American Sign Language Character Interpreter

Matthew Laws mdl4@williams.edu
Williams College Computer Science

1. Introduction

According to Mitchell and Young (2022) over 5 million Americans are completely deaf or have significant trouble hearing; however, only 2.8% of the population knows American Sign Language (ASL). For individuals that are hearing impaired, communicating with people who do not know ASL is difficult and hiring translators is expensive. Creating an tool that would allow people to translate between ASL and spoken English would help many more people to communicate, and expand everyone's knowledge.

The goal of this project is to develop a tool that can effectively translate ASL characters into English characters. To accomplish my goal, I trained a Convolutional Neural Network on a dataset of images of ASL characters. My network performs a multi-categorical classification that identify input images as of the 29 possible characters. Currently there exist several implementations of ASL character recognition; however, they are insufficient on several fronts. Firstly, they overfit to their data distribution. Using generic object detection fails to generalize well on hands different from the training data. Secondly, character level translation is not very useful considering people speak in words, thus a character specialized model does not have the same real world applications as a word or sentence level one.

I used hand tailor pre-processing to improve on the state-of-the-art and developed a model that accurately predicts images with strong generalization. Beyond that, I extended the project to predict a series of images in sequence as a word. I utilized a secondary model that took the individually translated characters and applied an auto correct algorithm to reduce the frequency of errors at the word level. Finally I implemented a video translation feature that uses the prior models to translate ASL fingerspelling in real time.

Overall the model produced strong results production a test accuracy of 97% on a withheld test set. Additionally, with auto correction, the model predicted words correctly 95% of the time. Finally the video was able to correctly identify letters with passable reliability.

2. Preliminaries

2.1 ReLU Activation

Many machine learning models including the ones used in this paper require a non-linear activation function. I choose ReLU due to its simplicity to calculate, and that its gradient does not approach zero if weights are large in magnitude. The ReLU function is piece wise function that is defined by $\mathbf{ReLU(x)} =$

$$\begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

2.2 Sparse Categorical Cross Entropy

For the loss function this model uses Cross Entropy which is simply defined as:

$$-\sum_y P^*(y|x_i) \log P(y|x_i; \theta)$$

Where P^* is the true probability that some input x_i belongs to a certain class y, and P is the prediction of the model given θ. Since this is modeling a probability distribution, a softmax modeling constraint is applied to normalizes the outputs to produce a valid probability distribution. When training the model the goal is to find the vector θ that minimize the cross-entropy.

2.3 Backpropagation

Backpropagation is a direct application of the chain rule of calculus that allows us to compute all gradients in a single backwards pass allowing us to leverage the cheap gradient principle.[1] The formula for the gradient with θ of length n is:

$$\nabla L(\theta) = \left[\frac{\partial L}{\partial \theta_1}(\theta), \frac{\partial L}{\partial \theta_2}(\theta), \ldots, \frac{\partial L}{\partial \theta_n}(\theta)\right]$$

The chain rule states that the derivative with respect to any node n with edges to k nodes $n_1, n_2, \ldots, n_k$ can be calculated as:

$$\frac{\partial L}{\partial n} = \sum_{i=1}^{k} \frac{\partial L}{\partial n_i} \cdot \frac{\partial n_i}{\partial n}$$

where L is the final loss. In a backwards pass the deviates $\frac{\partial L}{\partial n_i}$ would have already been calculated, thus using backpropagation the gradient can be calculated in linear time.

2.4 Adam Optimization Algorithm

Adaptive Moment Estimation (Adam) is a combination of two other extensions of SGD: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) and has the benfits of both. Adam can be thought of as behaving like a heavy ball with friction rolling around the loss function. Adam, with small constants β_1, β_2, and ϵ and learning rate α, uses the following rule to update the parameter array θ:

$$\theta_{t+1} = \theta_t - \tfrac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

1. The cheap gradient principle states that computing the gradient of the loss is bounded to being no more than a 5 times more computationally complex to compute

where m_t and v_t are defined as follows:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

And $\hat{m_t}$ and $\hat{v_t}$ are defined by:

$$\hat{m_t} = \frac{m_t}{1-\beta_1^t} \qquad \hat{v_t} = \frac{v_t}{1-\beta_2^t}$$

As shown, $\hat{m_t}$ captures the concept of momentum, if the gradient continually points in the same direction the model will begin to take larger steps.[2] $\hat{v_t}$ accounts for the variance of passed gradients, as it increase, the amount we update by decreases. This is because if the gradient is changing a lot we don't know which way is the correct way to go and don't want to take big steps. Finally we use $\hat{m_t}$ and $\hat{v_t}$ to decay our learning rate over to lead to better convergence (Ruder, 2017).

2.5 Fully Connected Neural Networks

A fully connected neural network (NN) is a network that contains an input layer, 1 or more hidden layers, and an output layer. The input layer contains one node per feature of the input. In the case of multi-class classification, the output layer has a node for each of the potential classes that represents the probability that a given input belongs to that class. A NN is a directed acyclic graph[3] where nodes in layer L_i can only have parents from layer L_{i-1}. In a fully connected NN L_i has all nodes in L_{i-1} as parents see figure 1. Each edge represents a learnable weight that forms a weight matrix θ that defines the values of the nodes of layer L_i as a function of the previous layer L_{i-1} by the following formula:

$$L_i = \text{Activation}(L_{i-1}[\theta]^{L_{i-1} \to L_i})$$

Where we use ReLU as our Activation and $[\theta]^{L_{i-1} \to L_i}$ is the vector of weights between layers L_{i-1} and L_i. The goal of NNs is to learn a vector θ that contains all $[\theta]^{L_{i-1} \to L_i}$ such that given some input X, after applying the weights of the NN the distance of the prediction ($\hat{Y}$) to the true value (Y) is minimized.

2.6 Dropout

Dropout is implemented to build robustness to overfitting. Dropout is as follows: for a neural network with n edges – parameters – and dropout probability p, a vector D of Bernoulli(1-p) random variables of length n is initialized. Then the element wise product of the edges and D each iteration. This effectively turns off certain edges by setting their value to 0. See figure 2.

2.7 Convolutional Neural Network

A convolutional neural network (CNN) is a network that uses convolutional layers to augment fully connected layers. CNNs are effective at learning features from the data using

2. like how a ball might roll down a hill
3. a directed acyclic graph is defined as a graph G where G is simple, contains only directed edges, and there are no directed cycles

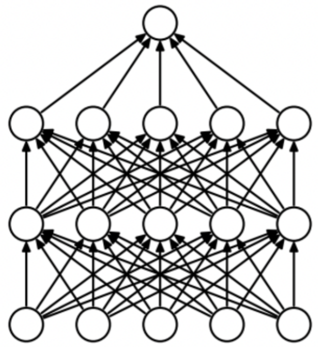


Figure 1:
Standard Neural Net

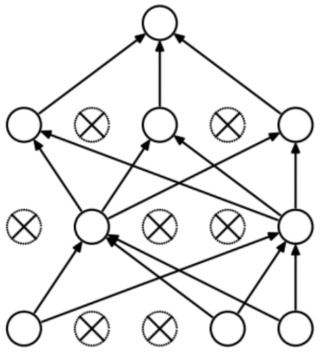


Figure 2:
Same NN with Dropout

locality. Overall CNNs use a combination of learned parameters to obtain a prediction vector $\hat{Y}$ given some input input X that estimates the true value Y. A convolutional layers relies on the convolution function to compute weights. The convolution of functions f and g is defined by the following:

For each $X_{i:i+k,j:j+k}$ in a image X defined as a $n \times m$ vector of pixels with 1-indexed domain: $1 \leq i \leq n-3; 1 \leq j \leq m-3$. A $k \times k$ filter is applied as seen below:

$$c_{p,q} = \sum_{i=1}^{k} \sum_{j=1}^{k} w_{i,j} \cdot x_{i+p,j+q} + \text{bias}$$

here (p, q) is the coordinate of the filter's top left corner and the weights ($w_{i,j}$) are learned through training the network. The output of a convolutional layer – which is also a two dimensional vector – is an aggregation of all the $c_{p,q}$ such that the feature at position (p, q) in the output layer is set to $c_{p,q}$ see figure 3.

An important feature of CNNs is that each feature in the new layer only sees a portion of the input, namely the size of the filter. By aggregating smaller sections of the image together into a single node, the network is able to learn features in a broader space than a single pixel. For example the position of a finger tip is a useful piece of information for identifying sign language numbers, a CNN could be able to identify a finger tip itself[4] as a characteristic rather than individual pixels (Amini, 2023). These learned features are then passed to a standard NN to complete the learning.

2.8 Max Pooling

A $k \times k$ max pooling function takes values in a convolutional layer L $L_{i:i+k,j:j+k}$ and returns the max of all the candidate values. The $k \times k$ filter is then shifted over by a set amount and the process is repeated. In order to maximally reduce dimensionality without explicitly loosing any features a shift of $k \times k$ is ideal. See figure 4 for example.

4. Or really a collection of pixels that resemble a finger tip

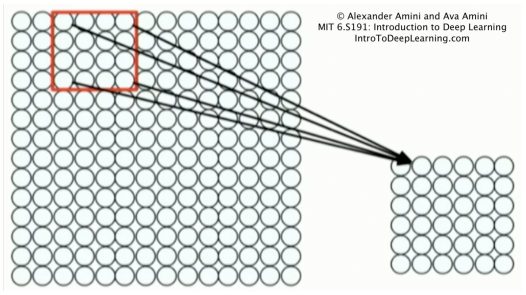

© Alexander Amini and Ava Amini
MIT 6.S191: Introduction to Deep Learning
IntroToDeepLearning.com

Figure 3:
Visualization of the construction of the convolution layer using a 4×4 filter and a 2×2 shift Amini (2023)

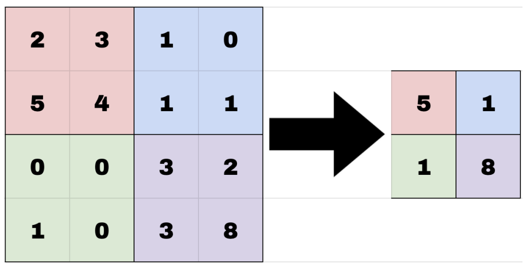


Figure 4:
Visualization of max pooling using a 2x2 filter and 2x2 shift

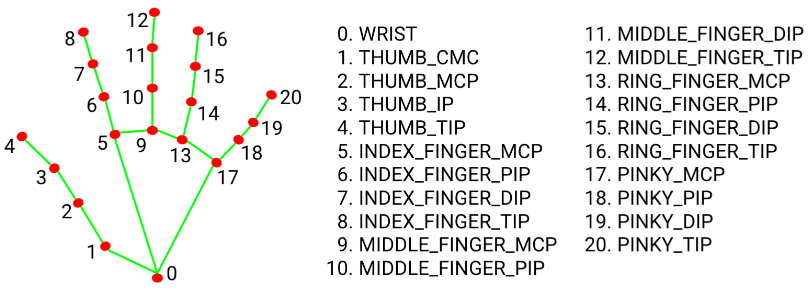


Figure 5: Hand Mapping Landmarks (Zhang et al., 2020)

2.9 SymSpell Auto Correction (Garbe, 2012)

Symmetric Delete Spelling Correction (SymSpell), is a spelling correction algorithm developed by Garbe (2012) and his company, SeekStorm. SymSpell relies on calculating the edit distance between a target word the rest of the words in a given dictionary; however, unlike traditional algorithms SymSpell only considers deletions as potential edits[5]. This allows the algorithm to examine fewer potential actions at each check greatly reducing the complexity. It also caps the max edit distance at 2 to further remove dimensionality from queries. Finally SymSpell completes a pre-calculation step for the dictionary that calculates and adds the deletion distances from each dictionary term. The addition of these terms allows for $O(1)$ look up of potential correction candidates greatly increasing run time with a negligible performance trade off see Garbe (2012) for more detail.

2.10 MediaPipe Hand Detection (Zhang et al., 2020)

MediaPipe's hand detection algorithm uses a two part machine learning model recognize hands at 21 key features of them, displayed in figure 5. The first part of the algorithm is palm detection. In order to narrow the search region, the model first solves the easier

5. This takes advantage of the fact that inserting / deleting are symmetrical

task of detecting the palm and returns a bounding box for the hand using a single-shot detector as described in Bazarevsky et al. (2019).[6] Then the 21 landmarks from figure 5 are detected in the image using regression techniques similar to Simon et al. (2017). This mapping returns the precise locations of all landmarks in 2.5D coordinates – x, y, and relative z. See Zhang et al. (2020) for full paper.

3. Data

3.1 Datasets

The data I used for my analysis was a collection of 87,000 images with 3000 images of each of the following characters: a-z, delete, space, and no sign. The data was provided by Nagaraj (2018). Each image is 200x200 pixels and contains several different background states such as lighting differences, and backdrop. An example image from the dataset is shown in figure 6. I augmented this dataset with a synthetic dataset from Lexset (2022) with a collection of an additional 27,000 images – 1000 images of each letter and 1000 blank images. These images were 512x512 pixels; however were resized to match the 200x200 scale before any pre-processing occurred.

3.2 Pre-processing

I complete two stages of pre-processing to both improve my model's learning / generalization ability and to reduce the dimensionality. The main method of pre-processing is hand detection as described in section 2.10. Using Zhang et al. (2020)'s landmark features I used openCV to superimpose lines between finger joints and a circle at each finger tip. Each finger was encoded with a different color so the model can distinguish them easier see Figure 7.[7] Overall, this concept builds off domain knowledge that suggests the relative positions of fingers and specifically fingertips are a strong way to identify ASL letters. After the hand is identified and encoded, in order to avoid overfitting to certain hands or backgrounds, the mapping is superimposed over a black screen and only the mapping is used by the network, see Figure 8. The second step is to reduce the parameter size. A 200x200 pixel color image – a rgb 3-tuple – has 120000 different parameters. In order to design a simpler model I resize the images to 70x70 pixels or 14700 parameters. 70x70 pixels was the chosen dimension through experimentation.[8] I experimented converting images to greyscale; however, that had negative effects.

6. Since the palm is rigid compared to fingers it is a much easier target to detect
7. Note, the colors shown in figures 7 and 8 are enhanced so they can be detected by the human eye, in reality all of the rgb values are 1 or 0 to represent the 5 colors to create a more standardized image. For example where we see green (0, 255, 0) mapping to the thump, in the pre-processing that is encoded as (0, 1, 0).
8. It was the lowest dimensionality that still clearly displayed accurate hand-mappings

3.3 Limitations

3.3.1 Z and J

Letters j and z in the ASL alphabet require hand movement, thus it is impossible to capture the correct sign using a static image. My datasets use images that resemble the beginning of the sign as their entries. Although this is limitation of the model, it only requires a small adaptation of standard fingerspelling to make it work. Furthermore, with video translation signing the full letter will likely still yield strong results.

3.3.2 Missing Hands

When a hand was unable to identified in an image where a hand was present, the image was labeled as none and a blank images was returned. Since the goal of the project was translating hand images to characters not identifying hands, I don't believe this is a significant drawback to this study. Furthermore, mediapipe is good at recognizing hands in standard (non 200x200 pixel) quality images. Through empirical evidence mediapipe has not failed to recognize a hand that was in frame when given a remotely clear image.

3.4 Data Splits

I use a 80%-10%-10% training, validation, test split of the data supplemented by a smaller adversarial validation set that contains a different hand, blurrier images, and backgrounds that contain significantly more complex objects, see figure 9. This data was acquired from Rasband (2017), pre-processed identically, and is used to test generalization properties of the algorithm.

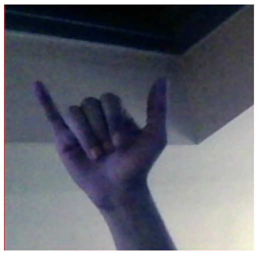


Figure 6: An image from the training set of the letter Y

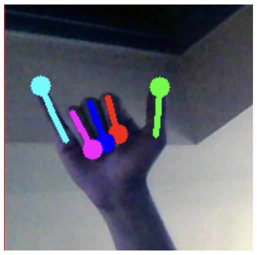


Figure 7: The same image after applying hand detection

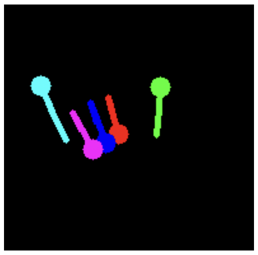


Figure 8: The same image after applying pre-processing

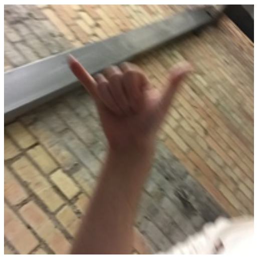


Figure 9: An image from the adversarial set of the letter Y

4. Training And Validation Of Models

The two main models I attempted to fit were a convoluted neural net (CNN), and a standard neural net (NN). After research I decided to use the adam optimizer with the following parameters: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.9$, $\epsilon = 10^{-7}$ as my optimiser for both. The creators of adam, Kingma and Ba (2017), suggest using these parameters to ensure the

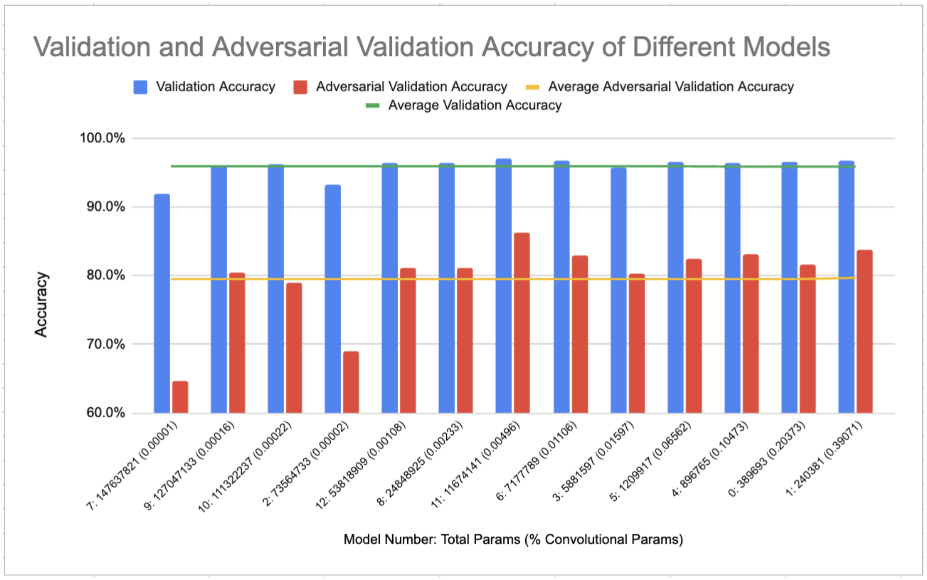


Figure 10: Comparison of Validation and Adversarial Validation Accuracy of Different CNN Models. The x-axis is sorted by most number of parameters to least. The percentage of parameters that were convolutional is also provided.

best learning. After some experimentation I trained two similarly complicated model, one NN and one CNN. The CNN outperformed the NN, see 7.3, thus I continued forwards with the CNN.

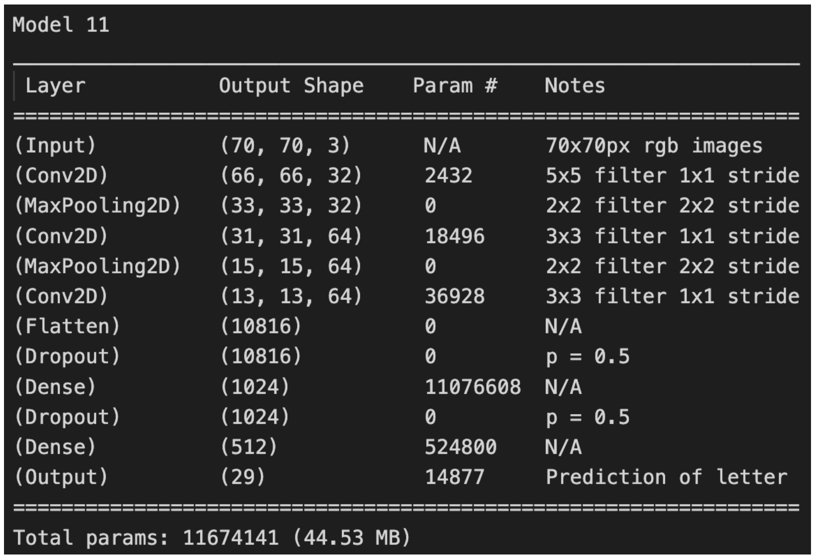
I trained a series of 13 different CNNs with varying number and size of convolutional and dense layers. Each trained for 10 epochs with an accuracy $\Delta < 0.1\%$ callback.[9] see figure 10 for a complete breakdown of the models. As seen in figure 10 model 11 performed the best on the adversarial and validation sets thus I chose this model as my final model. A summary of the layers of this model can be seen in figure 11.

5. Results

5.1 Baseline

As a baseline I trained a standard neural network with 43235357 total parameters – roughly 4x more complex than my final model – on unpre-processed data. This is a suitable baseline because it accounts for the two main ideas of this project, the convolution to detect features, and the hand-mapping to more clearly identify those features. Our baseline model achieved essentially naive test results with a training accuracy of 3.5%, test accuracy of 3.4%, and an Adversarial validation accuracy of 3.4%[10]. Despite this being a comparably large network it has no predictive power without additional insights.

9. If the model did not improve its accuracy by more than 0.1% in a given epoch it would stop early
10. This equates to guessing randomly because there are 29 potential outcomes and 100% / 29 = 3.4%



```
Model 11

 Layer           Output Shape    Param #    Notes
==================================================================
(Input)          (70, 70, 3)      N/A       70x70px rgb images
(Conv2D)         (66, 66, 32)     2432      5x5 filter 1x1 stride
(MaxPooling2D)   (33, 33, 32)     0         2x2 filter 2x2 stride
(Conv2D)         (31, 31, 64)     18496     3x3 filter 1x1 stride
(MaxPooling2D)   (15, 15, 64)     0         2x2 filter 2x2 stride
(Conv2D)         (13, 13, 64)     36928     3x3 filter 1x1 stride
(Flatten)        (10816)          0         N/A
(Dropout)        (10816)          0         p = 0.5
(Dense)          (1024)           11076608  N/A
(Dropout)        (1024)           0         p = 0.5
(Dense)          (512)            524800    N/A
(Output)         (29)             14877     Prediction of letter
==================================================================
Total params: 11674141 (44.53 MB)
```

Figure 11: Full description of all layers of the final model

5.2 Final Results

My final model ultimately achieved a 97.9% training, a 86.3% accuracy on an unlearned adversarial set, and a 97.1% test accuracy on an unseen testing set. This far out performed the baseline and demonstrates the strengths of my model.

I additionally conducted a word level recognition experiment. I created series of images from my test set that made up 999 common words with 50 different versions of each word. Then using the final model, I predicted each letter of a given word and concatenated them together to predict the word as a whole. Then I applied the SymSpell spelling correction discussed in section 2.9 to correct errors that the image prediction model made. Without SymSpell the model predicted the correct word 83.8%[11] of the time, and with SymSpell it improved to 94.9% of the time, demonstrating the efficacy of the additional spell check model.

I did not conduct any formal tests on the video detection, and there is still much to improve in these results. It was able to predict correct signs with decent empirical accuracy, but it is still in its early stages. The completion of this aspect of the project will require significant more time and is left as future work.

6. Ablation Study

For my Ablation Study I conducted a sensitivity analysis where I trained my model on the standard training set, but tested it on an unseen dataset gathered by Londhe (2021). The use of a different dataset meant a different hand, and represented a distribution shift from my training data. My model correctly predicted pre-processed images from the ab-

11. This is almost precisely what would be expected: with an average word length of 5.84 and a accuracy of 97.1% per letter one would expect a $97.1\%^{5.84} = 84.2\%$

lation set with only 35.4% accuracy. This seemed unusual given it had already proved to generalize well to unseen data with my adversarial set. However, I believe that my model performed poorly on the ablation set because it was left handed dataset. All of the images in the ablation set are mirrored to the initial dataset, see Appendix. This causes the model to struggle greatly because it learns based on relative finger positions, and when the hand is flipped the relative positions are changed. This was a interesting discovery, and certainly a drawback of my model. I could hopefully solve this issue by including left handed images in our training set; however, that will be left as future work.

7. Discussion and Conclusion

7.1 Discussion

Overall my model achieved strong results. As evidenced by the improvement from the baseline the combination of pre-processing and convolution worked well to create a model with strong generalization and resilience. Because CNNs learn spacial features to an extent that NNs are unable to, my model was able to better isolate the key features of a pre-processed input. This claim is evidenced by a 5% and a 15% decrease in test accuracy and adversarial validation accuracy respectively when a equally complicated NN was used for training. Pre-processing also played a large role. By eliminating all features that did not directly related to the letter – everything but the hand mappings show in figure 8 – the model only learns the important details for classification. This allowed the CNN to learn features that are much simpler than the original features.[12] Without having to identify hands in the learning stage, the model is able to more precisely learn features that correspond with ASL letters. This result is seen clearly when I train the same model on the same data but without the pre-processing: I get the following test and adversarial validation accuracies respectively: 98.7%, 43.2%. Although it gain a small amount of accuracy on data from the same distribution, it lost much of its ability to generalize. The CNN can more easily learn the relevant features detailed by pre-processing than in the base image itself. The model uses the combination of the accuracy provided by the CNN and the generalization of the pre-processing to be maximally effective.

7.2 Future Work

As future work I would like to implement several improvements to the project. The simplest is the add the left handed ablation data to my training set to aid in generalization. Additionally I want to implement ASL specific distance calculation. Certain ASL character look highly similar, A and E for example, and including these facts into my distance calculation would create a better spell checker tailored to ASL. Furthermore I would like to extend my auto correction to the sentence level so that the model could effectively translate sentences. Finally I want to finish implementing the video recognition. It is currently still in early stages of development and completing a system that could directly translate ASL fingerspelling to words on the page is the ultimate goal of this work.

12. For example: the difference between recognizing a blue circle and an index fingertip

7.3 Conclusion

Overall I learned a lot from this project. The most important lesson I learned was to apply domain knowledge and to not blindly use standard machine learning methods. I spend significant amounts of time attempting Canny edge detection[13]; however it did not compare the the efficacy of the hand mapping. Applying what I knew / learned about identifying signs helped inform which parts of the hand I decided to map instead of using the default mapping mediapipe ships. I would like to spend more time on this project pursuing the future work laid out in 7.2. This project fulfilled many of the goals it set out for, but seamless video translation remained out of reach. I hope to continue to work on this project and ultimately develop a seamless ASL to text translator.

13. a common object detection algorithm

References

Alexander Amini, Mar 2023. URL `http://introtodeeplearning.com/`.

Valentin Bazarevsky, Yury Kartynnik, Andrey Vakunov, Karthik Raveendran, and Matthias Grundmann. Blazeface: Sub-millisecond neural face detection on mobile gpus. *arXiv preprint arXiv:1907.05047*, 2019.

Wolf Garbe. 1000x faster spelling correction algorithm, 2012. URL `https://seekstorm.com/blog/1000x-spelling-correction/`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Lexset. Synthetic asl alphabet, 2022. URL `https://www.kaggle.com/datasets/lexset/synthetic-asl-alphabet`.

Kapil Londhe. American sign language, 2021. URL `https://www.kaggle.com/dsv/2184214`.

Ross E Mitchell and Travas A Young. How Many People Use Sign Language? A National Health Survey-Based Estimate. *The Journal of Deaf Studies and Deaf Education*, 28(1): 1–6, 11 2022. ISSN 1081-4159. doi: 10.1093/deafed/enac031. URL `https://doi.org/10.1093/deafed/enac031`.

Akash Nagaraj. Asl alphabet, 2018. URL `https://www.kaggle.com/dsv/29550`.

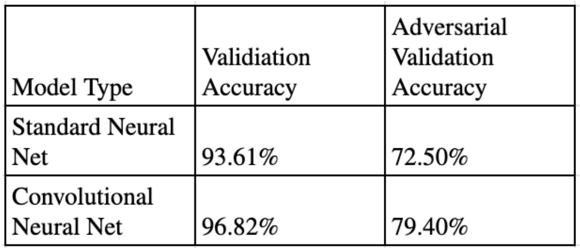
Dan Rasband. Asl alphabet test, 2017. URL `https://www.kaggle.com/datasets/danrasband/asl-alphabet-test`.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1145–1153, 2017.

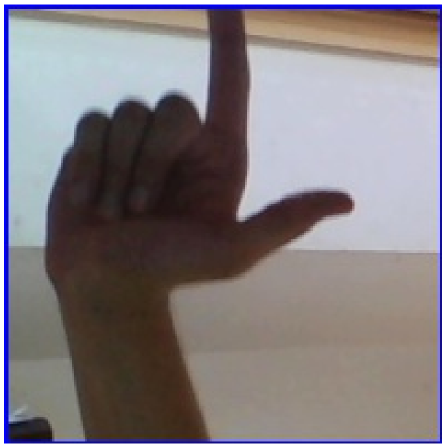
Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking. *arXiv preprint arXiv:2006.10214*, 2020.

Appendix

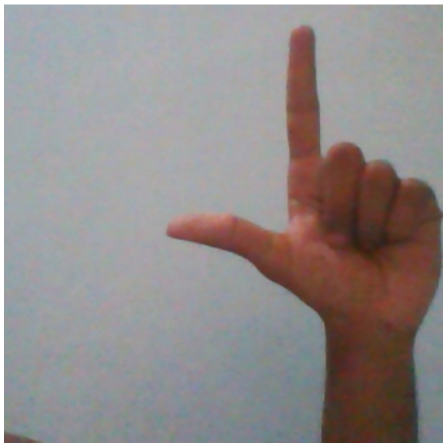


Model Type	Validiation Accuracy	Adversarial Validation Accuracy
Standard Neural Net	93.61%	72.50%
Convolutional Neural Net	96.82%	79.40%

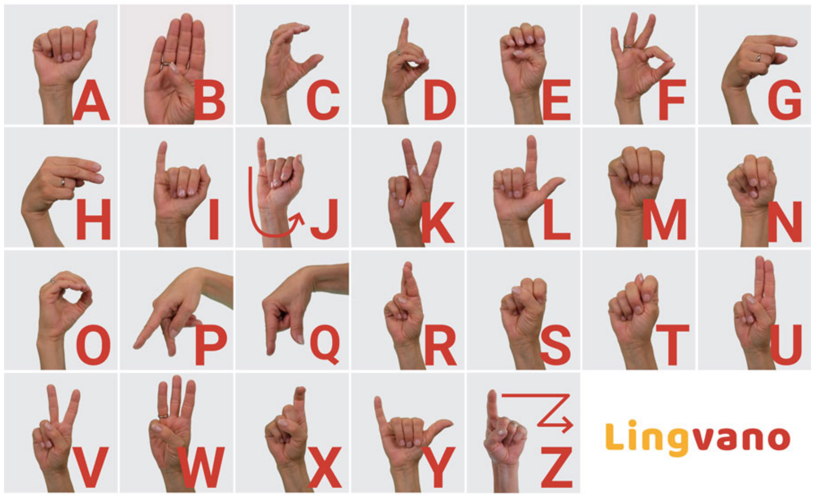
Comparison of Validation and Adversarial Validation
text Accuracy between Standard and Convolutional Neural Nets



L from the original dataset



L from the ablation dataset



Description of the ASL Alphabet