MAIS 202 Deliverable 3: ASL Alphabet Classifier

Cheng Lin

1. Final training results

I trained the SVM model from deliverable 2 on half of the available data using a VM provided by John. As expected, increasing the training dataset decreased the model accuracy slightly (it dropped to 94%). However, upon testing the model with photos of my own hand, I concluded that it wasn't performing well enough and was most likely overfitted. In particular, the model consistently misclassified close to all ASL letters I fed it, even after my photos were edited using OpenCV to resemble the training data.

The SVM approach assumes that HOG is the best classification feature for this problem. Because the SVM model was underperforming, I decided to switch to a convolutional neural network, as proposed in my first deliverable.

I began by preprocessing the images; I normalized their RGB values, formatted them as a 3D tensor, and fed them to a Pytorch Dataloader. For my labels, I transformed them with a sklearn LabelEncoder object like I did with the SVM model.

In my first architecture attempt, I implemented an LeNet-5 CNN. I chose this architecture because my images are formatted similarly and the classification is relatively simple. I based a lot of Pytorch code on an existing LeNet-5 implementation.

However, the LeNet was still overfitting to the training images and not picking out versatile enough features. Although the training accuracy was around 97%, when I tested it on pictures of my own hand, it didn't classify any correctly.

In my second attempt (with a lot of counsel from John), I added data augmentation to the image preprocessing stage. Specifically, I read in the images as a PIL image; performed a random change on the picture's brightness, contrast, saturation, and hue; performed a random crop; and performed a random rotation. I also switched from an LeNet to a VGG architecture, as the data augmentation would cause my images to be more varied, and the features more complicated. (As well, the LeNet was not training well with the data augmentations.) I tried to implemented an 11-layer CNN based on this github repository, but in the end chose to just import Pytorch's VGG without pretrained weights.

The 11-layer VGG I trained had an accuracy of 99.3% when tested on the validation set. When I fed it pictures of my own hand, it classified roughly 75% correctly. The figure below (Figure 1) is the confusion matrix for one 100-image batch in the validation set.

Because I have a decently-performing model and a short time-frame until the demo, I'll begin focusing on building the web app (see below). Moving forward, I could try to improve this model by training it with aggressive data augmentation techniques (more colour/contrast/brightness changes, more rotation, more cropping). This would force the CNN to pick out even better features.

MAIS 202 Deliverable 3: ASL Alphabet Classifier

Cheng Lin

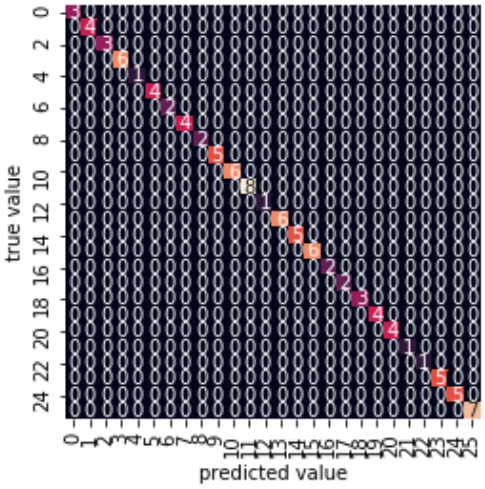

Fig. 1: confusion matrix of trained VGG-11 model

2. Final demonstration proposal

I plan on integrating my model into a web app. I want to use a platform that allows me to build this app with just Python to keep things simple. Based on online posts, I will use Flask's framework and Heroku's platform to build and host my app. Even though I don't have experience with either technologies, there are a lot of online resources I can refer to.

I anticipate that the biggest challenge in building my app will be creating the architecture to accept user input. To begin, I plan on letting the user upload a photo of an ASL letter to classify. For simplicity, this photo would need to be cropped, but I can later implement a sliding window approach to locate the ASL letter. If this works out, I can extend the web app by allowing user input directly from the laptop's webcam via the click of a button. However, I'm unsure of how to dynamically display the webcam's input until the user decides to take this photo, so I think implementing this feature will be the most challenging part. A rough sketch of the last stage of my web app can be found below. The red square outlining the letter can be removed if my sliding window approach works.