EECS 605 Final Project: Hand Gesture Recognition Model Training - Reproducibility Memo

This document provides an outline of the steps required to reproduce the training of the hand gesture recognition model that is then deployed as both a Heroku web app and as a DeepLens project. All necessary files should be available in the GitHub repository: https://github.com/dmanwill/Hand-Gesture-Recognition-Dataset-and-CNN-Model. The following sections detail the steps for collecting the dataset and then training the model.

Dataset Collection:

Training a deep-learning model from scratch to recognize something like hand gestures often takes thousands of images of training data and that data needs to be properly processed and curated. I had thought it would be simple to find a dataset online to train a model on and then deploy for the Heroku app and DeepLens project. However, somewhat to my surprise, there didn't appear to be a popular and easily available dataset of regular camera images of different hand gestures. I found a couple of datasets with either near infrared images where the hand stood out well against any background or of RGB-D video with depth information in the videos, but none of the available datasets I found appeared to be of a proper format to do well for regular RGB camera images such as would be processed on the DeepLens device. So, using the information learned in the EECS 605 class, I created a DeepLens project to capture images every 0.5 seconds, preprocess the images by cropping, converting to grayscale, resizing, and then upload them to an S3 bucket. Using this project, I created my own dataset of roughly 3,000 images to be used to train my deep-learning model. A few example processed images are shown below

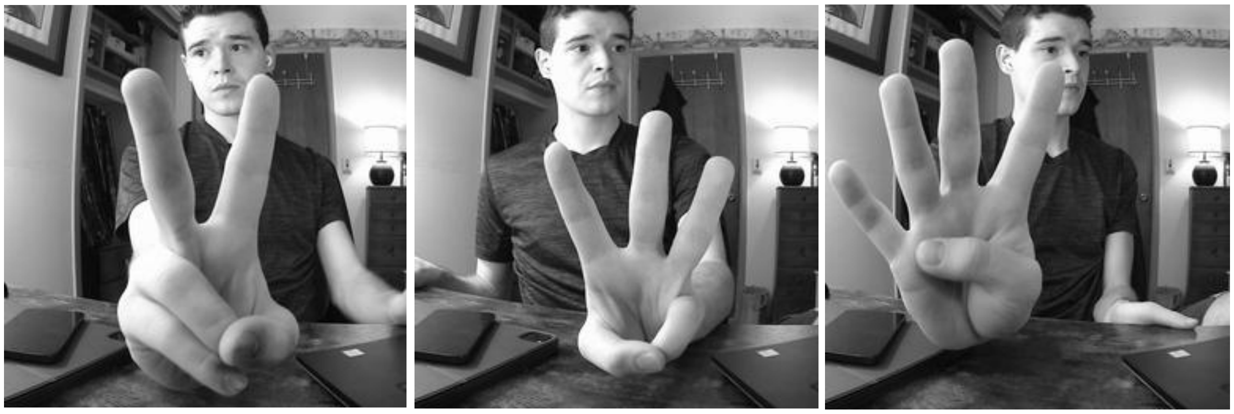


Figure 1: Example processed dataset images

Details of the steps required to collect this dataset are provided below:

Step 1: Create an S3 bucket for the data to be uploaded to

- Created an S3 bucket named "dmanwill-project-dataset"

Step 2: Create a Lambda function with the code to run on the DeepLens device to collect the data

- Created a Lambda function called "data-collection" with python 3.7 code environment
- Wrote and uploaded "data_collection.py" to provide a script for capturing images every 0.5 seconds, preprocess the images by cropping, converting to grayscale, resizing, and then upload them to the S3 bucket
- Uploaded "data_collection.py" to the Lambda function

- Published the Lambda function so that the DeepLens portal can access it

Step 3: Create an AWS DeepLens project

- Created an AWS DeepLens project called “dataset-200” to collect 200 images at a time
- Linked the data-collection Lambda function as the function for the project
- Linked a dummy model (a previous handwritten digits ONNX model) as the model. This was not used but to create a DeepLens project, you must link a model

Step 4: Deploy the DeepLens project to collect the data

- Deployed the dataset-200 project to my DeepLens device
- Opened the project stream to keep track of when each image was taken and when to switch hand gestures
- Opened the project log to make sure images were being collected properly and track how many had been collected

Step 5: Download the images from the S3 bucket

- Run the code in the “data_downloading.ipynb” notebook to download the images collected on one run of the data collection project to a local directory

Step 6: Deploy the model again as many times as needed till the needed number of images is collected

- Deployed the same project again and again many times to collect several thousand images

Discussion about the Dataset:

At first, I ran the project over and over to collect 2,000 images of 5 different hand gestures, the number of fingers held up. I then tried training a CNN deep-learning model and it almost immediately learned to perfectly classify every image in the training and validation set but when I tried testing with several cell phone images, it was completely wrong every time. I realized there must be something wrong with the dataset and eventually realized that I had captured each set of 200 images doing the same hand gesture over and over. I had moved my hand around to be at different distances and different orientations and I had even used both my right and left hand on different runs but I had sat with my chair in the same position throughout each run and then moved slightly for the next run of 200 images. The model then only had to learn to classify based on where my chair was and then correlate that to which kind of hand gesture I was making during that run. This was far easier to learn than to recognize the complex hand gestures, so the model easily learned to classify every training and validation image correctly, but it never actually learned to recognize the hand gestures. I then had to rerun everything to collect another 2,000 images but this time moving around my chair position as well as my hand position with each collected image. After training the model on this new data, it actually learned to recognize the hand gestures but seemed to not quite have enough data and maybe still slightly fitting to some features that were distinct between runs of collecting images of the same hand gesture. So, I collected another roughly 800 images with switching hand gestures every few images and added this to the dataset. This was more work to separate into the proper folders, but it definitely eliminated the fitting to anything else that was distinct between runs. All was not for naught with the original 2,000 images collected as these made for an excellent test dataset. I took about 40 images from each hand gesture to add to the training/validation dataset and then used the remaining images for a final test dataset. This gave roughly 3,000 training/validation images and 1,800 remaining test images. The model then had very good performance on the training/validation dataset and nearly equally good performance on the final test dataset.

Training the Model:

Once everything was figured out about collecting, processing, and properly curating the dataset, the deep-learning model training was fairly straightforward. The code for loading in the data, setting up the proper data structures, splitting into training and validation, defining the CNN model structure and training the model is available in the Jupyter notebook in the model folder. After testing several different models, I found that a CNN based approach with 4 convolutional layers and one dense layer followed by the final output layer gave the best performance on my dataset. The convolution layers use filter widths of 16, 16, 32, and 32 and have (3,3) convolution kernels. Each layer has ReLU activation except for the final output layer which has a softmax activation. The model was trained with the Adam optimizer with its default learning rate and was trained until there was an early-stopping callback on the validation loss. The model was trained using TensorFlow's Keras and then exported to an ONNX model. The code for creating the model and the training and test results are available in the same Jupyter notebook in the model folder. Below are plots of the training/validation loss and the training/validation accuracy:

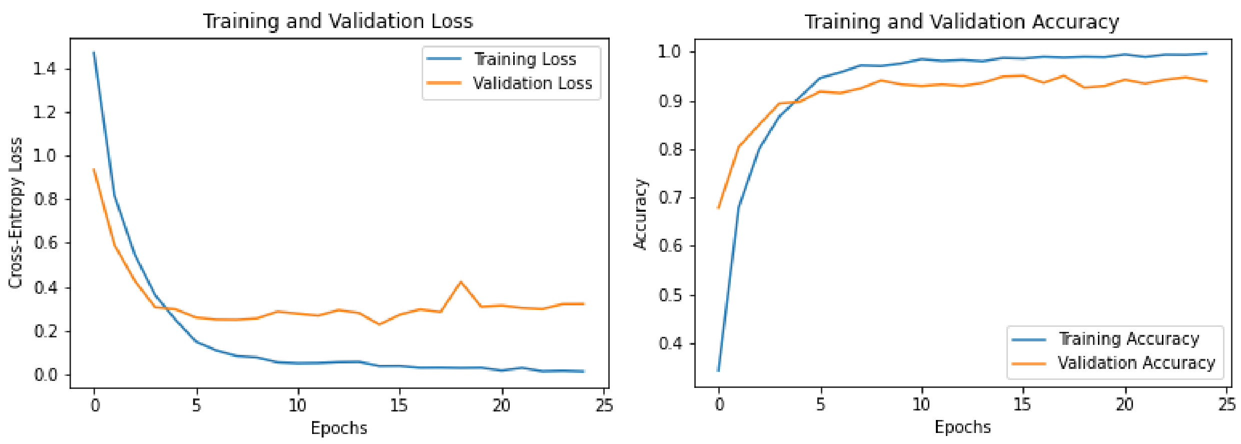
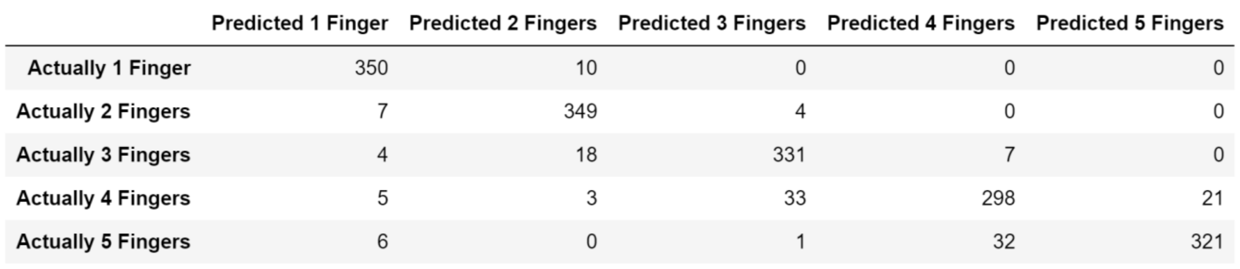


Figure 2: Training/Validation loss and accuracy plots

The training accuracy reached nearly 100% while the best validation accuracy was 95% with a test accuracy of 92%. It is likely that with a larger dataset, the validation and test accuracy could be increased slightly but this model definitely shows it can classify the 5 hand gestures very well. Below is a confusion matrix showing how the model performed on each class.



	Predicted 1 Finger	Predicted 2 Fingers	Predicted 3 Fingers	Predicted 4 Fingers	Predicted 5 Fingers
Actually 1 Finger	350	10	0	0	0
Actually 2 Fingers	7	349	4	0	0
Actually 3 Fingers	4	18	331	7	0
Actually 4 Fingers	5	3	33	298	21
Actually 5 Fingers	6	0	1	32	321

Figure 4: Model confusion matrix on the 5 classes

Overall, the model performed quite well across all classes. It looks like it had the most trouble distinguishing between 4s and 5s and this is somewhat understandable given that they have the same number of fingers held up and only distinct with a thumb sticking out. The model also occasionally confused any number of fingers off by 1 from each other. It also looks like the model equally confused any random gestures as a 1. Below is shown a few examples of test images that were incorrectly classified:

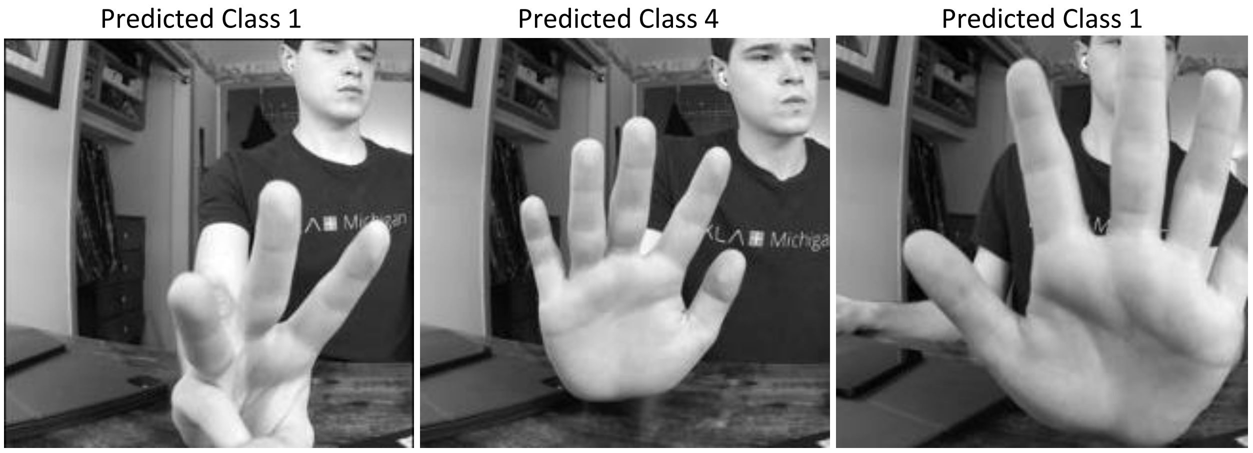


Figure 4: Example missclassified test images

A couple of these images make sense that they would be difficult for the model to classify correctly due to a weird angle or being cut off. However, the middle image doesn't appear to have any reason why it should have been misclassified. After reviewing these and other misclassified images, it doesn't appear that there is any consistent reason why certain images were misclassified but with more training data, the model accuracy could likely be improved and better classify some of these images.

One important note about this model. It is very possible that this model is overfit to my skin tone and may not work well or even at all for someone with a significantly different skin tone. Other factors such as significantly different lighting conditions may cause the model to fail. There may be other factors that could cause this model to fail and it certainly is not robust enough for a real-world application. More training data with varied skin tones, lighting conditions, and any other relevant factors would be required to make this model more robust. The preprocessing of the images is also dependent on the formatting of the original image. For testing images taken from my iPhone or iPad, I had to set the camera format to have a .jpg file output to be able to work.

This model was trained with the specific goal of performing well while deployed on the DeepLens device. The DeepLens camera has a fish-eye effect that distorts the sides of the images. However, with center cropping the images, this did not become an issue for this model. With the model trained on data collected directly from the DeepLens device, this allows the model to perform very well for its purpose of being deployed on the DeepLens device. Details about deployment of the model for both the Heroku web app and the final DeepLens project can be found in the other accompanying reproducibility memo document.