so that we could learn better features within those contexts. Our model suffered in live classification especially because of this lack in quality of data, so were we to continue this project, that would certainly be the next step.

7 Author's Contributions

7.1 Doanh Nguyen

Helped write abstract, introduction, and future research. Tried implementing advanced fine tuning techniques, SWA.

7.2 Brandon

Tried YOLO11 for detecting hands, it failed. Did some independent research, and tried to find pre-trained models for hand detection

7.3 Jose Chan

Wrote the abstract, introduction, Related work, initial baseline dataloader, training loop, and model files. Wrote sections 5.1.1 and 5.1.2. Wrote initial semantic segmentation file to create masks as targets. Generated some plots and table #1. Attempts at implementing Scyllan IoU loss,

7.4 Matthew Mizumoto

Pair-programmed the Section 3.3 CNN using YOLO– made the CNN, debugged the dataloader files, generated the loss and accuracy plots, created the accuracy function, created the code for YOLO-CNN pipeline (parsing frames and cropping). Fine-tuned resizing values for

7.5 Ken Vandeventer

Pair-programmed the Section 3.3 CNN with Matthew– helped debug the data loader files, generated the loss and accuracy plots, created the accuracy function. Helped fine-tune resizing values for inputs and LR for optimal CNN performance. Wrote the 3.2 and 3.3 method sections.

7.6 Diego Williams

Created the section 3.4 CNN– including own implementation, with accuracy function, and hooked up with OpenCV so live video could be used to detect ASL letters. Wrote the corresponding discussion, methods, and results sections. created a notebook for real-time testing of our alphabet recognition model.

References

- Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H. (2018). *Encoder-decoder with atrous separable convolution for semantic image segmentation*. Proceedings of the European Conference on Computer Vision (ECCV), 801-818. https://doi.org/10.1007/978-3-030-01234-2_49
- Long, J., Shelhamer, E., Darrell, T. (2015). *Fully convolutional networks for semantic segmentation*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3431-3440. https://doi.org/10.1109/CVPR.2015.7298965
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A. C. (2016). *SSD: Single shot multibox detector.* Proceedings of the European Conference on Computer Vision (ECCV), 21-37. https://doi.org/10.1007/978-3-319-46448-0_2
- Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). *You only look once: Unified, real-time object detection*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779-788. https://doi.org/10.1109/CVPR.2016.91
- Sincan, O. M., Keles, H. Y. (2020). *Efficient hand gesture recognition using 3D CNNs and an ensemble of GRU networks*. Applied Soft Computing, 97, 106798. https://doi.org/10.1016/j.asoc.2020.106798
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. (2015). *Going deeper with convolutions*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1-9. https://doi.org/10.1109/CVPR.2015.7298594