Real-Time American Sign Language (ASL) Recognition with Visual and Pose-Based Classification

Armando Borda Elisabeth Holm
Stanford University
aabd@stanford.edu, eholm@stanford.edu

Abstract

American Sign Language (ASL) is essential for communication in the Deaf community, yet technology struggles to recognize it accurately in real-world conditions. We developed and compared two ASL alphabet recognition models: a YOLOv11-based image classifier and a MediaPipe-based hand landmark classifier using a PyTorch multilayer perceptron. We trained both models on three datasets: D1 (controlled conditions), D2 (more diverse), and D3 (a combination of both D1 and D2).

Dataset test performance was nearly perfect for all models, however, real-time webcam testing revealed weaknesses, especially for moving or visually similar letters, as well as varied lighting conditions. The MediaPipe model consistently outperformed YOLOv11 in real-time settings, achieving 64.92% accuracy on D3 compared to YOLOv11's 61.25%. We also observed improved accuracy as training data became more diverse, highlighting the importance of robust datasets.

These findings emphasize the gap between controlled dataset performance and real-world usability. Insights from static letter classification can help guide future work on dynamic video-based sign recognition, moving towards real-time translation systems that support full ASL communication.

1. Introduction

1.1. Motivation

American Sign Language (ASL) is a primary tool for communication in the Deaf and hard-of-hearing community. However, the majority of the population is not fluent in ASL, creating barriers in education, healthcare, customer service, and everyday life. Bridging this gap with technology could increase accessibility and inclusion.

1.2. Project Objective

In this project, we address the problem of recognizing static ASL alphabet gestures using deep learning-based computer vision models. We focus on the classification of images representing the 26 letters of the ASL alphabet plus three additional non-alphabet classes (space, delete, and nothing). The goal is to build a system that performs robustly on both clean, curated datasets and in real-world settings, such as real-time webcam input.

We developed and compared two model architectures for multi-class image classification: one based on object detection with YOLOv11 [2], and another using hand landmark detection with MediaPipe [3] combined with a PyTorch multilayer perceptron (MLP). We trained these models on three datasets, which we refer to as D1, D2, and a combined dataset D3, which merged D1 and D2. We evaluated the models' performance by measuring accuracy on the datasets and during real-time webcam testing to assess their usability in live settings.

1.3. Overview of Results

Our results show that both models achieve high accuracy on curated datasets, but their performance drops significantly under real-world conditions. This highlights the challenges of generalizing to diverse backgrounds and lighting conditions. By combining multiple datasets, we improved real-world accuracy, suggesting that more robust and diverse training datasets are key for real-world applications. Across all three datasets, the MediaPipe model consistently outperformed the YOLO model in real-time accuracy, indicating that pose-based models are better equipped to handle variability in real-world settings. Looking ahead, we propose strategies for further improving generalization, such as incorporating additional data sources, and we outline potential extensions to dynamic video-based sign recognition using the MS-ASL dataset [4].

2. Related Work

To provide context for our work, we use prior research in the field of ASL recognition. One such study is Live Non-isolated Sign Language Recognition Using Transformers by Yang and Farah (2024) [8], which outlines the difficulties of building a live ASL interpretation system. Their work shows that accurately classifying individual static letters of the ASL alphabet is challenging but a necessary foundational step before attempting full sentence translation or live video interpretation. They experimented with different model architectures and they concluded that combining MediaPipe for efficient hand landmark detection with Vision Transformer-based models led to the best results. This combination of lightweight preprocessing and sequence modeling was very good in capturing subtle hand movements and puts emphasizes on breaking down ASL tasks into more manageable steps in order to build full translation systems.

Another relevant study we reference is Classifying Sign Languages and Fingerspellings with Convolutional Neural Networks by Vu, Chen, and Wong (2022) [7]. This project focused on recognizing static fingerspelled signs across multiple sign languages using a two-model pipeline: they combined a R-CNN-based hand detector with a CNN classifier, which executed what was called the "crop-and-classify" approach. The system first detected and cropped the hand out of the image and then fed it into a classifier that predicts both the sign language and the letter being signed. The results showed that the crop-and-classify method was around four times more accurate than classification without cropping. This finding shows that explicitly detecting and isolating the hand improves generalization, especially in settings where there is a lot of image-related noise. This approach worked really well in low-resource scenarios, suggesting that detection-first pipelines can be more resilient when working with smaller or more diverse datasets.

Moreover, ElSayed et al presented a more streamlined, CNN-based approach in their paper Vision-Based American Sign Language Classification via Deep Learning (2022) [1]. Their work focused specifically on classifying static ASL alphabet letters using a lightweight convolutional neural network that avoided the complexity of segmentation or transfer learning. One of the core strengths of their model lies is in its simplicity, by not requiring a lot of preprocessing or hardware-intensive methods. To address the common challenge of limited training data in ASL classification, they applied multiple forms of data augmentation like Gaussian noise and rotational transformations. In our case, given our large access to data, we didn't focus our efforts in data augmentation.

Our study balances performance with accessibility by focusing on lightweight models and realistic testing conditions (real-time environment). While many recent approaches in ASL recognition push toward increasingly complex architectures or specialized hardware, our goal was to develop a system that could be accurate and efficient. By comparing two fundamentally different models—YOLOv11 for image-based detection and MediaPipe with a PyTorch MLP for pose-based classification—we were able to explore trade-offs between computational cost, real-time performance and robustness.

3. Data

We used three datasets to train and evaluate our ASL gesture recognition models.

The first dataset (D1) is the Kaggle ASL Alphabet dataset [5], which consists of over 87,000 labeled images of hands captured through a webcam. These images were captured under controlled conditions with consistent lighting and uniform backgrounds, providing clean and high-quality samples for training. However, this homogeneity posed a challenge for generalization to real-world scenarios, where lighting and backgrounds vary significantly. This was particularly seen in the models' performance under real-time video testing, explained later in the report.

To address this limitation, we incorporated a second dataset (D2) from Kaggle [6], which contains 223,000 labeled images of the ASL alphabet captured through a webcam, but with greater variation in background, lighting, and hand positions. We chose this dataset to help the models learn to recognize gestures under more realistic conditions.

Because both D1 and D2 shared the same classes, we could combine them into a single dataset (D3) to increase the overall data diversity and quantity. This combined dataset allowed us to expose the models to a wider range of examples during training, helping them generalize better to real-world inputs.

Since the datasets already included augmented versions of their images, we did not do any additional data augmentation or preprocessing, other than changing the filenames of images to not overwrite each other during the dataset combination process.

4. Methods

We wanted to compare a visual-based approach (YOLO) with a landmark-based approach (MediaPipe + PyTorch) so we could evaluate the trade-offs between direct image classification and landmark feature extraction.

4.1. YOLOv11-based Classification

Our first pipeline uses YOLOv11 [2], a real-time object detection framework that we chose for its speed and efficiency, which allows for real-time inference. We adapted the final classification layer to output 26 letter classes + 3 non-alphabet classes (space, delete, and nothing) and trained the model using transfer learning on the ASL Alphabet dataset [5]. The training process involved standard preprocessing, resizing input images, and fine-tuning on our labeled dataset using the hyperparameters in Table 1.

We trained the model for 20 epochs, which we determined to be enough after observing that both training and validation loss plateaued by that stage. Training beyond 20 epochs resulted in diminishing returns and increased risk of overfitting—particularly for clean datasets like D1. Training took a total of 3 hours per model, running on a T4 GPU through Amazon Web Services (AWS).

A batch size of 32 was chosen to strike a balance between memory efficiency and gradient stability, as smaller sizes introduced noisier gradients. A size of 32 allowed us to process multiple examples per batch while keeping training time reasonable.

We used a learning rate of 1e-2, which is higher than typical default rates for deeper CNNs but very well-aligned with YOLOv11's lightweight structure. Since the model was already pretrained on ImageNet, a higher learning rate helped the classifier quickly adapt to the ASL-specific dataset without getting stuck in suboptimal minima.

The input images were resized to 224x224 pixels. We specifically used the YOLO11n-cls model: this framework was 8x faster than the largest one offered by Ultralytics, and was built with 1.6 million parameters and 0.5 billion FLOP capabilities.

Hyperparameter	**Value**
Image Size	224
Epochs	20
Batch Size	32
Learning Rate	1e-2

Table 1. Training hyperparameters used in our YOLO model.

4.2. MediaPipe + PyTorch-based Classification

To address the challenge of ASL gesture recognition in diverse real-world settings, we implemented a model that uses MediaPipe's hand landmark detection [3] to extract robust, position-invariant features from images. MediaPipe detects 21 key hand landmarks per image, each represented by 3D coordinates (x, y, z). We stored these landmarks in JSON files. Landmark extraction was the most time-consuming part of preprocessing and training, achieving a processing speed of approximately 27 images per second, which meant processing Dataset 2 took about 2 hours on a Macbook with an M1 chip.

After extraction, we flattened the 21 landmarks into a 63-dimensional feature vector per frame, which was then classified using a PyTorch-based multilayer perceptron (MLP). Training was relatively fast given the low dimensionality of the input feature vectors, taking roughly one minute per model, again on an M1 chip.

The PyTorch MLP consisted of three fully connected layers with ReLU activations and a final softmax output for the 29 gesture classes (26 letters plus space, delete, and nothing). We tuned the hyperparameters (Table 2) to match those used in our YOLO-based model where possible, allowing a fair comparison between approaches.

We chose this MediaPipe approach to leverage spatial rather than visual information, making the model less sensitive to background noise and lighting variations.

Hyperparameter	**Value**
Image Size	224
Epochs	20
Batch Size	32
Learning Rate	1e-3

Table 2. Training hyperparameters used in our MediaPipe model.

4.3. Real-Time Testing

To evaluate our models in a realistic scenario, we implemented a real-time testing pipeline using OpenCV and the computer webcam feed to continuously capture and process frames. We found this to be an especially crucial step in testing after seeing a large discrepancy between the dataset and real-time inference accuracy for the initial models, which were just trained on D1.

For the MediaPipe-based model, we first passed the image through the pre-trained hand landmark detection model. Then, the resulting 21 hand landmarks were again flattened into a 63-dimensional vector and passed through our trained PyTorch classifier to predict the corresponding ASL letter.

For the YOLO-based model, the input frame was passed directly through the trained YOLO classifier. In our real-time script, we captured continuous frames using OpenCV and ran YOLO inference on each frame to identify the predicted ASL class, with the model returning the top predicted class along with a confidence score.

We tested each letter class by capturing 150 frames (at 30 frames per second for 5 seconds) per letter, moving around our hand during each letter to test various angles, positions, and lighting. The program waited until the user pressed a key before continuing to the next letter, allowing time to transition their hand from one letter to the next. At the end of testing each model on each dataset, the program displayed the accuracy for each letter by calculating the percentage of correct predictions, which we then averaged to get the total real-time accuracy.

A demonstration clip of the real-time test for the MediaPipe-D3 model can be found here.

4.4. Alternative Approaches Considered

We initially considered using only an image-based classifier (e.g., YOLO) for static ASL alphabet recognition, as

	YOLO	MediaPipe
D1	100%	97.11%
D2	99.90%	97.24%
D3	99.92%	98.49%

Table 3. Accuracies on test data for both the YOLO and MediaPipe pipelines.

it would be the simplest solution. However, given the high similarity among certain ASL letters and the variability of real-world, we hypothesized that using hand landmark features could provide better generalization. This is why we incorporated the MediaPipe-based approach. We also considered integrating temporal information (e.g., dynamic video-based classification) for continuous gesture recognition but deferred that extension to future work due to the time constraints of the quarter.

5. Experiments

5.1. Dataset Results

Both of our pipelines achieved near-perfect accuracy on testing data for all three datasets, as shown in Table 3. These results indicate that the models are very capable of distinguishing between ASL letters under controlled conditions. Additionally, for YOLO, we performed sample predictions by selecting random images, passing each one through the model, and printing the predicted label, true label, and confidence score. All the predicted labels were correct and the confidence score ranged from 98.99% to 99.99%. This further showed that the dataset prediction task didn't present major challenges to YOLOv11.

However, the discrepancies in real-time accuracies show that we must be cautious, not relying solely on dataset accuracies before deploying applications to the real world.

5.2. YOLO Real-Time Results

After running the real-time test of the YOLO models trained on D1, D2, and D3, we calculated the per-letter accuracy, with a comparison of model results shown in Figure 1. The exact numerical per-letter accuracies are in Appendix A.1.

For the real-time testing, we experimented by varying three aspects of the hand language gestures we showed: the position in front of the camera, the angle, and slight differences with finger positioning (while still representing the same sign). Ilumination also played a big factor. With the real-time test, we could assess not just static dataset performance but also practical usability in live settings.

To summarize, D1 was greatly outperformed by D2 and D3 trained models. The D1-trained model for the YOLO Real-Time experiment performed very poorly with multiple letters, including P, Q, R, S, T, where it's accuracy was near 0%. The D2 model saw drastic improvement, obtaining an accuracy of 80% or higher for 10 letters. The D3-trained model improved from the D2-trained model but not as much as the change from using D1 to D2: although only 9 letters received an accuracy of 80 percent or higher, overall it received higher scores by obtaining a total accuracy of 61.25% compared to the 60.44% of D2. This difference was marginal compared to the very poor 26.71% accuracy of the D1 trained model.

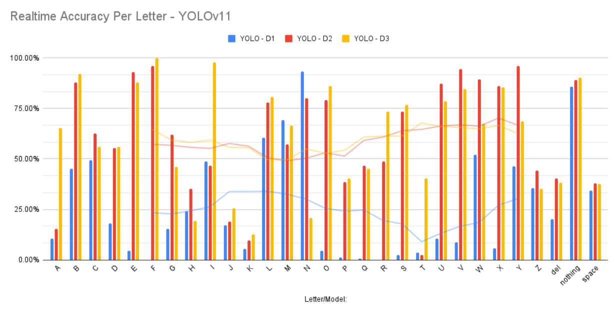


Figure 1. Per-Letter Realtime Accuracy for the YOLO model trained on D1, D2, and D3.

Some exceptions where the D1-trained model outperformed the other two was for the letter M and N. Meanwhile, the D2-trained model performed the best for the letters C, E, G, H, Q, U, V, W, X, Y, Z, del and space. However, the difference between some of the letter accuracies between D2 and D3 when D2 had a higher accuracy was small (a couple of percentage points); The D3 model, when it had the highest accuracy for a letter, sometimes outperformed by a much larger amount.

5.3. MediaPipe Real-Time Results

Similarly to YOLO, after running the real-time test of the MediaPipe models trained on D1, D2, and D3, we calculated the per-letter accuracy, with a comparison of model results shown in Figure 2. The trendlines are running averages of accuracies, and while the horizontal trend doesn't mean much, the vertical stack shows an overall ranking of the models with D3 performing best, then D2, then D1 last. Again, the exact numerical per-letter accuracies are in Appendix A.1.

As we can see, D3 significantly outperformed D1 and D2 on most letters, except for S and U, which performed better under D1 and D, E, K, M, and T, which performed better under D2. D2 and D3 performed comparably for some letters, likely due to D2 making up a majority of D3. Some letters, such as C saw a $\tilde{9}0\%$ increase in real-time accuracy from D1 to D3, aligning with our hypothesis that a more robust dataset would increase real-time accuracy by increasing generalizability.

Certain letters still presented challenges. For example,

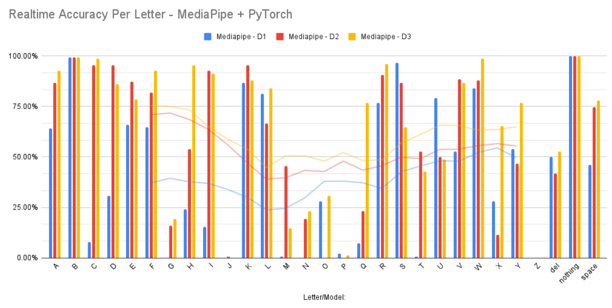


Figure 2. Per-Letter Realtime Accuracy for the MediaPipe + PyTorch model trained on D1, D2, and D3.

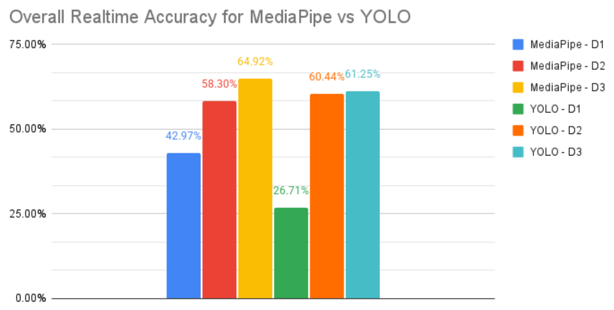


Figure 3. Overall Realtime Accuracy for the MediaPipe and YOLO models trained on D1, D2, and D3.

letters with similar shapes (e.g., M, N, and T) or those with subtle hand orientations (e.g., Q, P, and Z) showed persistently low accuracy across all datasets.

Some letters, such as B, consistently achieved near-perfect performance, suggesting that they have distinctive and easily separable landmark patterns. Since the landmark detection model was very good at detecting hands, the ”nothing” class had 100% accuracy for all models, as it returned ”nothing” if no hand was detected in the frame.

These results exhibit the importance of dataset diversity and robustness for generalizing to real-world scenarios. They also illustrate the limitations of using only landmark-based features, which may struggle with occlusions, left-handed signs, and sometimes unconventional hand orientations.

5.4. Comparison of Models

In Figure 3 we see the overall real-time accuracies for each model trained on each dataset.

We see that the MediaPipe model trained on D1 (the clean but homogeneous dataset) achieved a total real-time accuracy of 42.97%, highlighting significant difficulties in generalizing from clean images to varied real-world conditions. Training on D2 improved the overall real-time accuracy to 58.30%. The best performance was from D3, the combined dataset, with a total real-time accuracy of 64.92%.

In comparison, the YOLO-based model achieved even lower real-time performance on D1, with a total real-time accuracy of only 26.71%. This result highlights the YOLO model’s sensitivity to domain shift, likely due to its reliance on raw image features rather than the abstracted hand landmarks used by the MediaPipe model. Training on D2 improved YOLO’s real-time accuracy to 60.44%, and on D3 to 61.25%, showing a similar trend to MediaPipe in benefiting from more diverse training data.

YOLO’s overall performance consistently lagged behind the MediaPipe model across all datasets, suggesting that landmark-based approaches may be better suited to this task, especially in real-world conditions with varied lighting and backgrounds.

6. Conclusion

In this project, we explored the challenge of American Sign Language (ASL) letter recognition using both a YOLO-based image classification model and a MediaPipe-based landmark detection approach. Our experiments demonstrated that while both models performed well on curated, static datasets, real-time testing revealed significant drops in accuracy, particularly for letters with similar hand shapes (such as ‘M’ and ‘N’) or those involving motion (such as ‘J’ and ‘Z’). This highlights the importance of testing in realistic, dynamic environments rather than relying solely on static datasets.

We found that incorporating additional data diversity by combining datasets significantly improved real-time performance, exhibiting the need for models that can generalize across a range of lighting conditions, backgrounds, and hand variations. Our MediaPipe-based landmark model consistently outperformed the YOLO-based approach across all datasets, suggesting that pose-based methods offer better robustness to background noise and lighting changes.

While visual-based models like YOLO show promise, they may benefit from further preprocessing steps such as cropping to the hand region before classification as to mitigate their sensitivity to background clutter.

6.1. Future Work

Future work could explore several promising directions. First, incorporating hand detection and cropping in the YOLO pipeline could improve performance by reducing background noise. Second, extending the system to handle dynamic gestures, such as ‘J’ and ‘Z’, would require integrating temporal information, for example by using sequence models or recurrent architectures. Third, expanding the dataset with additional real-world variations, including different lighting, backgrounds, skin tones, and camera perspectives, would help improve generalization. Incorporating multi-angle training data would also be a big boost, either through more diverse dataset collection or through synthetic data generation (e.g., using 3D hand models to simulate rotations).

Insights from our tests on static images can inform future work on dynamic sign recognition. Specifically, combining pose-based features with temporal models (e.g., LSTMs or Transformers) could help the system track hand motion over time, enabling classification of continuous ASL word signs and phrases. Ultimately, we envision a world with real-time translation systems that support full ASL communication to bridge the gap between the deaf and hearing communities.

References

[1] Elsayed et al. 2024. "Vision-Based American Sign Language Classification Approach via Deep Learning." *Arxiv.* https://arxiv.org/abs/2204.04235.

[2] Jocher, Glenn, and Jing Qiu. 2024. *Ultralytics YOLO11.* https://github.com/ultralytics/ultralytics.

[3] Google. 2019. *MediaPipe.* https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker.

[4] Microsoft. 2024. "MS-ASL American Sign Language Dataset." July 15, 2024. https://www.microsoft.com/en-us/download/details.aspx?id=100121.

[5] Nagaraj, Akash. 2018. "ASL Alphabet." *Kaggle.* https://www.kaggle.com/grassknoted/asl-alphabet.

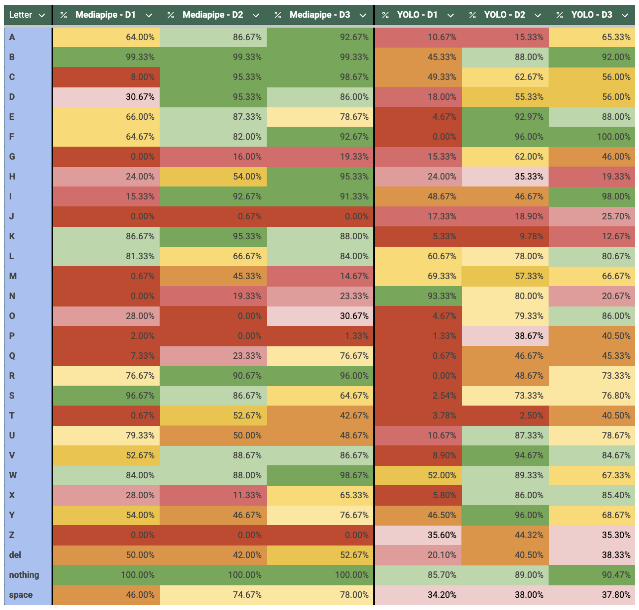
[6] Sau, Debashish. 2021. "ASL(American Sign Language) Alphabet Dataset." *CS231N.* https://cs231n.stanford.edu/reports/2022/pdfs/23.pdf.

[7] Vu at al. 2022. "Classifying Sign Languages and Fingerspellings with Convolutional Neural Networks." *Semantic Scholar.* https://api.semanticscholar.org/CorpusID:273696110.

[8] Yang, Daniel and Ethan Farah. 2024. "Live Non-isolated Sign Language Recognition Using Transformers." *Semantic Scholar.* https://api.semanticscholar.org/CorpusID:273696110.

Appendix A

A.1. MediaPipe and PyTorch D1, D2, and D3 Model Accuracies Per-Letter for Real-Time Testing



Letter	% Mediapipe - D1	% Mediapipe - D2	% Mediapipe - D3	% YOLO - D1	% YOLO - D2	% YOLO - D3
A	64.00%	86.67%	92.67%	10.67%	15.33%	65.33%
B	99.33%	99.33%	99.33%	45.33%	88.00%	92.00%
C	8.00%	95.33%	98.67%	49.33%	62.67%	56.00%
D	30.67%	95.33%	86.00%	18.00%	55.33%	56.00%
E	66.00%	87.33%	78.67%	4.67%	92.97%	88.00%
F	64.67%	82.00%	92.67%	0.00%	96.00%	100.00%
G	0.00%	16.00%	19.33%	15.33%	62.00%	46.00%
H	24.00%	54.00%	95.33%	24.00%	35.33%	19.33%
I	15.33%	92.67%	91.33%	48.67%	46.67%	98.00%
J	0.00%	0.67%	0.00%	17.33%	18.90%	25.70%
K	86.67%	95.33%	88.00%	5.33%	9.78%	12.67%
L	81.33%	66.67%	84.00%	60.67%	78.00%	80.67%
M	0.67%	45.33%	14.67%	69.33%	57.33%	66.67%
N	0.00%	19.33%	23.33%	93.33%	80.00%	20.67%
O	28.00%	0.00%	30.67%	4.67%	79.33%	86.00%
P	2.00%	0.00%	1.33%	1.33%	38.67%	40.50%
Q	7.33%	23.33%	76.67%	0.67%	46.67%	45.33%
R	76.67%	90.67%	96.00%	0.00%	48.67%	73.33%
S	96.67%	86.67%	64.67%	2.54%	73.33%	76.80%
T	0.67%	52.67%	42.67%	3.78%	2.50%	40.50%
U	79.33%	50.00%	48.67%	10.67%	87.33%	78.67%
V	52.67%	88.67%	86.67%	8.90%	94.67%	84.67%
W	84.00%	88.00%	98.67%	52.00%	89.33%	67.33%
X	28.00%	11.33%	65.33%	5.80%	86.00%	85.40%
Y	54.00%	46.67%	76.67%	46.50%	96.00%	68.67%
Z	0.00%	0.00%	0.00%	35.60%	44.32%	35.30%
del	50.00%	42.00%	52.67%	20.10%	40.50%	38.33%
nothing	100.00%	100.00%	100.00%	85.70%	89.00%	90.47%
space	46.00%	74.67%	78.00%	34.20%	38.00%	37.80%

Per-Letter Accuracy (num_correct/150 tested frames) for each class and each model. We assigned colors based on the accuracy values, grouped by the tens place (eg 0%-10%, 10% - 20%, etc). The bar graphs in Figure 1 and 2 show the same information, but this shows the numerical detail.