CSE 803: American Sign Language Classifier

Kevin Le Jarod Collier Sean Depke

December 12, 2022

Abstract

The classification of images of American Sign Language (ASL) letters helps in providing a realistic implementation of a real world use of Neural Networks. By experimenting with different models that have proven to be robust helps in solidifying their uses in potentially other implementations as well. This project uses a live camera to supply frames to the model to predict what letter a user is signing.

1 Introduction

This project is used to experiment and implement a real world use case for image classification. More specifically, our models will be shown an image frame from a video feed of someone's hand signing out a letter in ASL and the model will output the corresponding alphabet letter. See below in Figure 1 for an example.

Figure 1: ASL Alphabet

ASL was used as the subject for the importance of communication and how more prevalent it has become to make media more accessible to those deaf or hard of hear. For example, a rise of sign language interpreters in almost all areas of televised content where there is consistently a sign language interpreter for every presidential address or political debate. The goal is to showcase a helpful use case for the real world. With classification, next could come the reverse where words or letters can showcase a sign instead. A good stepping stone to developing an interpreter for ASL, would be to first create a baseline project, which is computer recognition of ASL letters.

With the understanding that the more data in the training set, the more realistic it would be to achieve good results, a large data set was sought out. The data set was decided to be from Kaggle and a sample is found in Figure 2 below. The data set contains 29 signs, the alphabet, space, delete, and images of nothing.

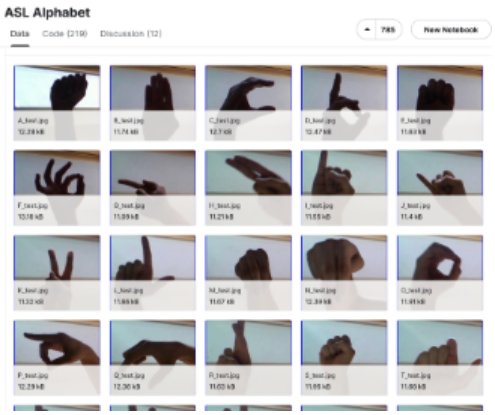

Figure 2: Kaggle Data set

It was decided to implement and compare three different models, a simple Convolutional Neural Network (CNN), AlexNet, and ResNet, with which will process the

aforementioned Kaggle data set. In the comparison, an understanding of the architectures of these models is sought after, alongside potential improvements and modifications, and what this means for future use.

2 Related Work

The project draws upon the research from AlexNet and ResNet for the model implementation, and the simple CNN was referenced from a YouTube tutorial by Keith Galli.

2.1 Simple CNN

The simple CNN was referenced by a neural network tutorial by Keith Galli. The Model was first set up with an average pooling layer to average the feature map of the image. Then followed by two convolutional layers to filter out the image and help indicate locations of strength to detect features, ReLu is used for the activation.

This takes any value that is less than zero and sets it to 0. Any value that is greater than 0 is kept at that value. This is then further intensified by a max pool layer which is applied to prevent over fitting, and then the next layer flattens the results into a 1D array. This gets fed into two Dense layers to start classification of the image. First Dense layer uses ReLu for activation, and the final layer with 29 neurons uses SoftMax.

2.2 AlexNet

AlexNet was the first paper that showed you could use CUDA (GPUs) to achieve a much larger scale of model and in turn, number of learnable parameters. This was so large that they needed to split the model between two GPUs. At this time, CNNs were not immediately used for this type of image learning, and this has been a hugely influential paper through the years.

ReLU Nonlinearity was one of the major design choices made instead of the commonly used sigmoid or hyperbolic tangent, as they were inspired by anatomical neurons. This was chosen because of the slowness of the saturating nonlinearities gradient descent for these neuron-inspired choices versus the non-saturating linearities of reLU, providing additional speed into the model. Researchers note that the training error rate was six times faster.

Dropout was also used. According to the original authors, this implemented better techniques to reduce risk of over fitting. However, some modern researchers argue that may not have necessarily reduced over fitting, but allowed for normalization across the model. Instead, the large data set used with the model was more of the solution that removed the over fitting.

AlexNet was modeled with 2 convolutional layers, each followed by a Max Pool, then 3 more convolutional layers with a single Max Pool layer. 3 classification or dense layers were used to categorize the image. Originally it would categorize 1000 labels, but for the team's purpose, 29 was used instead.

2.3 ResNet

ResNet was built on what was created in AlexNet, and increased the number of learnable parameters by an order of magnitude. At the time, researchers understood that increasing the layers too far would cause a drop in performance. This drop was understood to be not caused by over fitting. Most of the models being created were similar to AlexNet, and most popularly, VGG-nets. These VGG nets would operate by performing (sometimes several in sequence) convolutions, then follow up with down scaling/pooling.

With the understanding that the networks learn more and more abstract features as you go up the layers, the exact localization of these features becomes less important. The VGG networks took advantage of this by down scaling these features.

Residual Connections is the major breakthrough for ResNet. This implemented skip connections so that there are less overall FLOPs. In comparison VGG-19 has 19.6 billion FLOPs versus ResNet with 3.6 billion. Overall this provided an incredible increase in performance by including these skip connections.

The ResNet Model that was used for this project had 4 ResNet blocks created with sub blocks of identity and convolutional blocks and 2 classification layers. The identity blocks are used to correspond to the case where the input activation has the same dimension as the output activation. The convolutional block was used when the input and output dimensions don't match. With a total of 13 identity blocks and 3 convolutional blocks, this would total to 35

convolutional layer in just the 4 ResNet blocks. In addition to the initial convolutional layer, it would be 36 in total.

3 Method

Though these three models were created almost a decade ago, the team wanted to leverage them as more modern models tend to increase in scale, complexity, and time to train. Once the team started to attempt to train the ResNet model, it was discovered that each epoch took around 3 hours to complete. With the intention to run each model for at least five epochs, training ResNet for 15 hours was unreasonable, especially if the team had to tune any of the hyperparameters. The team had success using a GPU when implementing neural networks in a PyTorch environment, but since this project aimed to use TensorFlow, the team could not get the versioning to work right away. The final environment that worked for the team was to ensure that the package "tensorflow-gpu" was installed at version "2.10.1" because that worked with NVIDIA Cuda Development 11.7. Once one team member could use a GPU, the training time difference between the ResNet model and the AlexNet model was still large, but not on the scale of multiple hours. This allowed the team to tune the ResNet model and use it to detect a user's sign language in a live video feed. It is important to note that the results across the different models were all very good, especially considering that some of the models are relatively old in the field.

3.1 Simple CNN

The Simple CNN did a great job. The team trained the Simple CNN for 5 epochs, each with 2175 iterations. In Table 1 below, the results of the training and evaluation can be seen. The Simple CNN ended up with a training accuracy of 96.23% and a validation accuracy of 98.31%. The Simple CNN also had a great final training loss of 0.1109 and a validation loss of 0.0536. When the team evaluates the model with the test set, the results are a test accuracy of 92.86% and a loss of 0.0536. Table 1 also shows us that the Simple CNN model was the faster to complete, with the total time training completing after 25 minutes. The Simple CNN was fitted using the Adam optimizer, a loss function of $SparseCategoricalCrossentropy()$, an image shape of 200 x 200, and 3,710,525 trainable parameters.

Epoch	Training Loss	Training Accuracy (%)	Validation Loss	Validation Accuracy (%)	Epoch Duration (s)
1	1.0774	67.33	0.3072	90.37	569
2	0.2986	90.07	0.1975	94.32	481
3	0.1870	93.65	0.0732	97.72	471
4	0.1334	95.36	0.0624	97.96	456
5	0.1109	96.23	0.0536	98.31	457

Table 1: Simple CNN Results

When fitting a model using TensorFlow, it outputs a $history$ object and that allows the user to look at the accuracy and loss over each epoch. In Figures 3 and 4 below, the history for the Simple CNN is displayed. The main finding from the history of the fitting for Simple CNN is that the model is essentially peaking after one epoch and 2175 iterations. The accuracy gets well above 90% for both the training and the test data after one epoch. The loss is also mostly minimized after one epoch as well. After the second epoch, there is not much more gains for the accuracy or the loss.

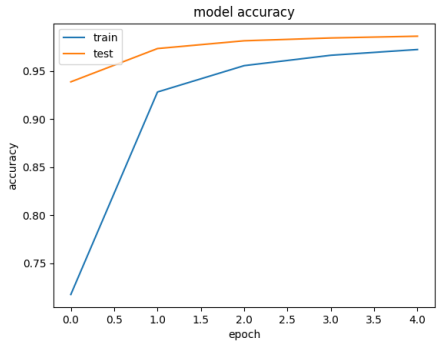

Figure 3: CNN Model Accuracy

3.2 AlexNet

The goal of using AlexNet was to expand off of the Simple CNN by following the AlexNet model. The team wanted to implement a well documented model in the computer vision field and evaluate its performance against the ASL data. AlexNet was much more detailed of a model compared to the Simple CNN and this can be seen by TensorFlow's output that said there were 58,402,589 trainable parameters. In contract, the Simple CNN had much less parameters with 3,710,525. Even though AlexNet had many

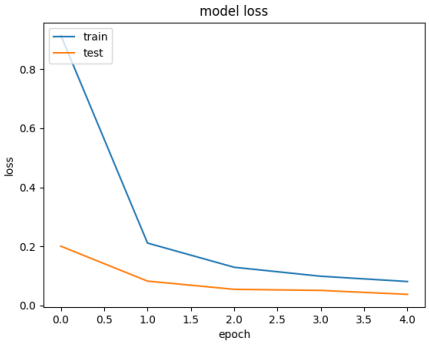

Figure 4: CNN Model Loss

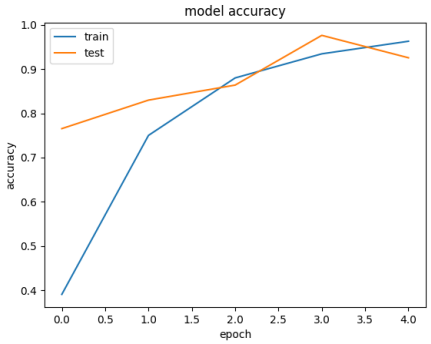

Figure 5: CNN Model Accuracy

more parameters, it only took around 10 more total minutes to train, finishing after 35 minutes. For consistency purposes, the team ran AlexNet again for five epochs with 2175 iterations in each epoch. The final training accuracy was 96.29% and the final training loss was 0.1166. The final validation accuracy was 97.63% and the final validation loss was 0.0904. These numbers can be seen below in Table 2. The evaluation of AlexNet proved to be very similar to the Simple CNN model, ending with 92.86% accuracy and a loss of 1.2979. AlexNet was fitted with the SGD optimizer, a learning rate of 0.001, the loss function of $SparseCategoricalCrossentropy()$, an image shape of 227 x 227, and as mentioned previously, 58,402,589 trainable parameters.

Epoch	Training Loss	Training Accuracy (%)	Validation Loss	Validation Accuracy (%)	Epoch Duration (s)
1	2.1343	39.09	0.9095	76.55	374
2	0.7677	75.502	0.5399	82.99	374
3	0.3700	88.00	0.4114	86.39	371
4	0.2037	94.45	0.0904	97.63	379
5	0.1166	96.29	0.2258	92.56	385

Table 2: AlexNet Results

In Figures 5 and 6, the $history$ object is again used to look at the accuracy and loss over each epoch for AlexNet. The first difference that is noticeable compared to the Simple CNN is that the model does not really peak until the third epoch. It is also possible that since the accuracy and the loss do not flatten out nearly as much as the Simple CNN, if AlexNet was trained over more epochs, then that could result in higher accuracy values.

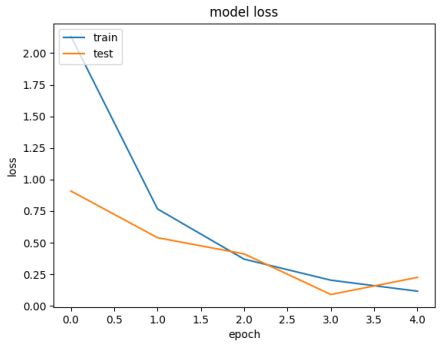

Figure 6: CNN Model Loss

3.3 ResNet

The ResNet model is the newest model in the field that the team implemented. The goal with using ResNet was to see how high high of an accuracy was possible. When implementing ResNet, it was quickly discovered that it was incredibly complex and took over three hours to train one epoch of 2175 itreations. With the goal to be consistent and run five epochs, that would have been 15 hours of training time and that is far too long. The team attempted to use Google Collaborator's GPU, but that quickly ran out of RAM. The team did figure out how to use Jarod's GPU and that brough the total training time of all five epochs down to one hour and 40 minutes. The results of each epoch can be seen in Table 3, below. The final training accuracy was 99.11% and the final training loss was

0.0296. The final validation accuracy was 98.78% and the final validation loss was 0.0373. The evaluation accuracy was 92.86% and the evaluation loss was 2.2747. ResNet was fitted with the Adam optimizer, the loss function of $SparseCategoricalCrossentropy()$, an image shape of 200 x 200 25,516,317 trainable parameters. It is important to note that there are much less trainable parameters in the ResNet compared to AlexNet.

Epoch	Training Loss	Training Accuracy (%)	Validation Loss	Validation Accuracy (%)	Epoch Duration (s)
1	2.1343	39.09	0.9095	76.55	374
2	0.7677	75.502	0.5399	82.99	374
3	0.3700	88.00	0.4114	86.39	371
4	0.2037	94.45	0.0904	97.63	379
5	0.1166	96.29	0.2258	92.56	385

Table 2: AlexNet Results

In Figures 7 and 8, the $history$ object is again used to look at the accuracy and loss over each epoch for ResNet. The results are similar to the Simple CNN results in that after one epoch of 2175 iterations, the accuracy and loss spike close to their best values. For some reason in the third epoch, the loss and accuracy both get worse before ending better. The team suspects that running the ResNet with more epochs might help to flatten this curve out. Overall, ResNet did an incredible job getting very close to 100% and almost 0 loss.

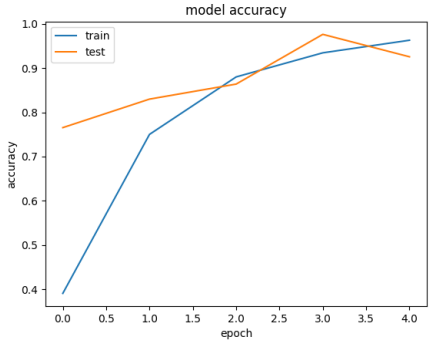

Figure 7: CNN Model Accuracy

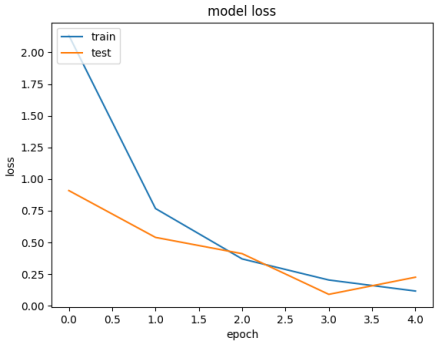

Figure 8: CNN Model Loss

3.4 Video Feed and Data Augmentation

After successfully implementing three high performing models on the Kaggle data set the team wanted to take this implementation back into as close of a real-world implementation as possible. The team ideated on the concept of a real-time ASL translation phone-based app. After performing research on this, it was found that there were no available apps that offer this service. The closest was an app that would give someone feedback on their ASL as they practiced. The team sought out additional research and found minimal research applying the translation in real time. Most notably, this was found in a paper in IJERT. However, only one model was developed, though it showed high performance.

With this understanding that there is a potential gap in research here, the team decided to apply their models to a real-time environment. Each of the three models was run in real time, using an architecture for the real-time algorithm found by Vikram Menon. In addition to getting the architecture to work for the models the team developed, this required practice of ASL finger spelling as well. There are several instances where even slight orientation changes result in a different letter entirely.

After applying the models in real time, it was noted that some of the letters were much easier to get the algorithm to recognize than others. With performance on the test and validation data set of mid to high ninety percent, the team expected all of the letters to show up well. A further investigation into the data set found the answer. Despite

the 87,000 images in the data set, the team noticed that almost all of the images had strong parallel lines in the background of the images in almost the same place every time. The team had been using a flat background for the test camera, and tested to see that there was higher performance when the camera was pointed higher to the corner of the ceiling. The team noted that the coloration of the skin was also different on the tester versus the training data, and that this would contribute to the lesser performance of the algorithm.

With the understanding that this is a large limitation when bringing it to the real world, the team wanted to create a better training environment. However, creating an entirely new data set would be extremely time consuming and was decided to be out of scope. It was decided that the images in the test data set could have their backgrounds removed through image masking. Further research found that there is a high performing Google model called MediaPipe that, when implemented, could do this removal. Data Flair has a tutorial on its implementation and this was followed to do the background removal in real time. The code was then modified so that it could be applied to a sample of the training data for each individual image.

A new data set was created by combining a subset of the original images and the removed images alongside each other. In order to train the data in a reasonable amount of time, only 14,500 of the original images were used, making a total of 29,000 images in the data set. This was then tested. This had higher performance as it was much faster at recognizing live finger spellings. However, this was still not working for many of the ASL letters.

Though originally decided to be out of scope, the team decided to do a partial implementation of their own data set. The live feed code was modified to take a specific image on command and an image was taken for each ASL letter. These images already had their background removed, as it was using the live background removal tool at the same time. With the understanding that this single image for each letter would have minimal impact on the overall data set and that taking many photos would be incredibly time consuming, it was determined that Keras data augmentation would be used. An example of this is found in Figure 9 below.

Figure 9: Sample Augmented Image

Following previous work from Ravindu Senaratne, data augmentation with Keras was performed on the personalized dataset. It was important to not have the images moved too far out of their original positions, as for several of the letters, this would result in overlap with a different letter. Additionally, the data could not be flipped, as this would result in conflicts as well. Some of the most time consuming part of this process was setting up the code for reusability in future projects.

After this was performed, the final dataset consisted of 950 images for each letter of the original dataset and 50 images for each letter of the personalized and augmented images. This ended up with a total of 29,000 images for training. Final testing was performed with a test word of "THANKS", in ASL. Using several different lighting styles for the background, it was found that the highest performance out of any was using a backlight and no background removal tool. This is shown in the comparison video in the presentation. A screenshot from the testing video is shown below in Figure 10. It is also noted that we expect higher performance when this is run using a more diverse dataset, and with more included data from the original dataset. Even with this minimal data, the team was happy to accurately detect all live signed letters with a high level of reproducibility with this tool.

4 Conclusion

In conclusion, the models all had relatively similar accuracies and losses with the training data. In context of the

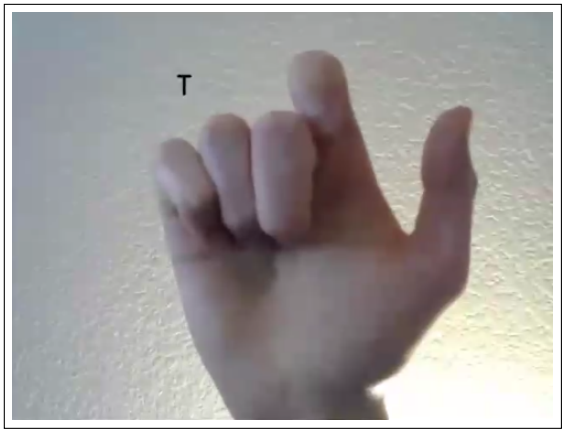

Figure 10: Successful Backlight Test

model development, the simple CNN would have been the most appropriate model to use for this project with only 29 labels and quicker training times. AlexNet and ResNet was designed to train on millions of images with up to 1000 labels. Thus, training time is noticeably long in comparison to the simple CNN.

Some problems the models had while training was the inconsistency of backgrounds. Due to the lighting and background of the training data set, it would cause inaccuracies in the prediction of the video feed frames. There was a need to do some image augmentation and background sanitation to remove the variance from the training data. Another issue is dealing with signs that would require movement. The current implementation can only handle predicting still images. Thus, any signs that require complicated movement or shape changing would have trouble being classified. As an example, signs "J" and "Z" require the drawing of that letter. Technically, either letter can be signed with drawing with the pinky or index finger. The current data set that the models trained on does not account for that.

For expansion beyond just letters, more labels and images would require a more robust model. In that case, AlexNet and ResNet would be more appropriate to use. The data set used would also have to have a variety in it's training data set to limit the issues stated before with the current implementation.

5 Contribution

The team mostly met up in meetings and contributed to the work together, however the following will illustrate the main contribution each author provided.

5.1 Kevin Le

Kevin was in charge of finding and implementing the models for the project. He would also help in debugging the implementation with Jarod, who would train the models on his machine. Kevin also created the presentation template and left placeholders for Jarod and Sean to put their work in. Kevin also did the initial compilation of the project report in LaTeX

5.2 Jarod Collier

Jarod was responsible for training the models as it turned out that his computer had the least number of issues trying to use TensorFlow with a GPU. Jarod, Kevin, and Sean worked together to debug the models and find run-time issues. He also collected all the data for the implementation to help with the report out. Jarod contributed to the results and implementation section of the report and presentation.

5.3 Sean Depke

Sean was responsible for the video capture script and background removal software used to sanitize the data set. He was also in charge of the progress report and proposal for the project. Sean also found great reference material to help the team understand what the different models were doing.

6 References

[1] K. Galli. *Real-World Python Neural Nets Tutorial (Image Classification w/ CNN) | Tensorflow & Keras*. YouTube, 2020.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *arXiv.org*, 10-Dec-2015. [Online].

[3] Y. Kilcher. *[Classic] ImageNet Classification with Deep Convolutional Neural Networks (Paper Explained)*. YouTube, 2020.

[4] Y. Kilcher. *[Classic] Deep Residual Learning for Image Recognition (Paper Explained)*. YouTube, 2020.

[5] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet: Classification with deep convolutional Neural Networks.” [Online].

[6] Team, D. F. (2021, October 27). *How to remove background of images in python?* DataFlair. Retrieved December 11, 2022, from https://data-flair.training/blogs/python-remove-image-background/

[7] Menon, V. (2020, April 4). *Using AI to translate sign language in real time*. Medium. Retrieved December 11, 2022, from https://towardsdatascience.com/using-ai-to-translate-sign-language-in-real-time-96fe8c8223ed

[8] Senaratne, R. (2020, July 21). *How to augmentate data using Keras*. Medium. Retrieved December 11, 2022, from https://towardsdatascience.com/how-to-augmentate-data-using-keras-38d84bd1c80c