**Training/Testing Split**

I went through several phases when splitting my data into a training and test set, but I'm only going to mention the one that changed how I calculated accuracy and improved my overall performance. I think it worked better because I actually knew what I was doing to calculate accuracy after we talked about it in class. ☺

I began by creating a function that let me control how many files I wanted to read in, and how many samples of each genuine and imposter pairs I wanted to created.

While reading in the data, I created a dictionary. The keys in the dictionary were the label of an image (where the label told us who the person was). The value to each person label was a list of indexes. These indexes told us the location of distinct images in "data" of the person. After reading in all of the data into the dictionary. I was able to create a list of pair indexes that were either the same person, or not. I did this by iterating through dictionary of persons. When a person had more than 1 picture of themselves, I created tuples of combinations of the images indexes. I then added that tuple of a list of genuine indexes. When a person had only 1 picture I added them to a list of indexes. I did this to collect all the indexes of pictures that had no genuine match to later create combination pairs of indexes that were imposters.

Though my method, I controlled how many samples I wanted for each of the combinations: genuine and imposters. Then I create 2 lists of images. One list was of genuine pairs of images, and the other was of imposter pairs of images. The labels were implied. Before I would grab the number of pairs I desired, I always randomly shuffled the list of pairs.

I also was able to control the number of test samples I wanted through this method. However, I created a list of randomize pairs of genuine and imposter images with a parallel list of their respective labels, where the label was 0 if the pairs were genuine and 1 otherwise.

When I would read in all the files, I would get 16,552,692 pairs of imposters and 484,514 of genuine pairs. Therefore, I would just do make sure I picked a size that was reasonable enough to test and run with. On my own computers I would usually do batches of 1, but on the supercomputer I'd use batches of 100 images. On the super computer I'd do 10,000 sample pairs of each, giving me 20,000 sample pairs total. On the super computer, it would take about 2 minutes an epoch while training.

**ResNet Architecture**

To be able to complete my Siamese network efficiently, I created a ResNet Function so I can call it twice and share its variables.

1. I passed in an image into my ResNet and did a linear transformation to prepare it to go through a convolution.

2. Following the ResNet paper, I first did a convolution with 64 filters, 2 strides, and a 7 k-size. (Following this convolutions, the rest of the convolutions I performs had k-sizes of 3. )
3. After the convolution, I did a batch normalization and did a Relu activation on it
4. Next, I did a maxpool on the convolution
5. Now, I begin with the identity functions! A *block* describes the identity function performed in the ResNet. I created forloops around blocks that had the same number of filters to make the number of blocks flexible.
    a. A block consisted of 2 convolutions
    b. First, I did a convolution with a stride of 2
    c. Then a batch normalization
    d. Finishing this convolution with a Relu activation
    e. Then I did another convolution
    f. And then again another batch normalization
    g. If the number of filters increased, I would add padding around the input "image" and add the input "img" and the output of f. (just before).
    h. Then I would do a Relu on the sum
    i. This gave me the identity block
6. After my maxpool, I do 2 blocks with 64 filters and strides of 1
7. 3 blocks of 128 filters
8. 3 blocks of 256 filters
9. 2 blocks of 512 filters
10. and I end it with an average pool
    a. I do a reduce mean of the output from the convolution before
    b. Then a linear transformation
    c. And end it with a softmax

**Final Performance**

In my Siamese network, I compute the energy of a batch (with all the pairs either genuine or all imposters) by doing the sum of the absolute value of the difference between the two results from each of the resnets.

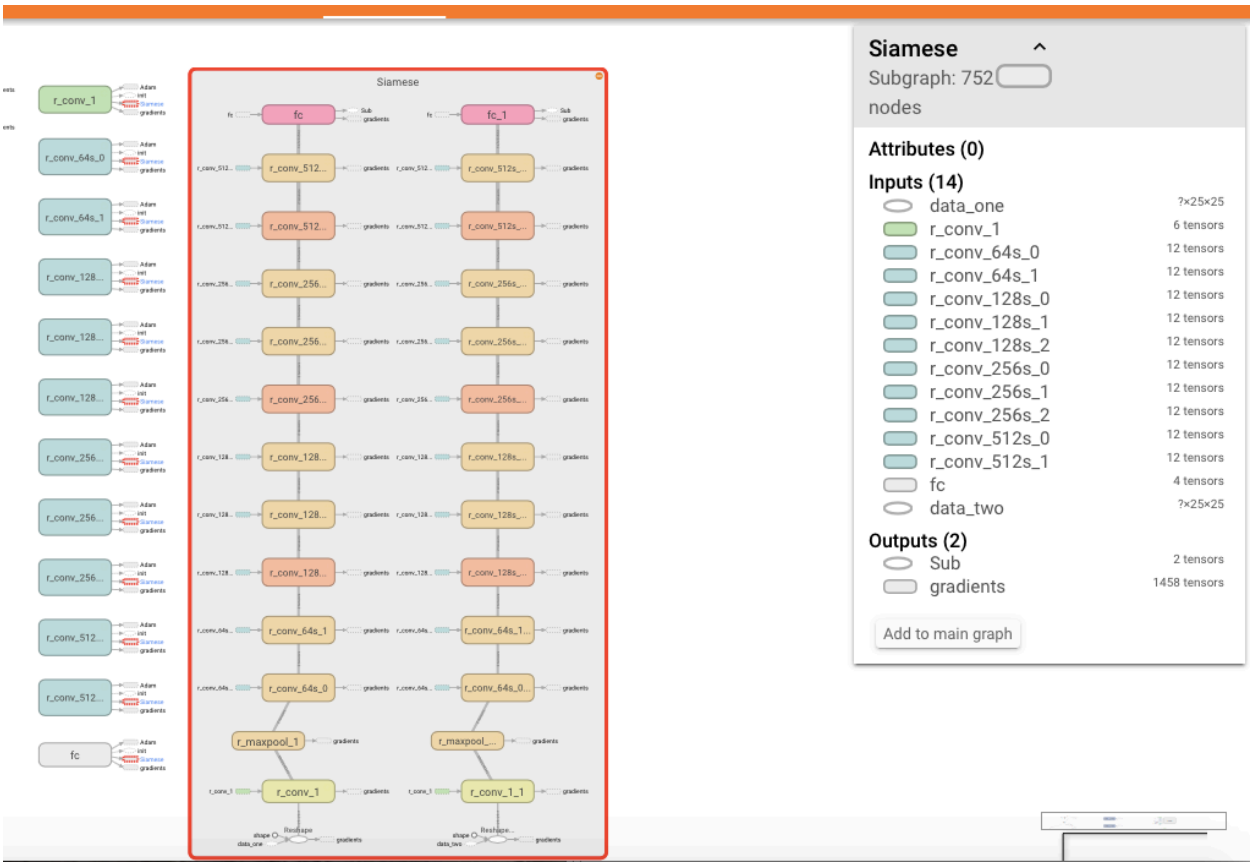
Then I use the energy to compute the contrastive loss.

To calculate accuracy, I used half of the margin (used in the loss function) to be my threshold. If the energy was less than the margin, then it was considered to be predicted as an imposter, otherwise, genuine. It was weird. Whenever I would switch it, my accuracy would start at 50% and decrease to 2% or so. So instead I changed it.

By setting my learning rate in my adam optimizer to .00001, I was able to learn pretty past. I only did 12 epochs as well to train.

To test, I used 2000 samples with 50% of imposter pairs and 50% of genuine pairs. Using the sample sizes described before, I achieved an accuracy of 99% on training and 100% on testing…which I know is weird, but as I looked at the energies, they were fulfilling the requirement. My loss decreased from 548 to 12 when it was done training

If I made the network any smaller, of if I made the learning rate larger, my training data would overfit by the 3rd epoch. Which honestly made me pretty happy.




```
Staring Epochs...
epoch 0  time: 120.764539 loss 548.06496 accuracy 43.50
epoch 1  time: 116.659581 loss 417.17166 accuracy 42.00
epoch 2  time: 117.352691 loss 403.58289 accuracy 47.50
epoch 3  time: 117.557977 loss 383.25751 accuracy 62.00
epoch 4  time: 117.666451 loss 354.38918 accuracy 78.00
epoch 5  time: 117.654236 loss 313.77161 accuracy 93.00
epoch 6  time: 117.612380 loss 258.07369 accuracy 97.50
epoch 7  time: 117.580366 loss 201.78833 accuracy 99.50
```

```
epoch 8  time: 117.679984 loss 153.15106 accuracy 100.00
epoch 9  time: 117.556329 loss 110.32362 accuracy 100.00
epoch 10  time: 117.321457 loss 75.41175 accuracy 100.00
epoch 11  time: 117.179944 loss 48.10359 accuracy 100.00
epoch 12  time: 116.773954 loss 29.72087 accuracy 100.00
epoch 13  time: 116.391557 loss 18.58235 accuracy 100.00
epoch 14  time: 116.186736 loss 12.29125 accuracy 100.00
----------test model----------
Test Accuracy: 100.00
```