Section 1: CIFAR 10

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.**

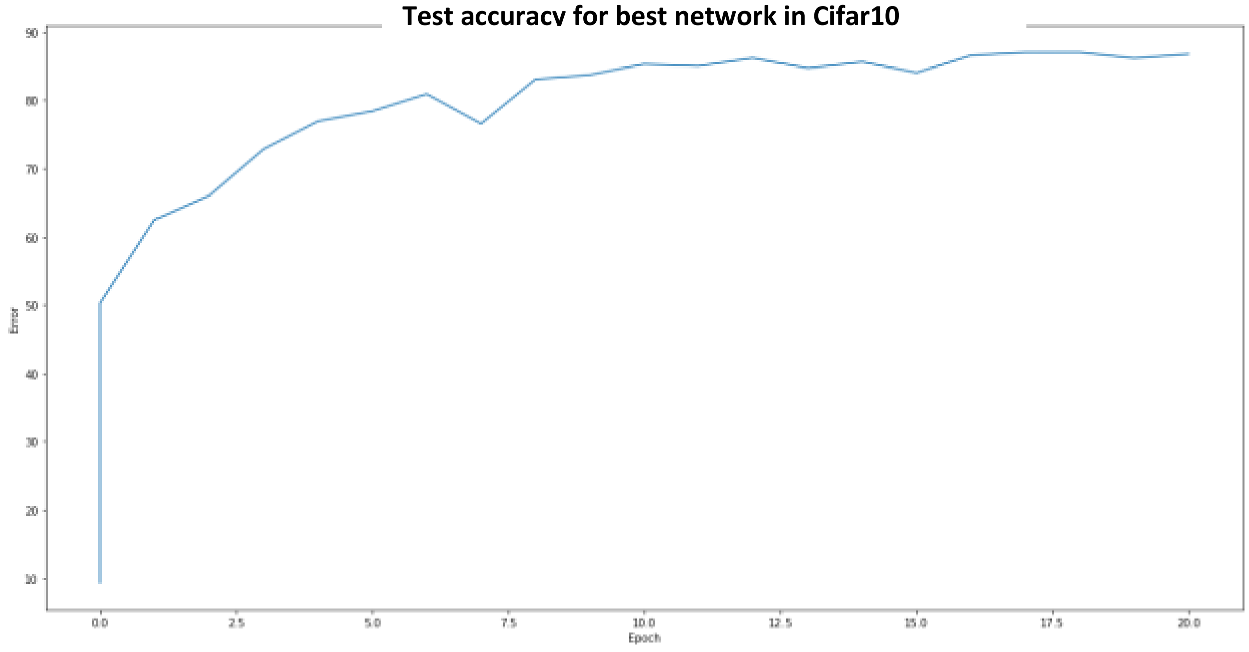
   For an apple to apple comparison, the performance of a model is given for a fixed number of epochs.

   Settings:
   - Batch size = 256
   - Test batch size = 10
   - Learning rate = 0.01
   - Momentum = 0.9
   - Number of epochs = 20
   - Seed = 0
   - Augmentations:
     1. RandomCrop(32, padding=4)
     2. RandomHorizontalFlip()
     3. ToTensor()
     4. Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))

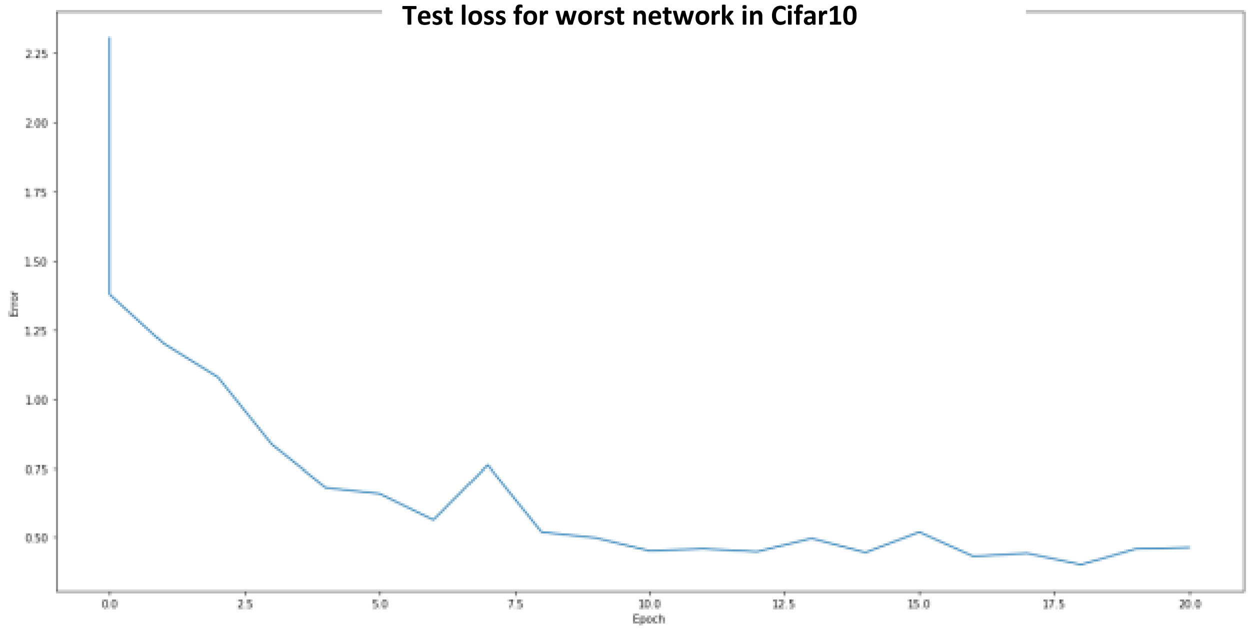
   Best network performance:
   - Final train loss is **0.151191**
   - Final test loss is **0.4632**
   - Final test accuracy is **87% (8670/10000)**

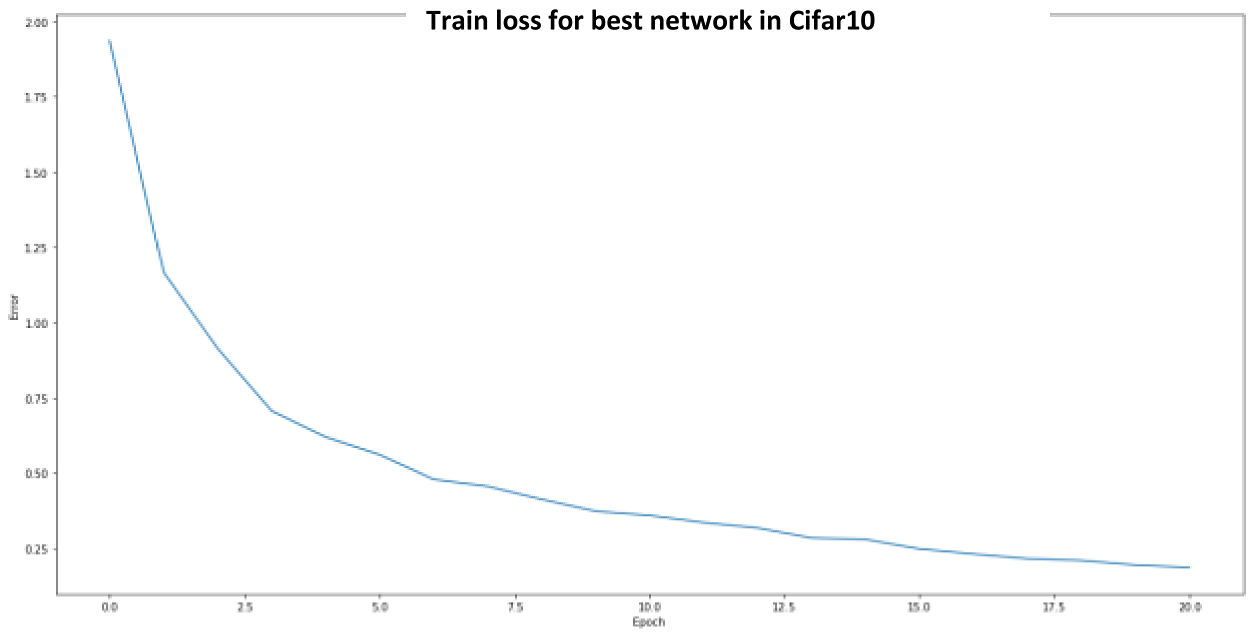
   Detail description of accuracy:
   Accuracy of plane: 79 %
   Accuracy of   car: 94 %
   Accuracy of bird: 73 %
   Accuracy of   cat: 67 %
   Accuracy of deer: 89 %
   Accuracy of   dog: 80 %
   Accuracy of frog: 95 %
   Accuracy of horse: 97 %
   Accuracy of ship: 98 %
   Accuracy of truck: 92 %

Test accuracy for best network in Cifar10



Test loss for worst network in Cifar10





Network structure:

```
CifarNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU()
  (conv2_0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm2_0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2_0): ReLU()
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2_1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm2_1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2_1): ReLU()
  (conv2_2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm2_2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2_2): ReLU()
  (conv3_0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm3_0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu3_0): ReLU()
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4_0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm4_0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu4_0): ReLU()
  (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4_1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm4_1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu4_1): ReLU()
  (conv4_2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm4_2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu4_2): ReLU()
  (pool5): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (flatten5): Flatten()
  (ff5_1): Linear(in_features=256, out_features=10, bias=False)
)
```

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

Settings:

Batch size = 256
Test batch size = 10
Learning rate = 0.01
Momentum = 0.9
Number of epochs = 20
Seed = 0
Augmentations:

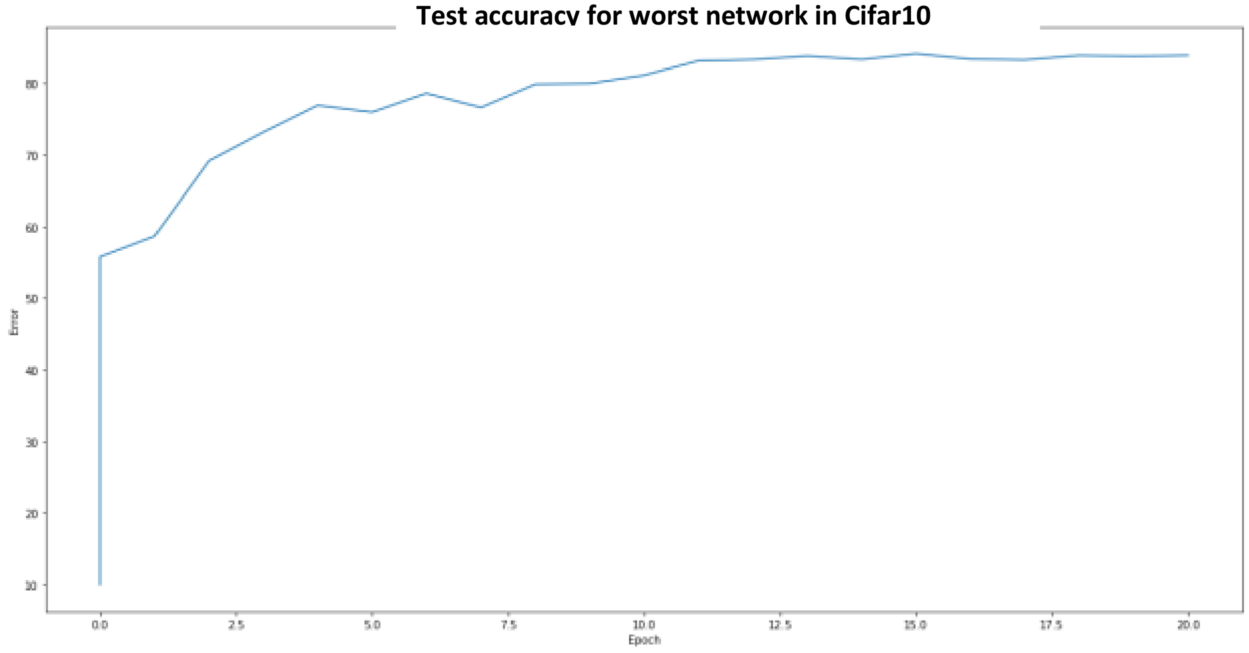
1. RandomCrop(32, padding=4)
2. RandomHorizontalFlip()
3. ToTensor()
4. Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))

Worst network performance:

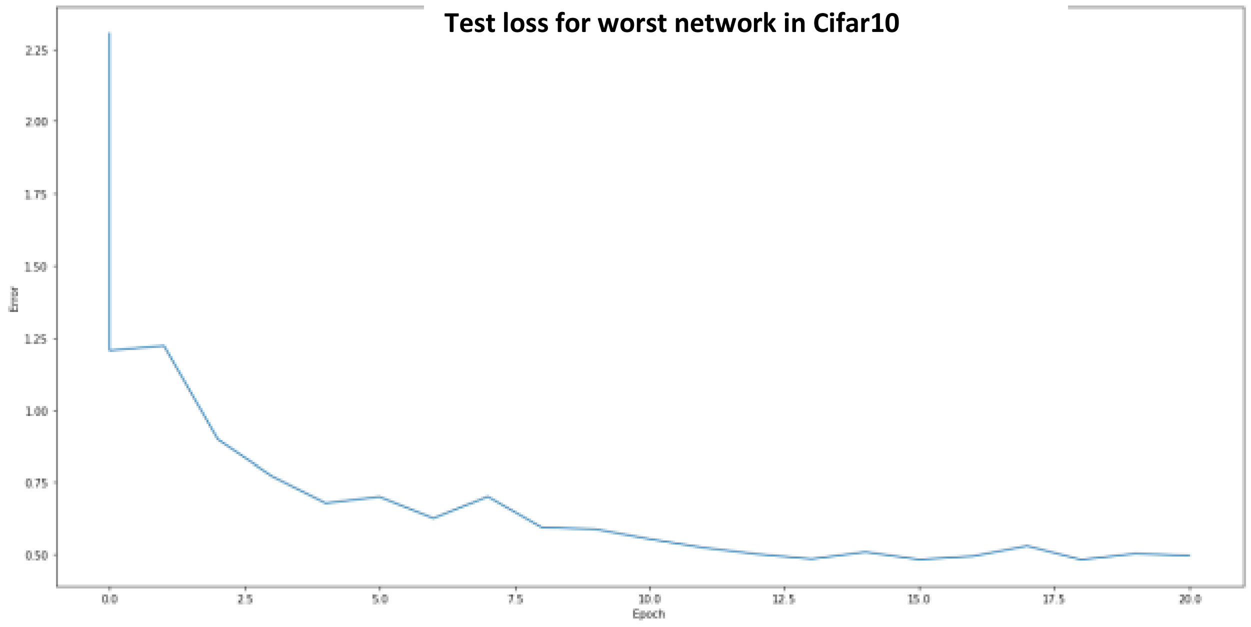
Final train loss is **0.277197**
Final test loss is **0.4956**
Final test accuracy is **84% (8386/10000)**

Detail description of accuracy:
Accuracy of plane: 86 %
Accuracy of   car: 86 %
Accuracy of bird: 83 %
Accuracy of   cat: 61 %
Accuracy of deer: 80 %
Accuracy of   dog: 83 %
Accuracy of frog: 83 %
Accuracy of horse: 86 %
Accuracy of ship: 91 %
Accuracy of truck: 88 %

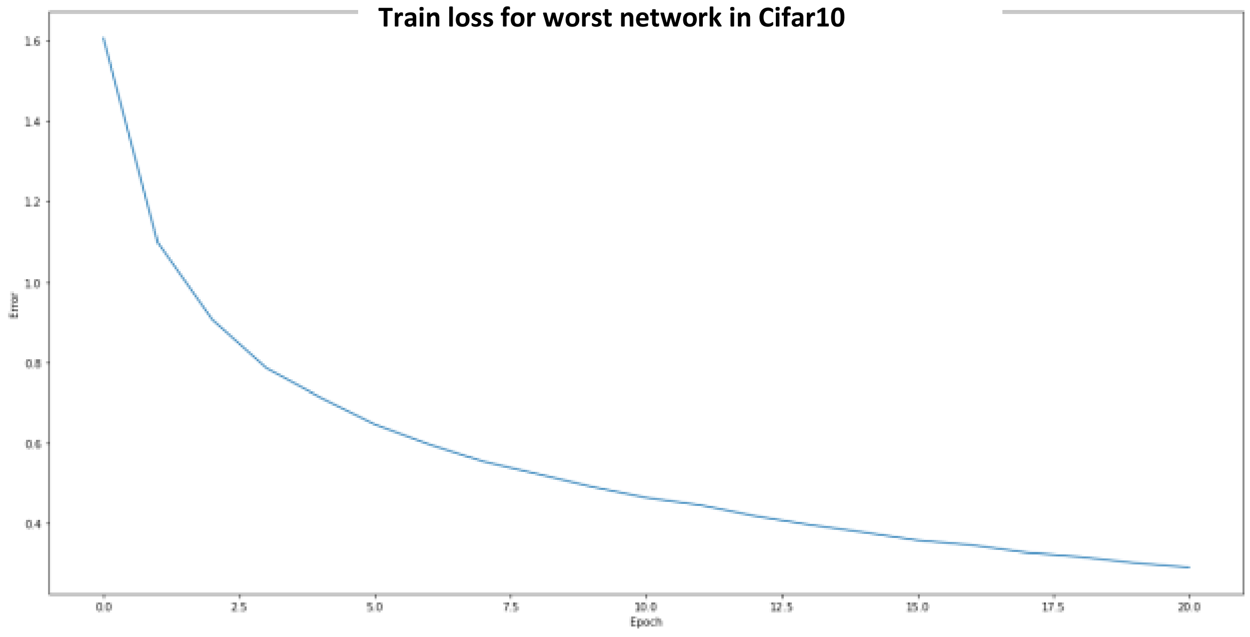
Test accuracy for worst network in Cifar10



Test loss for worst network in Cifar10



Train loss for worst network in Cifar10



Network structure:

```
CifarNet(
  (conv1): Sequential(
    (0): Conv2d(3, 10, kernel_size=(5, 5), stride=(1, 1), padding=(3, 3))
  )
  (resblock1_1): Sequential(
    (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (resblock1_2): Sequential(
    (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (resblock1_3): Sequential(
    (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (resblock1_4): Sequential(
    (0): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(10, 10, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (conv2): Sequential(
    (0): Conv2d(10, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (resblock2_1): Sequential(
    (0): Conv2d(100, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(100, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (resblock2_2): Sequential(
    (0): Conv2d(100, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(100, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (resblock2_3): Sequential(
    (0): Conv2d(100, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(100, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (resblock2_4): Sequential(
    (0): Conv2d(100, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(100, 100, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (conv3): Sequential(
    (0): Conv2d(100, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (resblock3_1): Sequential(
    (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (resblock3_2): Sequential(
    (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (resblock3_3): Sequential(
    (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (resblock3_4): Sequential(
    (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (conv4): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1))
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (maxpool_1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (maxpool_2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (maxpool_3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (flatten): Flatten()
  (ff_1): Linear(in_features=512, out_features=512, bias=True)
  (ff_2): Linear(in_features=512, out_features=500, bias=True)
  (ff_2_2): Linear(in_features=500, out_features=500, bias=True)
  (ff_3): Linear(in_features=500, out_features=100, bias=True)
  (ff_4): Linear(in_features=100, out_features=100, bias=True)
  (ff_5): Linear(in_features=100, out_features=100, bias=True)
  (ff_6): Linear(in_features=100, out_features=10, bias=True)
  (batchnorm_1): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (batchnorm_2): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (batchnorm_3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (batchnorm_4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)
```

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

The best one worked well because of the large number of filters despite a shallower network. Worst one worked poorly due to smaller number of filters despite the deeper network with a lot of residual connections. Adding more layers at some point will start to be a bad tradeoff for training time and accuracy gain.

Section 2: Tiny ImageNet

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.**

Settings:

Batch size = 256
Test batch size = 10
Learning rate = 0.01
Momentum = 0.9
Weight decay = 0.0005
Number of epochs = 10 *After 8 or 9 epochs losses and accuracy starts to saturate
Seed = 0
Augmentations:

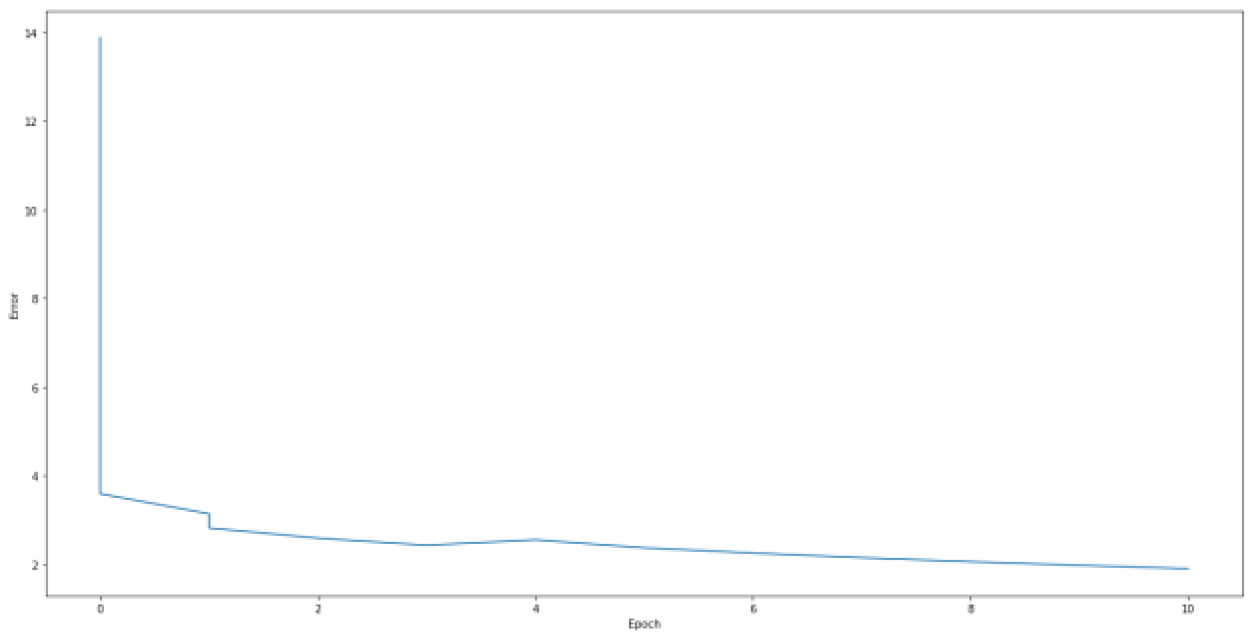
1. ToPilImage()
2. RandomHorizontalFlip()
3. ColorJitter()
4. RandomRotation(30)
5. ToTensor()
6. Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))

Best network performance:

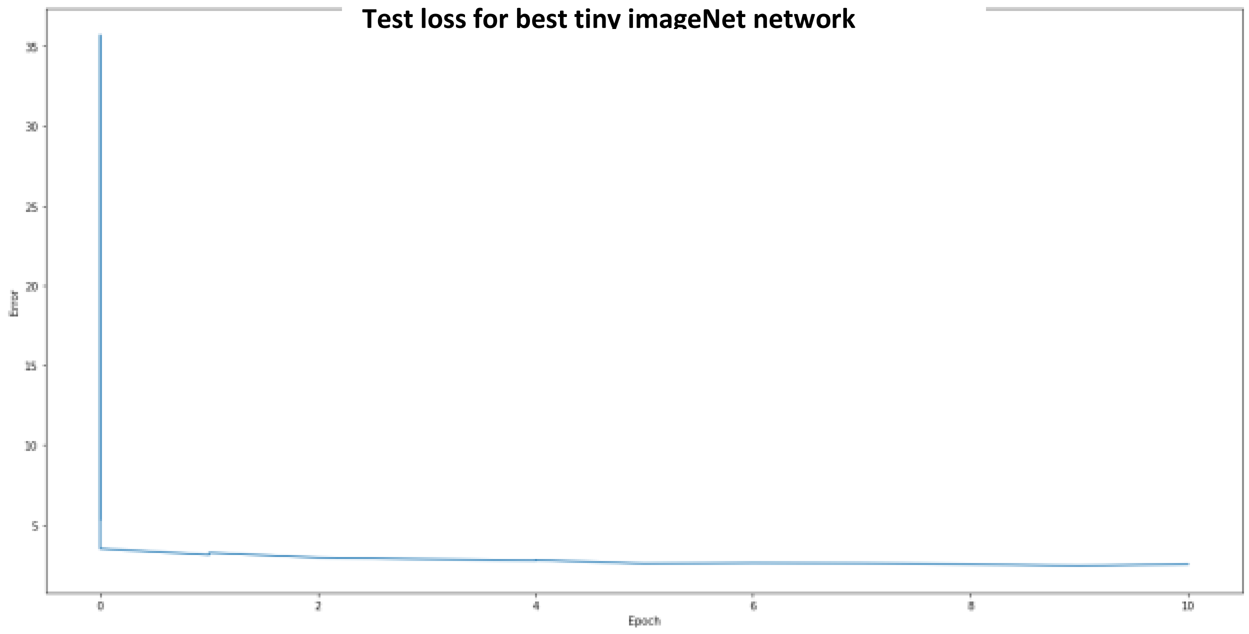
Final train loss is **0.2075371**
Final test loss is **02.4071**
Final test accuracy is **44% (3496/8000)**

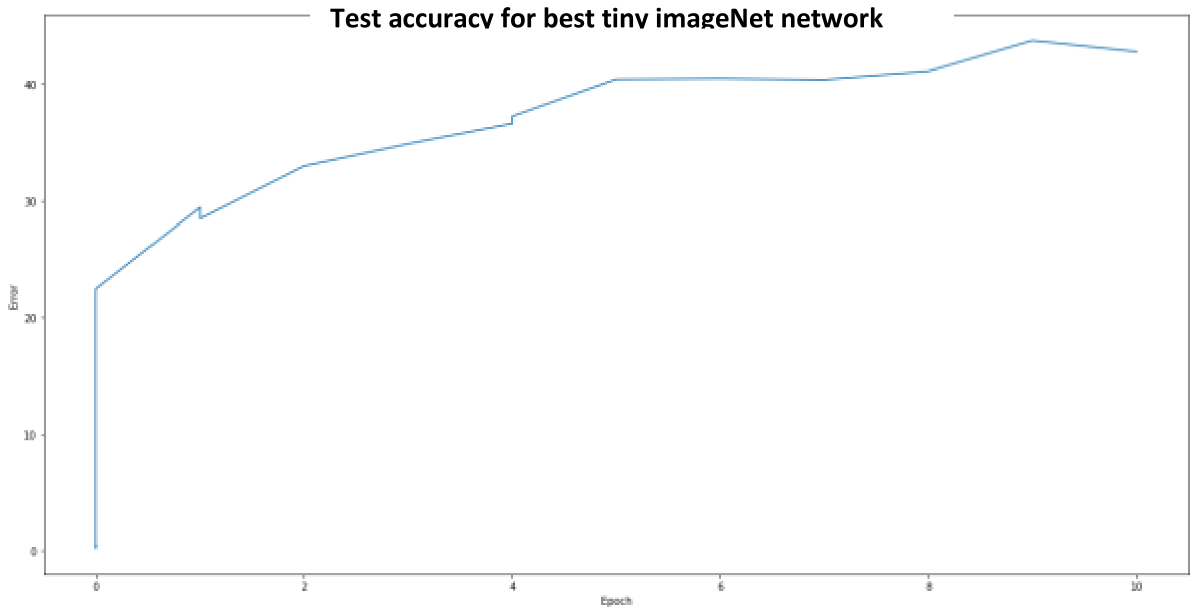
Detail description of accuracy:
Accuracy of plane: 86 %
Accuracy of car: 86 %
Accuracy of bird: 83 %
Accuracy of cat: 61 %
Accuracy of deer: 80 %
Accuracy of dog: 83 %
Accuracy of frog: 83 %
Accuracy of horse: 86 %
Accuracy of ship: 91 %
Accuracy of t ruck: 88 %

Training loss for best tiny imageNet network



Test loss for best tiny imageNet network





Network structure:

```
TinyImagnetNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU()
  (conv2_0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm2_0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2_0): ReLU()
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2_1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm2_1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2_1): ReLU()
  (conv2_2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm2_2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2_2): ReLU()
  (conv3_0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm3_0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu3_0): ReLU()
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4_0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm4_0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu4_0): ReLU()
  (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4_1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm4_1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu4_1): ReLU()
  (conv4_2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm4_2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu4_2): ReLU()
  (pool5): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  (flatten5): Flatten()
  (ff5_1): Linear(in_features=4096, out_features=200, bias=False)
)
```

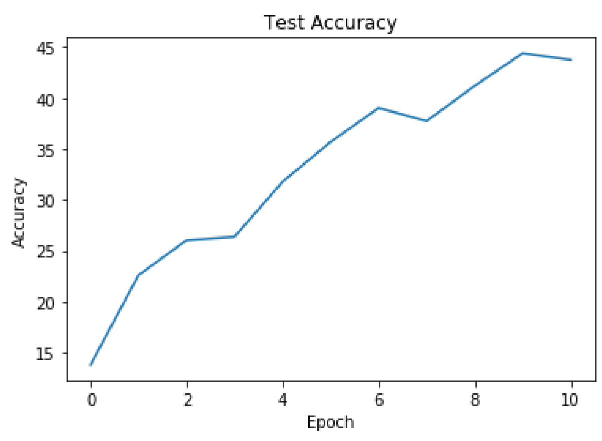
2. **Were you able to use larger/deeper networks on TinyImageNet than you used on CIFAR and increase accuracy? If so, why? If not, why not?**

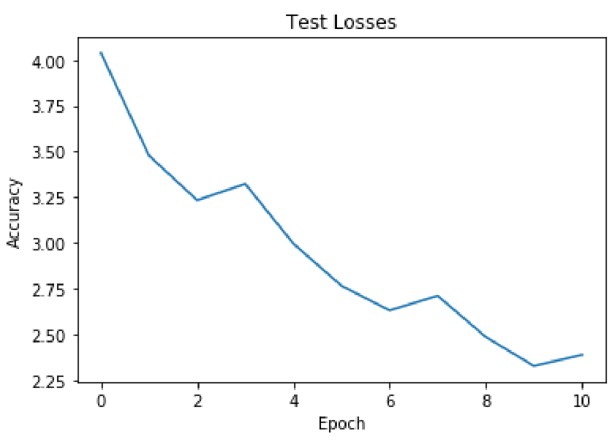
Settings:

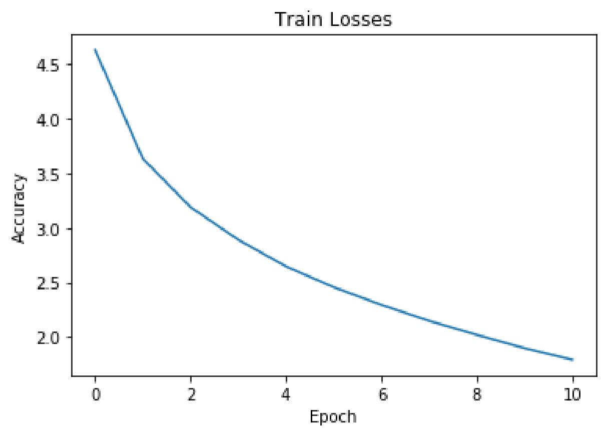
Batch size = 256
Test batch size = 10
Learning rate = 0.01
Momentum = 0.9
Weight decay = 0.0005
Number of epochs = 10
Seed = 0
Augmentations:

1. ToPilImage()
2. RandomHorizontalFlip()
3. ColorJitter()
4. RandomRotation(30)
5. ToTensor()
6. Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))

With a deeper convolution resnet network, I could not increase the efficiency. The efficiency of this deeper network is still the same peaking at 44% given that all other parameters are the same. By increasing the depth, I increases the number of features extracted from the images, but that might not be sufficient as I kept the dense fully-connected layer the same depth, which limits the expressivity of the network.







Network architecture:

```
TinyImagenetNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU()
  (conv2_0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm2_0): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2_0): ReLU()
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2_1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm2_1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2_1): ReLU()
  (conv2_2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm2_2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu2_2): ReLU()
  (conv3_0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm3_0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu3_0): ReLU()
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4_0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm4_0): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu4_0): ReLU()
  (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv4_1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm4_1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu4_1): ReLU()
  (conv4_2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm4_2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu4_2): ReLU()
  (conv5_0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm5_0): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu5_0): ReLU()
  (pool5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv6_0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm6_0): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu6_0): ReLU()
  (pool6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv6_1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm6_1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu6_1): ReLU()
  (conv6_2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (batchnorm6_2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu6_2): ReLU()
  (poolff): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (flattenff): Flatten()
  (linearff): Linear(in_features=512, out_features=200, bias=False)
)
```

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.**

If the images were twice as large, I could still use the same convolution layer because convolution does not care about the size, but I would need to change the number of connections between the end of the convolution network layer and the start of the fully-connected network layer for classification. Now, if we are facing a computation limit due to poor hardware, we could try to increase the number of pooling in the network, to reduce the dimension of the image. To compensate for that, we could double the number of filters that we use. I think that the accuracy would increase with a large image because more features will be present in the images.