homework2_colab Short Answer Questions

CIFAR

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.

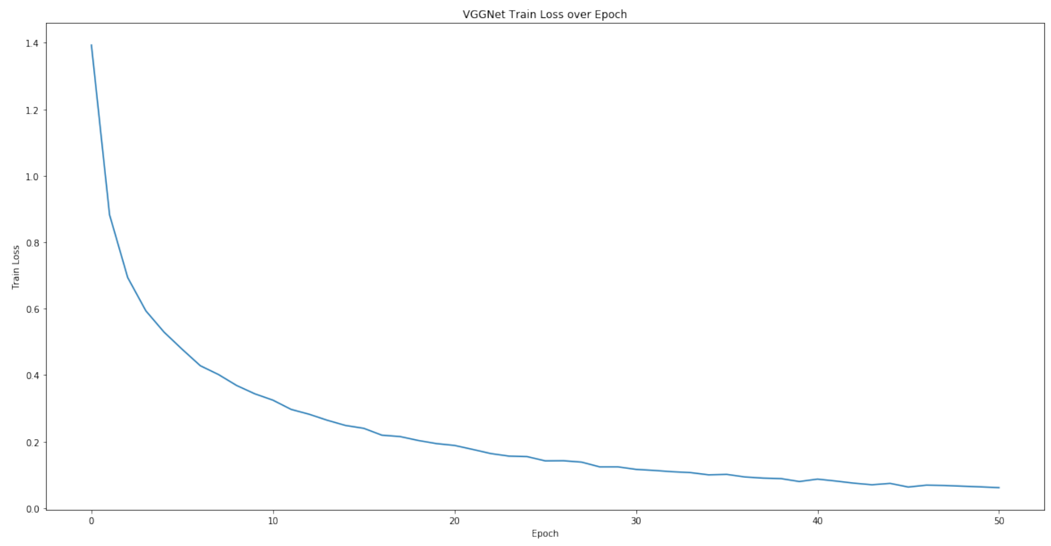
The best design achieved test accuracy 63% after 1 epoch, 80% after 5 epochs and 90% after 44 epochs.

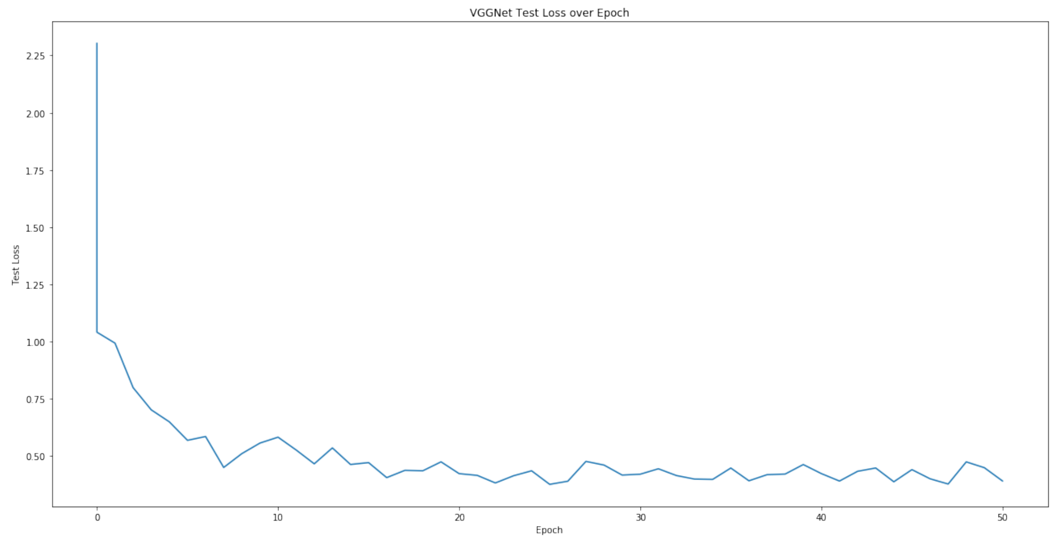
Network design (Reference to VGGNet):

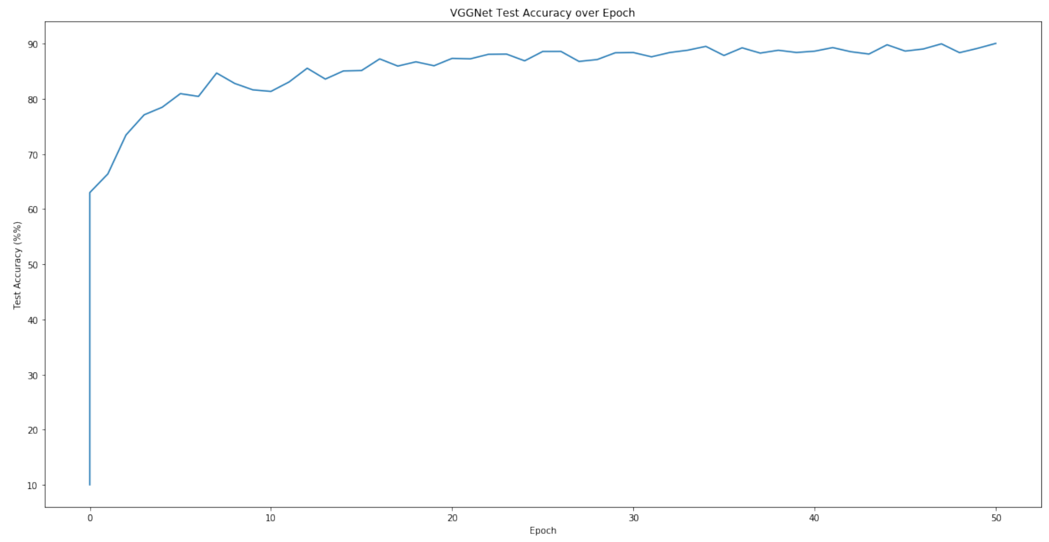
3*3Conv2d(3, 64) → BatchNorm2d → ReLU →
3*3Conv2d(64, 64) → BatchNorm2d → ReLU →
2*2MaxPool2d →
3*3Conv2d(64, 128) → BatchNorm2d → ReLU →
3*3Conv2d(128, 128) → BatchNorm2d → ReLU →
2*2MaxPool2d →
3*3Conv2d(128, 256) → BatchNorm2d → ReLU →
3*3Conv2d(256, 256) → BatchNorm2d → ReLU →
2*2MaxPool2d →
3*3Conv2d(256, 512) → BatchNorm2d → ReLU →
3*3Conv2d(512, 512) → BatchNorm2d → ReLU →
2*2MaxPool2d →
3*3Conv2d(512, 512) → BatchNorm2d → ReLU →
3*3Conv2d(512, 512) → BatchNorm2d → ReLU →
2*2MaxPool2d → 1*1AvgPool2d →
Linear(512, 10)

batch_size = 256
epochs = 50
learning_rate = 0.01
momentum = 0.9
weight_decay = 0.0005

Final train loss: 0.061487658633565416
Final test loss: 0.39041937673091887
Final test accuracy: 90.03%
Plots:


VGGNet Train Loss over Epoch
Train Loss
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
10
20
30
40
50


VGGNet Test Loss over Epoch
Test Loss
Epoch
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
0
10
20
30
40
50


VGGNet Test Accuracy over Epoch
Test Accuracy (%%)
Epoch
10
20
30
40
50
60
70
80
90
0
10
20
30
40
50

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

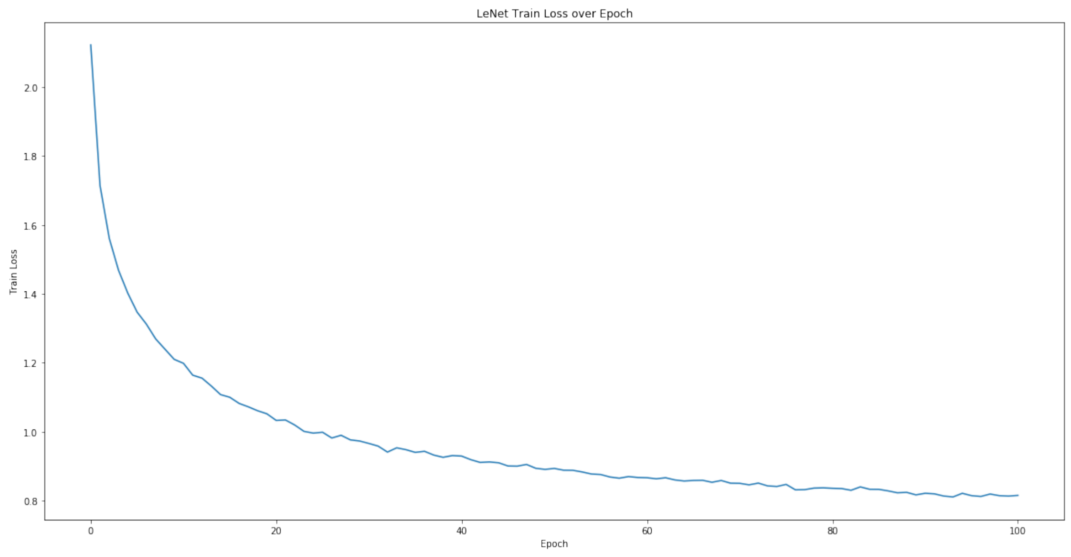
The worst design achieved test accuracy 33% after 1 epoch and 73% after 100 epochs.

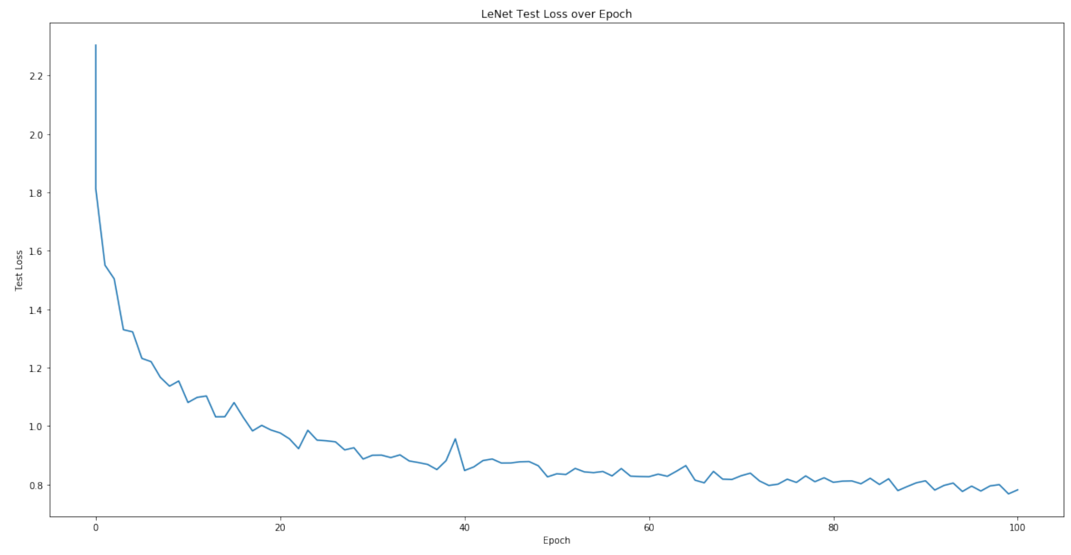
Network design (Reference to LeNet, see notebook for details):

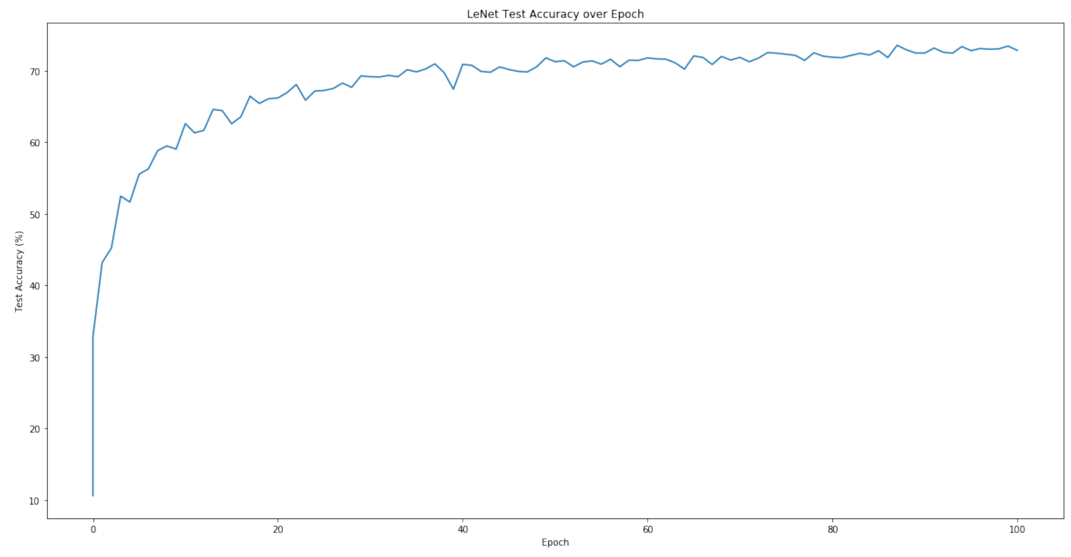
5*5Conv2d(3, 6) → ReLU →
2*2MaxPool2d →
5*5Conv2d(6, 16) → ReLU →
2*2MaxPool2d →
Flatten →
Linear(16*5*5, 120) → ReLU →
Linear(120, 84) → ReLU →
Linear(84, 10)

batch_size = 256
epochs = 100
learning_rate = 0.01
momentum = 0.9
weight_decay = 0.0005

Final train loss: 0.8148480751076523
Final test loss: 0.7816184884309769
Final test accuracy: 72.81%
Plots:







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

The best design is referred to VGGNet, reaching 90% test accuracy after only 50 epochs. It uses 3*3 convolution layers with 2*2 max pooling. This design has deeper convolution layers and convert to linear classifier by average pooling. The worst design is a simple network referred to LeNet, with % test accuracy after 100 epochs. It uses two 5*5 convolution layers with 2*2 max pooling and convert to linear classifier by flattening. This design has a tradeoff between the simpler network architecture and lower test accuracy. In this case, the best design can beat the worst deign at very beginning, with 80% accuracy after 5 epochs which cannot be reached in the worst design. The parameters are almost the same except for the number of epochs. The network design is the main factor of the difference in network performance here. The best design has more convolution layers and uses average pool rather than flattening. The worst network design is not deep enough to detect the image details accurately.

TinyImageNet

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.

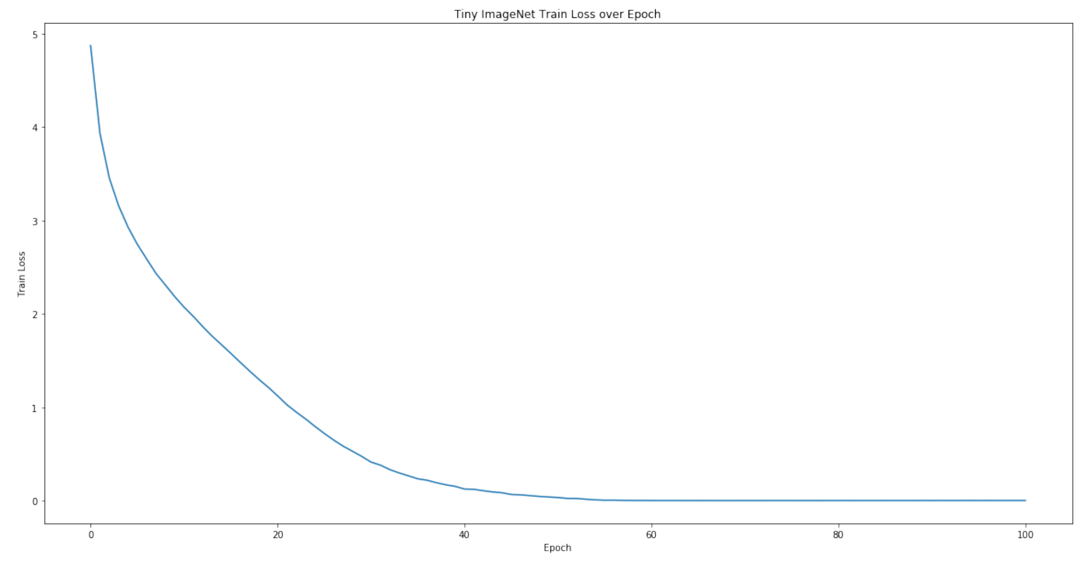
The best design achieved test accuracy 6% after 1 epoch, 40% after 13 epochs and 47% after 100 epochs.

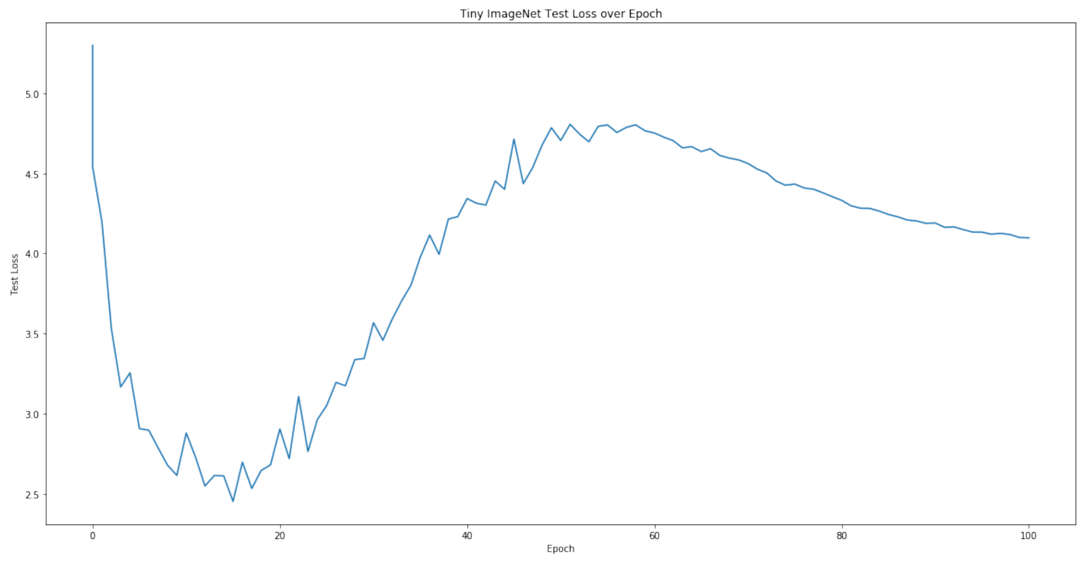
Network design (Reference to VGGNet, see notebook for details):

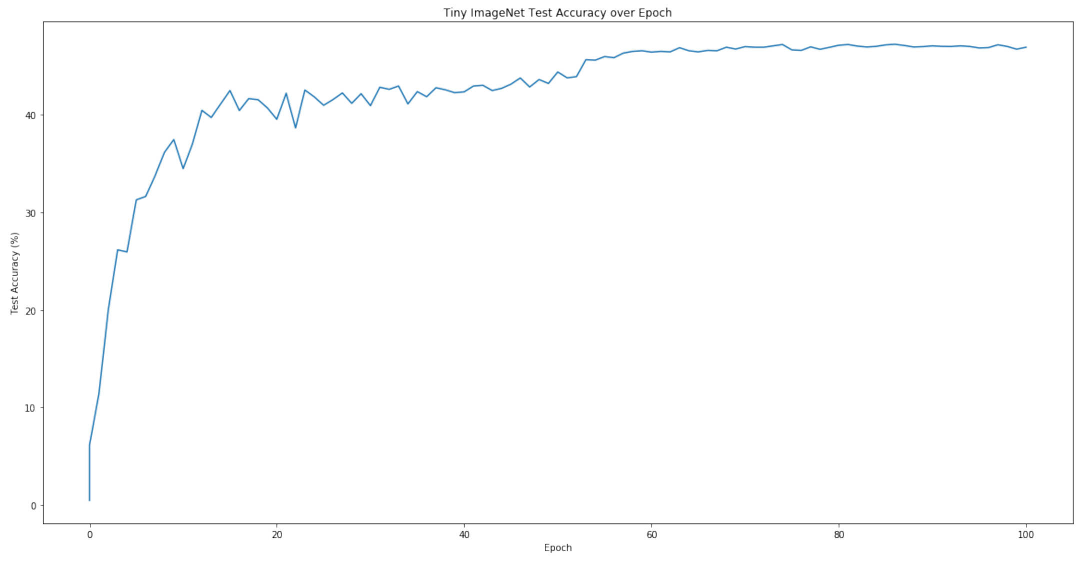
3*3Conv2d(3, 32) → BatchNorm2d → ReLU →
3*3Conv2d(32, 32) → BatchNorm2d → ReLU →
2*2MaxPool2d →
3*3Conv2d(32, 64) → BatchNorm2d → ReLU →
3*3Conv2d(64, 64) → BatchNorm2d → ReLU →
2*2MaxPool2d →
3*3Conv2d(64, 128) → BatchNorm2d → ReLU →
3*3Conv2d(128, 128) → BatchNorm2d → ReLU →
2*2MaxPool2d →
3*3Conv2d(128, 256) → BatchNorm2d → ReLU →
3*3Conv2d(256, 256) → BatchNorm2d → ReLU →
2*2MaxPool2d →
Flatten →
Linear(256*4*4, 1024) →
Linear(1024, 512) →
Linear(512, 256) →
Linear(256, 200)

batch_size = 256
test_batch_size = 10
epochs = 100
learning_rate = 0.01
momentum = 0.9
weight_decay = 0.0001

Final train loss: 0.0037294732569895512
Final test loss: 4.097113926500082
Final test accuracy: 46.9%
Plots:


Tiny ImageNet Train Loss over Epoch
Train Loss
Epoch
0
1
2
3
4
5
0
20
40
60
80
100


Tiny ImageNet Test Loss over Epoch
Test Loss
Epoch
2.5
3.0
3.5
4.0
4.5
5.0
0
20
40
60
80
100


Tiny ImageNet Test Accuracy over Epoch
Test Accuracy (%)
Epoch
0
10
20
30
40
0
20
40
60
80
100

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?

I was able to build a larger and deeper network on TinyImageNet than the one used on CIFAR. For both TinyImageNet and CIFAR, I implemented a modified version of VGGNet. Using the CIFAR network design on TinyImageNet could only reach 30% test accuracy after 20 epochs. Since the tiny image data is much larger and there are more targeting labels, I could use deeper network on TinyImageNet. Based on the network design for CIFAR, I used 3*3 convolution layers with 2*2 max pooling. After the flatten layer, I added more linear layers to get the 200-image-label classifier. The improved network increased the accuracy to 41% after 20 epochs. The final test accuracy is 46.9%.

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.

For larger images, I may add more convolution layers to gradually increase the number of output channels. The pooling stride may also need to be updated. After a flatten layer of an average pooling layer, I would increase more linear layers to gradually decrease the output size. Doing so can ideally keep or improve the accuracy for larger images. For smaller images, I may keep the current network design or do the inverse change, gradually reducing the number of layers. I expect the accuracy would be closed or improved.