Real-time American Sign-Language Recognition via Transfer Learning

Maria F. Harris
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, United States
mariaharris44@gmail.com

Roumen Guha
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, United States
roumen.guha@gmail.com

***Abstract*—The development of a system that can translate sign language alphabets on the fly is an important tool which can significantly reduce the communication gap between the deaf and those who are not well-versed with sign language. This paper presents a particular solution to the problem of real-time sign language translation. We utilize transfer learning techniques with pre-trained ResNet50 and VGG16 networks trained on the ImageNet dataset, and retrain the classification layers of both the models. We used a combination of different ASL datasets in order to bring more variety. We use the weights generated from this training to perform real-time classification of sign gestures using the user's webcam. Overall, the project has achieved favorable results - our detector has been able to consistently identify letters a, b, c, e, l, o, q, r, u, v, w and y in real-time. These promising results have prompted us to believe that producing a robust real-time sign language translator is very much achievable with a larger dataset and more experimentation with the model architecture.**

***Index Terms*—CNN, image processing, multiclass classification, linguistics, American sign language**

I. Introduction

At the time of writing, American sign language (ASL) has approximately half a million users in North America [1]. In a continent home to approximately 580 million people, the speakers of sign language are few and far between, and when communication becomes necessary between an ASL user and a non-sign language user, written communication becomes the only alternative left, and it is often cumbersome and time-consuming. This makes it especially bad for emergency situations, like it may when dealing with emergency responders.

The aim of this project is to present a working ASL alphabet recognition system, primarily as a proof of concept. This system is based only on individual frames, and so struggles with alphabets that require motion or gestures to convey. It also cannot classify words; that would be a much larger project. So presently, this is best described as a toy problem.

Our project thus consists of the following tasks:

1) Obtain frame of a user signing a letter
2) Classifying each frame as a letter in the English alphabet

However, as anyone familiar with classification on visual features knows, there are several issues with this approach:

Special thanks to Zhuwen Tu and Kwonjoon Lee for their support through a difficult time.

- Environmental concerns (including but not limited to: dynamic range of image sensor, camera pose relative to user, inadequate contrast, partial or complete occlusion)
- Classification of invalid signs
- Identifying when one sign gesture ends and the other begins.

This is not a new problem. In the next section, we will discuss previous attempts to solve the problem of sign-language detection in real-time.

A. Prior Work

Atwood et al., in [2], performed low-level image processing before attempting to use both principal component analysis (PCA) and feed-forward neural networks to classify hands. They briefly discuss the importance of finding a balance in the trade-off between pre-processing the images and ensuring they have enough distinctiveness for the machine learning methods to be able to perform well. While their neural network model was able to achieve training accuracies of 100% for every dataset, and the validation accuracies were in the close to perfect, their test accuracies were only around 50%. However, their dataset was composed of just 480 images, so the models likely were not able to generalize well to live-samples. Overall, we found their results encouraging, and you will find that convolutional neural networks (CNNs) are much better suited to the task of observing visual features.

One study we read attempted to use k-Nearest Neighbors (with $k = 3$ to classify ASL letters [3]. Though Aryanie et al. use a Microsoft Kinect camera, they discard all depth information, leaving only color images to be used for classification. Their dataset consisted of about 500 images for letters A-F, and they were able to achieve wildly varying accuracies on unseen images, ranging from 83% accuracy for B to 35% for F. They explain this as the effect of using PCA for reducing the feature dimensions, resulting in letters like B and F having highly correlated (i.e. redundant) features. Thus they suggest against using PCA for pre-processing, while also pointing out that lighting affected recognition accuracy. Finally, they note that the recognition time of their classifier is not ideal for a real-time solution, and propose the investigation of other machine learning methods.

In [4], Sun et al. also use a Microsoft Kinect camera and perform classification, both with and without depth information, with a latent support vector machine (SVM) formulation. They were able to obtain accuracies of 82% with their model, using HOG features only. With depth features included, this accuracy rose to 86%, which perhaps points to a future study with depth maps.

Our project is not the first to apply convolutional neural networks to real-time ASL letter recognition without motion data. For example, in [5], Garcia et al. utilized a pretrained GoogLeNet architecture trained on the ILSVRC2012 dataset. Their work is of some interest to us, and we found that some of our conclusions agree with theirs. We observed from our findings that models which have lower training and validation accuracy resulted in better real-time predictions. In their recommendations for future work, Garcia et al. suggest the exploration of different network architectures, such as a VGG or ResNet architecture, for comparison to their implementation. Their suggestion motivated this project; implementing both ResNet50 and VGG16 networks for transfer learning.

Further, in [6], Gao et al. use a two-stream CNN framework. In their implementation, the first stream operates on the RGB images to extract useful features for recognition, and the second stream operates on the depth image. The outputs of the two streams are then fused to achieve the final predictions. Unfortunately, since their work did not show results from just the image-stream, we cannot directly compare their results with ours, but since their results are promising, it is worth further study. We also note that their accuracies are quoted from a test set, which we find to be essentially unrelated to real-time performance. We shall discuss this further in the Sec. III-B.

II. Methodology

As mentioned previously, this project used transfer learning on two of the most popular image classification networks: ResNet50 and VGG16. Our approach consisted of combining two different datasets in an effort to improve the diversity in our training images. Once we had trained both our models, we then proceeded to put our models to the test by making them classify hand gestures in real-time using a computer's camera.

We will be explaining the aforementioned models and our approach in detail in this section.

A. Convolution Neural Networks and Transfer Learning

Over the years we have seen the huge success of Convolution Neural Networks in successfully handling problems related to images. The success of CNN can be attributed to the fact that the convolution layers act as automatic feature extractors for the images passed through them and also the local connections of a convolution layers are spatially local and governed by the kernel size, this leads to dimensionality reduction. Therefore, CNN's are particularly well-suited for problems involving images.

In general, it is very uncommon to train an entire convolution neural network because of the lack of a sufficiently large dataset/the very long training time involved/lack of sufficient computation hardware. Therefore, a pre-trained CNN model that was trained on a sufficiently large dataset (ImageNet) can be used as a feature extractor and we can retrain the classification layers with the new dataset. This approach is known as 'transfer learning' and is a fairly popular technique in making classifiers that can classify images other than those on which it was trained on.

For this project, we are using two convNets: ResNet50 and VGG16 - freezing the weights of the convolution layers and retraining the fully-connected (fc) classification layers. We carried out different experiments by using different number of fc layers during the training process. However, the classification layers had a similar pattern: The output fc layer used 'softmax' as the activation function since softmax (equation 1) gives out probabilities associated with each predicted letter and this allows us to select the predicted alphabet with largest probability.

$$\sigma(x_i) = \frac{\exp x_i}{\Sigma_j \exp x_j} \tag{1}$$

Furthermore, the NLL loss (equation 2) was used as the loss function during the training process

$$l(x, y) = \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n}} l_n \tag{2}$$

We had also considered using another transfer learning technique, 'Fine-Tuning', in which the the weights of the early convolution layers are also retrained; however, since our dataset wasn't large enough compared to ImageNet, therefore, training the early convolution layers of the network might lead to overfitting. Considering the early layers of the convolution neural network extracts rather generic features of any image, hence, we decided to only train the classification layer.

B. ResNet50

Residual Networks (ResNet) is a fairly commonly used neural network architecture for a number of computer vision problems. ResNet rose to prominence as the winner of the 2015 ImageNet challenge. Prior to ResNet, it was very difficult to train very deep neural networks due to the problem of vanishing gradients. However, ResNet resolved this issue by enabling the users to train a 150 plus layers successfully with the help of identity connections between the layers. At its heart, ResNet is essentially a Convolutional Neural Network and is similar to standard CNN architectures. However, the addition of an identity connection between the layers is what makes ResNet a residual network.

In this project we utilized ResNet50, a smaller of version of ResNet152, due to training time concerns. It has the same basic architecture but has reduced number of layers. Moreover, ResNet50 is frequently used for transfer learning. Therefore, it seemed as a viable option to opt for the smaller variant. Furthermore, ResNet50 isn't the only variant, there are many others: ResNet-18, ResNet-34 etc.

The network architecture of ResNet 50 is indicated in Figure 1.

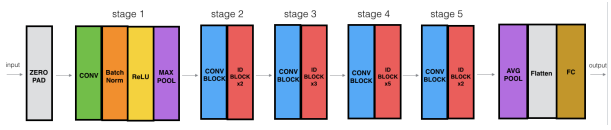

Fig. 1: ResNet50 Architecture

The ResNet50 architecture, shown in Figure 1, consists of 5 blocks, with each block divided into 2 sub-blocks: convolution and identity blocks. Each convolution sub-block contains 3 convolution layers and each identity block also contains 3 convolution layers. The classification layer consists of a single fully connected layer and this is the layer we retrained on our dataset and experimented with using transfer learning.

In total, ResNet50 has more than 20 million trainable parameters.

C. VGG16

Similar to ResNet50, VGG16 is also a convolutional neural network. VGG16 won the 2014 ImageNet competiton and has been extensively used for tackling a variety of computer vision based problems, and is a very popular architecture for transfer learning. The most distinctive aspect of VGG16 is that it replaces the large sized kernel filters of the convolution layers present in ImageNet 2012's winner: AlexNet with multiple 3 by 3 kernel-sized filters. It took several weeks for training VGG16 on the ImageNet dataset.

An overview of VGG16's architecture is presented in Figure 2.

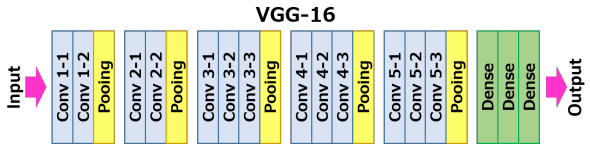

Fig. 2: VGG16 Architecture

As can be seen in Figure 2, VGG16 consists of 5 convolution layers followed by a classification block which consists of 3 Fully-Connected layers. We retrained these 3-FC layers in different experiments using transfer learning.

D. Data Description

We used two datasets, and combined them to suit our needs.

The first dataset [7] we considered contained 70 RGB images per alphabet, with the backgrounds segmented out. Each image's size was fixed with dimensions 400×400. The dataset contains signs from five (5) different hands, and each hand has its photo taken from different perspectives. However, the lighting conditions across the samples are very consistent. This is a very refined dataset, but unfortunately it does not contain many samples, and its consistency in light exposure perhaps does not generalize well to real-life situations. You can see examples of these images below. We also used data augmentation techniques like horizontally flipping the images to cater to both left and right-handed sign gestures, and introducing a small rotation (15 degrees in both directions) in an attempt to create an invariance to small rotations.

Fig. 3: Examples from Dataset 1. Hands 4 (left) and 5 (right) both signing 'A' from different relative poses.

Since our first dataset did not have a large number of images, we also trained with the second dataset [8]. This dataset contains 3000 RGB images per alphabet, across different lighting conditions, backgrounds, and camera pose relative to the sign being tested. These images have dimensions 200×200. Therefore this dataset could be considered exhaustive, except for the fact that there was only one subject and so there is little variation in the shape or complexion of the hands. You can see examples of these images below.

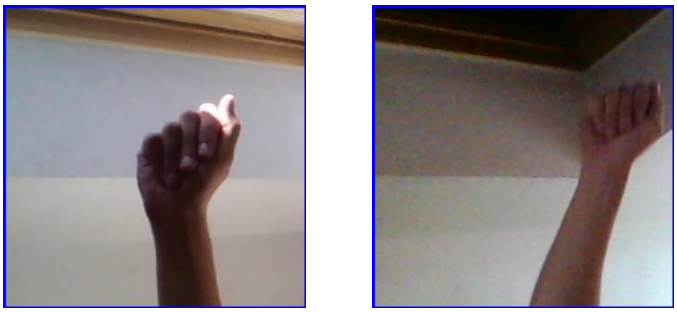

Fig. 4: Examples from Dataset 2. Same hand signing 'A' from different relative poses under different lighting conditions: inconsistent lighting across hand (left) and poor background contrast (right).

We will discuss how we combined these datasets to suit our needs for reasonable training times but without sacrificing accuracy too much in Sec. III-A2.

E. Real-Time Translation

Once we had trained our models, the next step involved creating the real-time recognition application. For this setup, the project member's laptop webcam was used for testing the performance of the trained model in real-time. The results of each classifier's performance in real-time will be discussed in Sec. III. There is a reference implementation in [9], but we developed our own independently. This can be used for comparison.

III. Experiments and Results

A. Training Experiments and Results

Since we are implementing both ResNet50 and VGG16, we attempted to keep as many hyperparameters consistent as possible so that the only difference in performance would be due to the model architecture itself. We go deeper in depth about the hyperparameters in Sec. III-A1.

Experiments were performed on UCSD's Datahub service, with the following hardware:

- Quad-core CPU
- 16GB RAM
- NVIDIA GTX 1080 TI (with 11GB of VRAM)

and using the PyTorch and Torchvision libraries for the training, and OpenCV for the real-time webcam interface.

In the table below, we present a brief overview of the attempted experiments. We will discuss these in more depth in the following subsections.

Dataset	Network	#Epochs	#Linear	#Dropout	Acc (%)	Loss
A-E	ResNet50	30	2	1	96.9	0.04
A-E	VGG16	30	2	1	96.5	0.06
A-E	ResNet50	16	3	2	96.81	0.07
A-E	VGG16	27	3	2	93.1	0.08
A-Z	ResNet50	-	-	-	-	-
A-Z	VGG16	30	2	1	90.9	0.32
A-Z	ResNet50	-	-	-	-	-
A-Z	VGG16	30	3	2	91.2	0.33

TABLE I: Where Accuracy and Loss are quoted as the best accuracy and loss on the validation set over all epochs. Two rows are left blank because the performance of the same models on smaller datasets is worse in real-time application, hence it was not worth the effort to investigate these models on larger datasets.

1) Hyperparameters: On this hardware, we found that a batch-size of 170 would work consistently for both networks and would be robust to minor changes in the networks (as we sought to see which hyperparameters result in the greatest changes to accuracy and learning). Thus, we used a batch-size of 170 for all of our experiments.

We also used Adam for the optimizer because, as [10] mentions, Adam is a more efficient variant of gradient descent, and does not require hand-tuning the learning rate. If we had not done this, an alternative would have been to mimic [11] by using Nesterov's Accelerated Gradient Descent (NAG) with a fixed momentum of 0.9, and initialize the learning rate at 0.003 and decrease it by 5% every epoch.

We trained each network for 30 epochs, however we also implemented early stopping: if the accuracy on the validation set did not increase after five (5) epochs, we terminate training.

2) Training Process: We followed a train, validation, test set ratio of: 60%, 20%, 20%.

However, early trials proved difficult: training on 26 classes of images, with over 3070 samples per image (i.e. the two datasets combined with no downsampling) leads to a dataset of about eighty thousand (80,000) images, calculations showed that it would take 45 hours for training to complete. This is likely due to the fact that a lot of data-streaming between GPU and memory is taking place.

So, instead, we began this study by investigating which network architecture performed best for a 5-category classification task. Namely, we attempted transfer learning for both ResNet50 and VGG16 with only classes A-E with each class containing 3070 samples, split according to the train, validation, test set split ratio stated earlier. Henceforth, we will refer to this A-E experiment as the 5-classifier. We would then train the model that performed best at this stage on an A-Z dataset with 26-classes (call this the 26-classifier), where we could afford to invest more time into training while having more confidence in results derived from them.

However, because the training time for 80,000 images was still too large, we combined the two datasets described in Sec. II-D to form the dataset for the 26-classifier. The procedure for forming this combined dataset is as follows:

1) Downsample dataset 2 by taking extracting every third image for training. Therefore this corresponds to a **downsampling factor of 3**. This results in every letter A-Z having 1000 samples. This was acceptable because there was a lot of redundancy in the dataset 2 for particular lighting conditions, pose of the sign being displayed, relative camera pose, background contrast, etc. Another reason why this should not affect the results too much is that we utilize PyTorch's Image Transforms to augment our training set.
2) Combine downsampled dataset 2 with dataset 1, so that we have 1070 samples per category.
3) Split this combined dataset according to the ratio we specified earlier (train, validation, test set ratio of: 60%, 20%, 20%).

The 5-classifier networks took less than an hour to train for both networks, whereas the 26-classifier networks took less than 7 hours to train on the combined dataset (with downsampling).

From the following training curves, loss curves, and confusion matrices, and from Table I, we can see that ResNet50's accuracy curves for training and validation loss tended to converge much faster, and to higher values than VGG16's accuracy curves. However, as we will discuss later in Sec. III-B, we will note that VGG16 performed better during live testing than ResNet50, despite having higher losses and lower accuracies during training.

3) A-E Recognition with 2-FC Retrained Layers: This experiment involved two fully-connected and one dropout layer for both models, with probability 0.2. This experiment operated on the dataset composed of 3070 images per letter, for letters A-E. As we can see from the comparisons, VGG16 seems to converge to a higher training loss and lower accuracy than ResNet50. This fact is also evident in the confusion matrices, with VGG16 misclassifying slightly more than ResNet50. The results of this experiment can be seen in the figures on the next page.

In attempting the real-time recognition of ASL signs via the weights obtained from this model, we notice that VGG16 per-

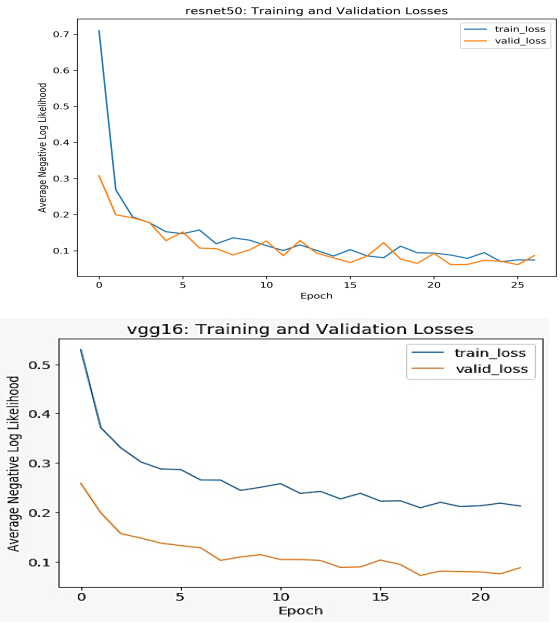

Fig. 5: A-E, 2-FC: Comparing loss curves between ResNet50 and VGG16.

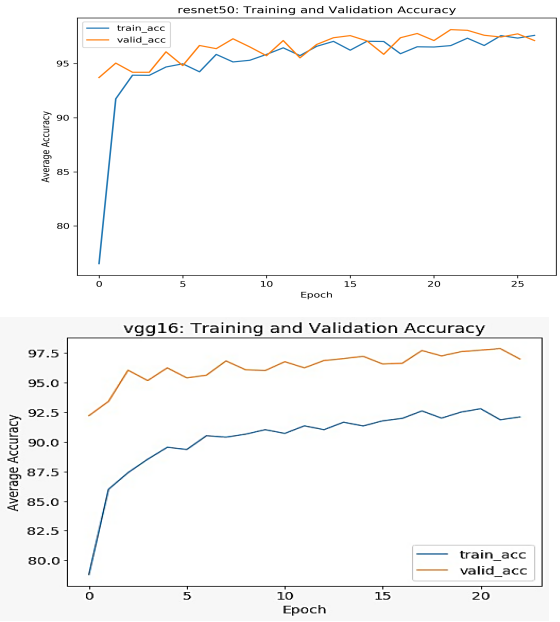

Fig. 6: A-E, 2-FC: Comparing accuracy curves between ResNet50 and VGG16.

forms better. Thus, we proceed by only training VGG16 (with 2-FC Retrained Layers) for the A-Z dataset. We discuss the real-time results corresponding to this experiment in greater detail in Sec. III-B.

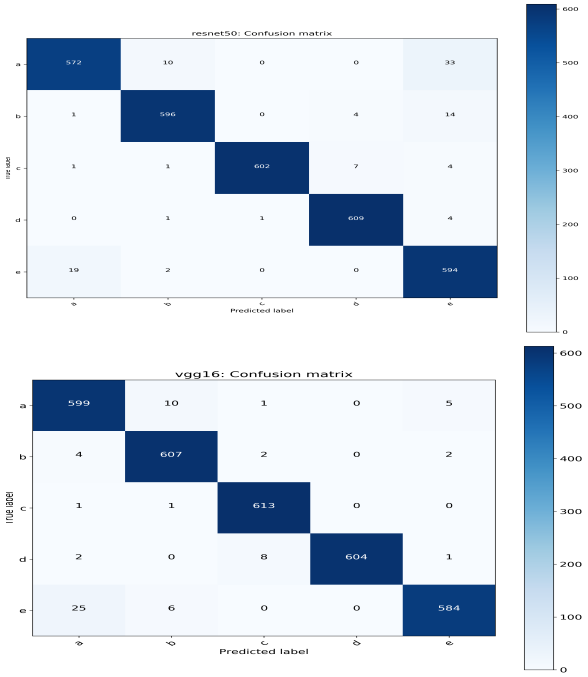

Fig. 7: A-E, 2-FC: Comparing confusion matrices between ResNet50 and VGG16.

4) A-E Recognition with 3-FC Retrained Layers: This experiment involved three fully-connected and two dropout layers for both models, with probability 0.2. This experiment, like the previous one, operated on the dataset composed of 3070 images per letter, for letters A-E. As with the previous experiment, VGG16 seems to converge to a higher training loss and lower accuracy than ResNet50. This fact is also evident in the confusion matrices, with VGG16 misclassifying slightly more than ResNet50. We also note that the training loss for both models seem to be higher than in the previous experiment (with 2-FC layers and one dropout layer), and the training accuracy lower. The results of this experiment can be seen in the figures on the next page.

In attempting the real-time recognition of ASL signs via the weights obtained from this model, we notice that VGG16 performs better. Thus, we proceed by only training VGG16 (with 3-FC Retrained Layers) for the A-Z dataset. We discuss the real-time results corresponding to this experiment in greater detail in Sec. III-B.

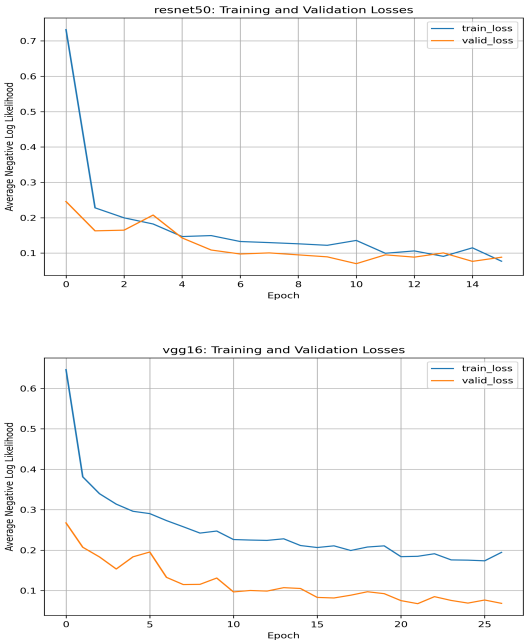

Fig. 8: A-E, 3-FC: Comparing loss curves between ResNet50 and VGG16.

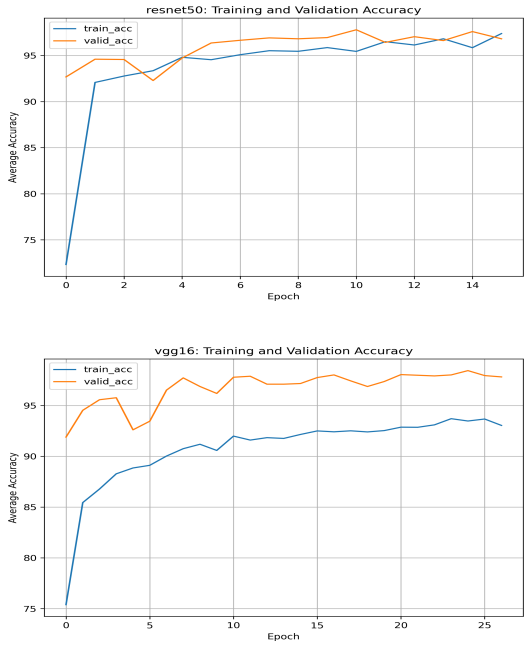

Fig. 9: A-E, 3-FC: Comparing accuracy curves between ResNet50 and VGG16.

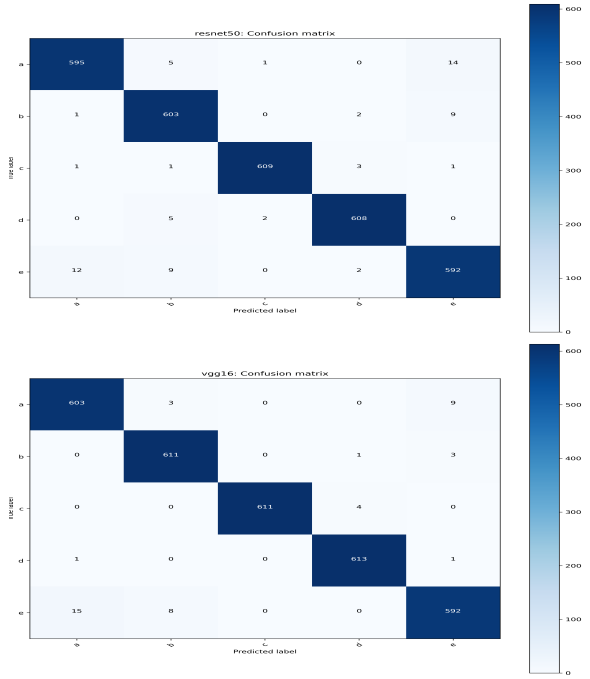

Fig. 10: A-E, 3-FC: Comparing confusion matrices between ResNet50 and VGG16.

5) A-Z Recognition with 2-FC Retrained Layers: This experiment involved two fully-connected and one dropout layer for both models, with probability 0.2. It operated on the dataset composed of 1070 images per letter, for letters A-Z. The model does well overall, reaching 91% accuracy on the validation set, and showing reasonable results on the confusion matrix. The results of this experiment can be seen in the figures on the next page.

We discuss the real-time results corresponding to this experiment in greater detail in Sec. III-B.

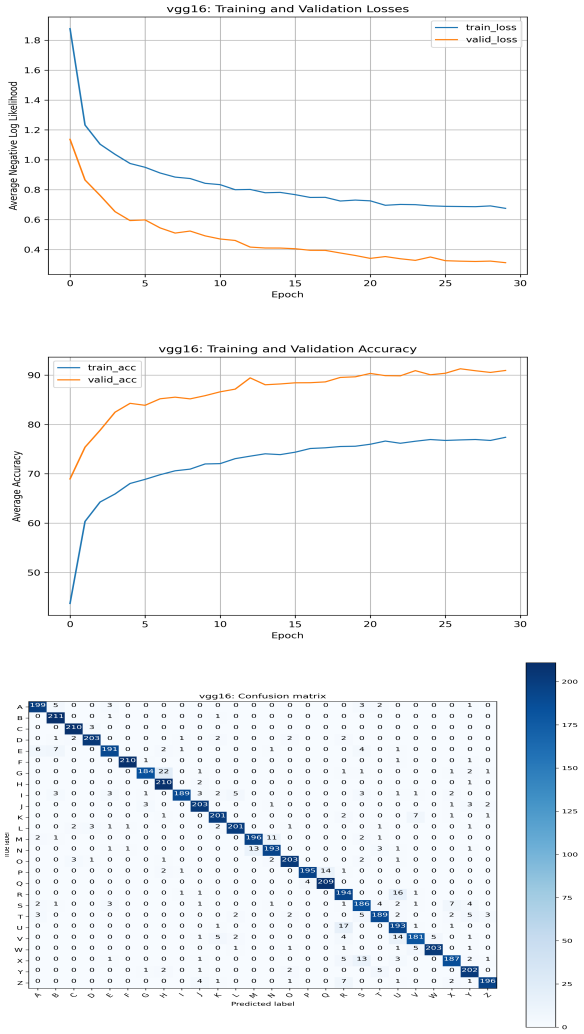

Fig. 11: A-Z, 2-FC: Loss, Accuracy, and Confusion Matrix.

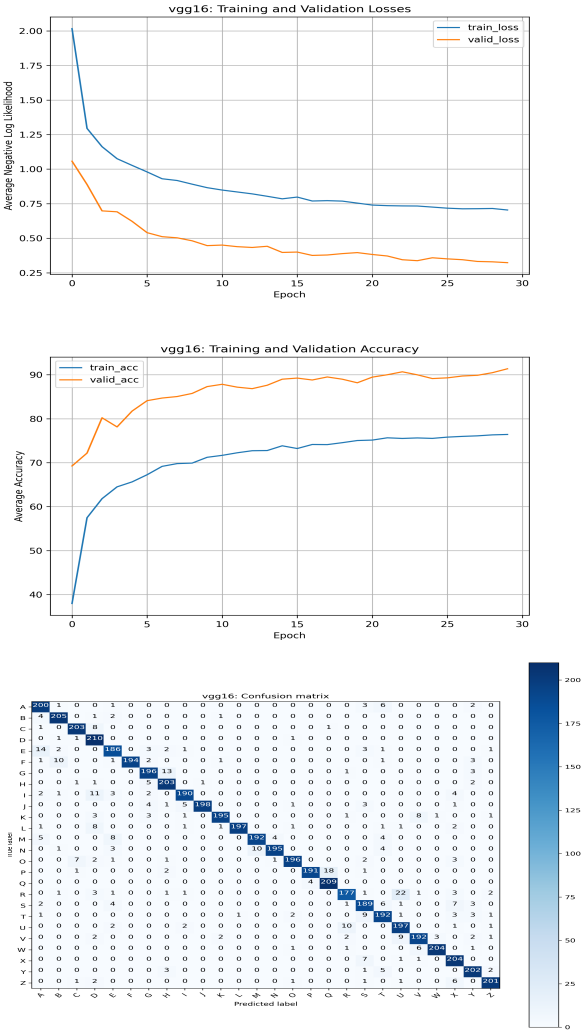

Fig. 12: A-Z, 3-FC: Loss, Accuracy, and Confusion Matrix.

6) A-Z Recognition with 3-FC Retrained Layers: This experiment involved three fully-connected and two dropout layers for both models, with probability 0.2. It operated on the dataset composed of 1070 images per letter, for letters A-Z. The model does well overall, reaching 91% accuracy on the validation set (like the 2-FC model), and showing reasonable results on the confusion matrix. We do note that the training loss was higher, and the training accuracy was lower for this 3-FC model compared to the 2-FC model. The results of this experiment can be seen in the figures in the next column.

We discuss the real-time results corresponding to this experiment in greater detail in Sec. III-B.

B. Real-Time Sign Recognition Experiments and Results

Using the models trained while conducting the various training experiments, they were put to the test by using live video from the project member's laptop camera. The frames of the video were standardized according to ImageNet standards, since that is what the original models were trained on. It was also ensured that the experiments were conducted under different lighting conditions and backgrounds in order to test the robustness of our trained model. In an effort to keep the report concise, we will be displaying a few of the results of the model which relatively produced the best results: VGG16 - 2-FC retrained on dataset from A-Z.

However, it is to be mentioned that there was a common issue faced by all the retrained models: The models were not invariant to the relative posture of the hand, e.g, if we turned our hand to a slightly different angle while signing the letter 'b', the models would misclassify it into some other letter. This is further discussed in Sec. IV.

1) A-E Recognition with 2-FC Retrained Layers: We used both the retrained ResNet50 and VGG16 models for this real-time experiment:

- ResNet50: This retrained model produced the worst results. It was unable to detect most of the sign-language gestures no matter which position, and lighting and background conditions were.
- VGG16: VGG16's performance was much better than ResNet50 on the real-time video. It was able to classify the letters 'b', 'c', 'd' and 'e' easily. However, it kept confusing 'a' for 'e' in almost all instances.

2) A-E Recognition with 3-FC Retrained Layers: After the disappointing performance of ResNet50 and due to the shortage of time, we decided only to experiment with the VGG16 model.

- VGG16's performance with 3 retrained fully-connected (FC) layers was very similar to VGG16 retrained using 2 fully-connected layers only.

3) A-Z Recognition with 2-FC Retrained Layers: As mentioned previously, we used only VGG16 for this experiment as well.

- VGG16's performance was favorable. It was able to recognize the following letters under different settings and lighting conditions: 'a', 'b', 'c', 'e', 'l', 'o', 'q', 'w', 'y'. In terms of performance, this model relatively had the best results. Some of the results are displayed in figure 13

Fig. 13: Correctly Translated Letters by VGG16

4) A-Z Recognition with 3-FC Retrained Layers: Again, we tested using only VGG16:

- This experiment's performance was slightly worse than the previous experiment (2-FC retrained layers). The model appeared to be more susceptible to making an error when the angle of the hand was changed. However, in terms of recognizing the letters, it was able to identify the letters similar to the 2-FC layered one, albeit with more difficulty.

IV. Discussion

In this section we will be discussing the performance of the real-time translator in-depth.

As mentioned in the previous section, all the trained models were not invariant to the relative posture of the hand. While the VGG16 model was able to classify the letters mentioned in the previous section, changes in the relative pose of the hand did force the model to misclassify. One major reason for this problem is the dataset on which the models were trained on; even though the dataset had images in different relative poses, those poses weren't at very different drastic angles. Therefore, when the model was trained on these images, it was able to generalize to only certain postures and positions of the hand for each letter.

Moreover, there were certain letters, e.g, 'p' and 'f' that were constantly getting misclassified as 'q' and 'w'. A possible explanation for this error is due to the fact that 'p' appears very similar to 'q' in terms of the way the fingers of the hand are positioned. Similarly, 'f' appears very similar to 'w' in terms of the way the three index fingers are positioned. This issue can best demonstrated by figure 14

Fig. 14: Top row: Incorrect classification of K and F. Bottom row: Actual Q and W.

We also note that letters like 'j' and 'z' require motion to be adequately communicated, and since our real-time detector classifies frame-by-frame; therefore, it would be very difficult for the detector to classify 'j' and 'z'. Perhaps it would be worth repeating this work with those categories removed from the dataset, since they will more likely lead to misclassifcation.

V. Conclusion

It is interesting to note in our experimentation that performance on the test set had almost no discernable relationship to real-time recognition performance. This leads us to believe that other results on classifying ASL letters via CNNs is not something we can take for granted; instead we must attempt real-time recognition obtained via their implementations. This essentially means that much previous work must be attempted again with this real-time goal in mind.

Overall, we are happy and proud of the work done on this project. We believe that our work is comprehensive and thus is deserving of a good grade.

A. Further Work

We believe that by implementing a image-preprocessing pipeline into the real-time implementation, as Atwood et al. did [2], would significantly improve the real-time performance of the sign classifier. For example, we would crop to the region of interest in the image, perform adaptive histogram equalization, and perhaps binarize the cropped region so that the edges are immediately apparent due to the contrast. However, this would likely involve creating our own dataset of handsigns that have been processed in this way, and require far

more effort to perform. Perhaps convolutional neural networks could be implemented for each of these steps too.

Additionally, we suspect that modelling the order of letters in American English via hidden Markov models (HMM) or an RNN will substantially aid real-time letter recognition, in the way described by Starner and Pentland in [12], [13], or by Bantupalli in [14].

Finally, a point of ethical concern: The real-time classification worked better for one of the authors than the other. We suspect this is because all the samples in the dataset were from people with fairer complexions. Thus, our implementation gives people with darker complexion the equivalent of a lisp in sign-language, which is a cause for concern and needs to be addressed before a public-use implementation can exist. We need better representation of human diversity in our dataset.

References

[1] R. E. Mitchell, T. A. Young, B. Bachleda, and M. A. Karchmer, "How Many People Use ASL in the United States?" 2005. [Online]. Available: https://web.archive.org/web/20110604191021/https://www.ncdhhs.gov/mhddsas/deafservices/ASL_Users.pdf

[2] J. Atwood, M. Eicholtz, and J. Farrell, "American Sign Language Recognition System," pp. 1–4, 2012. [Online]. Available: https://jasonatwood.io/wp-content/uploads/2012/12/Atwood-Eicholtz-and-Farrell-ASL-Recognition.pdf

[3] D. Aryanie and Y. Heryadi, "American sign language-based finger-spelling recognition using k-Nearest Neighbors classifier," *2015 3rd International Conference on Information and Communication Technology, ICoICT 2015*, pp. 533–536, 2015.

[4] C. Sun, T. Zhang, B.-k. Bao, and C. Xu, "Latent support vector machine for sign language recognition with Kinect," *ICIP*, pp. 4190–4194, 2013.

[5] B. Garcia and S. A. Viesca, "Real-time American Sign Language Recognition with Convolutional Neural Networks," *Stanford CS231n*, 2016. [Online]. Available: http://cs231n.stanford.edu/reports/2016/pdfs/214_Report.pdf

[6] Q. Gao, U. E. Ogenyi, J. Liu, Z. Ju, and H. Liu, "A Two-Stream CNN Framework for American Sign Language Recognition Based on Multimodal Data Fusion," *Advances in Intelligent Systems and Computing*, vol. 1043, no. January, pp. 107–118, 2020.

[7] A. Thakur, "American Sign Language Dataset, Version 1," 2019. [Online]. Available: https://www.kaggle.com/ayuraj/asl-dataset/metadata

[8] Akash, "ASL Alphabet, Version 1," 2018. [Online]. Available: https://www.kaggle.com/grassknoted/asl-alphabet

[9] A. Wan, "How To Build a Neural Network to Translate Sign Language into English," 2020. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-build-a-neural-network-to-translate-sign-language-into-english

[10] W. Koehrsen, "Transfer Learning with Convolutional Neural Networks in PyTorch," 2018. [Online]. Available: https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce

[11] L. Pigou, S. Dieleman, P.-J. Kindermans, and B. Schrauwen, "Sign Language Recognition Using Convolutional Neural Networks BT - Computer Vision - ECCV 2014 Workshops," pp. 572–578, 2015. [Online]. Available: https://core.ac.uk/download/pdf/55693048.pdf

[12] T. Starner and A. Pentland, "Real-time American Sign Language recognition from video using Hidden Markov models," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 265–270, 1995.

[13] T. Starner and A. Pendland, "Visual Recognition of American Sign Language Using Hidden Markov Models," p. 2013, 2013.

[14] K. Bantupalli, "American Sign Language Recognition Using Machine Learning and Computer Vision," 2018.