CSCI 1430 Final Project Report: Creating an ASL Translator using a CNN

ASL Translator: Noah Atanda, Theo McArn, Jorge Sanchez.
Brown University

Abstract

In this project we developed an algorithm that is able to classify the letters of the ASL alphabet. The motivation for this project was to promote ASL education and make it easier for people to interact with those who used ASL as their primary source of communication. In order to accomplish this task, we developed and trained a CNN to classify images of the different signs. In the end we managed to achieve an accuracy of 99% when we tested our trained model against the testing images from the data set.

1. Introduction

In today's multi-cultural society, the ability to communicate with others in a secondary or tertiary language is a necessity. Especially in the United States, speaking languages aside from English allows foreign raised residents to feel understood and welcomed. However, one language that is often overlooked is American Sign Language. Some humans are born with hearing disabilities or acquire them later on in life. Accommodations must be made for those who are deaf or hard of hearing. This is why sign language was created, so that those with disabilities could communicate with everyone else and vice versa. Unfortunately, not everyone understands or speaks sign language. As a result, the goal of this project is to make ASL a more accessible language. This program could be used as both a translation tool for those fluent ASL and as a teaching tool for people who do not know ASL. The overarching goal of this project is to make it easier for people with hearing disabilities to communicate with others and live their daily lives. This will allow for smoother communication and overall, an inclusive society.

2. Related Work

There has been a lot of research on this topic as is it has a lot of useful applications. There exist many datasets to train off of and many people have worked on developing optimized models with high accuracy.

For our final model, we relied on Dr. Kumar and his article about developing his own CNN for letter classification[2]. Kumar achieved extremely high accuracy on his model so we used his architecture in our own model as well. His article also included ways to acquire metrics for our model such as the plots and confusion matrix that were very useful.

In terms of datasets, there were many different ones to choose from. Initially, we were using a dataset from Roboflow [3] with large, color, images. However, we were not receiving good results. In the end we decided to go with a dataset from Kaggle, which Kumar used, that had more images that were much smaller and gray scale [4].

In addition, we used many python libraries in order to train our model, make the demo, and produce graphs. We utilized Numpy, Tensorflow, Keras, Pandas, skimage, matplotlib, and openCV among others in order to write our code. In addition, in order to make the visualization for our model, we utilized a library called visualkeras which was availible on github [1].

3. Method

The overall problem we dealt with was image classification. More specifically, the problem lied in recognizing a sign out of the 26 ASL letters. Thus, we needed a solution that would be able to ignore background clutter, adjust image brightness levels, and recognize images from a live video. The best way to go about this was to create a Convolutional Neural Network (CNN). The first thing we did was gather a dataset of ASL letters to use for training and testing. The next step was processing the images and creating a CNN that worked best with our data set for training. The third stage was training the network over multiple iterations until the accuracy of the predictions were acceptable. The last step was using the model and the acquired weights to evaluate new, never before seen images of ASL letters. Overall, this project highly dependented on getting a reliable CNN constructed.

For our project, we built two models, named old model and new model. At first, the old model's architecture was largely reminiscent of Homework's 5 VGG model. However, after the first run through of training, the accuracy was not what we had hoped. Thus, we sought out to find a model

that would work well with our images and specific problem. After a tedious series of trial and error, we felt we had done all we could and found a respectable model that was able to achieve about 52% accuracy on our testing data. The old model consisted of two blocks where each block had two convolutional layers and a max pooling. Then we used a flatten function and two dense functions. The first dense function used the rectified linear activation function with 128 units and the second dense function used the softmax function with 26 units. For the loss function, we decided to use the Sparse Categorical Cross Entropy function and for the optimizer, we decided to use the Adam optimizer. As for the hyperparameters, we decided to use 50 epochs, an image size of 32x32, a learning rate of 1e-4, and a batch size of 10. The figure below represents the architecture of our old model.

```
self.architecture = [
#Block 1
Conv2D(256, 3, 1, padding="same",
       activation="relu", name="
           block1_conv1"),
Conv2D(256, 3, 1, padding="same",
       activation="relu", name="
           block1_conv2"),
MaxPool2D(2, name="block1_pool"),

# Block 2
Conv2D(512, 3, 1, padding="same",
       activation="relu", name="
           block2_conv1"),
Conv2D(512, 3, 1, padding="same",
       activation="relu", name="
           block2_conv2"),
MaxPool2D(2, name="block2_pool"),
Flatten(),
Dense(128, activation="relu"),
Dense(26, activation='softmax')
]
```

From here, we decided to create a new model that would hopefully yield a higher accuracy. Finally, after going through many articles, we came across Dr. Vaibhav Kumar's article [2]. This helped us create a new model which consisted of three blocks with each block having a convolutional layer, a max pooling, and a dropout. Then, at the end of it all, the model used the flatten function, a dense function, a dropout function, and a dense function once again. The first dense function used the rectified linear activation function with 512 nodes and the second dense function used the softmax function with 25 nodes. Figure 1 gives a good visualization of what this new model looks like. For the loss function, kept the Sparse Categorical Cross Entropy function and for the optimizer, also kept the Adam optimizer. As for

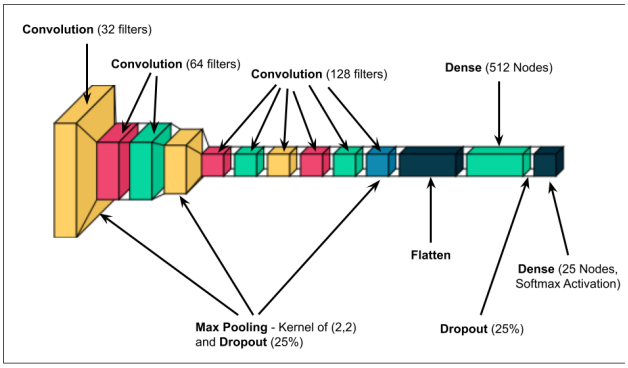

Figure 1. CNN Visualiztion

the hyperparametes, we decided to stick with 50 epochs, but use an image size of 28x28, a learning rate of 0.001, and a batch size of 512.

Shown here is the equation for the loss function: sparse categorical cross entropy. (Eq. 1):

$$CCE(p,t) = -\sum_{c=1}^{C} t_{o,c} log(p_{o,c}) \quad (1)$$

4. Results

As mentioned above, the results that we were seeing from our initial model were undesirable. On our best model, we only received a training accuracy of 52%. Even though the loss was decreasing and the accuracy was increasing, it was not adequate enough. As you can see from our loss values over time in Figure 2, even though our training losses were steadily decreasing, our validation losses stayed practically constant. This supports the fact that our model was clearly a poor fit. This is further proven by looking at the Accuracy over time shown in Figure 3. Similar to the loss, while the training accuracy increased, the validation accuracy leveled off very quickly. This further indicated the bad performance of this model. In Figure 4 we can see a sample of testing images and the predicted class. The predictions in red indicate a false classification. This further demonstrates the inability of the model to make correct predictions.

This low accuracy is also due in part to the variability in the test images. The signs photographed were taken from different angles, with different lighting and backgrounds. While this is beneficial for creating a robust model, it is definitely part of the reason why the accuracy was so low.

We realized that there must be a better way to create a model than simply guessing and checking. At this point we decided to outsource and look to see how other people trained similar models.

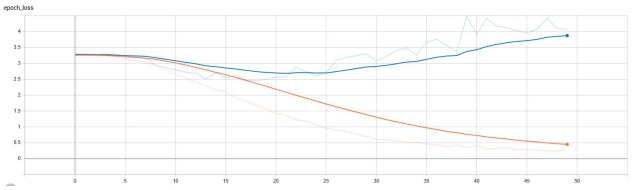

Figure 2. Original Model Loss

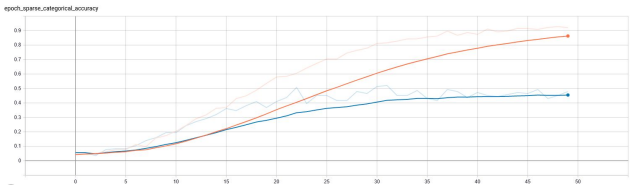

Figure 3. Original Model Accuracy

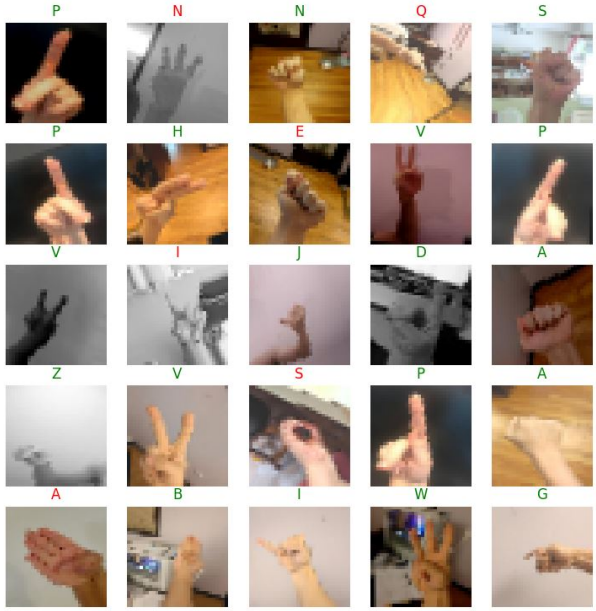

Figure 4. Original Model Sample of Predictions

Once we found Dr. Kumar's article and adapted our model to something similar, we immediately saw much better results. This new model had only about 100,000 parameters compared to the tens of millions in our former attempts. As a result, the training time was much faster, only about a minute compared to our old trials of about 40 minutes. Not only run time improved either. We saw drastic improvements in our accuracy and loss.

After 50 epochs with this new model, we achieved training and validation accuracy of about 99% and testing was about the same. The evolution of our training and validation accuracy over the epochs can be seen in Figures 5 and 6

This impeccable accuracy that we were able to achieve with this model can also be visualized in this confusion

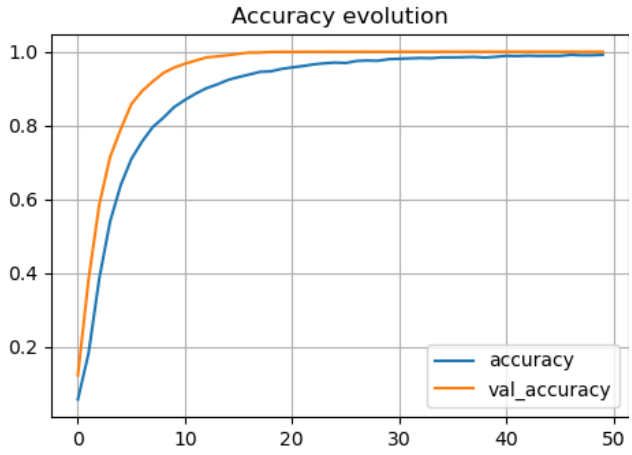

Figure 5. New Model Accuracy

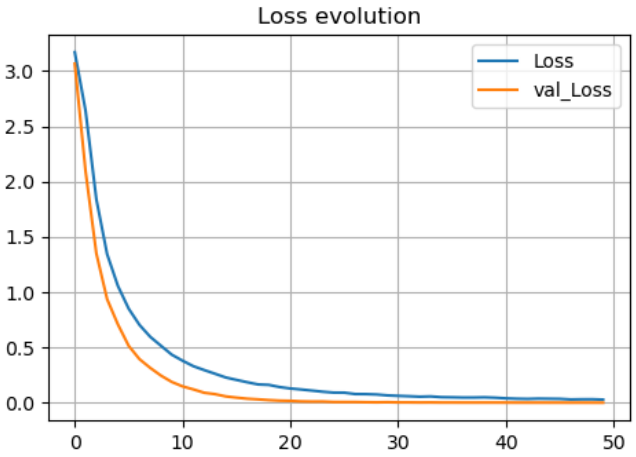

Figure 6. New Model Loss

matrix that we constructed with the insight from Kumar's article [2]. As you can see from the confusion matrix in Figure 7, there were very few miss classifications made when testing on this model.

We can also visualize the accuracy that we received by take a sample of the testing images and looking at the true and predicted classes for these images. Figure 8 is a sample of test images and you can see that in every case the prediction matches the ground truth label.

Some of the changes that can account for this significant improvement is the switch in datasets from the large, RGB images, to the much smaller, grayscale ones. This allowed us to simplify the problem and work with less data. As a result, the network could be smaller and have less parameters. In addition, this new data set had images that were much more uniform and consistent. The signs were always in the center of the frame the lighting was consistent, and the background was plain. As a result, this makes classification much easier

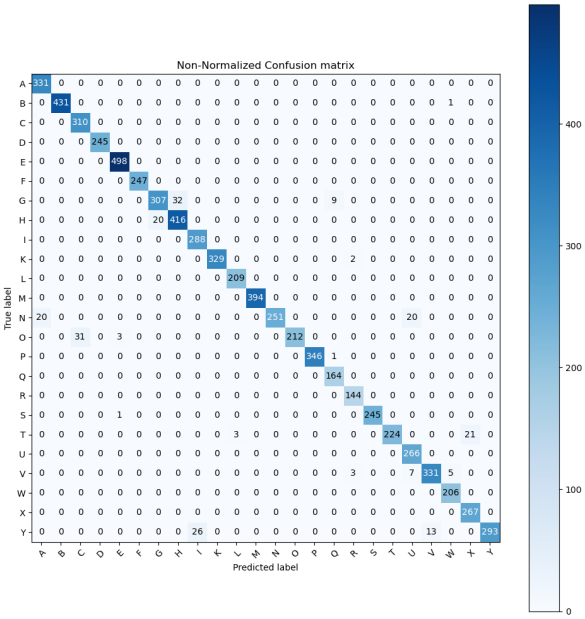

Figure 7. Classification Confusion Matrix

Figure 8. New Model Prediction Sample

for the CNN and is likely a large reason for the significant increase in accuracy.

However, despite these improvements in accuracy and run-time, this model does have some severe draw backs. Because we used the much more normalized data set to train, the model was unprepared for the noise and inconsistency of the 'real life' photos that would be input when making real time predictions. As a consequence, the accuracy of classification when running our model on new images was extremely low.

In Figure 4, we are able to see the images the old model used to train. For the most part, there seems to be minimal preprocessing. All one sees is a blurry image, thanks to the standardization and the occasional horizontal flip. At the same time, there was a lot of background noise which, while it impacted the accuracy, also led for a more robust model. In Figure 8, we see the images that the new model used to train. As opposed to the old data, these new images had significanly less variablility. The images have very similar orientation, lighing and backrounds. As a result we achieved higher testing accuracy but we loose the ability to predict letters when conditions are not perfect.

4.1. Technical Discussion

The old model was able to achieve an accuracy of 52 percent after many changes. The old model started off with an accuracy of about 3 or 4 percent which was the probability of guessing one of the letters. The problems with the model included low accuracy, long training times, high number of parameters, heavy preprocessing, and not enough images.

One of the first modifications we did was alter the preprocessing file. Previously, the images were coming out over saturated and so we removed an RGB filter as the image was already in color. This was able to fix the images and allowed the model to train on better quality images. On top of this, we played around with the amount of convolutional layers we had to see which would give us a reasonable amount of parameters. We went from around 80 million parameters to around 20 million. This was able to reduce the training time and increase the accuracy.

We also played around with the hyperparameters. For example, we tried various learning rates, various batch sizes, and various image sizes until we found a combination that worked well.

Lastly, we realized that we simply weren't using enough images to train. For each letter, we used about 40 or 50 images to train where most of these images were similar in the sense that they had similar backgrounds, brightness levels, and angles. All in all, our old model revealed to us how each element of the CNN contributes to the accuracy and they all depend on one another in some way.

Outside of accuracy, one thing about the old model and new model is the fact that it was not able to detect letters that involved motion. For example, the letter 'Z' involves using the pointer finger to draw the letter. Thus, we decided to exclude the letter 'Z' because we knew the model would only work with static images.

We needed to present our project during the presentation with the whole class. This would require use to allow anyone who wants to present any ASL letter and have our model print out what letter that is. First step we had to do was figure out how we were going to access our webcam. For this we

used the opencv library. This library helped us with not only accessing the webcam, but also interpreting the image that is displayed on it so that the computer can come up with an accurate output. Essentially the way this is done is that a rectangle is made on the screen, and the user must put the letter they want to output inside of said rectangle. This helps lower the amount of background noise and possible confusions for the model. Once the letter is inside of the rectangle, openCV reads in each frame and compares that frame to what it learned from the neural network. The first question that arose, was how is our model supposed to take that inputted frame, and analyze it using the model that we trained it on. Simply loading in the model is not enough, one must pair this loaded model with the function that is interpreting our inputted image from the webcam. What was required was that we obtain the weights from our model, as well as our configuration file. At first we attempted to obtain them separately however, we soon realized that our weights were saved in an h5 file. We had to download a pb file, which contained our weights, then from this file we obtained the pbtxt file which has our configuration file. The other built in library we used were tensorflow and keras. We used their built in function load model in order to obtain our weights and configuration file. Once we have the model the process is quite straight forward, using an infinite loop so the model continuously takes in data from the webcam and runs it, creating our rectangle for the user to put the letter in, printing out the predicted letter. However there were some hurdles that we were not able to overcome. A lot of times, even with the small rectangle, there were still items in the background that were disrupting our model's ability to decipher what the inputted letter was. This is likely due to a limitation of our training and testing data only having blank backgrounds. Another factor that limited the performance of our live demo was the fact that the inputted hands were moving, because it is very hard to keep one's hand perfectly still. Unfortunately the live demo was only able to accurately identify on perfectly still images.

4.2. Societal Discussion

We live in an ableist society. ASL is unfortunately still not a widely used language despite the 250,000-500,000 people in the US who use ASL on a daily basis . The goal of this project is to make ASL a more accessible language and close the language barrier divide. In an article by Sherman Wilcox [5], he explains, "In several states, ASL is mandated by law as acceptable in fulfillment of high school foreign language graduation requirements." This aligns nicely with our goal to close the language barrier divide. Having more students learn ASL can certainly be helpful and our translator can facilitate this. Additionally, the fact that many different people know ASL indicates that our data should be reflective of the people that use ASL, in other words, all kinds of people. Lastly, this ASL translator can be useful for the children of deaf parents so we are motivated to create a simple and easy-to-use translator that anyone can use.

There has been a lot of research pushing for more people to learn ASL at an earlier age, if not to communicate with loved ones, but to allow those with disabilities to communicate easier with those around them. This research has led to ASL becoming one of the most widely taught languages in schools and universities. As a result, our goal should be to create an easy to use translator that can one day be used in school in conjunction with a curriculum. Our algorithm, data, and analysis should remain the same but the way we present our results should change. We must be able to present our results in a manner that displays ease of use, and accurate outputs. In essence, a child should be able to use and understand the translator when communicating with someone.

The main stakeholders will be the users of the translator. For example, people who are deaf or hard of hearing, and people who want to communicate with them. This can include people who may or may not know ASL. One specific stakeholder could be the children of deaf parents. These stakeholders will benefit from the ASL translator and no one will be harmed. Other stakeholders can be the creators of the translator. Those who actually code it. Other stakeholders can be those who fund the production and sale of this translator, these people, along with the coders would be the ones responsible for actually making sure the translator is effective.

The only way a stakeholder can be harmed is if the program stores the images it encounters. This can be an invasion of privacy since the identity of the person using it can be exposed. At the same time, if the translation is inaccurate, it can give misinformation. Other than that, our ASL translator should be safe to use.

When considering biases, our data used images of the right hand which could make it harder to detect letters on the left hand. At the same time, our data used images of the hands of white people. This could make it harder for a non-white person to get a correct translation of the letter they are signing.

5. Conclusion

What we have done for our project is create an improved and faster communication system between those with disabilities, and those who do not speak ASL. This project possesses the ability to help people all across the world, in other languages. The impact of our project is a more inclusive and easily accessible world for everyone where anyone can reap the benefits of being able to interact more easily and with more people. More opportunities will also be provided to those with disabilities.

6. Hypothetical Future Steps

If we were to continue working on this project, it would be good to incorporate more noise and variability into our data set so that our model could be more robust to different environmental factors such as lighting and backgrounds.

In addition, incorporating an object detection algorithm into our demo which automatically isolates the hand within the frame would be beneficial. This way we would not need the user to center their hand in the middle of the frame.

Finally, something that we wanted to do but didn't have time for was incorporating a text to speech aspect to this demo so that the computer would read out loud the letters it was seeing. However, we thought that with such low accuracy our demo was getting, this added feature would be useless at this point.

7. Reflection

After completing this project and looking back on it there are a few things that we wish we had done differently. One key error that we made in this process was trying to jump into programming way too early. We simply started running random models and seeing what worked on the fly. In doing so we wasted a lot of time guessing and checking. Instead, we should have taken our time and done more research into what other people did and build off of them. In doing so we would have had much more time to improve in the other areas that needed work.

References

[1] Paul Gavrikov. Visualkeras. *Github*, 2020. 1

[2] Dr. Kumar. Hands-on guide to sign language classification using cnn. *Analytics India Magazine*, 2020. 1, 2, 3

[3] David Lee. American sign language letters dataset. 2019. 1

[4] User tecperson. Sign language mnist. *Kaggle*, 2017. 1

[5] Sherman Wilcox. Asl as a foreign language fact sheet. *University of New Mexico*, 1991. 5

Appendix

Team contributions

Theo McArn Theo worked hard to find an adequate data set that would be useful to our CNN model. He figured out effective ways to preprocess our images, in order to obtain a higher accuracy. He helped create a solid foundation for the new model/new convolutional neural network. This resulted in a glorious 99% accuracy. Lastly, he helped troubleshoot and set up the live demo.

Jorge Sanchez Jorge helped improve the CNN model's accuracy by testing different hyper parameters. These included batch size, learning rate, and image size. At the same time, he adjusted the amount of convolutional layers as he saw fit. He also worked on obtaining the lime explainer images of the old model. Lastly, he helped develop a solid foundation for the poster.

Noah Atanda Noah helped create and improve the CNN model. He thought of ways to improve the accuracy by testing out various proposals. For example, he thought of changing the dense or dropout values. He also worked diligently on setting up the live demo feature. This involved obtaining the weights and configuration file of the model in order to use the 'opencv' detection library to analyze images from the webcam.