Final Project - ASL Generation Model

Kelsey Hawkins

Introduction

ASL is the most used sign language in the deaf community, and regardless of the language being taught in schools, it is not a widely understood sign language. With that in mind, accessibility towards the deaf community includes communicating with people who do not understand sign language. By creating a program that can translate ASL is one milestone, however taking it to another level is a new task. Being able to generate ASL by giving a program the sentence or sequence of words you want to sign is a new challenge. With the utilization of generative models and time series models together, we can attempt to generate ASL signs to further the learning of ASL with the deaf and hearing communities.

Analysis

With the complicated nature of time series and generative models, a thorough approach is taken to prepare and format the dataset. Because ASL has signs that require motion, the dataset has to consist of sequential images that represent a time series. The original dataset found off of kaggle had one folder containing videos of people signing words and letters, adding up to over 2000 words and the alphabet.

To begin, the dataset has to be separated into the respective classes with folders representing each word. For the sake of this experiment, a smaller dataset is utilized with about 400 words and letters. The dataset contained a JSON file that acts as a dictionary for the video ID and correponding class it belongs to. This file also has information pertaining to bounding boxes, and the length of video. Each video is cropped to the bounding box coordinates to ensure the images are focusing on the subject and not the background. Then, the images are resized to a standard 256x256x3 shape.

Once the dataset has been reorganized into the respective classes and videos, a python package, CV2 is utilized to parse the videos into frame segments. Each video is parsed into 16 frames as each video is only three to five seconds long, taking four frames per second. Once the images have been created, teh dataset is ready to be analyzed. Considering the amount of data that

is present in this dataset, there are only two or three videos per class, which adds up to twenty thousand images. There is no class imbalance as there is always more than one video per class so certain classes will not have more weight than others.

In order to read in this complex dataset, a Video Generator class is created to handle the dataset reading in and standardizing for the model to intake. This class is able to take in many parameters pertaining to the labels, batch size, max frames, channels, classes, and what happens to each batch on the training epochs since it is handling time series data the images have to stay in order while staying in batches.

Figure 1: "Sample Sequence"

Methods

The model built is a combination of a Conditional GAN, and LSTM. Because a Conditional GAN consists of a generator and discriminator, both will contain LSTM to capture the long short term memory property of a recurrent neural network to preserve the time series of the images. The generator is also trained on labels and noise so the model can create predictions given noise and a label to generate a new image for. In the creation of this model, many versions of the model were created, from extremely basic generative models without a time series, to a fully working and tuned model with LSTM layers and conditional generative properties.

To begin, the videos are preprocessed into a dataset. The batch size is set to two for memory purposes, the number of classes is exactly 408 as well as the latent dimension, and the frames per video is 16. Videos that are longer or shorter than the required 16 frames are either cropped or the last image is duplicated to fill in the missing frames.

The conditional GAN model is conistent of a generator and discriminator. Both of which have their own loss functions that consist of binary cross entropy that are assessing the fake and real inputs that are generated from the generator, compared to the real input.

The generator itself takes in a label and noise. They are combined to account for the conditional part of the GAN. The model is consistent of a dense layer then a reshaping layer to ensure the images are flattened to be fed into the ConvLSTM layer as it is a 2D layer. The output of that layer is 128 channels which is normalized with BatchNorm and fed into another convLSTM layer with an output of 64, then normalized again. Once that is done, the images need to be recreated through a Transpose Convolution series. Utilizing a Time Distributed layer with conv2DTranspose layer inside, the images are upscaled from 64 to 256 to contain the original

image input shape of 256x256x3. Once complete, the generator is now able to take in noise and a label and be able to output a batch of 16 images that represent the 'video'.

The discriminator takes in the sequence of images and is outputting a predicted probability of the image being fake or real. This network is simpler, which is taking the image and downscaling it utilizing convolutional layers so the LSTM can take in the flattened image with 128 channels. There is only one LSTM layer which is then flattened and returned with a dense layer with an output of one, being the score of the image.

Once the two models are built, they can be utilized together to represent the conditional GAN. One training step, or epoch, will be discussed. The first step is to generate the random noise that is inputted into the generator. This is utilizing the random module from tensorflow to ensure it is a tensor of random values, then a label is one hot encoded for the generator as well. The generator is then given those two values and an output of a generated video is created. The discriminator then takes in the real video for that label and outputs the real predictions. The discriminator is again fed the generated video which outputs the fake predictions. The loss metrics are then calculated for the discriminator for the real and fake inputs, then added together to get the total loss. The generator also gets the loss from taking the fake predictions given by the discriminator. Once the loss has been calculated for both models, the networks are backpropagated to update the weights, biases, and other hyperparameters pertaining to the deep neural networks. This step is repeated for each training step, or epoch.

The conditional Gan is optimized with Adam, and loss functions are for the generator and discrimniator respectively with thier custom loss functions.

In order to maximize the model's output and ability to generate realistic images, a tuner will be utilized for the batch size, and learning rate. The Keras tuner, Hyperband is utilized, and the objective is to minimize the generator loss. The tuner is trained for twenty epochs and a hyperband iteration of 2. Callbacks including early stopping and model checkpoints are utilized and loss functions are monitored. The best model is then saved and the weights of both generator and discriminator and overall model are saved.

Results

The output of the model is not as expected. Through rigorous testing and tuning, the model did not succeed in generating ASL signs. In the following images, the untuned and tuned model outputted very different images for the same given label, or known as the ASL word "Tired".

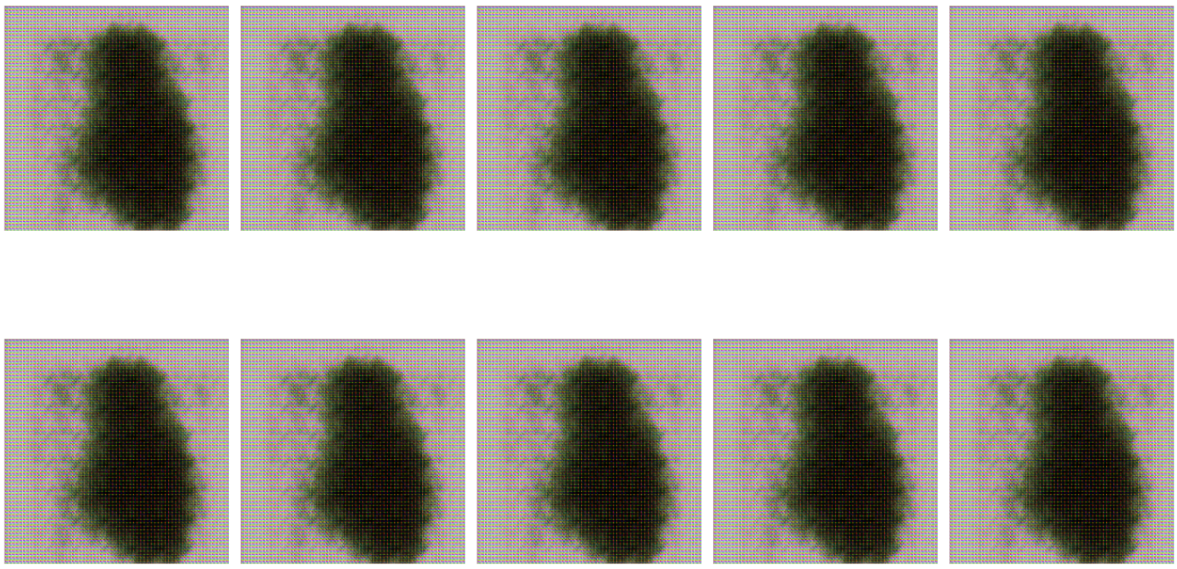

Figure 2: "'Tired' - Untuned Model

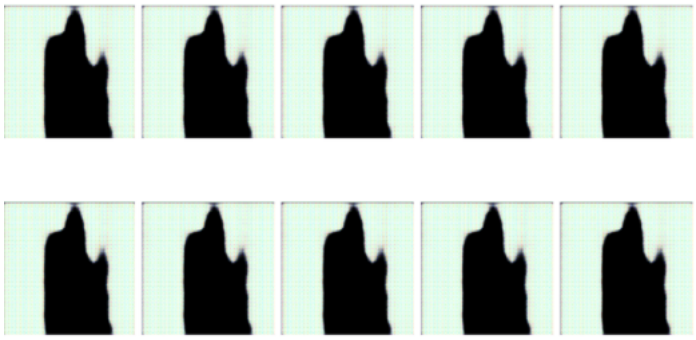

Figure 3: "'Tired' - Tuned Model

With those two predictions, each image going from top left to bottom right, it should be a sequential time series, however each image is the exact same image. The next issue that is very apparent is the fact that the people or subject are a black blob. I believe this is because the model mistaked the person as the only subject from the image and blurred them all together. Since it was a colored image, the three channels made everything more complex. I believe the model has potential but more time invested is needed. To analyze why this is happening, the discriminator may be too strong in its criticisms of the generator's outputs. Since there was an output that represented the shaddow of a person, I believe the generator and discriminator learned that the outline of the person was the realest representation of the training data. This is not great for this case, however the convolutaional neural network must have found that

to be the best feature to take out of the images it was given. Another issue that I believe caused the images to be the same images for all 16 frames is an issue with the dataset being fed into the recurrent neural network in batches. Since each batch contains the images in their sequential state, there might be something wrong with the model taking in those sequences of images. The model utilizing a ConvLSTM was a generic solution to utilizing time series data in deep learning, however it may not be the best model for this problem. However, given the time frame given to complete this project, thte current architecture is sufficient, however does not generate a sequence of images to represent a video of a ASL word signed.

Through the training, the generator was too strong compared to the discriminator. The loss averaged around 8 and the loss of the discriminator averaged around 1. The model that was tuned had the learning rate values between 1e-3 and 1e-6. This allows for fine tuning and wiggle room for the learning rate to change through tuning. To account for the discriminator being too weak for the generator, I wanted to increase the learning rate for the discriminator itself. This makes sense that the images are just black blobs, not just the dataset being imbanced is causing it, but the generator is too strong for the discriminator thus causing the images to be not learning from the discriminator labeling images as fake instead of thinking all of them are real. On average, the generator had a loss value of around 12.5, and the discriminator had a loss value of around 1.5. This is clear that the generator is much too strong and the discriminator is too confident in classifying the images as fake. From these issues, I decided to edit the learning rates so the discriminator had a smaller learning rate than the generator so the generator can catch up to the quick learning of the discriminator. Due to the simpler architecture, this makes intuitive sense since the discriminator has a smaller dimension to learn from the inputs, compared to the robust archetecture of the generator.

For the model loss and accuracy, unfortunately the code I was debugging did not want to allow me to print out the epoch training validation loss for the generator and discriminator. However, through training, the generator was much weaker than the discriminator with loss values multitudes higher than the discriminator. This shows that the discriminator is too strong for the generator and its confidence is much too high for classifying images as false.

Model and Results of Version 2, New Dataset Same Architecture

Due to having extra time, a new C-GAN was created with the ASL MNIST dataset. The model's goal is the same as the goal defined above, of generating signs for ASL words. Since this dataset only contains letters, the concept of fingerspelling will be used, for spelling out Names or proper nouns. The model consists of a generator and discriminator that is trained on labels and noise, with the classic architecture used for each model. No recurrent neural networks are used here and single outputs are created instead of sequences of outputs. To create the fingerspelled word, a list of letters are given to a custom method that will create predictions using the generator for each letter, creating the word spelled out in ASL. This model performed much better than the model trained on the dataset of videos due to the

nature of the dataset setup. The other model had a huge issue of having the dataset being extremely sparse and imbalanced with only two videos per word, and hundreds of classes which had no possibility of creating a strong model. It makes sense why the model failed and gave inspiration to complete this model, with the ASL MNIST dataset as a backbone for the dataset. It is a bit more challenging than the original MNIST dataset because it contains hands, so it is interesting how the model handles generating hand shapes. When playing around with this model, it became a bit more promising than the previous one. At first the basic model had loss values for the generator around 3.5 and for the discriminator around 0.1. This means that the generator is much stronger than the discriminator and needs more tuning and playing around with the model architecture. The first change I made was adding the learning rate scheuler and smooth the loss function of the discriminator to slow down its learning so that generator can catch up since the loss is multitudes higher and the ouptuts were quite bad. Training epochs were also increased drastically from 50 to 200. With these changes, the generator and discriminator were more balanced in their loss metrics over the epochs.

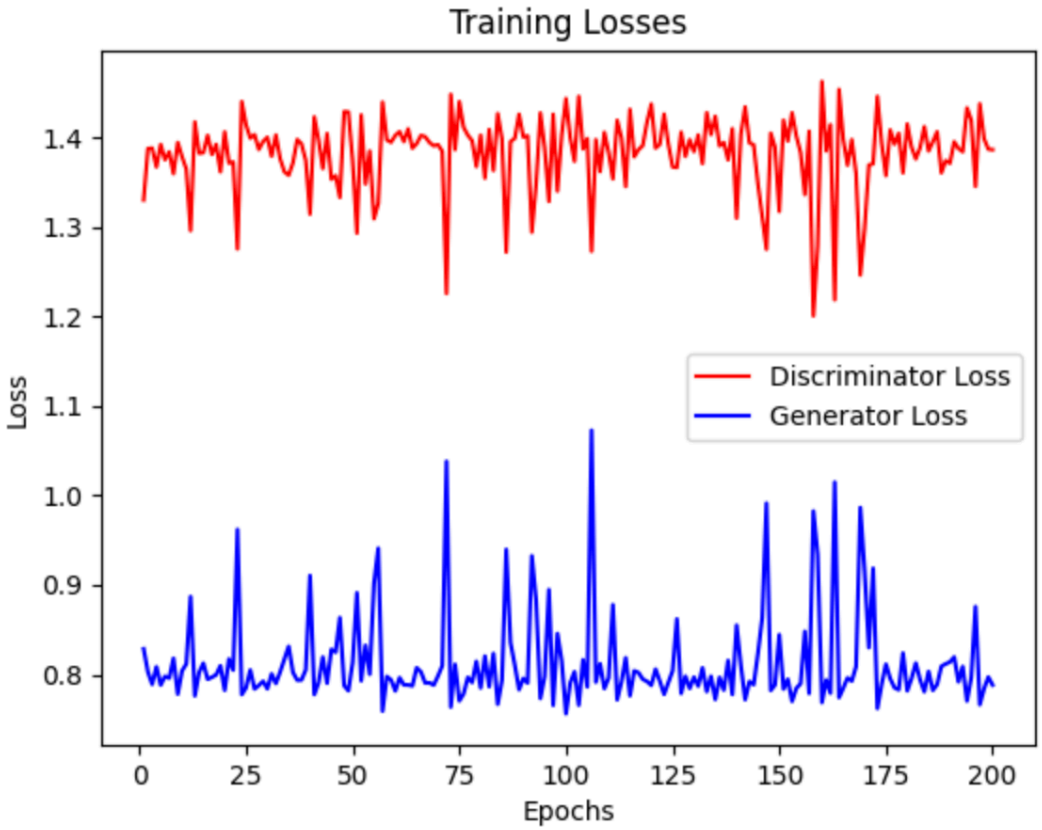

Figure 4: “Training Losses - ASL MNIST Dataset”

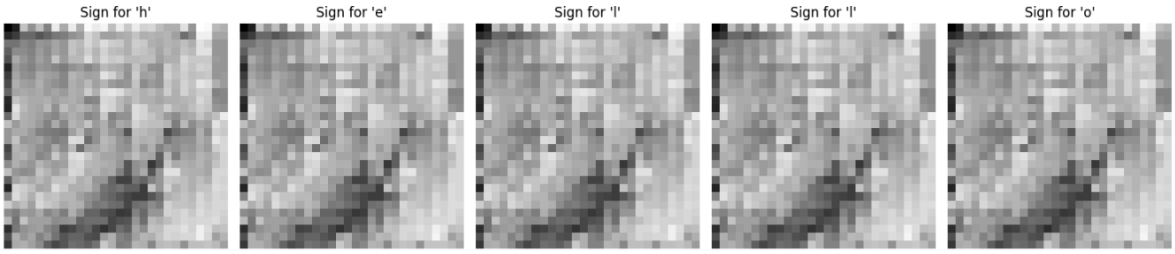

Figure 5: “Predictions for the word”Hello” - fingerspelled”

Reflection

This is one of the most challenging assignment or project I have had in my time. It not only pushed my boundaries as a person and academically, but the debugging process and completion of the model made me interested to improve upon this model. In the future, I would start even smaller than I did and not try to do a task that has barely any literature on in the data science community. The task itself is extremely difficult and barely achievable, but I was able to output some data that is vaguely representative of a working model. In the future, I would give myself more than a month to do the project, and maybe try different architechtures that are meant for image segmentation as well as generation. In the coming months, I will be taking this dataset and experimenting with utilizing Image Segmentation in pair with recurrent neural networks, or diffusion models to be able to generate videos while keeping the ASL language readable and usable.