1. See if you can improve the MNistResNetwork architecture using more ResNetBlocks. What's the highest accuracy you achieve? What is the architecture (you can paste the output from `print(network))`.

The original network has only two blocks, and I tried three blocks, 4 blocks, 5 blocks, 1 block. None of them achieved 99% accuracy, and as blocks become more, the accuracy drops down a little. I think it is because the MNIST datasets is simple dataset, large complex network is not suitable and may cause over fitting.

```
        (0): ConvLayer: Kernel: (5, 5) In Channels 1 Out Channels 6 Stride 1
        (1): MaxPoolLayer: kernel: 2 stride: 2
        (2): ReLULayer:
        (3): ConvLayer: Kernel: (5, 5) In Channels 6 Out Channels 16 Stride 1
        (4): ResNetBlock:
            (conv_layers): SequentialLayer:
                (0): ConvLayer: Kernel: (3, 3) In Channels 16 Out Channels 16 Stride 1
                (1): ReLULayer:
                (2): ConvLayer: Kernel: (3, 3) In Channels 16 Out Channels 16 Stride 1
            (add_layer): AddLayer:
            (relu2): ReLULayer:
        (5): MaxPoolLayer: kernel: 2 stride: 2
        (6): ReLULayer:
        (7): FlattenLayer:
        (8): LinearLayer: (784, 120)
        (9): ReLULayer:
        (10): LinearLayer: (120, 84)
        (11): ReLULayer:
        (12): LinearLayer: (84, 10)
    (loss_layer): SoftmaxCrossEntropyLossLayer:
```

2. Do you get any improvement using a different non-linearity? Be sure to change it back to ReLU before you turn in your final code.

I get just a little improvement by using leaky relu.

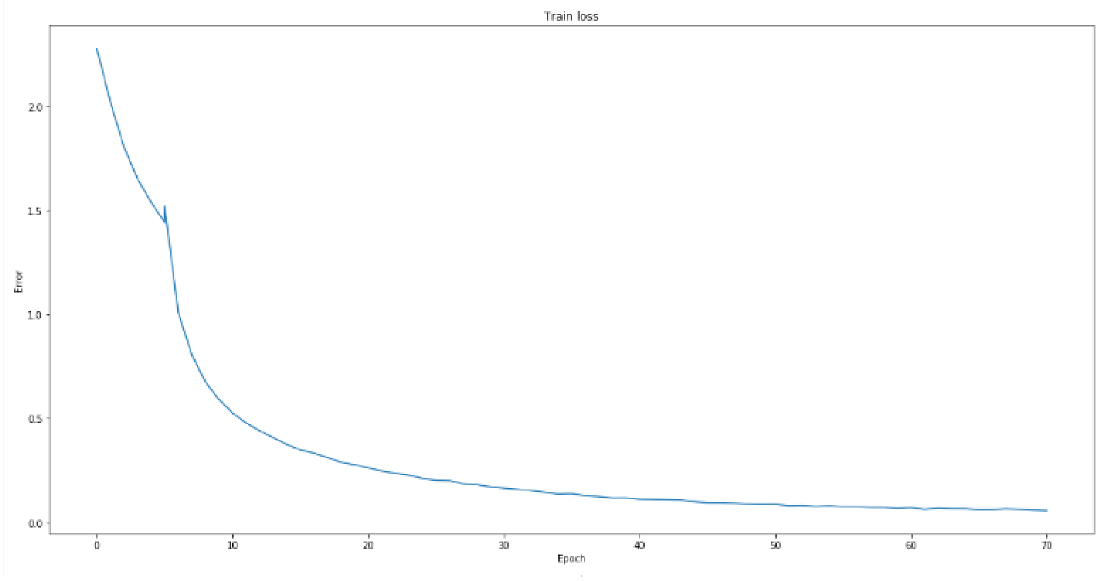
3. Can you come up with an architecture which gets even higher accuracy? Again, include the output from `print(network)`.

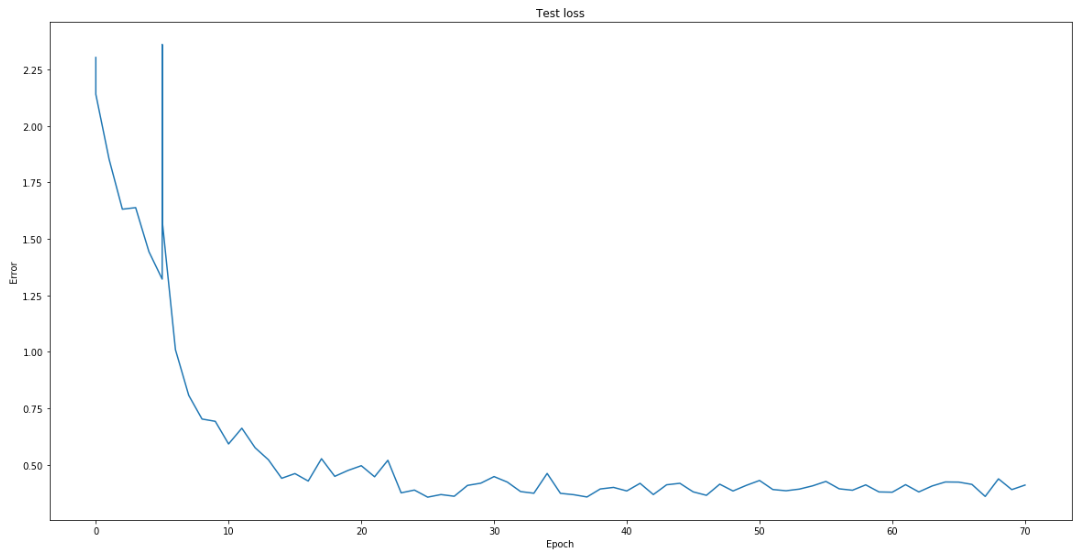
I cut off one fc layer and get 98.9%.

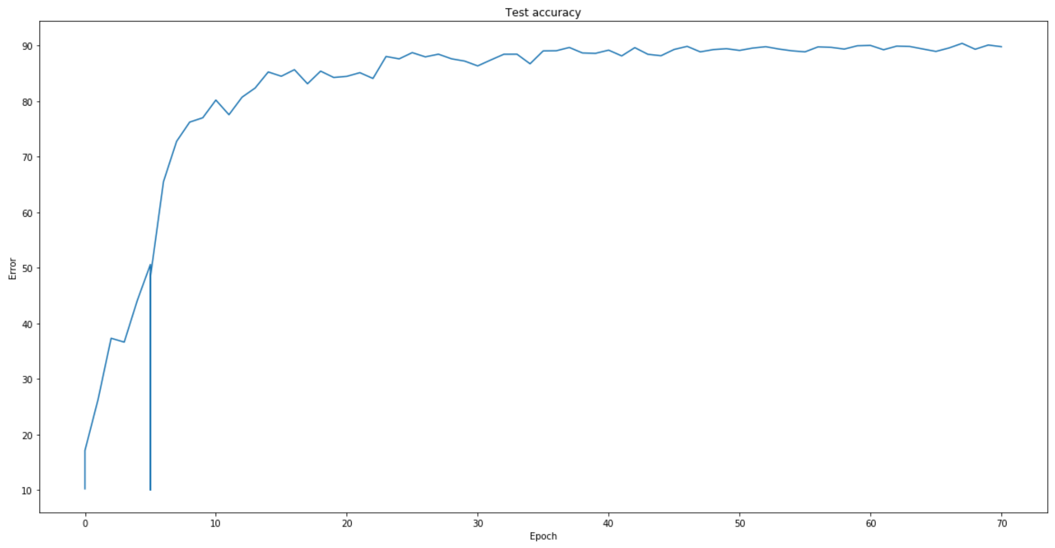
```
        (0): ConvLayer: Kernel: (5, 5) In Channels 1 Out Channels 6 Stride 1
        (1): MaxPoolLayer: kernel: 2 stride: 2
        (2): ReLULayer:
        (3): ConvLayer: Kernel: (5, 5) In Channels 6 Out Channels 16 Stride 1
        (4): MaxPoolLayer: kernel: 2 stride: 2
        (5): ReLULayer:
        (6): FlattenLayer:
        (7): LinearLayer: (784, 120)
        (8): ReLULayer:
        (9): LinearLayer: (120, 10)
```

CIFAR

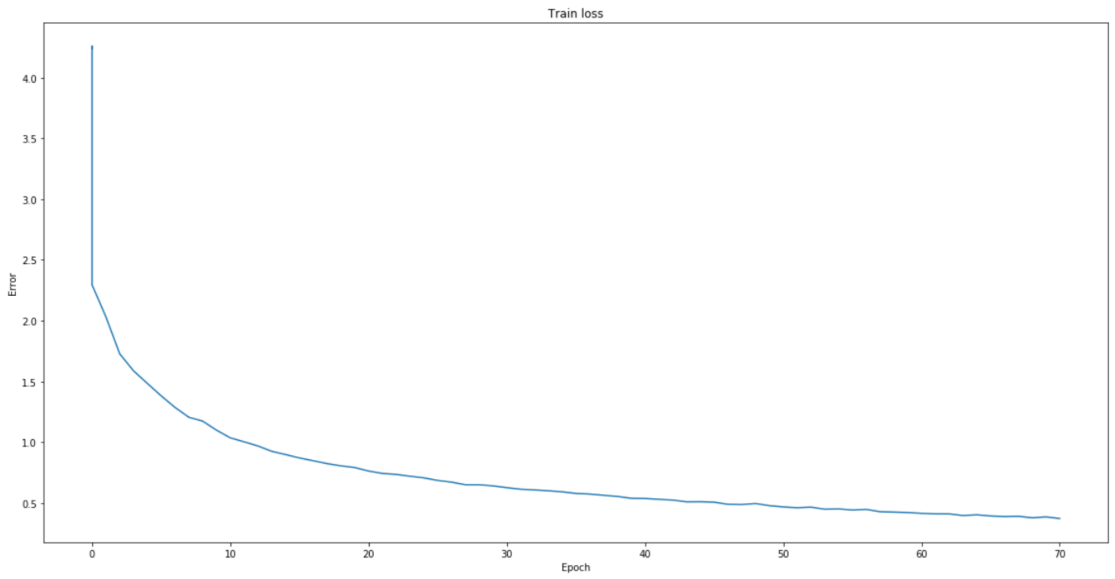
1. What design that you tried worked the best? This includes things like network design, learning rate, batch size, number of epochs, and other optimization parameters, data augmentation etc. What was the final train loss? Test loss? Test Accuracy? Provide the plots for train loss, test loss, and test accuracy.
   VGG16 worked well. Learning rate: 0.01, batch size: 256, number of epochs: 70. Train loss: 0.049, test loss: 0.3894, test accuracy: 90%

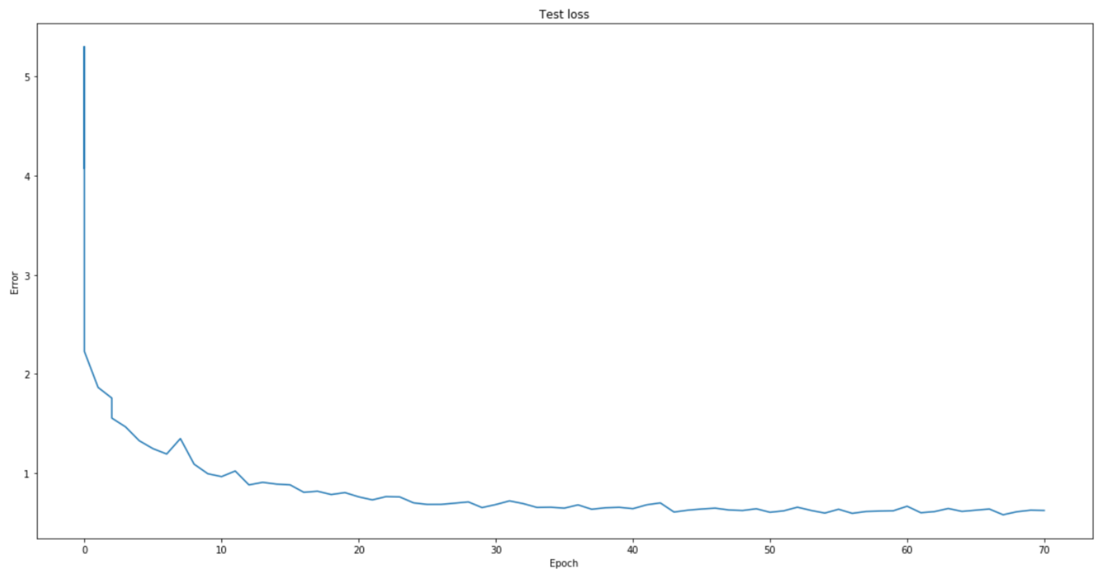


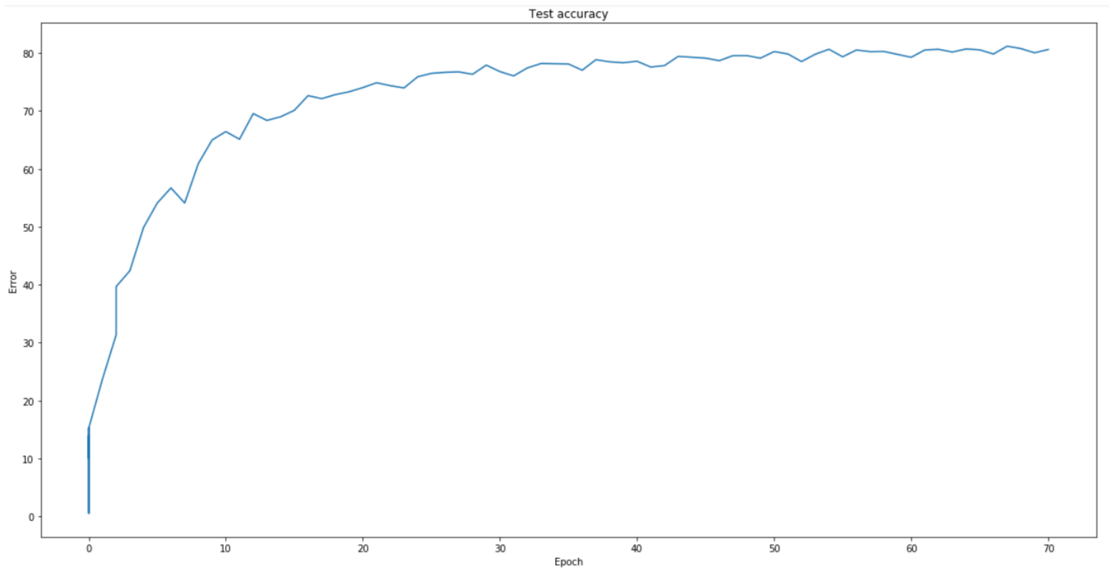




2. What design worked the worst (but still performed better than random chance)? Provide all the same information as question 1
   AlexNet worked the worst. Learning rate: 0.01, batch size:256, epochs: 70. Train loss: 0.425, test loss: 0.623, test accuracy: 81%.
3. Why do you think the best one worked well and the worst one worked poorly.
   Because AlexNet is too simple, and VGG16 is deeper, it has more conv layers and pool layers.

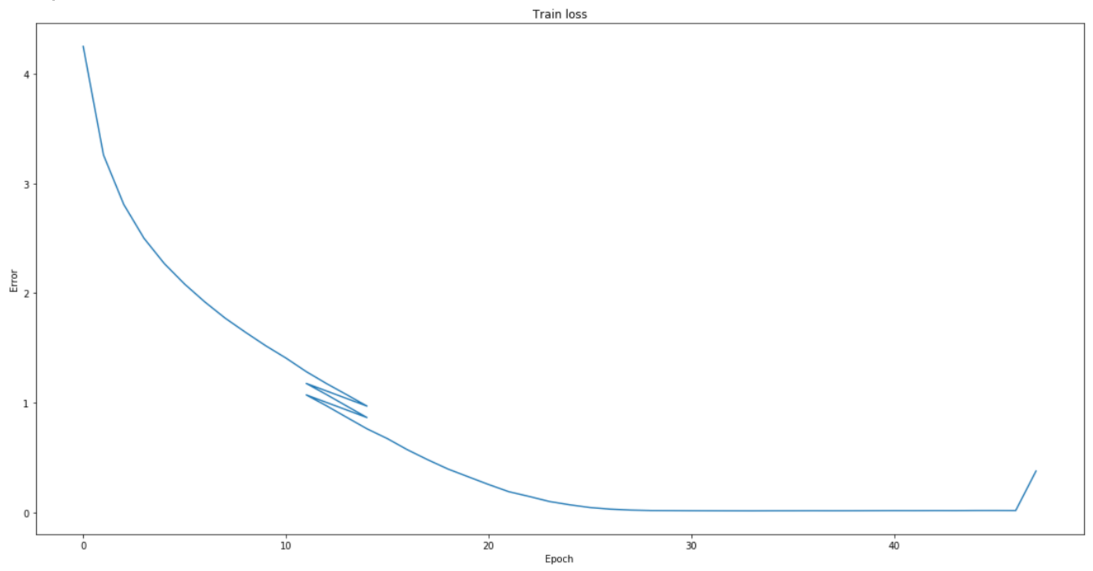


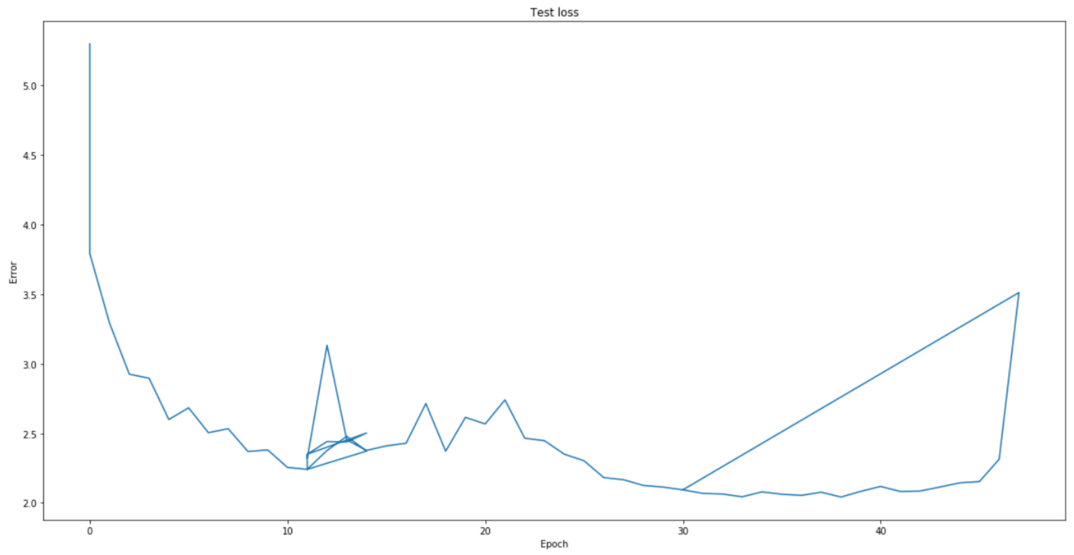


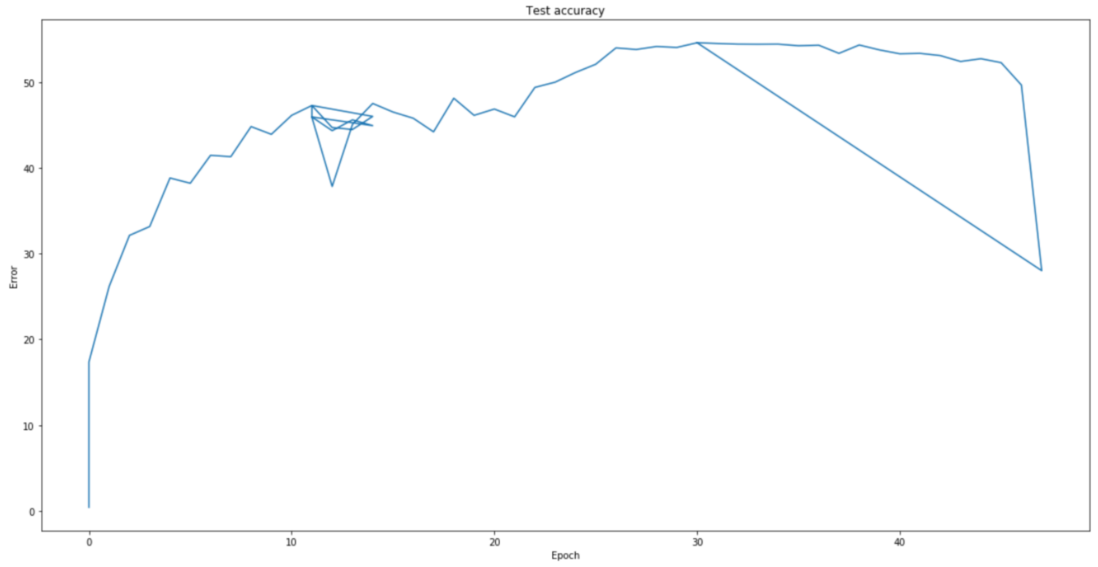


TinyImageNet

1. What design that you tried worked the best? How many epochs were you able to run it for? Provide the same information from CIFAR question 1.
   I tried AlexNet, VGG16 and ResNet, ResNet worked best. I ran it for 50 epoches. Batch size: 256, learning rate:0.01. In epoch 30(best), train loss : 0.014921, test loss : 2.0938, test accuracy : 55%







<u>Note</u>: Since I lost connection in the process of training , there are some weird parts in the plot.

2. Were you able to use larger/deeper networks on TinyImageNet than you used on CIFAR and increase accuracy? If so, why? If not, why not?
   Yes, VGG16 worked well on CIFAR, while it could only achieve 40% on TinyImageNet, so I used ResNet on TinyImageNet. Because the picture data on TinyImageNet is larger, 64x64, and the class is more than CIFAR, so we should use more deeper and larger networks.
3. 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.
   In my opinion, no matter how large is the image, the proportion of object in the image doesn’t change too much compared with background. So instead of resizing the images, I could use a larger kernel and stride in the first conv layer, which could help us extract features in the image. First, we trained a mode in the fixed image size, then we change the image size and cut off the first layer(change kernel size) and re-train it. I thinks as image goes larger, the accuracy would go down.