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

This document provides an outline of the steps required to reproduce the deployment of the hand gesture recognition model to 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 deploying the Heroku web app and the final DeepLens project.

Heroku Web App:

To create a working Heroku web app, there are several steps that need to be followed to setup all the necessary AWS processing that the code on the Heroku app will call when it is run. Then the Heroku app deployment is fairly straightforward. Here we will assume that everyone is familiar with the steps learning in the EECS 605 class so some lower-level details may be glossed over.

Step 1: Create an S3 bucket for the project and upload necessary files

- Created an S3 bucket named "heroku-deployment"
- Uploaded the final best trained model file "my_model.onnx"
- Uploaded the zipped lambda function code files

Step 2: Create a lambda function, link its code, and change the function configuration

- Created a lambda function called "heroku_deployment" with python 3.6 code selected
- Linked the code uploaded as a zip file to the S3 bucket to this lambda function
- Changed the permissions to give it full S3 access
- Changed the configuration to give it a 42 second timeout and 500 MB of maximum memory (the timeout far exceeds normal runtime but the 500MB of maximum memory was required because of how large image files are processed in the code).

Step 3: Create a password for the Jupyter notebook so the Heroku web app remains restricted to those with the password, thus maintaining the security of the AWS credential information

- Ran `jupyter notebook password`
- Followed the prompts to set password for Jupyter Notebook on your local machine
- Copied the hashed password from the `jupyter_notebook_config.json` file into the appropriate line in the Dockerfile

Step 4: Create a Docker environment and push it to a Heroku web app!

- Installed Heroku on my command line
    - `pip install heroku`
    - `heroku login`
    - `heroku container:login`
- Created a Docker environment and pushed it to my Heroku app using heroku command line tools (must have Docker installed). Called my app eecs605test.
    - `heroku container:push web -a eecs605test`
    - `heroku container:release web -a eecs605test`
    - See https://devcenter.heroku.com/articles/container-registry-and-runtime for more details on these steps

Step 5: Set AWS credentials as Heroku app environment variables

- Logged into my Heroku dashboard and selected the eecs605test app
- In the settings, added two Config Vars matching my AWS credentials
    - AWS_ACCESS_KEY_ID
    - AWS_SECRET_ACCESS_KEY

Step 6: Test to see that it's working!

- Went to http://eecs605test.herokuapp.com
- Typed in the password I set for the Jupyter notebook
- Opened "Project Notebook.ipynb"
- Ran the first cell and followed the steps to select and uploaded one of the images from the zPhone_Test_Data folder from the dataset
- Should upload to AWS and come back with a proper prediction of the number of fingers!

DeepLens Model Deployment:

To create a working DeepLens project there are necessary steps for setting up the device, connecting it to a local computer, setting a project stream with the necessary certifications, and installing necessary python packages on the DeepLens device. These instructions will assume all of those preliminary steps covered in EECS 605 labs have been done and that you are ready simply deploy a new project.

Step 1: Create a Lambda function and upload the code for the final project model

- Created a Lambda function called "project-run" with python 3.7 code environment
- Uploaded the zipped code folder from the Final Model Deployment folder
- Published the Lambda function so that the DeepLens project can see it

Step 2: Create an S3 bucket and upload the saved ONNX model

- Created an S3 bucket called "deeplens-project-model"
- Uploaded "my_model.onnx" to this bucket

Step 3: Create a new DeepLens model

- Created a new DeepLens model called "project-model" by importing the ONNX model that had been uploaded to the deeplens-project-model S3 bucket

Step 4: Create a new DeepLens project with the Lambda function and the new DeepLens model

- Created a new DeepLens project called "final-project" and linked the project-run Lambda function and the project-model ONNX model to the project

Step 5: Deploy the project and see it correctly classifying the number of fingers held up!!

- Deployed the project to the DeepLens device
- Opened the project stream in my browser and recorded a video of it working!

A video demonstration of the Heroku web app and the DeepLens project working is provided in this Project Documentation folder.