Short answer questions in colab:

Please answer these questions, and put the answers in a file called homework2_colab.pdf in your repository.

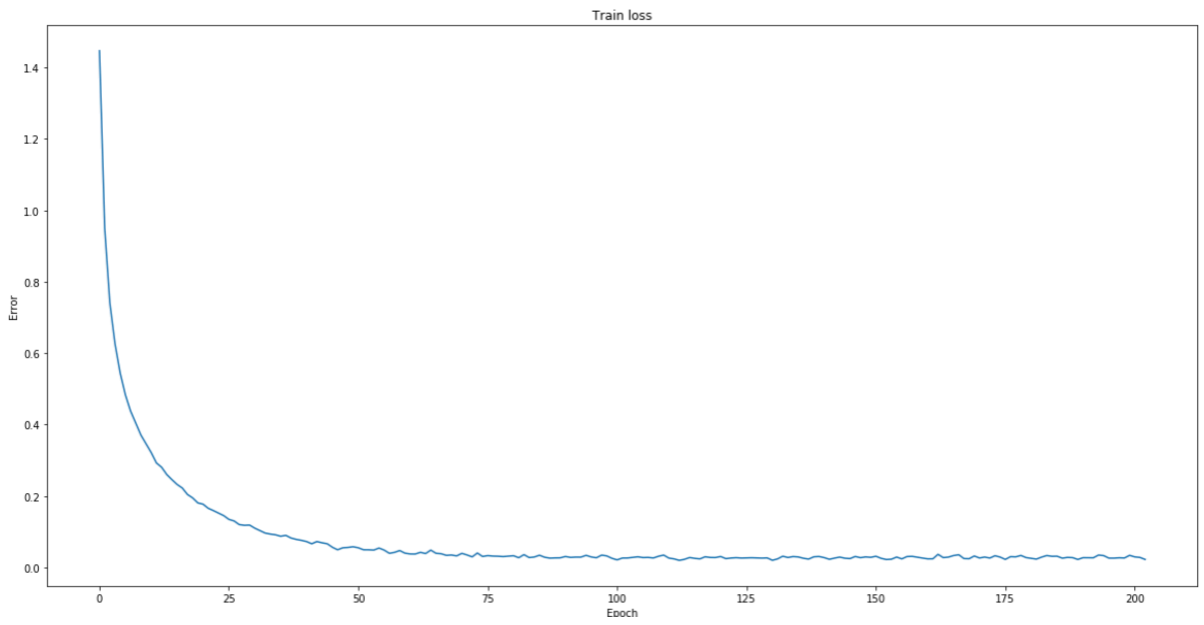
CIFAR

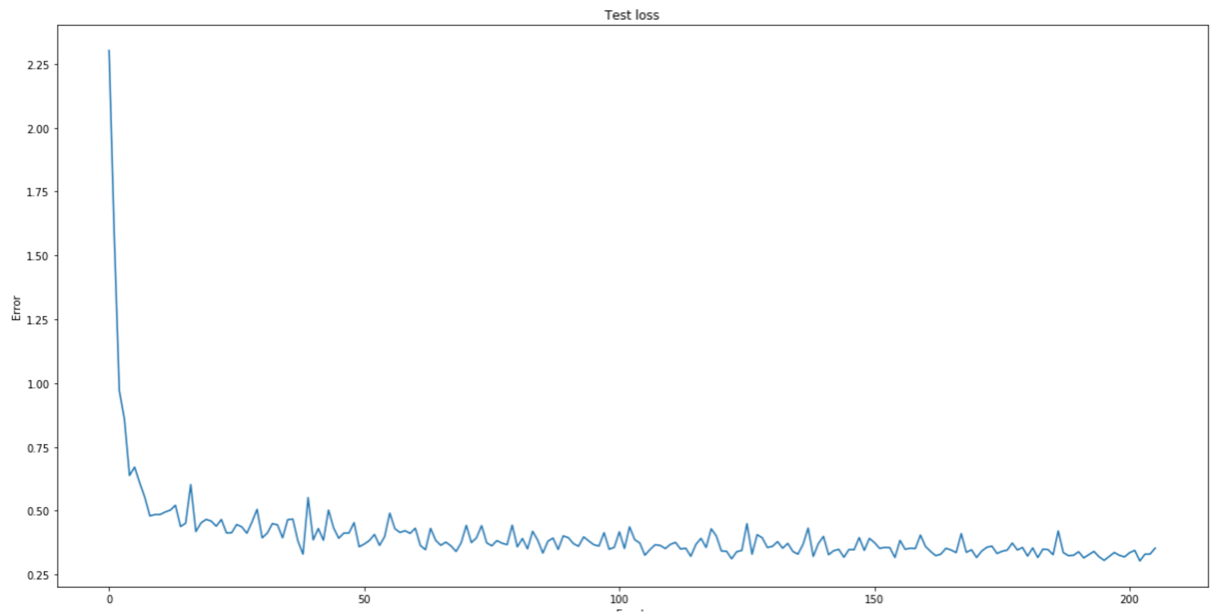
1. What design that you tried worked the best? This includes things like network design, learning rate, batch size, number of epochs, and other optimization parameters, data augmentation etc. What was the final train loss? Test loss? Test Accuracy? Provide the plots for train loss, test loss, and test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

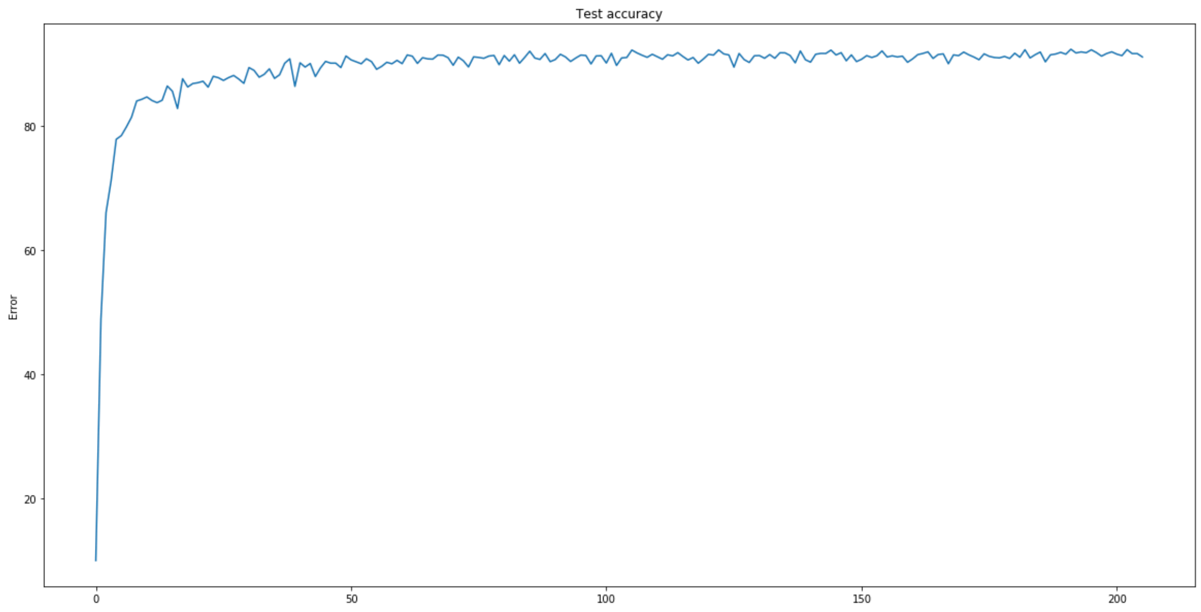
   The best is resnet18 network. Learning rate is 0.01, batch size is 256, epochs are 200.

   The final train loss is 0.024359 . Test loss is 0.3293. Test Accuracy is 92%.

   ### print :





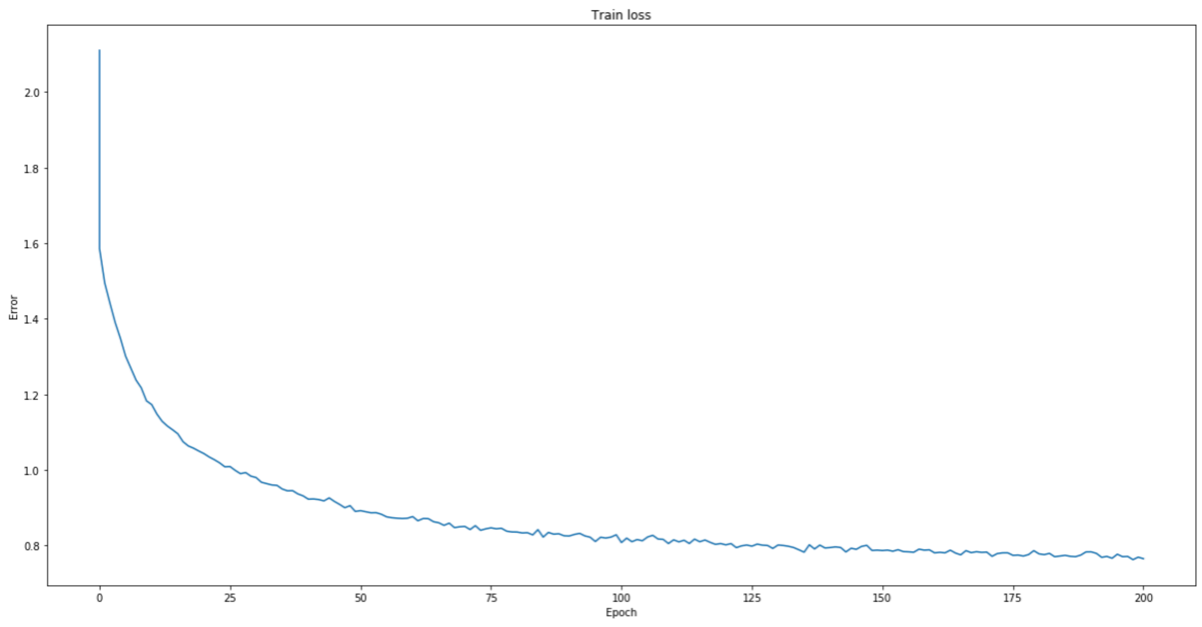


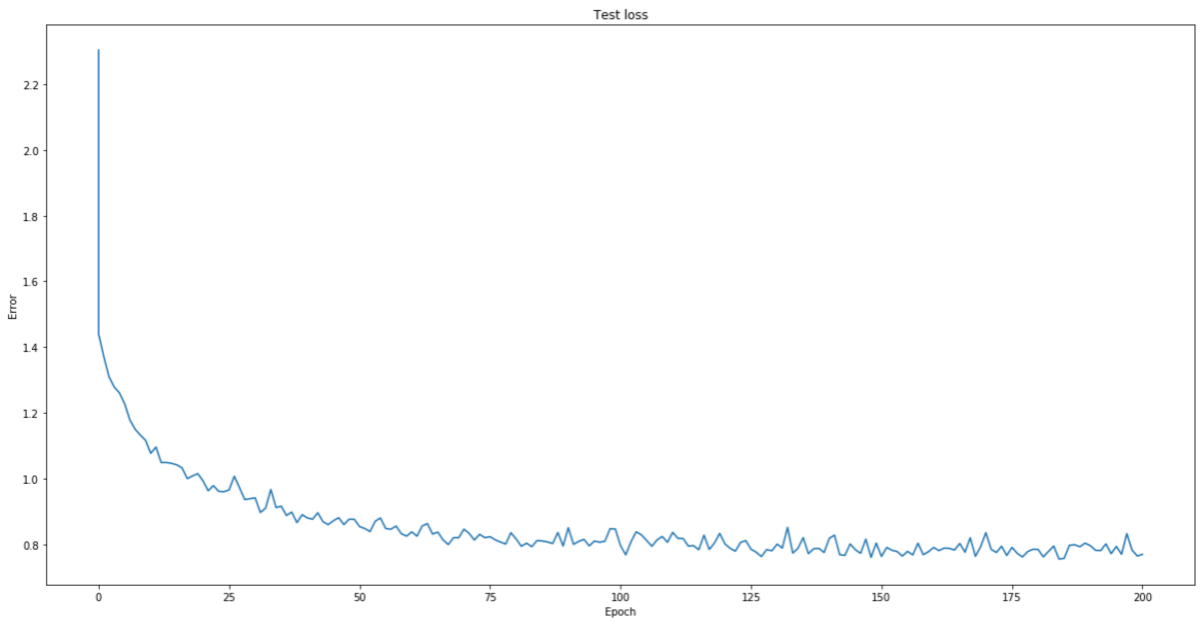
2. What design worked the worst (but still performed better than random chance)? Provide all the same information as question 1.

   Just use the 3- layer convolution net with bitch normalization with Learning rate is 0.01, batch size is 256, epochs are 200 will gain 73% finally.

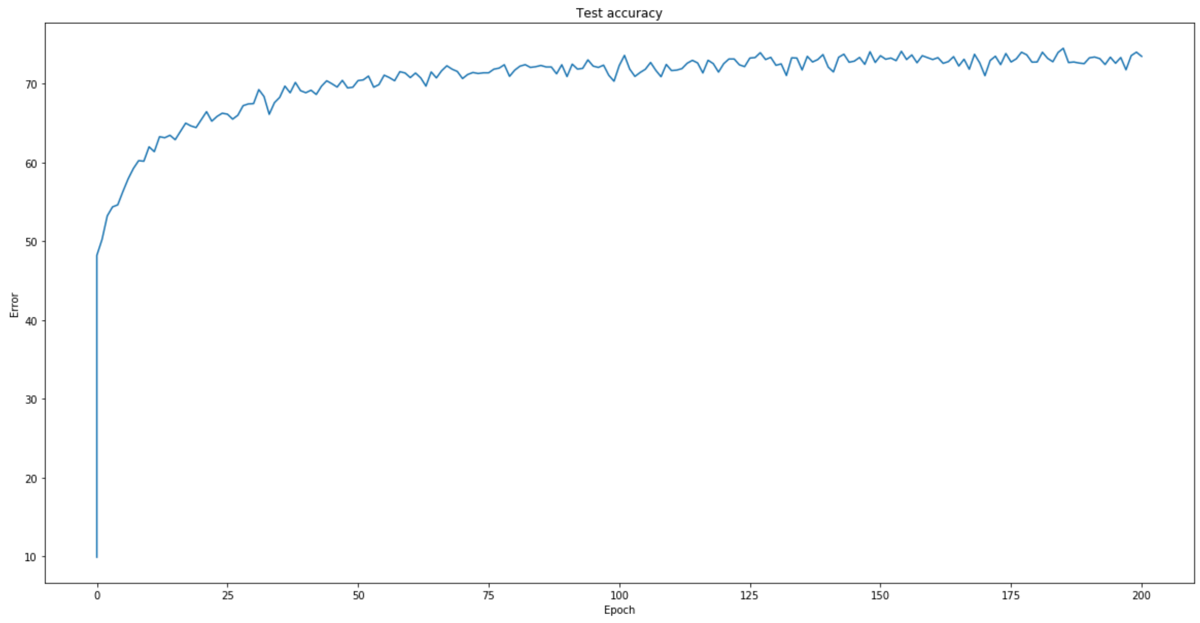
   ## Print:

```
ed Oct 30 05:12:22 2019 Train Epoch: 200 [0/50000 (0%)]        Loss: 0.683675
ed Oct 30 05:12:29 2019 Train Epoch: 200 [25600/50000 (51%)]   Loss: 0.765592

est set: Average loss: 0.7707, Accuracy: 7344/10000 (73%)

aved /gdrive/My Drive/colab_files/homework2/cifar/checkpoints/200.pt

aved /gdrive/My Drive/colab_files/homework2/cifar/checkpoints/200.pt

aved /gdrive/My Drive/colab_files/homework2/cifar/checkpoints/200.pt
```





Epoch



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

   Two reasons. Best the best one have more layers and use the residual to void the error exploration or vanishing. So it can work better.

TinyImageNet

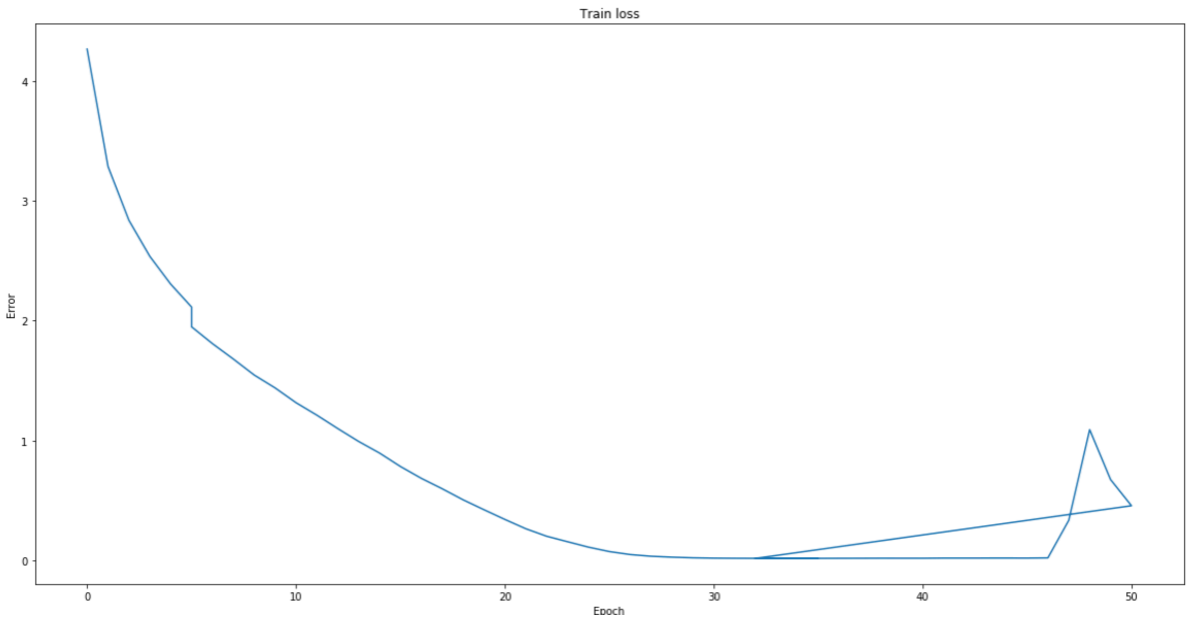
1. What design that you tried worked the best? How many epochs were you able to run it for? Provide the same information from CIFAR question 1.

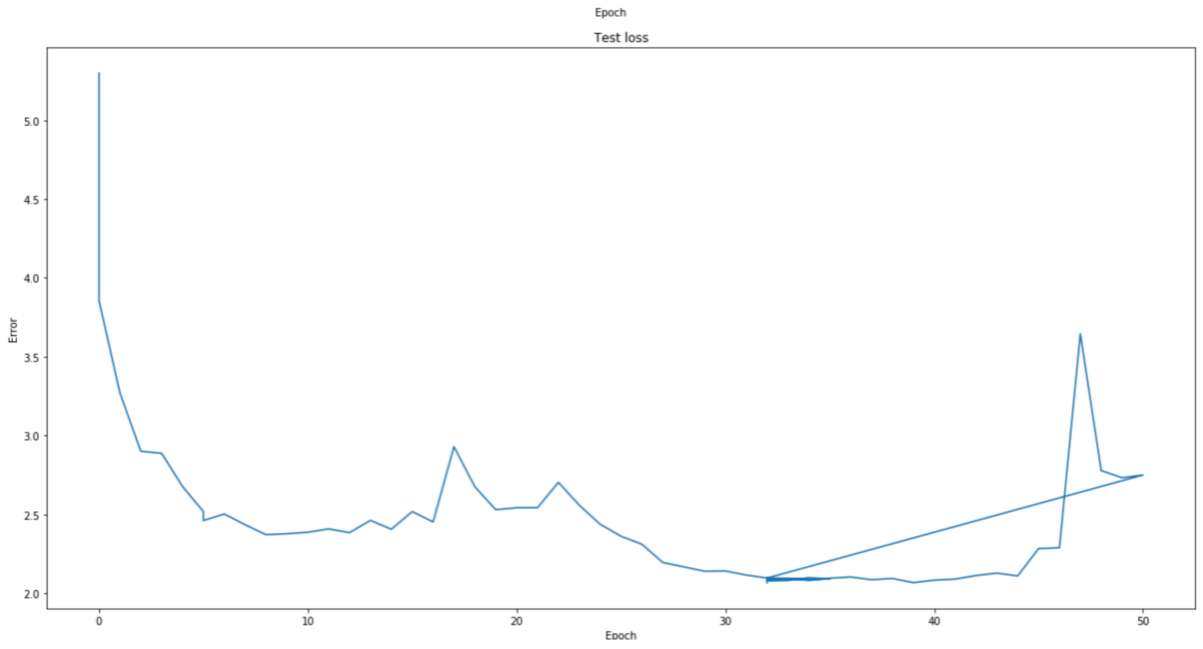
   I use the resnet18, learning rate is 0.01, batch size is 256, epochs are 32.The final train loss is 0.017059. Test loss is 2.0665. Test Accuracy is 55%.

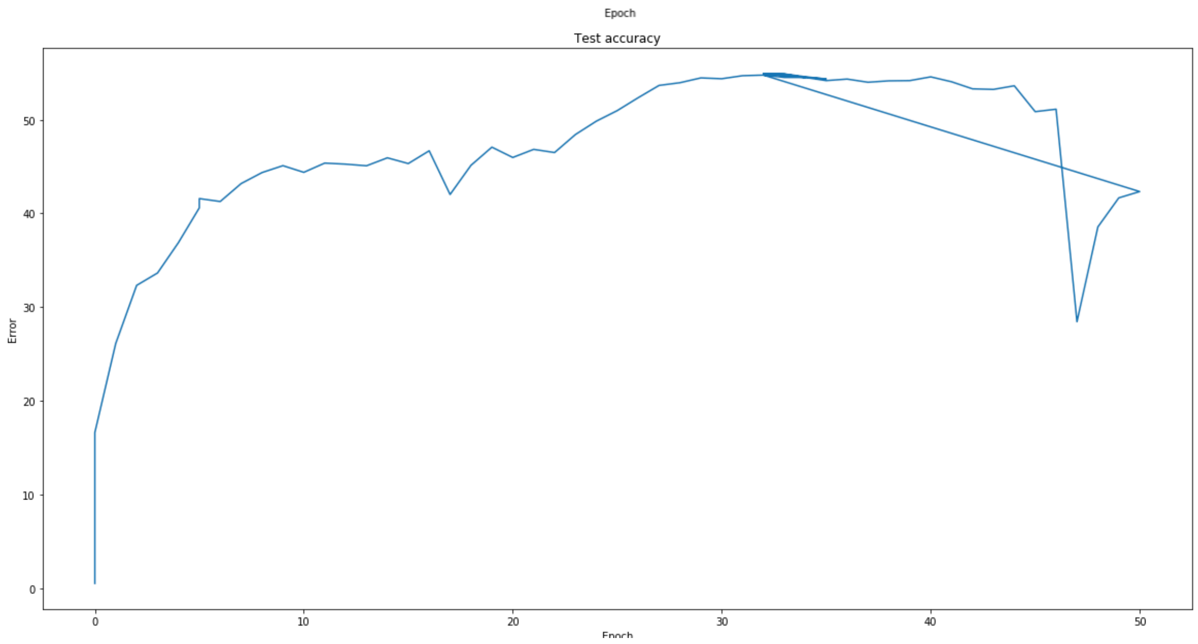
   ## print:

   (sorry for the retrain line ,this makes a little confusing), but we still can see it has go over the 55%.

```
Test set: Average loss: 2.0665, Accuracy: 4386/8000 (55%)

Saved /gdrive/My Drive/colab_files/homework2/tiny_imagenet/checkpoints/032.pt
```







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?

   I use the 18 layers residual network, thanks to usinng the residual to void the error exploration or vanishing. So it can work. (compare with 3-layer convolution neural network.) and because the class are more than first one, we should use the deeper one the increase the accurarate.

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 image is larger, we can use the segmentation first to find the key feature of the picture, but in a smaller size than before. Then use the Spatial Pyramid Pooling to deal with featured picture to pooling with same network. The second idea is to change the fully connected layer to keep more features.

   If the size is really small, we can go deeper network like the resnet105 to increase the accurate. (already fit the input size of the neural network). Or we can use the net with several output to decrease the computational difficulty. (like the google network) The second idea is to change the fully connected layer, because the performance maybe worse in small input size.