ELEC 475 Lab 5 (Final Project)

William Dormer, 20176544

What model did you use and why did you select it?

The first step to choosing a model is to understand the problem. We have a set of RGB colour images that have (x,y) positions representing the labels of where the nose appears in the image. Thus the input to the model has to be the colour images, and the output should have 2 values, representing that x and y position.

Properties that we want in the model:

1. Translation Invariance (we don't care where the image of the nose appears in the image), but it still has to have some concept of where in the image the nose was to be able to predict its location.
2. Scale Invariance (there are images of multiple scales with bigger or smaller noses)
3. It should work regardless of whether the pet is a cat or dog
4. It should have a fast inference time
5. Reasonable train time
    a. not much more than the resnet18 that we used, 11.7 million params
6. It should have a high accuracy (low average euclidean distance of predictions to labels)

It could be useful to harness a pre-trained model to take advantage of the low level features that have already been learned, thus reducing the training time, so first I'll look at pytorch's pre-trained models to see if any can be fit to the task at hand:

Some notable ones:

- ResNet (used in the previous lab)
    - familiar, would probably perform pretty well.
- VisionTransformer or other attention based solutions
    - Even the small ones have 8 times as many parameters as our resnet one. Although I could simply freeze the weights
    - would be fun to play around with.
- VGG (used in previous labs)
- Spatial Transformer Networks
- Keypoint detection networks like OpenPose
    - Keypoint R-CNN is available in pytorch
- Hourglass networks
- YOLO modification

I first experimented with using a vision transformer because I thought they were interesting and wanted to experiment with the technology. So I modified pytorch's smallest vision transformer with the final layer to regress a 2 dimensional vector. This did not yield promising results. I was having problems getting it to overfit to even a small training set, and it was unclear if I needed far more training time, or had a bug in implementation.

I then decided to take a step back and use a smaller more familiar model, resnet18. This worked very well. Thus, my final implementation uses resnet18 with the default weights, fine tuned for regression of the two dimensional position of the nose. To enable this, I had to modify the final FC layer to output 2 instead of the default.

link to the base model:
https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html

I'm aware that I could have implemented some of the networks we discussed in class for keypoint detection, but I decided to use the simplest approach first, and it achieved good results.

Training information:

Hyperparameters:

Learning rate: 0.001
epochs: 50
batch size: 64
optimizer: Adam
scheduler: None
Transforms:

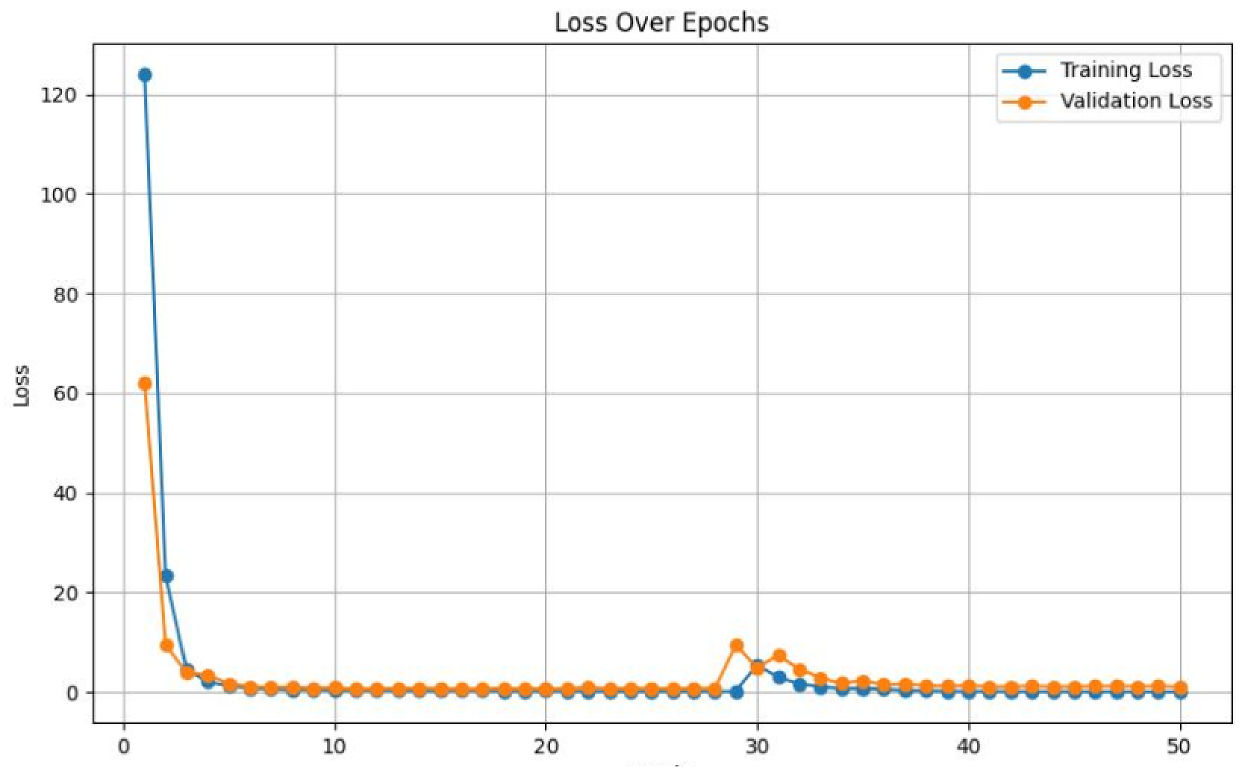
1. Resize image to 3x224x224
2. Normalize batch to mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]

Hardware:

All training was done on my desktop's GTX 1070 GPU.

Loss Plot:

I split the training set into train and validation, leaving the test set completely isolated.



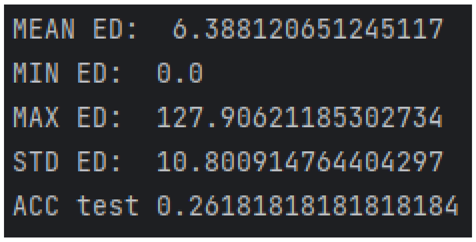
Interesting behaviour at around 28 epochs. The top model saved was before that point.

Train Time:

27 min for 30 epochs

Localization Accuracy Statistics:

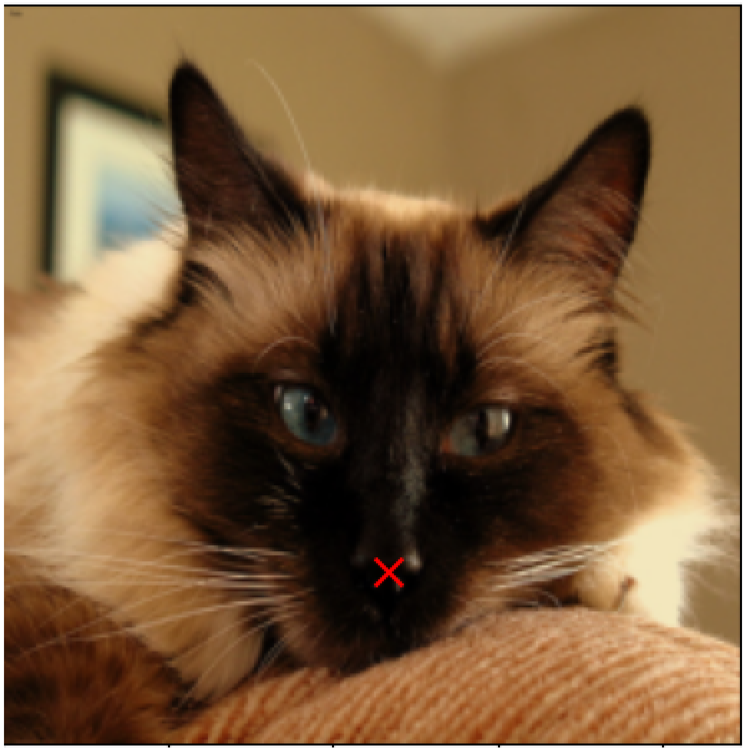
on test set:

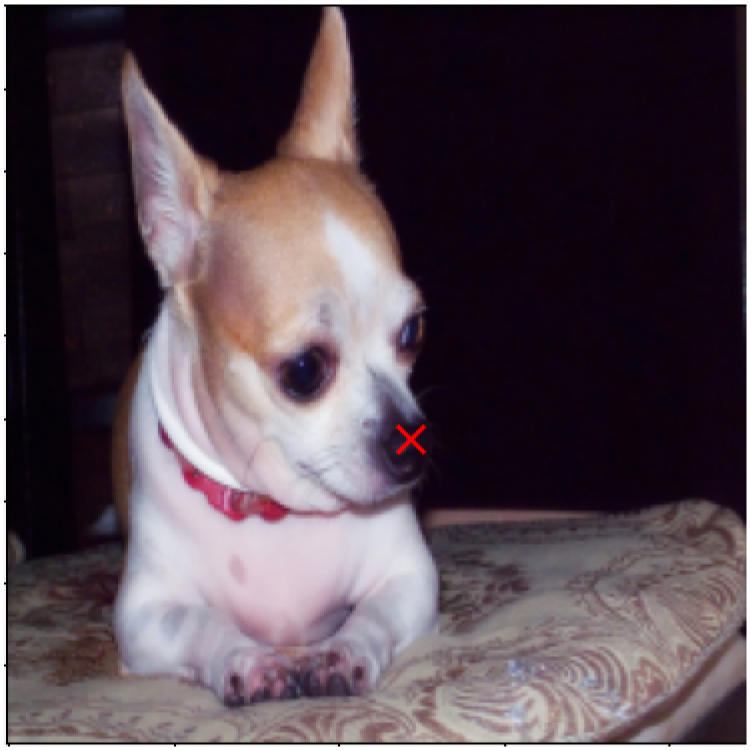


```
MEAN ED:  6.388120651245117
MIN ED:  0.0
MAX ED:  127.90621185302734
STD ED:  10.800914764404297
ACC test 0.26181818181818184
```

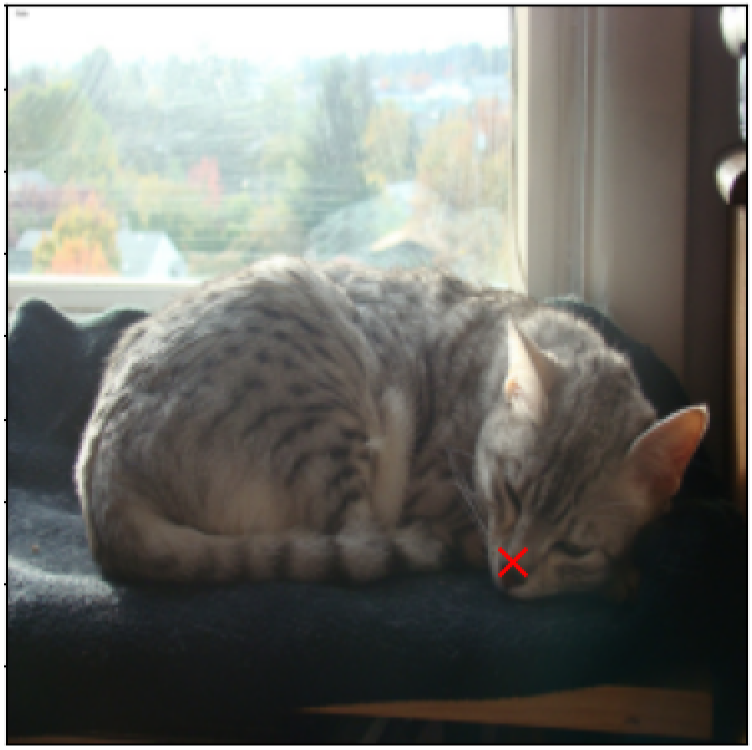
How well did it work?

For the majority of pet images, it works very well, but there are some adversarial examples where it struggles, or misses completely.



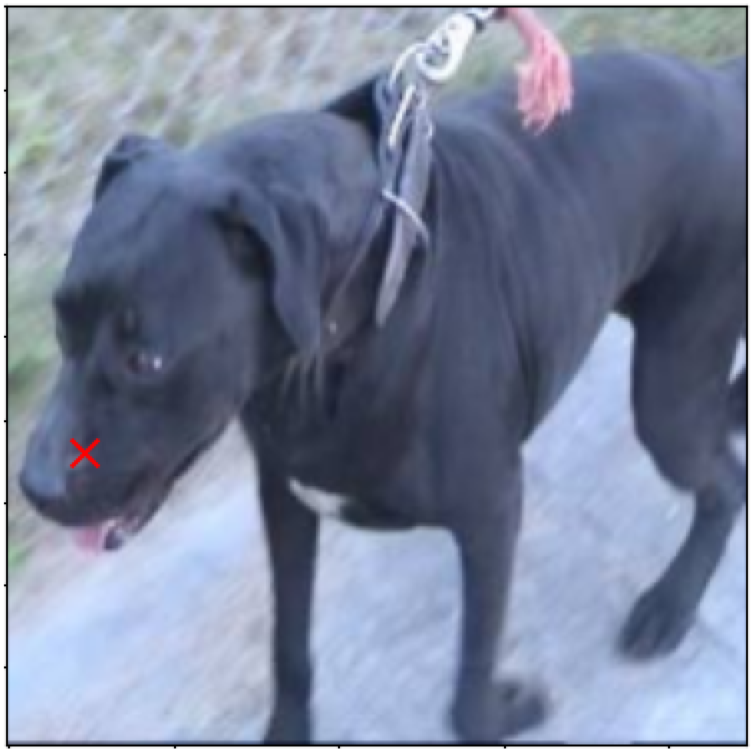


Look at this sad chihuahua. This is my favorite test image.

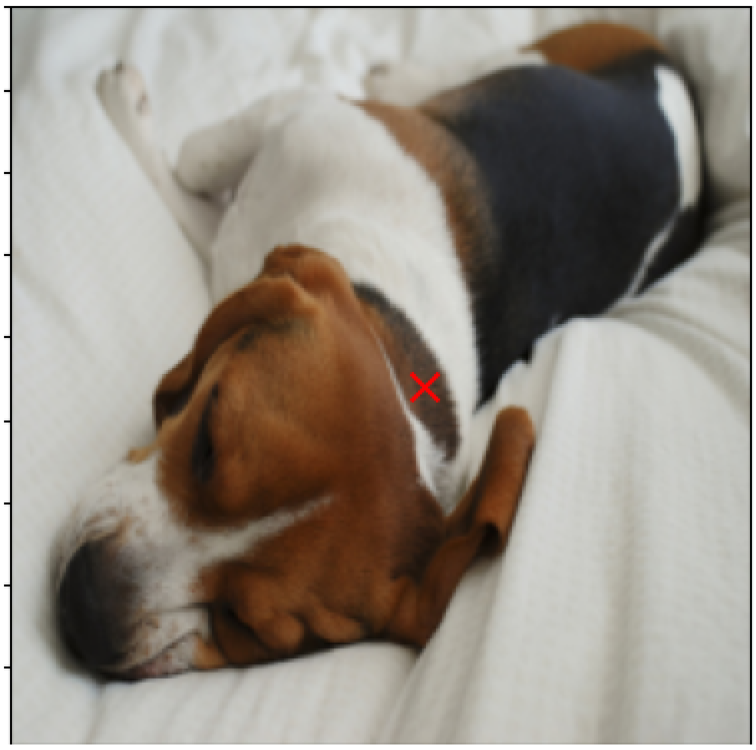


As you can see, the performance on the majority of the images is pretty well spot on, well within the range of labeling accuracy.

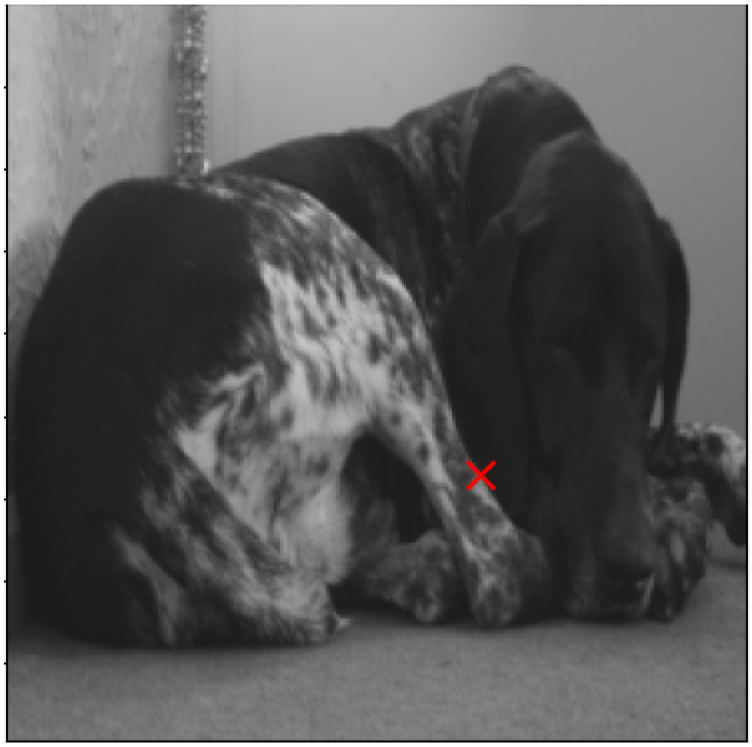
Adversarial examples: Ones I've cherry picked for the bad predictions



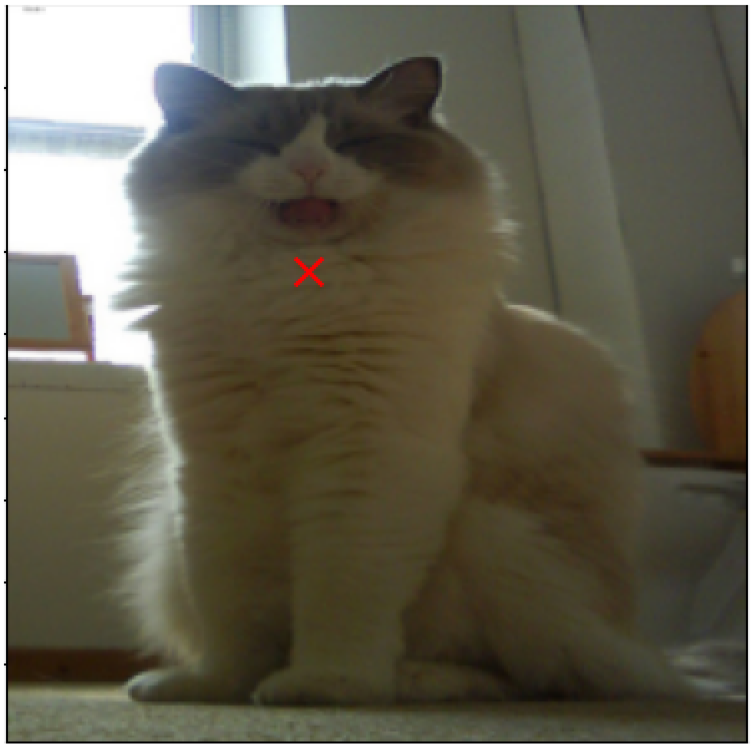
Colour of the dog is same as the nose, and in a pose that is not common.



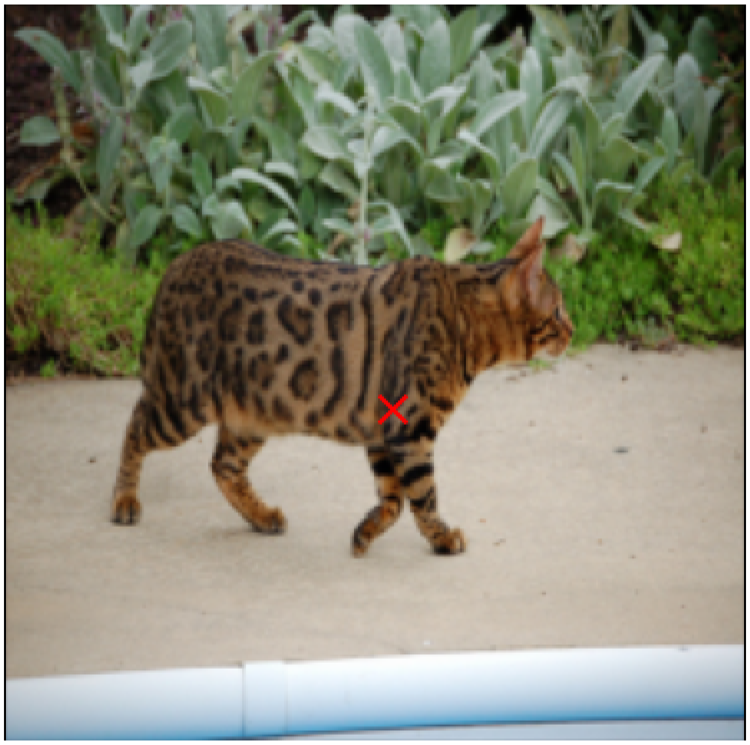
Looks like the model may have been fooled by the spot on the back, since it's closer to the center of the image, and the dog is in a non-typical pose



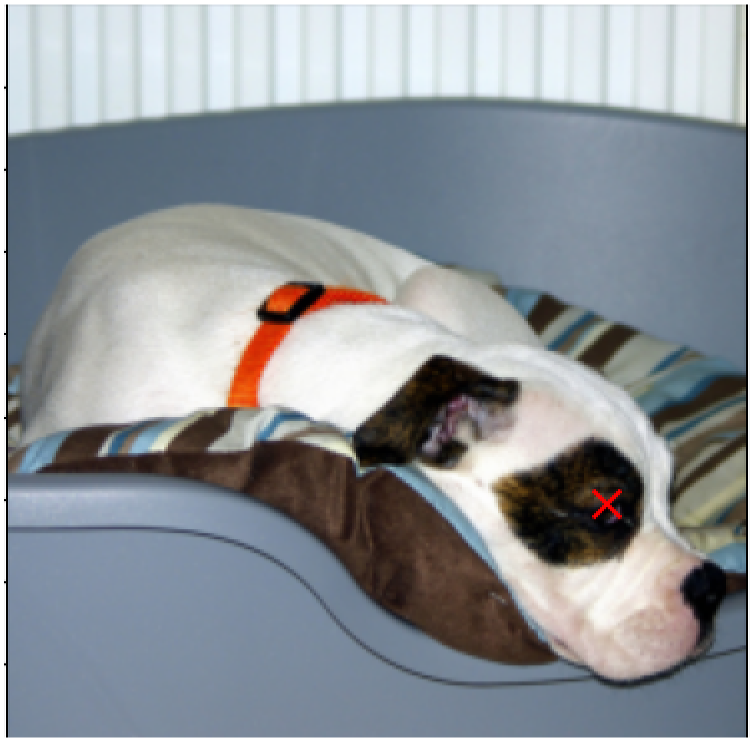
This is a tough image for even a human to spot the nose with their first glance. Also it's gray scale, which is different than the distribution of most of the images.



I think the model may have been confused by the open mouth, or the slightly strange shape of the cat's neck.



Nose is barely even visible in the picture. Model may have been confused by the spots, which is very interesting because it sort of matches how camouflage patterns are meant to confuse prey visual systems.



It’s clear on this one that it was confused by the eye coloring.

So as you can see, the examples where it's getting them wrong are mostly ones where having some conceptual understanding instead of purely visual information could really assist in the classification. This is the reason why I wanted to go with a vision transformer initially. However, I believe that overall the results are satisfactory.

Inference hardware:

Using the same GPU as listed above.

Performance:

0.012877 seconds per image approximately, so about 13 msec per image.

Discussion:

Overall I feel that the performance of the model is adequate for the task. It predicts the noses in the majority of normal cases to a high accuracy. I could make a better model if I had more time, and it was necessary to, however I don't feel that it would be worth the extra effort for such a modest boost in performance. I suspect the error in labeling is around 5 pixels, due to the fact that we had human annotators, and thus I don't think doing better than that without overfitting is likely. So the only real way, in my estimation, to improve the model would be by eliminating the corner cases as shown above.

Possible improvements:
- scaling the images to keep the aspect ratio (may be why the labels are off horizontally a little)
- adding more regularization methods (noise, dropout, L2 norm)
- creating more "fake" training data through cropping, data augmentation, flipping etc.
- upsampling the adversarial examples
- use the test set for validation instead of splitting up the train into validation and train (could give a little more data to train, especially given how small the train set is)
- maybe the model could predict if it's a cat or a dog along with the regression box to force it to learn different pathways
- more complex models with more training time

One challenge I had in this lab was trying to get the transformer version of my code to fit at all. When I simply tried training the final FC layers on the dataset, it was incapable of performing the regression task. It could not even overfit to a tiny training dataset. At that point, I tried allowing the whole transformer to fit the set, but this took a very long time, and did not yield good results either. It is unclear if I simply would have needed to run the training longer, or if there is a fundamental issue with my implementation. I overcame this by using a different model, namely resnet18, and fine tuning all of its parameters. The rest of the implementation was almost identical.

My biggest challenge in this lab was debugging the testing code. For a long time I was confused because I was getting a low average euclidean distance on the training and validation partitions, but when I ran the same model on the test partition, it was getting terrible results. Eventually I was able to determine what I had done wrong. It was not in the model itself but the way I was generating the singular test outputs, because I was taking the batch normalization of one image, which of course gives you the center of the distribution. I needed to prepare a fake batch to do the normalization with so that the normalized image was appropriate for the distribution the model was used to. Once I did this, the results were as expected.

P.S.
I really enjoyed this course. I felt that the labs gave hands-on skills that will be transferable long after this course is over, which is quite rare. I hope that it sticks around for future years.