Capstone Final Project Report

Nikhil Korlipara, netID: nk2742

1. Problem Statement

In this project, I worked on the state prediction of a 4-fingered robot provided by RGBD. I implemented a simple supervised learning algorithm that imputed the RGBD images (3 different camera angles) of the robotic hand from a top view and outputted the positions of the tip of each finger. The objective of this project was to achieve as low of an RMSE score as possible and, in turn, achieve as high an accuracy as possible.

2. Methodology

After downloading the test and train data provided in Kaggle onto my hard drive, I used the *lazyload* function that I created, which provided access to both the train and test data, and did not cause the RAM to crash. This data was normalized, reshaped as NumPy arrays, and saved as test and train data files before being processed by the model.

```
data_train = lazyload('./Downloads/Capstone_Project/lazydata/', isTrain = True)
data_test = lazyload('./Downloads/Capstone_Project/lazydata/', isTrain = False)
```

I then apply the processing class on the test and train data files, *norm_data*, converting the tensor to the shape of an array using *conv_tensor_to_array* and applying *norm_depth*, *norm_image*, a*dd_depth_to_img*, and *data_reshaping* to further normalize the image and depth arrays. I also call garbage collection here, as my RAM kept crashing in this part initially.

```
gc.collect()
img0_array_test, img1_array_test, img2_array_test, depth_array_test, field_id_array =
dp.conv_tensor_to_array(data=data_test, isTrain=False)
img0_array_train, img1_array_train, img2_array_train, depth_array_train, y_array =
dp.conv_tensor_to_array(data=data_train, isTrain=True)
depth_array_test = dp.norm_depth(depth_array_test)

img0_array_test = dp.norm_image(img0_array_test)
img0_array_test = dp.add_depth_to_img(depth_array_test, img0_array_test)
img0_array_test = dp.data_reshaping(img0_array_test)
```

gc.collect()

(I do the same for image1array and imag2 array, and also do the same for train)

testX_img0 = [img0_array_test, field_id_array]

testX_img1 = [img1_array_test, field_id_array]

testX_img2 = [img2_array_test, field_id_array]

train_img0 = [img0_array_train, y_array]

train_img1 = [img1_array_train, y_array]

train_img2 = [img2_array_train, y_array]

I then use the dump function under joblib, which persists an arbitrary Python object, here ***testX_img0, etc.,*** into one file, here ***preprocessed_test_img0.joblib', etc.***

dump(testX_img0, 'preprocessed_test_img0.joblib')

dump(testX_img1, 'preprocessed_test_img1.joblib')

dump(testX_img2, 'preprocessed_test_img2.joblib')

dump(train_img0, 'preprocessed_train_data0.joblib')

dump(train_img1, 'preprocessed_train_data1.joblib')

dump(train_img2, 'preprocessed_train_data2.joblib')

Lastly, I call the *mainmethod* of *test_CNN()* on the train data and using specifically using resnet34, I train the model and then send the data to a CSV file using *submit* of *Submission()*. To define Resnet34, I implemented my BasicBlock class within.

[device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

print(torch.cuda.is_available())

torch.cuda.empty_cache()

cnn_model = test_CNN()

cnn_model_0 = cnn_model.mainmethod(loadname = 'preprocessed_train_data0.joblib',

pre_trained_model = None)

model_scripted = torch.jit.script(cnn_model_0) # Export to TorchScript

model_scripted.save('res34_pretrained_model_0.pt')

sub = Submission()

df = sub.submit(filename = 'preprocessed_test_img0.joblib', modelname =

'res34_pretrained_model_0.pt')]*and do the same with 3 images combined

What interested me was managing large amounts of data, transforming it, and identifying parts of the image. I also had to deal with introducing depth into the model and its use in recognizing/identifying the image and improving CNN accuracy.

3. Results and Interpretation

For Epoch=11, Res34 and Img0, RSME=0.02222

For Epoch=11, Res34 and Combined on Img0, RSME=0.17161

For Epoch=11, Res34 and Combined on Img2, RSME=0.01325

Combining all of the images and testing on Img2 provides me with the best RSME value, regardless of the epoch–I found that it is actually better than for a single image, as I tested on Img0 for the first. Additionally, increasing the epoch does not decrease the RSME value after a certain threshold due to overfitting.

4. Improvements for the Future Work

For future work, I would prefer to run these same algorithms on hardware that supports the CUDA library. I would also like to test this CNN using other Resnets aside from Resnet34, and see whether they have better performance.

5. Works Cited

https://campuspro-uploads.s3.us-west-2.amazonaws.com/6c251796-3233-438a-8cca-69b700b79782/70a59545-1ca7-4e1e-85f6-c78ba25d63ab/sample_submission.csv

https://campuspro-uploads.s3.us-west-2.amazonaws.com/6c251796-3233-438a-8cca-69b700b79782/aa163af3-7ff3-4ff5-afc2-df5915d71152/submission.py

https://www.analyticsvidhya.com/blog/2021/09/torch-dataset-and-dataloader-early-loading-of-data/