An Exploration of ASL Object Detection and Classification Methods and Modalities

Cameron Ervin
Jonathan Hunnicutt
Ben Brock

1 Introduction

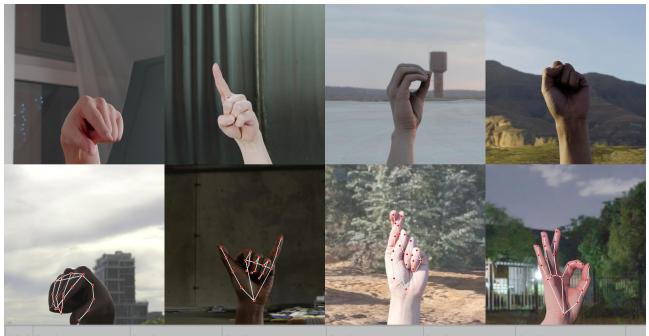
1.1 Motivation

American Sign Language (ASL) is used by approximate 6.4 million hearing able people and approximately 1 million deaf or hard-of-hearing people in the United States of America. [10] Despite the size of this community and the fact that across nation, ASL is offered in high schools as foreign language, even getting an accurate count of ASL users is quite difficult, since the United States Census does not track users of ASL. The commonly cited estimate of 500,000 is over 50 years old [17]. Although the exact count of the number of ASL users is unknown, we do know the implementation of translation based language models would bring a great benefit to the ASL community.

Since ASL is a non-spoken and gestural language, requiring a completely different network architecture than other translation models, we wanted to explore the feasibility of creating ASL translation model by using current object tracking and object detection models for a simple classification task.

Though American Sign Language is its own language, with its own grammar and structure, with several thousand of the most common words having their own sign. A classification task of several thousand signs (some dynamic) is too large for an exploratory study. But ASL also finger-spells rare words with individual signs for each letter. Most ASL communication does not spell out every word, letters are important for unknown/unusual words.

We therefore begin exploring ASL translation with a finger-spelling task of identifying signed letters for transcription. This work could be expanded upon with other static signs, and eventually built upon for classifying dynamic signs.



label	hand_0_x	hand_0_y	hand_0_z	hand_1_x	hand_1_y	hand_1_z	
J	0.427092224359512 00	0.686869204044342	3.14358942432591E-07	0.506993174552918 0	0.653293073177338 0	-0.0873574391007423 0	...
J	0.623823225498200 0	0.602519690990448	-5.31736787934278E-08	0.545216798782349 0	0.594301223754883 0	-0.0548820926713940 0	...
J	0.315064579248428 00	0.671338677406311	-7.65629550869562E-08	0.393225014209747 0	0.614807188510895 0	-0.0085636125877499 60	...
J	0.359287828207016	0.628463447093964 0	-1.40890654165566E-06	0.497453570365906 00	0.578961730003357 0	0.130519747734070 00	...

Figure 1. A sample of the base ASL letter image data, MediaPipe Hand overlay ASL letter image data, and MediaPipe Hand landmark tabular data.

1.2 Main Research Questions

This was exploratory research, broadly attempting to discover which aspects are most important for decoding and transcribing ASL hand gestures. Each model then had an implicit theory of decoding/transcribing. By training these models and assessing the comparative performance, we collected evidence which suggested which aspects are more important for decoding and transcribing hand signs.

These broader concerns can be broken down into concrete research questions:

1) What is the relative importance of the 2D images of the hands vs the 3D models of the hands for decoding and transcription of ASL?

Expanding upon this, is what matters the visual images of the hands, or merely that the visuals are representing the 3D model of the hands? Does the 3D hand landmark location data offer enough information for the decoder/transcriber, and thus the visual is almost entirely unnecessary. Or do the visual 2D images include information helpful for hand sign classification beyond the hand landmark model?

2) What is the relative importance of 2D images to 2D hand landmark overlays upon the original image?

Basically, does adding hand landmarks to the 2D image improve model performance, or is this flattened hand landmark information extraneous to the visual. Along with the 3D hand landmarks, we can compare whether the presence of the landmarks is what is important, or the dimensionality (2D vs. 3D) is important.

3) What is the relative importance of normalizing 3D hand landmark data?

Does hand size/image size impede or increase performance? Normalization often increases performance of models in Machine Learning, but not always. Scaling the hand landmark tensors is likely to be helpful, but shifting the origin could hinder progress, since MediaPipe uses the wrist

point as the origin, and normalizing the x,y,z locations together would shift the origin to the minimum x,y,z value, likely meaningless, instead of leaving it at the wrist.

1.3 Hypothesis

For Research Question 1: We hypothesize that the 3D hand landmark information is relatively more important part of the information and that the visual is unhelpful. Therefore the models with hand landmark information in any form (even just a 2D overlay) should perform better than those without. The models with 3D should perform better than the models with 2D.

For Research Question 2: We hypothesize that the 2D hand overlay should increase performance of the model over mere image processing.

For Research Question 3:

Based on machine learning best practices, we hypothesize that min-max normalization of the x,y,z increases the performance of the model.

1.4 Related Work and Context

ASL recognition and classification are not new use cases for the field of computer vision and machine learning. Since the early 2000s, researchers have used numerous different methods to recognize and classify ASL letters and signs [3]. As the field of computer vision within ASL has evolved, there have been several trends within the approach researchers take within this space.

Earlier methods focused more on Bayesian networks such as hidden Markov Models (HMM)[7, 21, 24]. These models often implemented some sort of wearable technology, such as gloves with sensors, that captured the physical location of different landmarks within the arms and hands. Fatmi, et al utilized Myo gloves to track the arms and body of participants in their research[3]. Other researchers used 3D depth sensors such as the Microsoft Kinect and Leap Motion to capture the location and features of each gesture [3, 20]. These physical locations were recorded for different signs and gestures and then a Bayesian network such as an HMM was used to classify the time series data provided by the wearable technology output. However, despite the heavy prevalence of HMMs, there were also other models used such as random forest regression. Zafrulla, et al utilized Gabor filters to capture the edges of sign images and random forest regression to classify these edges[25].

Following HMMs, researchers began to implement rudimentary feed forward artificial neural networks (ANN)[15]. These ANN models again utilized the wearable technology or 3D depth sensors that previous researchers used for HMMs. However, HMMs were replaced by neural networks [7]. Additionally, other researchers implement different feed forward network approaches. For example, Garg implements a radial basic functioning neural network (RBFNN) to convert sign language to voice[5]. Garg implements PCA, LDA, and then uses the RBFNN to classify signs to text based on different body landmarks identified in a video.

Next, came more advanced neural network architectures such as convolutional neural networks (CNN) and recurrent neural networks (RNN)[6]. Up until this point, previous models tried only to extract features from an image and classify the extracted features rather than classifying the images themselves[2]. Ameen and Vadera made use of a CNN architecture to do just that and classify entire images of ASL letters[2]. In their work, they explored the performance of a CNN model with raw ASL letter images as input to other models such as a CNN model with Gabor filtered ASL letter images as input. Two Stanford researchers, Garcia and Viesca, also implemented CNN architectures with the addition of transfer learning to classify and evaluate ASL letter images [4]. Garcia and Viesca extended the work previously done in ASL classification while implementing CNN architectures by performing transfer learning and fine tuning GoogLeNet for the specific task of classifying ASL letter images. Although convolutional models had a clear application to ASL letter classification, RNN models were applied to this task as well[8].

Today, research within the ASL classification space is often performed using the most up to date neural network architectures such as YOLO and other detection algorithms such as Google's MediaPipe Hand [13]. Over the course of evolution of first R-CNN and then YOLO, ASL research has continued to be applied to each iteration of these CNN based models due to their state of the art performance. For example, Panchamia, et al utilized YOLOv2-v4 as well as YOLO-tiny[11]. Other researchers have utilized and compared YOLOv5-v8[12, 19, 22]. YOLOv9 was released earlier this year as the new state of the art object detection and classification model[23]. Furthermore, other research implement MediaPipe Hand for ASL classification [26]. These models take the hand landmark output of MediaPipe Hand as input to RNN models[1, 16].

2 Methods

2.1 Models

To answer our research questions, we developed four machine learning models to compare.

1) A YOLOv9 model with labeled ASL letter images as input and a bounding box and classification of the ASL letter as output. Each output includes the corresponding letter label, height, width, and x-y coordinates of the bounding boxes in txt file. The results of each model are evaluated on several different metrics including mAP, precision, and recall. Ultralytics stores these outputs after training and testing.

2) A YOLOv9 model with labeled ASL letter images with a 2D skeletal overlay from MediaPipe hand landmarks as input and a bounding box and classifcation of the ASL letter as output. This model first preprocesses the image data with Mediapipe Hand and plots 21 hand landmarks on top of each

Table 1. Example Ultralytics YOLOv9 results.csv Output

epoch	...	metrics/precision(B)	metrics/recall(B)	...	metrics/mAP50-95(B)	...	val/box loss
0	...	0.00064	0.00549	...	6E-05	...	2.6873
1	...	0.40411	0.07033	...	0.01003	...	2.2077
2	...	0.29104	0.39874	...	0.1681	...	1.4703
3	...	0.58448	0.66134	...	0.54258	...	0.97323
...	...	...	...	...	...	...	...

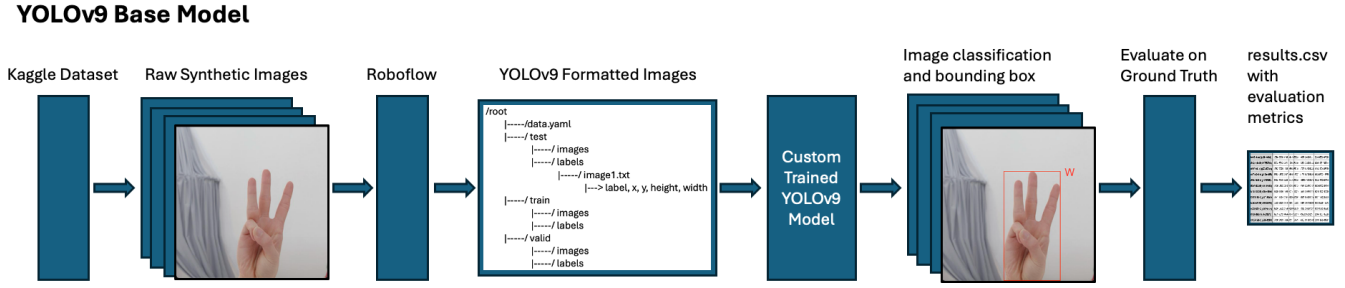


Figure 2. A diagram depicting our YOLOv9 base model processes.

image. The combined image is then fed into YOLOv9, and results in the same output as the previous model.

3) An XGBoost model with the **unnormalized** 3D coordinates of the MediaPipe hand landmarks as input and classification of the ASL letter as output. This model preprocesses the data with the Mediapipe hand landmark detection, which outputs a tensor handedness (left or right) and the 3D positions of 21 hand landmarks (with the wrist being the origin). The mediapipe hand coordinate tensor is outputted to a CSV, which includes 65 features (two being the image file name and the label and 63 being the x,y,z coordinates for each landmark). This is then run through an XGBoost classifier.

4) An XGBoost model with the **normalized** 3D coordinates of the MediaPipe hand landmarks as input and classification of the ASL letter as output. Same as the previous, except the x,y,z coordinates are normalized on a per row basis using min-max scaling which scales every number to a value between [0,1].

2.2 Building Our Models

2.2.1 Data Sourcing and Processing.

2.3.1.1 ASL Letter Images. Our original dataset before processing into the YOLOv9 format was a Kaggle dataset of synthetic ASL letters[9]. The background, lighting, and skin tone are all randomized using Lexset Seahaven.

2.3.1.2 Roboflow and YOLOv9 Data Formatting. In order to create our YOLOv9 dataset, we used Roboflow[14]. Roboflow is a web based application that allows for the labeling, augmentation, and exportation of a dataset into the necessary format for a model. We hand labeled the ground truth bounding boxes on all 2600 original images. All of the images were then augmented with the intent of improving model generalization. The augmentation steps can be seen in the table below. After augmentation, 88 percent, 8 percent, and 4 percent of our total dataset were split into train, valid, test splits respectively. The stratification of our splits is the result of the training set being the only split to have augmented data. As a result, the train split grew substantially in size after augmentation. Even though there is a large percentage of data in the train split comparatively, we argue that this split does not adversely impact the performance of our model. Like any language, there is a very large population of possible ASL images per letter. As a result, having a large train set is beneficial to our model. Additionally, our test and valid data are distributed evenly across each letter class making our split a viable split for our models. After augmentation, we then exported to the necessary YOLOv9 format which Roboflow performs natively.

Table 2. Augmentation Steps Performed in Roboflow to increase model generalization

Augmentation	Value
resize	640x640
flip	horizontal
crop	-5% to +20%
hue	+/- 15 degrees
exposure	+/- 10%
blur	2.5px

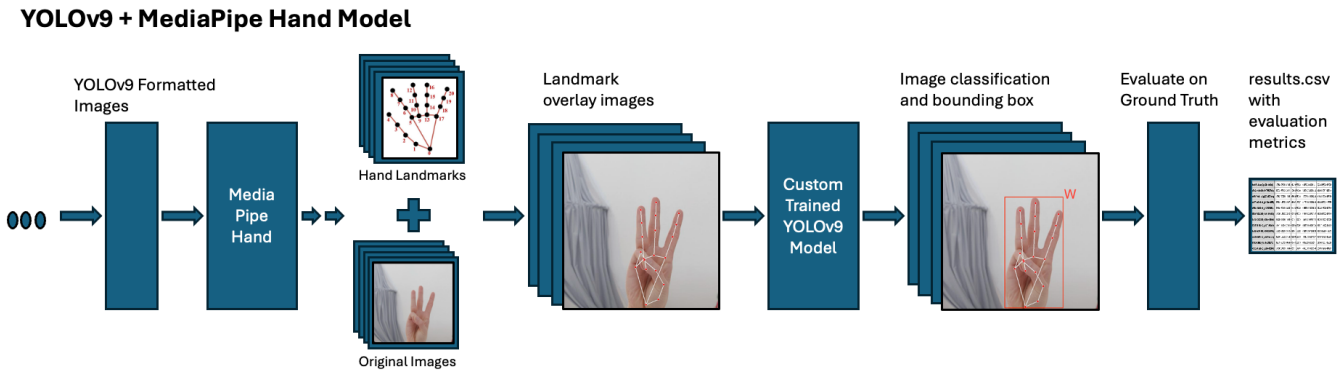


Figure 3. A diagram depicting our YOLOv9 + MediaPipe Hand overlay model processes.

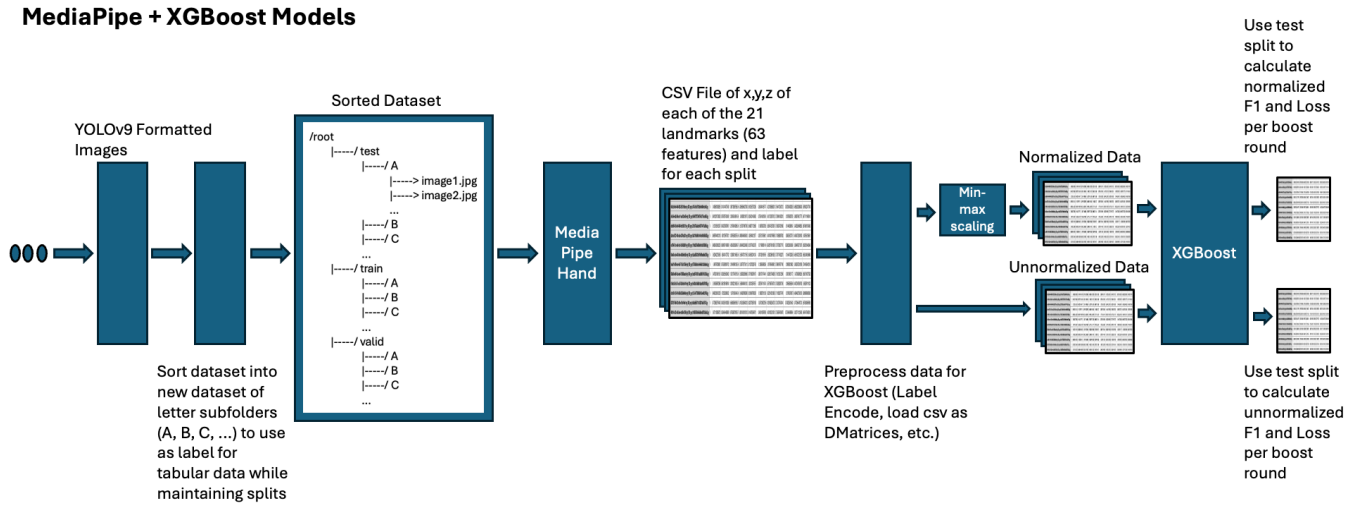


Figure 4. A diagram depicting both our MediaPipe Hand landmark + XGBoost model processes.

2.3.1.3 MediaPipe Hand 2D Overlay. In order to add an overlay of the Google MediaPipe Hand landmarks, we took our YOLOv9 formatted dataset and ran each image through the Hand pipeline. We then saved the landmarks on top of our original image using the draw landmarks utility function of MediaPipe Hand. We then created a new dataset with our new images while maintaining the same images in each split. Additionally, since the orientation of the images has not changed, we used the same labels for each image within each split respectively.

2.3.1.4 MediaPipe Hand 3D Landmark CSV. To save the 3D x,y,z coordinates of all 21 hand landmarks, first, we reorganized our original YOLOv9 dataset. We looped through each image and its respective label folder and sorted all of our images into a new dataset sorted by labels while still maintaining the same splits. This was performed to allow the ability to add the letter folder name as the label for our coordinate data.

Table 3. Example MediaPipe Hand landmark data. Each image equates to one data entry in a csv file for each split with 65 features including 21 hand landmarks in the x,y,z dimensions, image file name, and image label (A,B,C,...). This data is used as input to the XGBoost model with the label as the y column.

image	label	hand 0 x	hand 0 y	hand 0 z	...
(...).jpg	A	0.486322...	0.811409...	-7.62301...E-07	...
(...).jpg	A	0.510374...	0.752208...	-2.98105...E-07	...
(...).jpg	A	0.488164...	0.832668...	-7.25295...E-07	...
...	...	...	...	...	...

Next, we again ran each image of our new sorted dataset through the MediaPipe Hand pipeline. We then accessed the results of each pipeline output and saved the x,y,z coordinates for each of the 21 hand landmarks. Each letter coordinates and label was saved to a Pandas DataFrame and then later

Table 4. YOLOv9 and YOLOv9 +MediaPipe Hand Overlay Model Parameters

Parameter	Value
Optimizer	SGD
lr	0.01
momentum	0.9
154 weight	decay=0.0
161 weight	decay=0.0005
160 bias	decay=0.0

exported to a csv. This was performed for each the train, test, and valid splits. Each split was saved to a new, separate csv file.

2.2.2 Results Metric. Before touching on our model training methods, it is important to note that we chose to use F1 score as our results metric to measure the performance of our four different models. We chose F1 score because not only is it a way to evaluate both object detection and tabular models, but F1 score also has real world implications with ASL letters. Like the English language, not all letters are used at equal rates. For instance, there has not been a single occurrence of the letter "Z" in this paragraph. Therefore, it is important to use an evaluation metric that weights less common signs as equally as more common signs. F1 score captures both predicted positive observations with precision and predicted positives to actual positives with recall.

2.2.3 YOLOv9 Base Model. We trained the YOLOv9 model from scratch in Python using our train ASL letter images data with the max number of epochs set to 300. The Ultralytics YOLOv9 model uses standard YOLOv9 architecture.

We used the auto argument for the optimizer which determines the best optimizer for the given model. The YOLOv9 model determined the best optimizer to be SGD(lr=0.01, momentum=0.9) with parameter groups 154 weight(decay=0.0), 161 weight(decay=0.0005), 160 bias(decay=0.0).

The model converged after 193 epochs using early stopping and a patience of 100 epochs. After training our model, Ultralytics creates a csv file with the results of the model. In the csv file are precision and recall along with other metrics for each epoch. We utilized these two columns to calculate the F1 score per epoch, so we could later compare to our other models.

2.2.4 YOLOv9 + MediaPipe Overlay. This model follows the same steps as the previous model except now using the ASL letter images with the 2D MediaPipe overlay. We trained our model from scratch, and again the auto optimizer used the same parameters. It converged after 192 epochs using early stopping and a patience of 100 epochs. Again, we calculate the F1 score per epoch using results.csv and the precision and recall per epoch.

2.2.5 MediaPipe CSV XGBoost (Normalized and Unnormalized). To create our XGBoost models, we loaded in the train, test, valid 3D hand landmark csv files as Pandas DataFrames. Next, we divided our DataFrames into the 3D coordinate data and the letter label as the X and y datasets respectively for our train, test, and valid DataFrames. We then label encoded all three of the y datasets.

Next, we created separate, normalized, X datasets for our train, test, and valid splits. This was performed using min-max scaling. We will later compare the performances of the normalized and unnormalized X data. After, we then create XGBoost DMatrices for both the normalized and unnormalized data.

Finally, we use the XGBoost library in Python to create our XGBoost models. We run our DMatrices through the model with the following parameters: objective = multi:softprob, num class = 26, eval metric = mlogloss. Furthermore, we create our own negative F1 eval metric and use this as the feval argument when training. We used negative F1 since XGBoost aims to minimize the loss function. Therefore, we calculate the negative F1 of each of our 300 boost rounds. Next, we save our negative F1 and loss metrics for each of our 300 boost rounds using the evals result argument. Lastly, in order to compare to our other models, we take 1-(neg F1) for each boost round to find our final F1 values. We then save this output as a Pandas DataFrame and export to a csv for evaluation.

Table 6. XGBoost Parameters

Parameter	Value
objective	multi:softprob
num class	26
eval metric	mlogloss
feval	neg f1 (custom)
num boost rounds	300

2.3 Analysis Methods

To evaluate the differences between these models, we used a linear regression of F1 scores. The regression was dummy coded (one hot) in the regression model. To avoid multicollinearity, one model's dummy codes was droped, with the model chosen as the 'base model' which other models are compared to. To compare the final models, trained for a different number of epochs, the epochs were centered at the endpoints for ease of interpretation. [18]

To compare the performance of all models to the other models, the linear regression was performed with different models as the 'base model' to test significant difference. Picking different 'base model' for regression allows us to directly compare to answer the research questions.

However, despite the ease of interpretation of the linear regression, these growth models violate the assumptions

Table 5. Example XGBoost Output after converting the F1 columns from negative F1 to normal F1

boost round	test mlogloss	test F1	train mlogloss	train F1	...
0	0.704063...	0.962634	0.734370...	0.965309	...
1	0.536748...	0.962747	0.535278...	0.963179	...
2	0.415880...	0.962679	0.400154...	0.961995	...
3	0.331298...	0.967506	0.303098...	0.959595	...
...	...	...	...	...	...

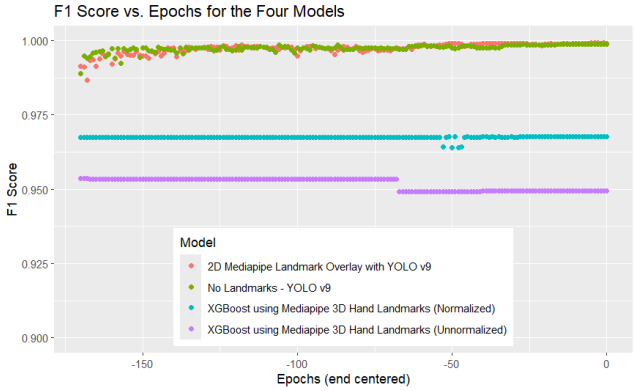


Figure 5. F1 Score vs. Epochs for all four models.

of linear regression (the epoch data is not randomly sampled, but the performance of each epoch depends on the previous epoch), we checked the robustness of the linear regression results by adding a covariate for epoch and three auto-regressive covariates for the previous epochs' F1 scores. Eight lagging auto-regressive terms were tested but only the first three were significant. The epoch covariate controls for any linear learning trend, while the three auto-regressive terms somewhat control for the non-randomly sampled nature of the data.

As with the linear regressions, these robustness checks were performed with different base models.

3 Results and Conclusion

3.1 Results

3.1.1 For Research Question 1 and Hypothesis 1 concerning the importance of 3D hand landmark modeling vs. 2D images. As seen in Figure 4, the visual only Yolo model and the Mediapipe hand skeleton 2D overlay models performed best with no significant difference between the two ($p = 0.491$), contrary to our first hypothesis that the 3D hand landmarks were the only relevant information. The Yolo only final F1 score was .998, and the Mediapipe 2D Overlay Visual final F1 score was .999.

However, the two 3D skeleton only models performed well, but significantly worse ($p < .001$) than the YOLO and Mediapipe 2D Visual overlay model.

To check the robustness of these results, covariates for number of epochs and auto-regressive terms were added, the differences between the performances of the 3D hand landmark modes and the Visual models remained significant ($p < .001$).

Therefore the 2D image focused models performed better than the 3D hand landmark models.

3.1.2 For Research Question 2 and Hypothesis 2 concerning the importance of 2D hand landmark overlays modeling vs. unaugmented 2D images. There was no significant difference between the performance of the YOLO model and the 2D Mediapipe Overlay Model ($p = 0.491$). When checking the robustness of these results with three autoregressive covariates and an epoch covariate, the difference between the YOLO model and the 2D Mediapipe Overlay Model remained insignificant ($p = .877$).

We also tested the differences of other reported machine learning metrics. The Mediapipe Overlay Model performed slightly better (.01, 1 percentage point) than the YOLO model on the training dataset box loss, training dataset cls loss and training dataset dfl loss. The validation dataset metrics were not significantly different between the two, with the exception of the validation dataset dfl loss metric where YOLO was better (.00389) by a significant amount ($p = 0.027$).

Thus the Mediapipe 2D Hand Overlay Model did not increase model performance contrary to our second hypothesis.

3.1.3 For Research Question 3 and Hypothesis 3 concerning normalization of 3D hand landmarks. Xgboost without normalization averaged an F1 score of .95, while the Xgboost model with normalization performed significantly better at an F1 score of .97 ($p < .001$). After checking the robustness of these results with additional autoregressive terms, the difference between the Normalized and Unnormalized 3D hand landmark models remained significant ($p < .001$)

As with many other machine learning tasks, normalizing this data increased performance as we hypothesized.

3.2 Implications

Several implication follow from the analysis concerning our research questions. In general the hand landmark data was

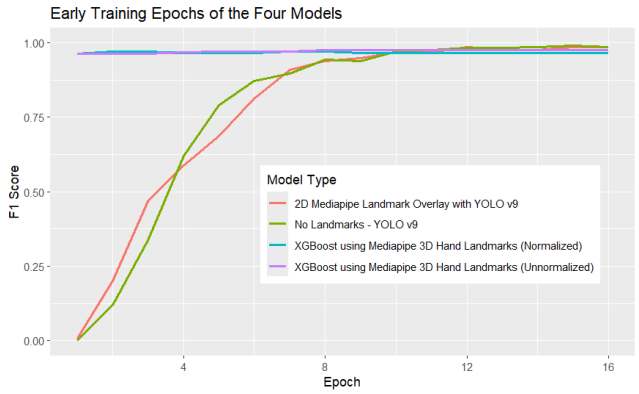


Figure 6. Early training epochs of the four models. F1 vs Epoch.

not as critical for increasing performance, since the YOLO model did consistent as good as the 2D Mediapipe Overlay Model.

The superior performance of the 2D Mediapipe Overlay Model and the YOLO model suggest that the 3D hand landmark data isn't capturing all the relevant information for distinguishing signs. The image data carries enough extra information to boost F1 Score performance by approximately 3 percent.

However, the lack of significant difference between the 2D Mediapipe Overlay model and the YOLO model suggests that the 2D hand landmarks added nothing but extra preprocessing to model performance. Whatever information YOLO was focusing on to classify the hand signs was enough.

The better performance of the normalized hand landmark tensor suggests that different hand sizes (and different image sizes) slightly impeded task classification for the unnormalized model. Since scaling the hands to a common scale increased performance, combined with other results, this suggests that hand size was not the extra information from the 2D images which increased classification performance of the YOLO and 2D Mediapipe Overlay Model.

Despite performing worse that the Yolo and Mediapipe 2D overlay, the results of the Xgboost models were still impressive for two main reasons. The mediapipe algorithm reduces the hand to a tensor of 63 float16s, which is 100 to 1000 times less information to process than images. Similarly while the YOLO and Mediapipe overlay models took 10-12 hours to train, the Xgboost models trained almost instantaneously as seen in Figure 5.

The xgboost models using the 3D hand landmarks were more computationally efficient and training efficient, and still very high performing after normalizing the coordinates. In a sense the 3D hand landmark data provided 97 percent of the necessary information to classify the alphabet hand signs.

References

[1] Reham Mohamed Abdulhamied, Mona M. Nasr, and Sarah N. Abdulkader. 2023. Real-time recognition of American sign language using long-short term memory neural network and hand detection. *Indonesian Journal of Electrical Engineering and Computer Science* 30, 1 (2023), 545–556. https://doi.org/10.11591/ijeecs.v30.i1.pp545-556

[2] Salem Ameen and Sunil Vadera. 2016. A convolutional neural network to classify American Sign Language fingerspelling from depth and colour images. *Expert Systems with Applications* Volume 34, Issue 3 (2016). https://doi.org/10.1111/exsy.12197

[3] Rabeet Fatmi, Sherif Rashad, and Ryan Integlia. 2019. Comparing ANN, SVM, and HMM based Machine Learning Methods for American Sign Language Recognition using Wearable Motion Sensors. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. 0290–0297. https://doi.org/10.1109/CCWC.2019.8666491

[4] Brandon Garcia and Sigberto Alarcon Viesca. 2023. *Real-time American Sign Language Recognition with Convolutional Neural Networks.* Technical Report. Stanford University, Stanford, CA. Available online: http://vision.stanford.edu/teaching/cs231n/reports/2016/pdfs/214_Report.pdf.

[5] Anirudh Garg. 2012. *CONVERTING AMERICAN SIGN LANGUAGE TO VOICE USING RBFNN.* Master's thesis. San Diego State University, San Diego, CA.

[6] Md Asif Jalal, Ruilong Chen, Roger K Moore, and Lyudmila Mihaylova. 2018. American Sign Language Posture Understanding with Deep Neural Networks. In *2018 21st International Conference on Information Fusion (FUSION)*. 573–579. https://doi.org/10.23919/ICIF.2018.8455725

[7] Taehwan Kim. 2016. American Sign Language fingerspelling recognition from video: Methods for unrestricted recognition and signer-independence. arXiv:1608.08339 [cs.CL]

[8] C.K.M. Lee, Kam K.H. Ng, Chun-Hsien Chen, H.C.W. Lau, S.Y. Chung, and Tiffany Tsoi. 2021. American sign language recognition and training method with recurrent neural network. *Expert Systems With Applications* 167 (2021), 114403. https://doi.org/10.1016/j.eswa.2020.114403

[9] Lexset. 2024. Synthetic ASL Alphabet. Available online at Kaggle. https://www.kaggle.com/datasets/lexset/synthetic-asl-alphabet

[10] Ross E Mitchell and Travas A Young. 2023. How many people use sign language? A national health survey-based estimate. *Journal of Deaf Studies and Deaf Education* 28, 1 (2023), 1–6.

[11] Nemil Panchamia, Jay Mehta, Priyesh Ghosh, and Jalpa Mehta. 2021. ASL Tutor Using Deep Learning. In *2021 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)*. 1–6. https://doi.org/10.1109/SMARTGENCON51891.2021.9645843

[12] I. Gethzi Ahila Poornima, G. Sakthi Priya, C. A. Yogaraja, R. Venkatesh, and P. Shalini. 2024. Hand and Sign Recognition of Alphabets Using YOLOv5. *SN Computer Science* 5 (2024), 311. https://doi.org/10.1007/s42979-024-02628-4

[13] Miguel Rivera-Acosta, Juan Manuel Ruiz-Varela, Susana Ortega-Cisneros, Jorge Rivera, Ramón Parra-Michel, and Pedro Mejia-Alvarez. 2021. Spelling Correction Real-Time American Sign Language Alphabet Translation System Based on YOLO Network and LSTM. *Electronics* 10 (2021), 1035. https://doi.org/10.3390/electronics10091035

[14] Roboflow. 2024. *Roboflow: Optimize Your Computer Vision.* https://roboflow.com

[15] Khadidja Sadeddine, Fatma Zohra Chelali, and Rachida Djeradi. 2015. Sign language recognition using PCA, wavelet and neural network. In *2015 3rd International Conference on Control, Engineering Information Technology (CEIT)*. 1–6. https://doi.org/10.1109/CEIT.2015.7233117

[16] Gerges H. Samaan, Abanoub R. Wadie, Abanoub K. Attia, Abanoub M. Asaad, Andrew E. Kamel, Salwa O. Slim, Mohamed S. Abdallah, and Young-Im Cho. 2022. MediaPipe's Landmarks with RNN for Dynamic Sign Language Recognition. *Electronics* 11 (2022), 3228. https://doi.

org/10.3390/electronics11193228

[17] Jerome D Schein and Marcus T Delk Jr. 1974. The deaf population of the United States. (1974).

[18] Judith D Singer and John B Willett. 2003. *Applied longitudinal data analysis: Modeling change and event occurrence*. Oxford university press.

[19] Chiranjeev Singh, Dolly Sharma, Aditya Prakash Dubey, and Neha Tyagi. 2023. Sign Language Detection Using CNN-YOLOv8l. In *2023 International Conference on Advances in Computation, Communication and Information Technology (ICAICCIT)*. 12–17. https://doi.org/10.1109/ICAICCIT60255.2023.10465792

[20] T. Starner, J. Weaver, and A. Pentland. 1998. Real-time American sign language recognition using desk and wearable computer based video. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 12 (1998), 1371–1375. https://doi.org/10.1109/34.735811

[21] Sudeep D. Thepade, Gandhali Kulkarni, Arati Narkhede, Priti Kelvekar, and Seema Tathe. 2013. Sign language recognition using color means of gradient slope magnitude edge images. In *2013 International Conference on Intelligent Systems and Signal Processing (ISSP)*. 216–220. https://doi.org/10.1109/ISSP.2013.6526905

[22] Shobhit Tyagi, Prashant Upadhyay, Hoor Fatima, Sachin Jain, and Avinash Kumar Sharma. 2023. American Sign Language Detection using YOLOv5 and YOLOv8. *Research Square* (2023). https://doi.org/10.21203/rs.3.rs-3126918/v1

[23] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. 2024. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. arXiv:2402.13616 [cs.CV]

[24] Honggang Wang, Ming C. Leu, and Cemil Oz. 2006. American Sign Language Recognition Using Multi-dimensional Hidden Markov Models. *Journal of Information Science and Engineering* 22 (2006), 1109–1123. Received August 16, 2005; accepted January 17, 2006.

[25] Zahoor Zafrulla, Himanshu Sahni, Abdelkareem Bedri, Pavleen Thukral, and Thad Starner. 2015. Hand detection in American Sign Language depth data using domain-driven random forest regression. In *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, Vol. 1. 1–7. https://doi.org/10.1109/FG.2015.7163135

[26] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. 2020. MediaPipe Hands: On-device Real-time Hand Tracking. arXiv:2006.10214 [cs.CV]

A Model Background

A.1 YOLOv9

YOLOv9 is the State of the Art Model in object detection. YOLOv9 which came out in February of 2024 in the paper "YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information" by Chien-Yao Wang and others builds upon previous YOLO models by implementing Programmable Gradient Information (PGI) and Generalized Efficient Layer Aggregation Network (GELAN) to address the large amount of data lost in layer-by-layer feature extraction and spatial transformation. Minimizing data loss is fundemental in maintaining accuracy and efficiency in object detection tasks. Furthermore, YOLOv9 makes implements PGI and GELAN to address the challenges of Information Bottleneck Principle and Reversible Functions which describe two of the challenges that cause data loss in networks.

The Information Bottleneck Principle describes how as data passes through successive layers in a network, the potential for information loss increases.The Information Bottleneck Principle is represented mathematically as: I(X, X) >= I(X, ftheta(X)) >= I(X, gphi(ftheta(X))) where I denotes mutual information, f and g represent transformation functions with parameters theta and phi. PGI helps address this problem by preserving data that is considered essential throughout the depth of the network. This allows for more reliable gradient generation and better model performance.

A Reversible Function is a function if it can be reversed and there is no loss in information. Mathematically, this is expressed by X = vzeta(rpsi(X)) where psi and zeta are parameters for the reversible function and its inverse. By utilizing reversible functions, a network can retain complete information flow during the back propogation and feed forward actions of the network.

YOLOv9 implements reverse functions, especially later in the network to ensure the preservation of critical data.

Another challenge of data loss is as a model loses parameters, the chance of information loss increases. This means that lighter weight models are often more prone to data loss even though they are more efficient. However, despite YOLOv9's lighter GELAN model, the use of PGI and reversible functions ensures model performance despite being streamlined. GELAN allows for integration of various computational blocks making it adaptable to a wide range of applications.

A.2 MediaPipe Hand

Google's MediaPipe is an open source framework that blends the digital and physical world together through various machine learning pipelines. The MediaPipe Hand pipeline uses a machine learning pipeline to infer the location of 21 hand landmark locations in dimensions. Initially, the pipeline performs palm detection using a single-shot detector which utilizes a single deep neural network to combine multibox detection and convolutions to predict object classes and bounding boxes. After the palm of the hand is recognized, the model proceeds to pinpoint 21 knuckle coordinates using regression.

The original model was trained on a ground truth dataset of about 30 thousand ground truth images that are all annotated by hand, landmark by landmark. As a result, the MediaPipe Hand model performs well at pinpointing the 21 knuckle locations even when the hand is partially visible or occluded.

A.3 XGBoost

XGBoost (short for eXtreme Gradient Boosting) is a an optimized gradient boosting method that utilizes tabular data. It constructs decision trees sequentially, with each new tree correcting the errors of its predecessors.

XGBoost's objective function is a combination of a loss function and a regularization term. The objective is defined as:

Objective Function (Obj) = Sum of Losses + Sum of Regularization Terms

Here, the loss function quantifies the difference between the actual and predicted values across all instances in the dataset. The regularization term is added to control the model's complexity, helping to prevent overfitting. This term is important because it penalizes the model for having too many branches or leaves in the trees, thus keeping the model simpler.

The regularization term itself is a function of the number of leaves in a tree and the scores on these leaves, adjusted by two parameters that control the complexity penalty. The model iteratively adds trees, focusing on the gradients of the loss, effectively reducing the error over time.

XGBoost stands out due to its efficiency and performance, incorporating features like handling missing data, supporting various objective functions, and optimizing computational resources. It's a powerful tool in the machine learning toolkit, favored for its speed and accuracy.