UNITED STATES MILITARY ACADEMY

FINAL PROJECT

CS485: APPLIED NEURAL NETWORKS

SECTION G2

MAJOR DANIEL RUIZ

By

CADET JACK SUMMERS ’22, CO A4

WEST POINT, NEW YORK

2 MAY 2022

_J.S__ MY DOCUMENTATION IDENTIFIES ALL SOURCES USED AND ASSISTANCE
RECEIVED IN COMPLETING THIS ASSIGNMENT.

_____ I DID NOT USE ANY SOURCES OR ASSISTANCE REQUIRING DOCUMENTATION IN COMPLETING THIS ASSIGNMENT.

SIGNATURE: 

Problem:

The original goal was to create a model to identify the sign language alphabet and translate each letter into a string. The string must then be sent via email to the desired user. The problem requires a convolutional neural network that can classify 26 letters and a space bar symbol. Additionally, the camera frame must split into two separate areas. One for "signing" messages and one for commands. The three commands will be "M" for message, "T" for the "to" line, and "S" for sending. After training over ten versions, each of which took several hours to train, there was not enough time to complete the original goal. Instead, the model classifies the sign language alphabet and translates each letter into a character in real-time. Ultimately the goal of the project was not to create something entirely new or groundbreaking. The project goal was to solve a complex problem by making a model from scratch to demonstrate machine learning capabilities.

There are several similar examples of machine learning models used to detect sign language letters. Some programmers created tiny datasets using images of their own hands [4]. Models that utilize images from only one person may perform well for that user, but they will likely not generalize well to other people due to hand size, shape, and color. However, there are also examples of models that utilize large databases like the datasets this project uses. These models have accuracy rates above 90%, but the developera only use static images and they do not detect in real-time [2]. Ultimately, the problem is approached in many ways. The model created in this project is unique because it was made from scratch. It is also unique because it contains the following characteristics: two large data sets, detecting in real-time, no transferer learning/ pre-trained models. There are examples of models that use different combinations of the previous characteristics. However, to the best of the author's knowledge, there is not an example that integrates all of them. Using a large dataset may make the model more generalizable to many users, even though it is not as accurate as other available models. Other models may be more accurate than this project because the data set is tailor-made to one user, or they use transfer learning with powerful pre-trained models.

Evaluation:

The original desired end state of the project was to be able to use sign language to send emails. However, too much time was used in the span of two months attempting to optimize the model and prepare the data. In turn, the final project produced a model that can translate sign language letters into characters in real-time. To measure success with the new baseline, first, the validation accuracy and training accuracy should be within five percentage points of each other and therefore not overfitting greatly from a subjective point of view. Secondly, the training and validation accuracy should be above at least 70%. The model will succeed when evaluating with separate test data if it correctly predicts at least 90% of the time. Finally, the model will be deemed a success when tested in real-time if it can correctly predict at least 75% of the alphabet via the sign language detector colab notebook.

Validation Process:

Ten pecent of the training data was set aside for validation purposes to evaluate the model. Additionally, a small separate dataset was used for evaluating the model with .evaluate(). The validation and evaluation data underwent the same normalization and resizing process as the training data. The final test involved testing in real-time. A separate colab virtual machine was used to take a live camera feed and predict what the user was signing. The detector colab notebook was initially used to detect human faces in real-time [3]. The notebook is unique because it utilizes JavaScript to allow camera input into the virtual machine [3]. Approximately 90% of the detector notebook (separate from the training notebook) was copied [3]. The final 10% was created to enable the use of a sign language translation model instead of the original human face detection model. First, the sign language model is loaded onto the vm. Then the JavaScript image is converted to an OpenCV image [3]. That image is then fed into a training generator to normalize and resize the image into the appropriate (64, 64, 3) dimension. The .predict() module then predicts the image class, outputting a np array with twenty-six 0s and a single 1. The index of 1 corresponds to the predicted class. Then, a custom decode function translates the np array prediction into a single character. Initially, the model did not perform well in real-time. Therefore, a second custom function was created to help optimize the sign language detector vm. The "mode" function takes the predictions from the ten most recent frames, then finds the most common character from the ten frames. In turn, the user must sign with a specific cadence. Meaning, that the user must sign each letter for approximately 1 second while also keeping their hand still in front of the camera. In turn, while the user stays still, the majority of the ten frames per second should be the character that the user intended.

Vectorization and Normalization:

The first versions of the sign language translation model experienced severe overfitting, and the validation accuracy topped off at approximately 50%. There was little success in optimizing the model for several days to solve the overfitting and accuracy problems. The solution involved collecting more data to expand the training set. The first step in data processing for training and evaluation involved extracting the directories into the colab workspace. After the two separate data sets were on the colab vm [1,2], a bash script combined the two datasets by first removing unwanted classes such as "nothing" and "delete." Then a bash for loop combined the two datasets into one, creating a dataset with 141,118 JPG images and 27 classes.

The next step in data preprocessing involved the use of training generators. The generators connect the training datasets on the colab vm to the training process. Additionally, they format the image data into the appropriate dimensions and augment it, artificially expanding the dataset. The original images varied in size but were greater than 200 x 300 (figure 1). The relatively large images originally increased the training time drastically. To reduce training time, a target_size of 64 x 64 was set. In turn, the final dimension of each training image was (64, 64, 3). Finally, the following commands

normalized the data: samplewise_center = True, samplewise_std_normalization = True (figure 2).

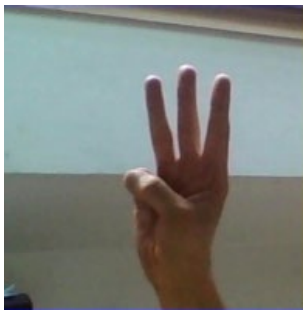

Figure 1

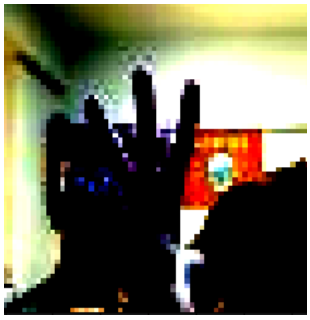

Figure 2

Model:

A naive model would make random predictions. Meaning it would guess between the 27 classes; it would obtain an accuracy rate of 1/27 = 0.037. The baseline to beat would be 3.7% when evaluating the model. The final model ended training with a training accuracy of 84% and a validation accuracy of 77%. There is a small amount of overfitting, but it has not negatively affected the model when generalizing to users outside the dataset. When evaluated with test data, the model achieved an accuracy of 100%. The sharp 33% increase may be because only one image from each class was set aside for testing purposes. Regardless, correctly predicting all 27 classes is a success. The final test involved using the sign language detector notebook. Each letter of the alphabet was signed in the view of a laptop camera. The accuracy rate was calculated in real-time as: the number of correctly predicated letters / 27. The model correctly predicted 59% of the alphabet in real-time. See figures 3-5 for an example of a correctly classifed “a”, “b”, and “c” in real time. The model could not correctly classify J, M, N, T, U, V, X, Z, and space. The model may struggle to classify J and Z because they are not meant to be static symbols. Meaning they each require a sweeping motion while signing. The motion helps humans distinguish a Z from a D and a J from an I. Due to the fact that the dataset uses images of Zs and Js at their final resting place, the images of Zs look like Ds, and Js look like upside-down Is. The model has difficulty predicting Zs and Js because it cannot factor in the sweeping motion. Additionally, Vs look very similar to Ks, and Ws look very similar to Fs. The error pertaining to the remaining letters M, N, T, U, X must be due to the fact that the validation accuracy only achieved a rate of 77%.

Figure 3

Figure 4

Figure 5

The final model utilizes a relatively large network of Conv2D layers with dropout to fight overfitting. Experimenting with the kernel size and stride helped to optimize the model [2]. Next, the output is flattened before input into the dense layers. Not many neurons were needed in the dense layers due to the fact that the model was originally overfitting. Finally, the model was compiled with categorical cross-entropy because the project solves a multiclass problem. One way to further develop the model would be to include max pooling. Additionally, converting the image data to grayscale would help make the model more generalizable by eliminating skin tone from the model. Finally, data augmentation layers could be added to the model instead of relying on the generators.

Bibliography

[1] K. MUVEZWA, “Significant (ASL) Sign Language Alphabet Dataset,” *www.kaggle.com*, 2019. https://www.kaggle.com/datasets/kuzivakwashe/significant-asl-sign-language-alphabet-dataset/code (accessed Apr. 30, 2022).

[2] D. B, “Running Kaggle Kernels with a GPU,” *kaggle.com*, 2018. https://www.kaggle.com/code/dansbecker/running-kaggle-kernels-with-a-gpu (accessed Apr. 30, 2022).

[3] T. A. Guy, “colab-webcam,” *GitHub*, Apr. 08, 2022. https://github.com/theAIGuysCode/colab-webcam (accessed Apr. 11, 2022).

[4] N. Renotte, “Real Time Sign Language Detection with Tensorflow Object Detection and Python | Deep Learning SSD,” *YouTube*. Nov. 05, 2020. [YouTube Video]. Available: https://www.youtube.com/watch?v=pDXdlXlaCco

[5] K. Team, “Keras documentation: ModelCheckpoint,” *keras.io*. https://keras.io/api/callbacks/model_checkpoint/ (accessed May 01, 2022).