Akshat Shrivastava
CSE 599 G1: Deep Learning
Homework 1

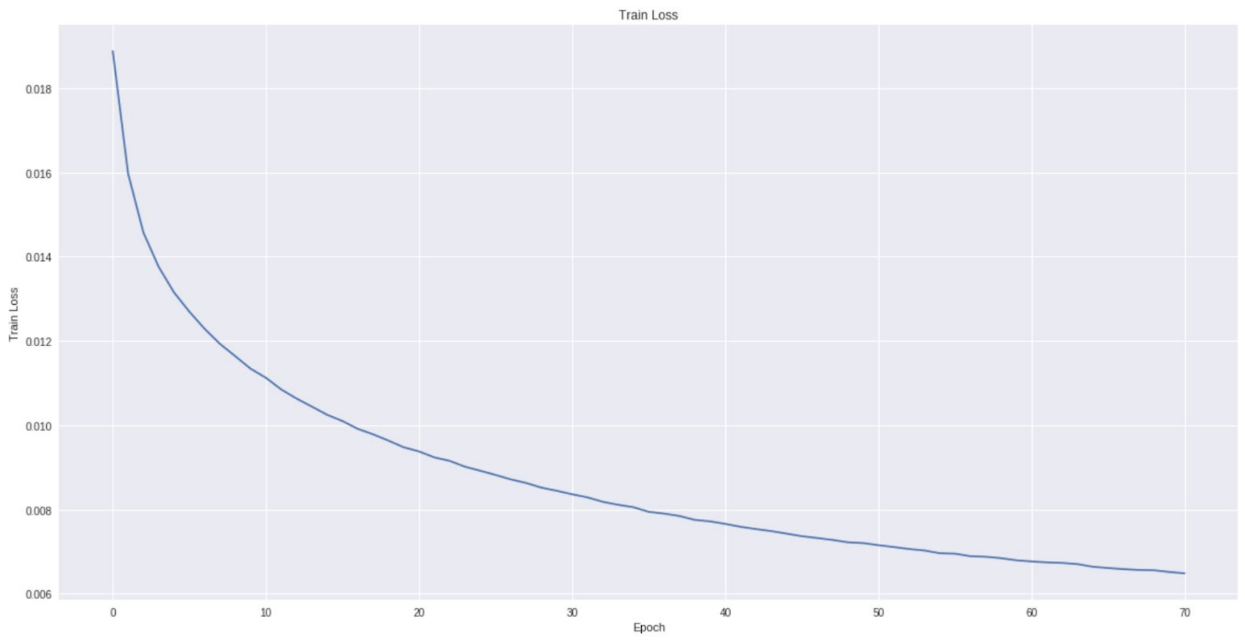
Tiny ImageNet

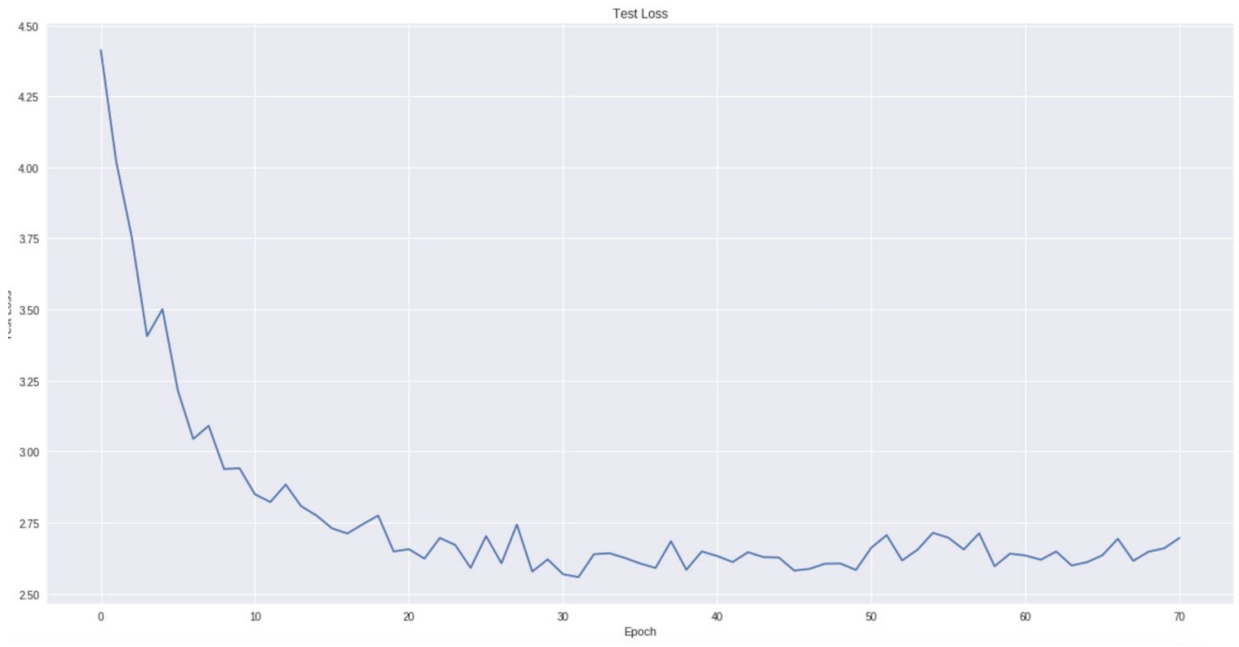
What design that you tried worked the best? This includes things like network design, learning rate, batch size, number of epochs, and other optimization parameters, data augmentation etc. What was the final train loss? Test loss? Test Accuracy? Provide the plots for train loss, test loss, and test accuracy.

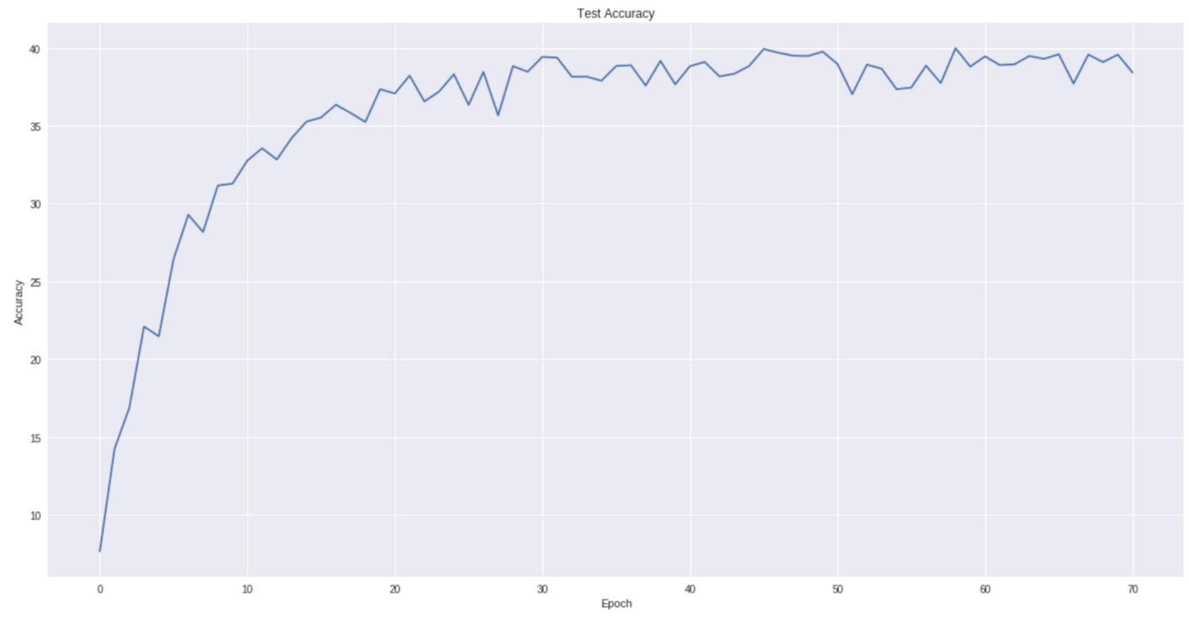
For Tiny ImageNet I experimented with quite a few different things, the model I turned in ended up being very similar to Inception Net with 5 Inception layers that compute 1x1 3x3 and 5x5 convolutions and concatenate them, with batch norm and dropout applied throughout. The optimal hyper parameters appeared to be 256 batch size with a learning rate of 0.01, when testing I found 0.1 learning rate to decrease performance and a batch size of 128 was too low as well.

I experimented with adding residual connections and further dropout + a 7x7 convolution in the inception block, but all of those things resulted in an accuracy that was lower or more overfitting.

The best model I had topped out at 40% accuracy as you can see in the plots below, and then began overfitting.



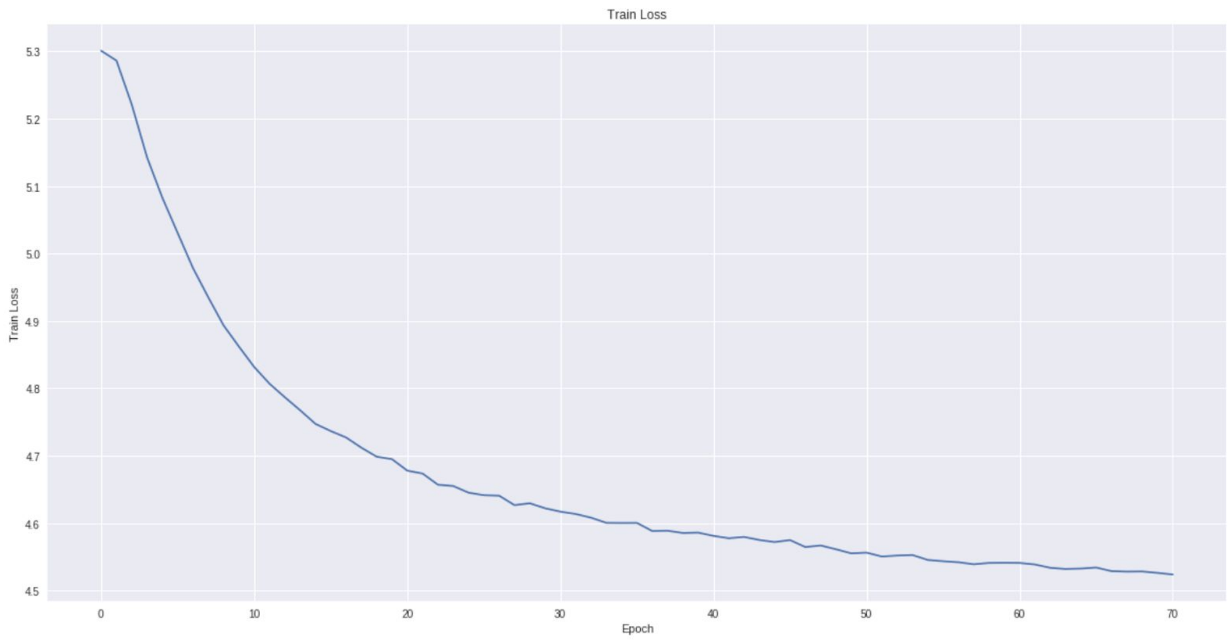


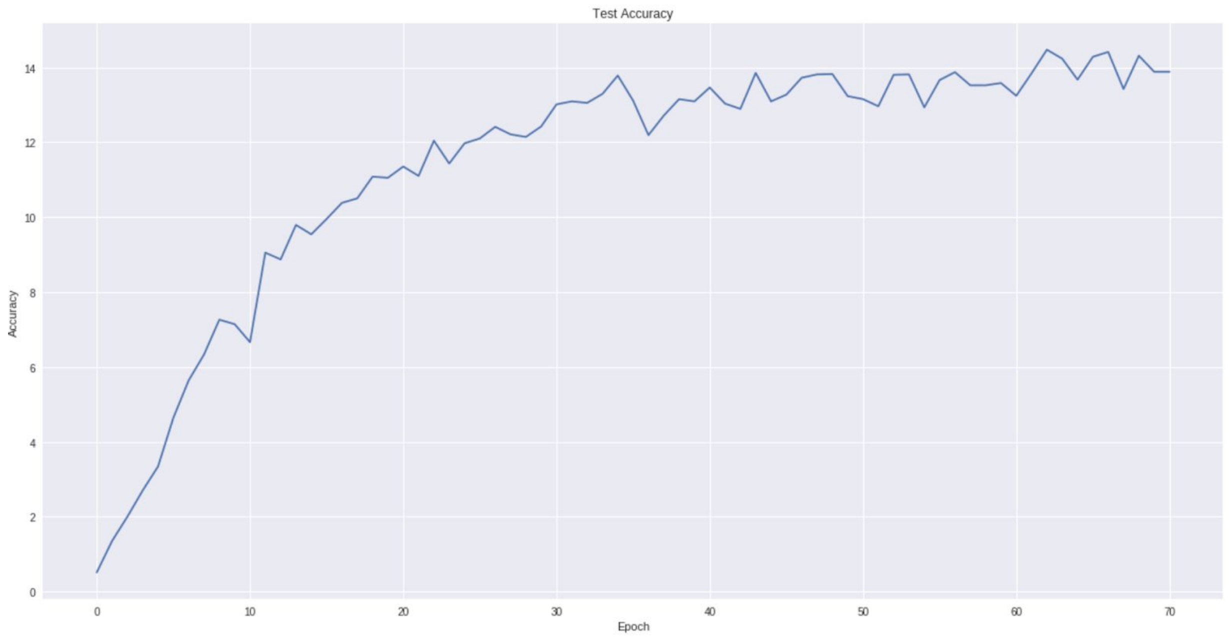


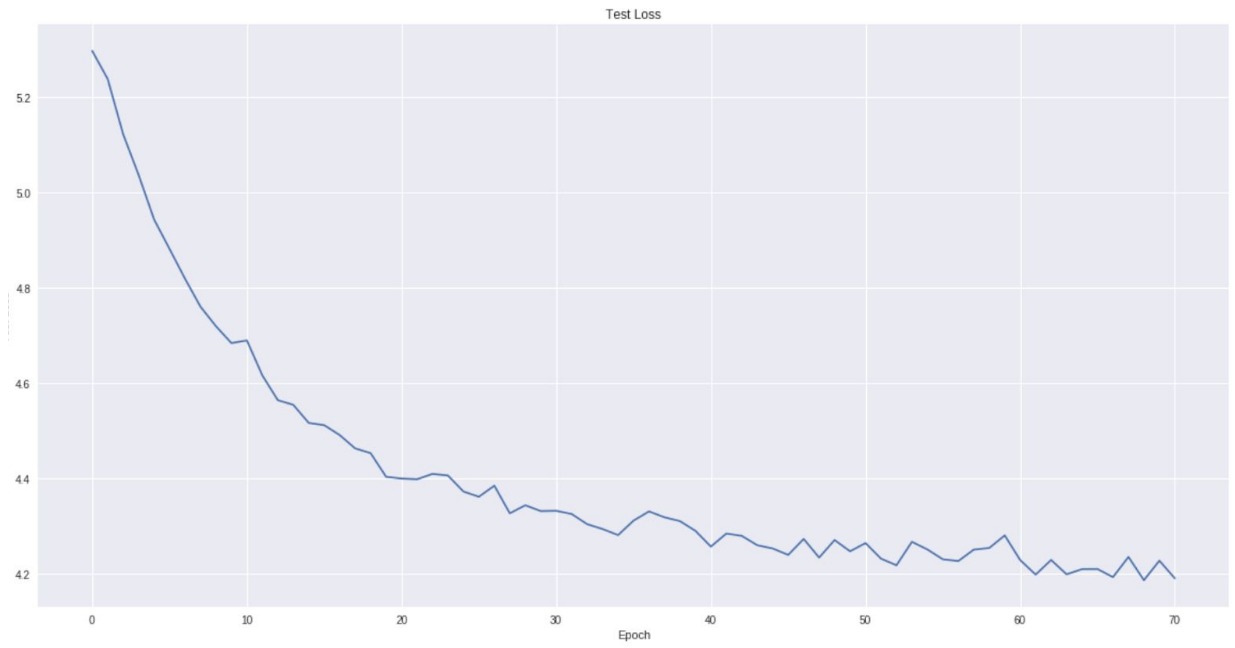
Note: For the train loss the y axis scale is messed up because I forgot to account the batch size when averaging the train loss. The lower point 0.008 is equivalent to a loss of 2.04 for reference.

What design worked the worst (but still performed better than random chance)? Provide all the same information as question 1.

The worst model I tried was the basic MNIST model adjusted for this problem with slightly bigger convolutions and output size. This model topped out at around 14% accuracy which is better than random guessing, but still not quite as performant as the other models I tried out. The plots are included here:







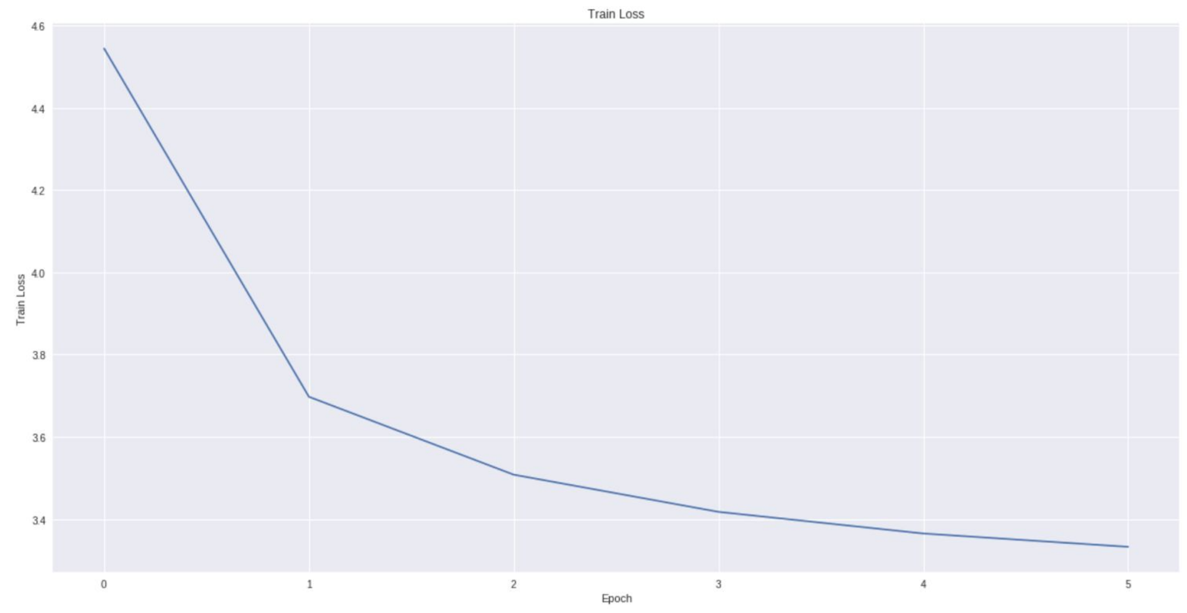
Why do you think the best one worked well and the worst one worked poorly.

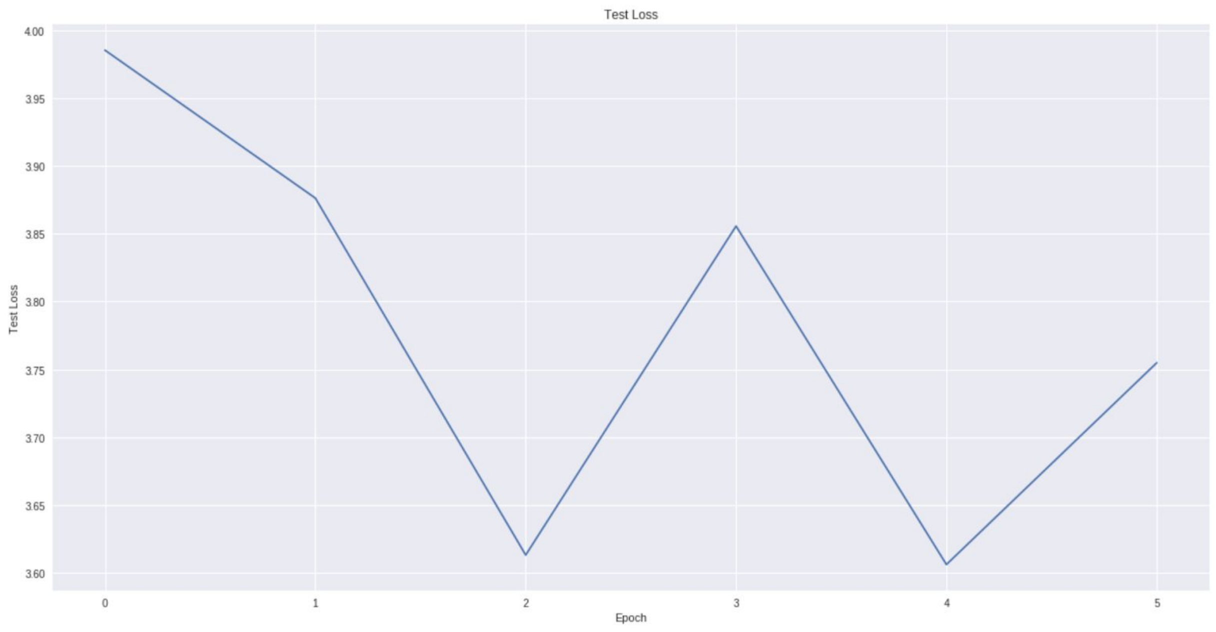
The complexity in the inception like model allowed the network to specialize more for such a large dataset in comparison to MNIST and CIFAR, where as the networks were too small in MNIST and CIFAR to capture all the information in the TinyImage Net pictures.

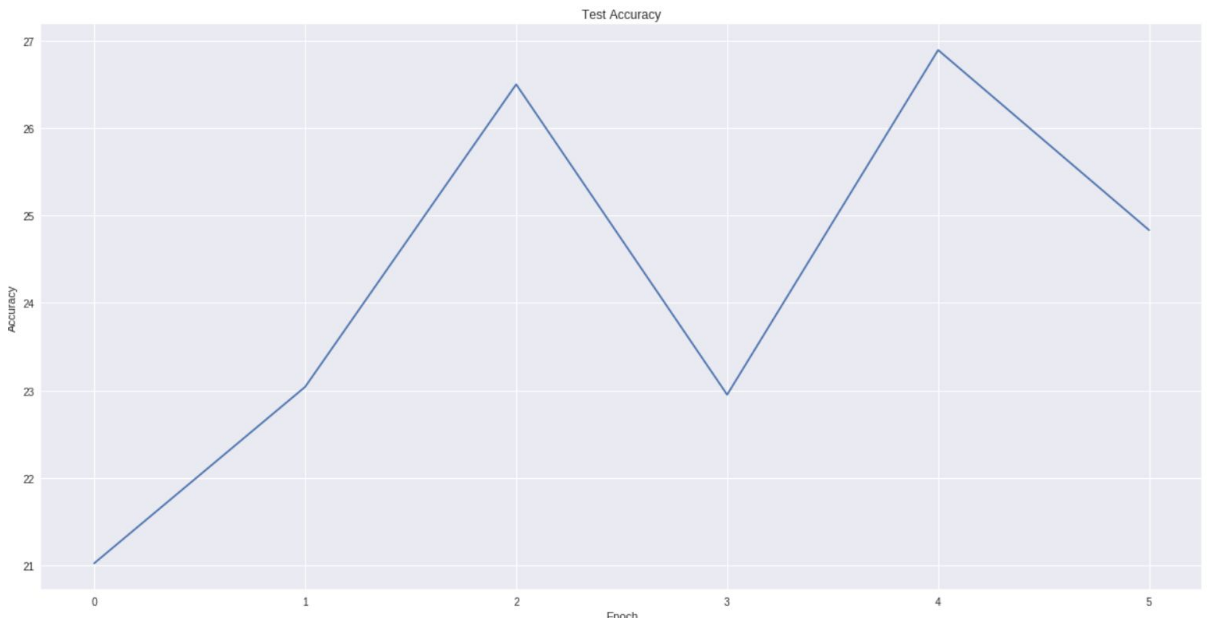
Full ImageNet

What design that you tried worked the best? How many epochs were you able to run it for? Provide the same information from Tiny ImageNet question 1.

Since Full Imagenet took too long to train, I worked on getting a good architecture in Tiny Image Net and scaling it up to handle the larger images and output dimension for the full image net problem, and this seemed to work pretty well, achieve 35% within 1 epochs, however this network took too long to train to generate graphs, so I made a much smaller network that is on part with the one I used for TinyImageNet scaled slightly up to handle the larger images and 1000 output classes, the results are as follows:


Train Loss
Train Loss
Epoch
4.6
4.4
4.2
4.0
3.8
3.6
3.4
0
1
2
3
4
5


Test Loss
Test Loss
Epoch
4.00
3.95
3.90
3.85
3.80
3.75
3.70
3.65
3.60
0
1
2
3
4
5



I ran this for 5 epochs, and the network peaked at about 27% accuracy, taking roughly 100 minutes to complete an epoch.

Were you able to use larger/deeper networks on Full ImageNet than you used on Tiny ImageNet and increase accuracy? If so, why? If not, why not?

I was able to train a larger network for Full Image net, I did run into some issue though. Primarily due to the larger images, larger training set, and larger network, the train time was quite slow and did not allow me to experiment too much. The larger network however did work at a pretty good rate, which I suspect is because it is able to capture more of the details through the larger number of parameters in the network.

The real ImageNet dataset has significantly larger images. How would you change your network design if the images were twice as large? How about smaller than Tiny ImageNet (32x32)? How do you think your accuracy would change? This is open-ended, but we want a more thought-out answer than "I'd resize the images" or "I'd do a larger pooling stride." You don't have to write code to test your hypothesis.

If the images were larger, I would add another convolutional- maxpool layer to the start of the network, to follow a similar design to VGG and Darknet where a convolutional layer + maxpooling will strategically downsample the image and add hopefully enough parameters to learn more from the larger image. Likewise if the image was smaller I would remove a layer from the start for the same reason. I would have to retune this models, but hopefully adding the layer at the start will increase accuracy from the additional information being provided, and removing the layer in the smaller images will stop overfitting since the original network has a large number of parameters.