DS 5220 Project Report: Classifying ASL Signs using a Convolutional Neural Network

Abhinav Nippani **Larson Ost** **Adwait Patil**

1 Abstract

American Sign Language (ASL) is a visual language used by over 500,000 people in the United States and Canada, predominantly in deaf communities. For those without a reason to learn ASL, communication with their non-verbal peers proves to be a difficult language barrier. Now that video technology and video communication is increasingly more popular, we believe that we can help reduce this barrier by translating ASL speakers for their audience. This will be completed by capturing real-time video of ASL and processing the information with a convolutional neural network.

2 Introduction

Video communication technology is a commonplace tool for remote interaction. People who are hearing-impaired must rely on the video aspect to communicate, but if their audience is unable to understand ASL, they will be at an impasse. This project aims to use the video feed of remote video calling as an information-gathering source to resolve this communication problem by automatically translating ASL. This can also be applied to in-person communication because if the people in conversation do not have a sign language interpreter, they would be able to use this translation technology to communicate. This has proved useful in Google Translate, as speakers of different languages can use it in real-time to communicate.

This technology can be used for real-time communication on platforms such as ZOOM Technologies, Skype, FaceTime, social media platforms, etc. Also, this only requires a video feed, so this can be applied to create closed captioning for prerecorded videos and transcription from ASL to English. In terms of our project group's personal interest in this topic, none of us have ever had to learn or use ASL, so we find that it would be an interesting challenge to work with a new domain for this project.

The actual machine learning for this project will be completed by collecting video frames from a video at certain intervals of time, then passing the image's information to a convolutional neural network (CNN). In this project, we will experiment with several CNN architectures to explore the possible use cases of each one. This method was chosen because of the unparalleled ability for a convolutional neural network to process large amounts of image information. We will be requiring to reduce thousands of high-dimensional images with medium resolution and color for preprocessing. Other machine learning methods would be far too computationally expensive. Also, due to ASL being performed by people in a real-time video, we will need the results to be translation invariant, which CNNs can perform. Other models such as decision trees would perform very poorly at feature processing and will be computationally demanding. Also, during our research, we found no other methods performed by other project groups yielded good results.

The key components of this project will be in primarily creating the program required to collect the real-time video feed. This required a video feed with hand-recognition technology that would be able to send frames to the neural network in real time. Once this is in place, we are able to feed it to CNN. This project will consider multiple CNN architectures and several data augmentations in preprocessing to improve accuracy. We will primarily evaluate our models on the static test images given with the training data. We will then evaluate our models on the real-time video where the test images will be provided by ourselves.

The main limitations of this project come from computational power and real-time video logistics. The real-time video feed will only be able to process images every 1.5 seconds. This is fine for our proof of concept, but would need to be processed on a more powerful machine before commercial implementation. Also, the dataset we are using has images that are very similar to one another. The images are monochromatic and homogeneous in background, lighting, and hand placement. We will need to perform augmentation to mitigate this issue.

3 Preliminaries

3.1 Dataset description

The dataset for this project is provided by "ASL Alphabet" on Kaggle ([4]). The dataset is comprised of a training and testing set. The training set has 3000 images per category, and there are 29 total categories of ASL signs: every English letter, "space", "delete", and blank images. The testing set has one image of each category for a total of 29 images. The images themselves are 200 by 200 pixels. Below is an example of an image from each category ("GT" stands for ground truth, and is the true label of the image):

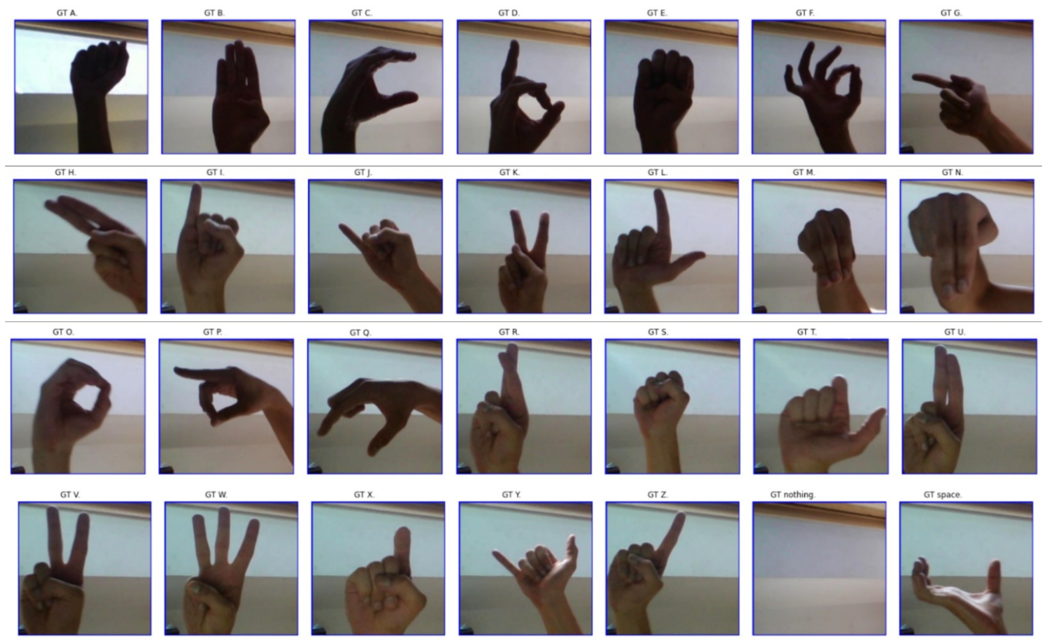


Figure 1: Example images from every category in the dataset

From a basic view of the images, it is obvious there are some limitations. The dataset appears to have been created by extracting frames from a single video. Consequentially, the images all appear monochromatic. They have the same lighting, the same background, the same ASL signer, and relatively similar hand positioning. This will mean that in order for real-time video to work, we will need to augment the data for training, so the model will be able to adapt to new settings, lightning, etc. Also, another difficulty comes from the similarity of ASL letters. Letters "K" and "V" are both represented by two fingers with differences in thumb positioning. "A", "E", and "S" are all similar to a fist with slight differences. "I", "J", and "K" and represented by the pinky being pointed outward. Below are two pairs of images: "K" compared to "V", and "A" compared to "S" to represent the similarities.

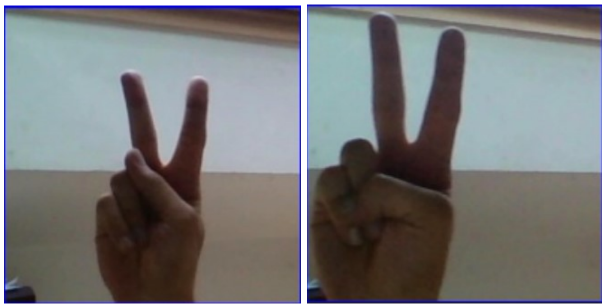


Figure 2: ASL for "K" (left) and "V" (right)

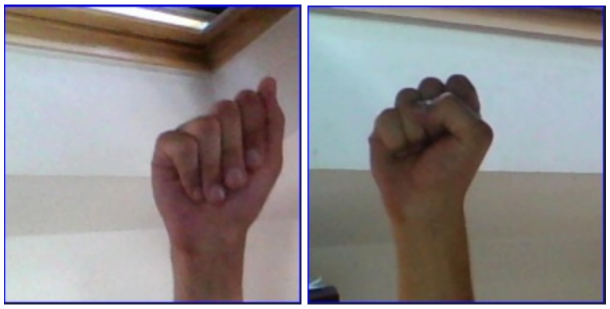


Figure 3: ASL for "A" (left) and "S" (right)

Also, letters "J" and "Z" use motion when being signed. We will have to consider for our purposes that these letters are represented with a non-moving hand. We will evaluate our models' performances on these images considering the limitations and perform appropriate augmentations.

3.2 Experimental setup

The ultimate goal of the project is to detect ASL letters in a real-time video. This would first require the ability to detect hand positioning in a video. Using a python library named Mediatape, we were able to create a framework for detecting hands using two important functions: mphandsḢands() and mp.draw_utils(). The mphandsḢands() function used to detect the user's hand positioning requires a level of acceptable certainty that it has correctly located the hand. The program will return a range of X and Y coordinates of which it calculated the hand will be contained within. We will take the minimum and maximum of the X and Y coordinates so that we are completely certain the hand lies within those coordinates. Finally, we will draw a box around the hand's expected position in the video. The mp.draw_utils() function draws the specific contours of the hand overlaid on the video. This includes the finger and joint positioning of the user. This does not provide any additional benefit for our calculations, but it is helpful to the user to visually see how their hand is being detected by the software.

The second library that assisted us in creating the hand detector was OpenCV. The OpenCV library provides a function to capture frames from a live video which can be converted into an array format and subsequently read as an image by the matplotlib library's imread() function. The final result would be passed to our neural network for processing. Additionally, it uses coordinates provided by the aforementioned hand tracking module to draw a box around the detected hand which is eventually cropped and fed as an image to our model in order to predict which ASL letter is been shown by the speaker.

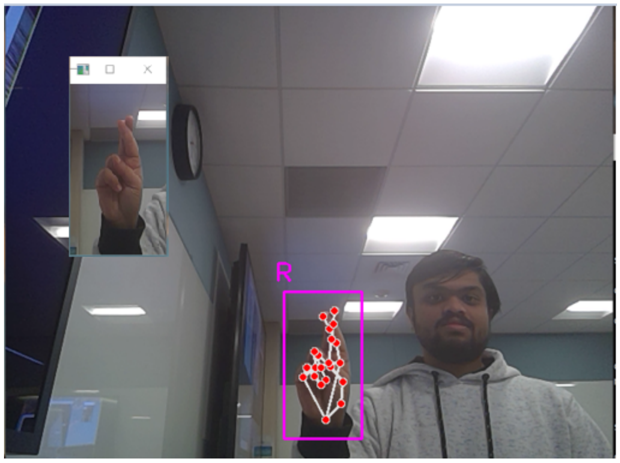


Figure 4: Real-time hand detection

3.3 Problem setup and neural networks' architectures

For this project we will experiment with three separate models: A CNN created from scratch with the TensorFlow library, the ResNet model, and the Inception-ResNet model.

3.3.1 Model from scratch

As discussed in the introduction, neural networks have proven to be the default choice for image classification or detection. CNNs have especially been the best choice to tackle image data and we chose that to be the starting point of our base model. This initial model consists of 3 2-Dimensional (2D) convolutional layers, where each layer is followed by a maximum pooling 2D layer. The last two convolutional layers are followed by two Dropout layers. Once the convolutional layers have extracted features from the images, we pass them through a flatten layer to convert them into a 1D array that can be processed by fully connected layers. The model uses two dense layers: one with 128 neurons accompanied by the ReLU activation function and the final output dense layer with 29 neurons (representing the number of classes). The model concludes with a softmax activation function to read the output as a set of probabilities.

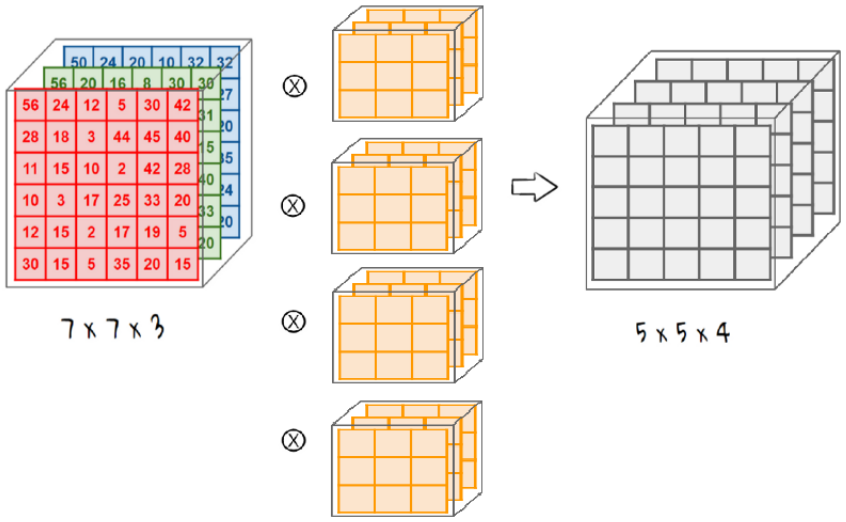


Figure 5: Convolution on a 2D image with 3 channels
([1])

The convolutional layers use a (4x4) Kernel. The kernel size was determined by cross-validation. The first two convolutional layers use 64 filters, so each input image that is received is convoluted with a 4x4x64 filter resulting in an output image with 64 channels. This result is later max pooled by a (2x2) max pool layer to extract the most important features from the images. This is then followed by a dropout layer which randomly drops out 40% of the neurons, which is utilized to to prevent over-fitting. The last convolutional layer uses 128 filters which extract even more contextual features that the previous filters were unable to find. Similarly, this output will undergo max pool using a (2x2) max pool layer and pass them to the dense layers, dropout layer, and a Flatten layer.

3.3.2 Transfer Learning

Transfer learning involves initializing weights from a pre-trained model instead of random initialization. Using the machine learning approach of Transfer Learning, a previously trained model is used for another task which enables quick progress or enhanced performance.

For instance, a model that has been trained on a sizable dataset of bird photos would include learned characteristics like edges or horizontal lines that are transferable to your dataset. We may benefit from pre-trained models in various ways such as saving time by utilizing a model that has already been trained.

There are two major methods used for transfer learning:

- Fine tuning: The pretrained weights and biases are used as the starting point to retrain the entire network on the new dataset. This approach is useful when the new dataset is different from the dataset used to train the original neural network, as it allows the network to learn new features.
- Feature extraction: This consists of starting from a trained network, then updating only the weights of the output layer. This approach is useful when the new dataset is similar to the dataset used to train the original neural network, as the network will already have learned some of the features of the new dataset.

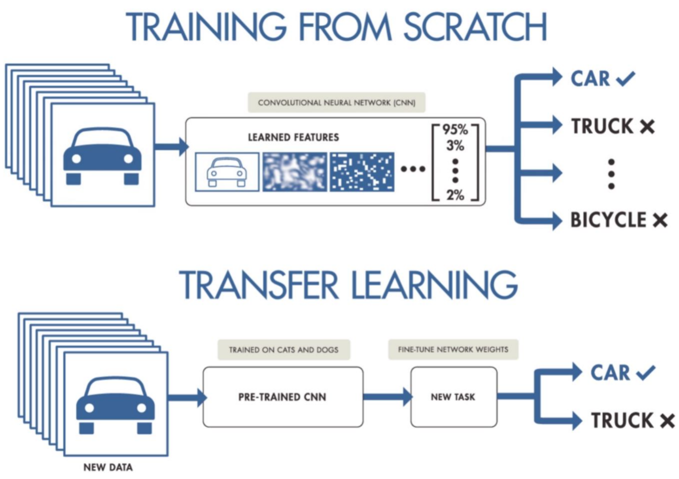


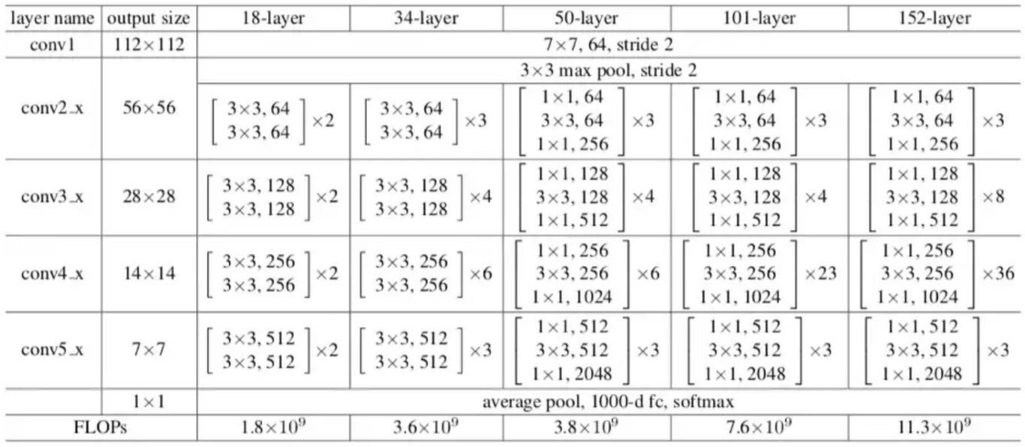
Figure 6: Transfer Learning Architecture
([2])

For this dataset, a pretrained Resnet34 is used as the pre-trained model for training. The pretrained model is frozen and further linear and dropout layers are added to the model. Adam optimizer and cross-entropy loss are used for training, the batch size of training data was set to 128 and a learning rate of 0.001 is used.

3.3.3 ResNet model

One of the bottlenecks of deep convolutional neural networks was that they couldn't go deeper as they would start to lose their generalization capability. This is due to the fact that when the network is excessively deep, the gradients used to compute the loss function simply drop to zero after a number of chain rule applications. As a result, there is no learning taking place because the weights' values are never updated. This problem is termed vanishing gradients. The problem of vanishing gradients arises when layers using certain activation functions are added to neural networks, and the gradients of the loss function approach zero, making the network hard to train. Certain non-linear activation functions often are insignificant to input changes because they map everything into a range of 0 to 1, which will result in the derivative being very small. This does not arise in shallow neural networks but only in deep neural network, causing the gradient to be too small to work effectively and update parameters.

ResNets (Residual Networks) solves this problem with the use of skip layers. With ResNets, gradients from initial layers can flow straight through skip connections to layers further in the network. Depending on how many layers the model contains, ResNets can range in size. The most popular ResNets are ResNet18, ResNet34, ResNet50, ResNet101 and ResNet152. The same pattern is followed by each layer in a ResNet: Every two convolutions they skip the input and conduct a 3x3 convolution with a fixed feature map dimension (F) of [64, 128, 256, 512] correspondingly. Furthermore, across the whole layer, the measurements of width (W) and height (H) stay constant.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 7: Sizes of outputs, convolutional kernels for various ResNet Models
([3])

After trying various ResNet models, ResNet34 is selected for the purpose of this project. Resnet34 is an image classification model that consists of a 34-layer convolutional neural network. This model has already been pre-trained using the 100,000+ images across 200 distinct classes that make up the ImageNet dataset. It differs from traditional neural networks in that it utilizes the residuals from each layer in the linked layers that come after it.

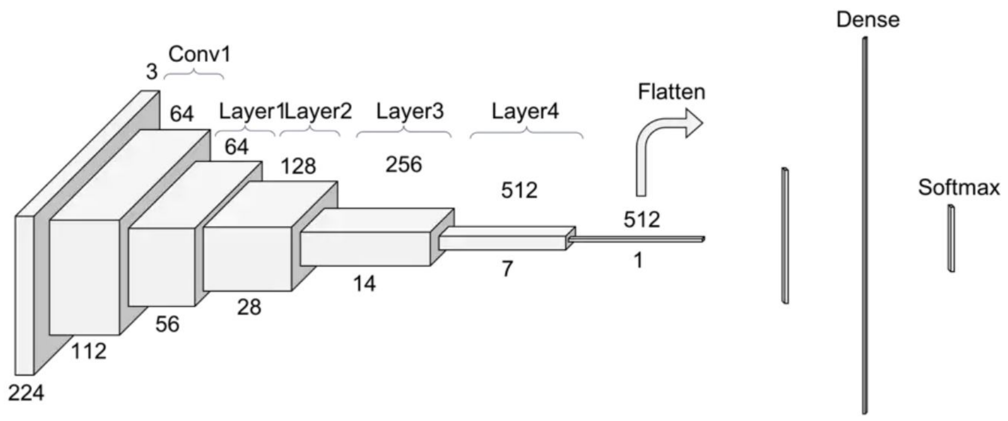


Figure 8: ResNet34 Architecture
([3])

3.3.4 Inception-ResNet model

Our final model, which would be applied when moving to real-time video, is the Inception-ResNet-V2 transfer learning model provided by Keras library ([8]). In summary, the model was trained on over 1 million images from ImageNet (an image collection database for research) and has 164 layers. It utilizes the concept of residual layers as described in the previous section, and this model additionally uses inception layers. A single inception module will include the following:

- Input layer
- 1x1 convolution
- 3x3 convolution
- 5x5 convolution
- max pooling
- concatenation

Inception layers aim to prevent overfitting and lower computational demand. After the input layer, the information will be processed by a 1x1 convolution. This convolution, while not enhancing the learning of the model individually, will lower the number of input channels for the next layer, and reduce computational demand. This will allow the model to learn more overall.

The 3x3 and 5x5 convolutional layers provide learning for the model on several different scales. Also, instead of running in series, these layers will run in parallel, meaning several kernels will be performed on the same image. Max pooling will be performed on each kernel before the information is concatenated in preparation for the next layer. The following image demonstrates the "stem" of the network, where we can observe the 1x1, 3x3, and 5x5 convolutional layers.

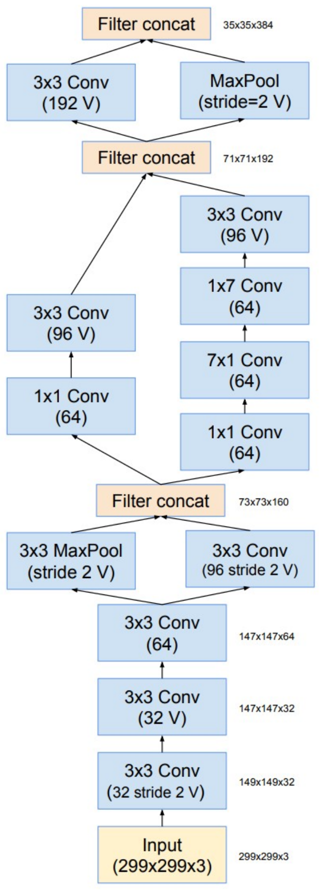


Figure 9: The schema for the "stem" of the CNN

Running the models convolutional layers in parallel and utilizing the 1x1 convolutional layer technique will greatly reduce the overfitting of the model without compromising computational demand, especially while integrating residual layers introduced in the previous section for ResNet. The full schema of the model is shown in the figure below.

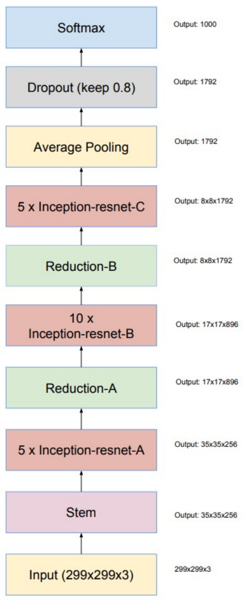


Figure 10: Full schema of Inception-ResNet-V2

4 Results

4.1 Main results

4.1.1 Scratch model results on the still images

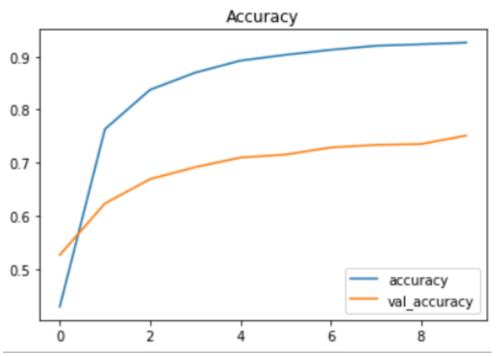


Figure 11: Accuracy on train and validation sets

The base model gave a really promising train accuracy of 93% but it clearly overfitted which can be derived from its performance on the validation set where it gave only 75% accuracy. These results can be attributed to the little to no variation in the input images, each having the same background and some letters having similar orientations as discussed in the dataset limitations. Also, these results suggest this model's archisture is not sufficient for this type of problem. We will be moving to transfer learning models for further improvement.

4.1.2 ResNet's performance on the still images

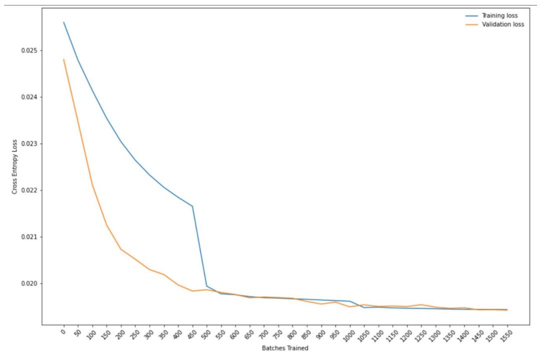
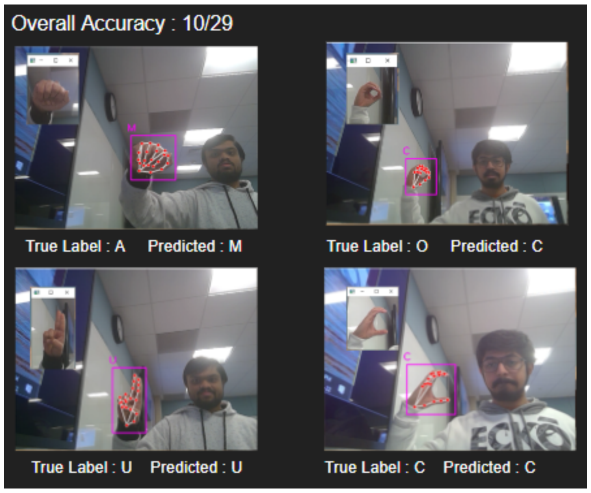


Figure 12: Train and validation loss across batches trained

Resnet34 gave a really promising result on the training set as well as the validation set. It gave an accuracy of 98% on the validation set. Furthermore, we achieved an accuracy of 100% on the 29 test images provided by the ASL dataset which made us confident about using resnet on real-time detection.

4.1.3 ResNet's performance on the real-time video

Even after achieving a near-perfect result on the test set using ResNet, it was unable to reproduce those results for real-time detection. It correctly detected only 10 out of the 29 hand symbols. These results further varied depending on the lighting condition or the background of the video. From experimentation, we concluded that ResNet was learning the background along with the hand gestures which can be attributed to the limited variation in the images as well as the skip connection between the initial and final layers. We would once again be testing an additional learning model to search for improved accuracy on the real-time video.



Overall Accuracy : 10/29
True Label : A Predicted : M
True Label : O Predicted : C
True Label : U Predicted : U
True Label : C Predicted : C

Figure 13: Examples of resnet's performance on real-time detection

4.1.4 Inception-ResNet's performance on the real-time video

For real-time video testing, we would need to work with the Inception-ResNet-v2 model, due to ResNet's imperfect performance in the previous section. Similar to the previous section, we will evaluate the model's performance by performing the 29 possible signs to the camera and recording its results.

Inception-ResNet was able to perfectly classify all tested images for a 100% testing accuracy (29/29 predicted correctly), as exemplified from a subset of the models predictions shown below.

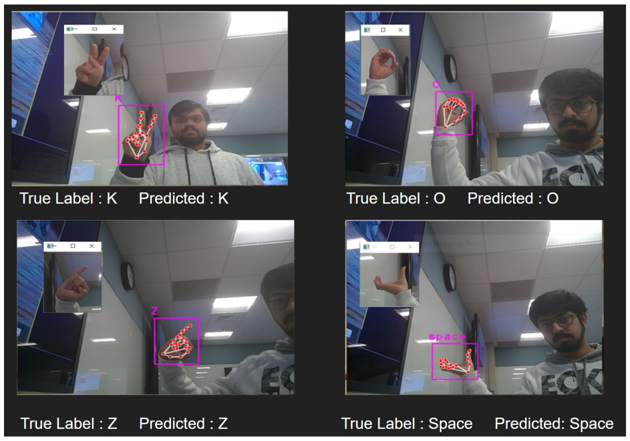


Figure 14: Examples of the correct classifications during real-time testing

4.2 Supplementary results

4.2.1 Effects of data augmentation

To improve the accuracy of our models, we considered several data augmentation techniques for preprocessing. These data augmentations would need to be analogous to real life scenarios that would cause input data to match the augmentation. For example, if the user had a blurry camera we could perform blurring to the input data. Here are the following augmentations considered with their real-life analogous situation:

- Greyscale images: converting the color images into black and white images. This would be realistic if the user did not have a color camera or were trying to reduce the file size of their images and video.
- Low resolution: reducing the image sizes. This would be typical for a user with a low resolution camera.
- Blurring the images. Blur would be useful for users who have blurry cameras or an obstructed lens.
- Off center: changing the location of the hands placement in the image. This would be useful if the hand detection software varies in its exact placement of the hands location, or if the user cannot steady their hand.
- Increasing and decreasing light. This would assist users who are in brighter or darker settings when using ASL.
- Flipping the image horizontally. This would be useful for applications such as ZOOM that allows the user to flip their camera axis perception for their audience.

Below is a visual representation of all considered augmentations:

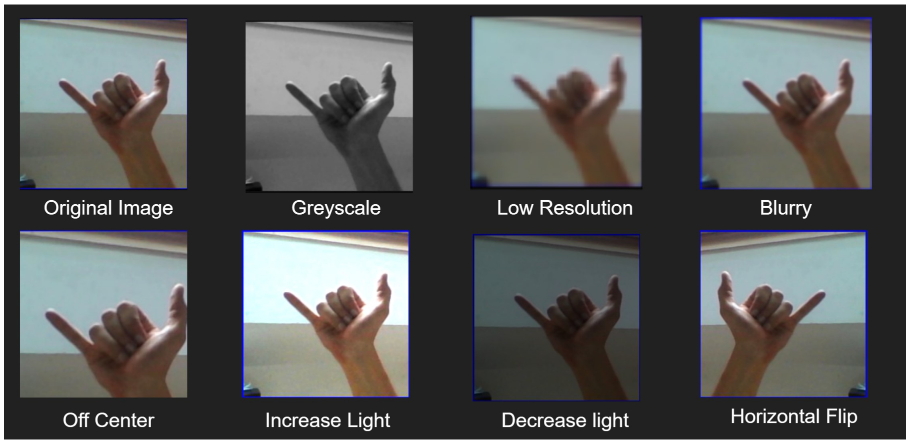


Figure 15: Examples of all considered data augmentations

Due to the limitations discussed in section 3.1, we found that any image that alters the positioning would lead to the sign becoming another symbol entirely, due to their similarities. During testing, we found that off-center and horizontal flip augmentations would not increase the accuracy of the model. Also, due to letter similarity, any reduction in the information being sent to the model would not increase the accuracy. This includes blur, lowering the resolution, and converting the image to greyscale.

The remaining augmentations under consideration were increasing and decreasing lighting. Through testing both, we found that increasing the contrast by 1.5x on the original image increased our overall accuracy from 96.55% to 100% on the real-time data. This means the only augmentation that would increase the performance of the model is to increase lighting. Therefore for our final round of model training, we would increase the dataset by including a duplicate of every image with increased lighting. This result is most likely due to our training data apparently being conducted under direct lighting.

5 Discussion

5.1 Comparison to related work

From the accuracy on both the provided test dataset from Kaggle and the accuracy on our real-time video testing, we consider our models extremely successful at classifying both still images of ASL letters and classifying our own real time demonstrations.

Satisfied with our results, we conducted research on similar projects to compare our results with theirs. Browsing the Kaggle notebooks associated with this project shows that everyone with very good results and high accuracy have also used transfer learning models. For example, a user reported over 99% accuracy when using the 'resnet26d' model ([5]). Another user who used the same model as ours 'InceptionResNetV2' reported 95% accuracy ([6]). Our higher accuracy is most likely due to our data augmentation and improved model parameters. Based on similar projects, we believe we have achieved either superior or similarly near-perfect results as other projects on the testing data from Kaggle.

We also wanted to assess the power of our real-time testing by comparing the results to others projects. We would not be able to look to Kaggle for comparisons because the real-time video was not a subject on those pages. There have been many notable projects using gloves with sensors or even multi-angle video feeds to gather data. While these projects usually yielded extremely good results, we will not consider them for comparison as they use different information-gathering techniques. There is a project from students at Stanford University using transfer learning models to capture real-time video ASL letters. They were able to "attain a validation accuracy of nearly 98% with five letters and 74% with ten" ([7]). Once again, we were able to outperform their model.

Concluding our attitude toward our accuracy relative to other projects, we believe we have a novel preliminary solution to creating real-time video translation for ASL speakers to a non-ASL speaking audience, as it has very good accuracy relative to comparative projects of this level.

5.2 Future work

This project leaves room for further development. One substantial area of improvement could come from processing the neural network on a more computationally robust system. We are currently limited to processing an image from the video feed every 1.5 seconds. With improved speed, we could provide a more seamless translation of ASL letters to the audience.

Also, this model could be paired with a language API to provide real-time translation as well, instead of being limited to purely English letters. Similarly, this could be paired with a transcription program to be able to convert recorded or real-time videos to written words.

6 Conclusion

In this project, we have created several convolutional neural networks to classify American Sign Language letters. The networks were evaluated on both still images given to us with the original dataset, and also a real-time video feed using our personal images as test images. We have been able to have 100% accuracy on our real-time testing set, and achieved 96.34% accuracy on our still images. We were able to optimize our results by utilizing transfer learning concepts, data augmentation, and varying hyperparameters. Our model performs quite well against comparative models, and we have suggested improvements that could be completed for future work.

References

[1] J. Jeong, "The most intuitive and easiest guide for CNN," Medium, 17-Jul-2019. [Online]. Available: https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480. [Accessed: 15-Dec-2022].

[2] "Transfer Learning," Google image result for https://miro.medium.com/max/3200/0*azmg0aura-oru2GB. [Online]. Available: https://images.app.goo.gl/CMXHnJS1ZKY3kSvB9. [Accessed: 15-Dec-2022].

[3] P. Ruiz, "Understanding and visualizing ResNets," Medium, 23-Apr-2019. [Online]. Available: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8. [Accessed: 15-Dec-2022].

[4] Akash. "Asl Alphabet." Kaggle, 22 Apr. 2018, https://www.kaggle.com/datasets/grassknoted/asl-alphabet.

[5] Abrorshopulatov. "ASL Baseline FASTAI Model with &lt;1% Error Rate." Kaggle, Kaggle, 24 Nov. 2022, https://www.kaggle.com/code/abrorshopulatov/asl-baseline-fastai-model-with-1-error-rate.

[6] Tuhinkumardutta. "Asl Alphabet: INCEPTIONRESNETV2: 95% Accuracy." Kaggle, Kaggle, 22 Sept. 2021, https://www.kaggle.com/code/tuhinkumardutta/asl-alphabet-inceptionresnetv2-95-accuracy.

[7] Garcia, Brandon, and Sigberto Alarcon Viesca. "Real-Time American Sign Language Recognition with Convolutional Neural Networks." Stanford.edu, http://cs231n.stanford.edu/reports/2016/pdfs/214_Report.pdf.

[8] Team, Keras. "Keras Documentation: INCEPTIONRESNETV2." Keras, https://keras.io/api/applications/inceptionresnetv2/.