Report

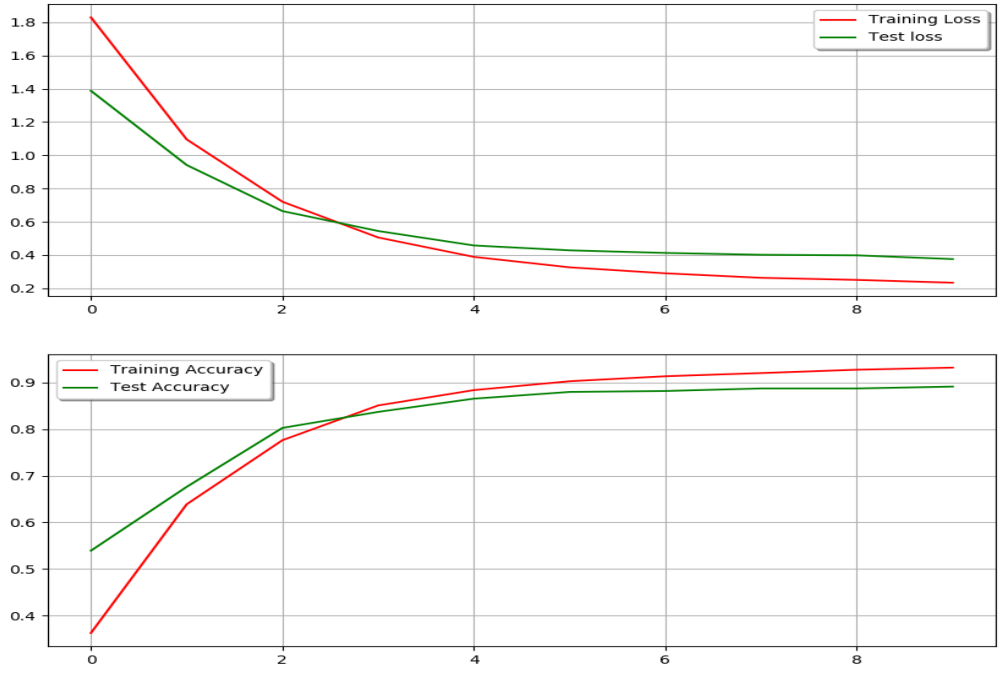
Task1

I designed three neutron networks: Fully, Locally, and CNN. I trained all three models in 10 epochs, because it is stable after 10 epochs.

1. Fully Connected that has five layers:

For n-1 and n-1, any node in n-1 is connected to all the nodes in n-1. That is, when each node in the n-1 layer is evaluated, the input of the activation function is the weight of all nodes in the n-1 layer. If the number of data is too much, the speed of training will be very slow. But we only have 7291 train data sample and 2007 test data, it's fine. As the result, we can see the curve, that has slow learning.

```
model.add(Dense(512,activation='relu', input_shape=(256,), ))
model.add(Dropout(0.2))
model.add(Dense(512,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10,activation='softmax'))
```

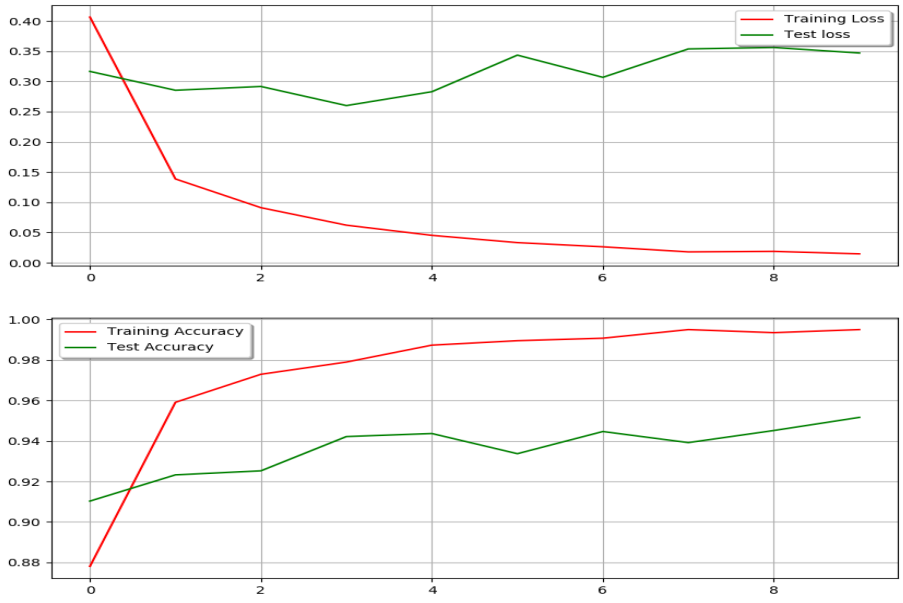


This is the result of Fully.

The train accuracy tends to 0.97. The test accuracy tends to 0.90. And the speed of training is fast.

2. Locally Connected, that has six layers:
   Locally connection is considered to cut off the direct connection between two nodes, which greatly reduces the amount of calculation during training. Before doing that, I think the training is fast because they reduce many neutrons. So, we can see the curve.

```
model.add(LocallyConnected2D(64,(3,3), input_shape=(16,16,1),init='VarianceScaling'))
model.add(Activation('relu'))
model.add(LocallyConnected2D(32,(3,3)))
model.add(Flatten())
model.add(Dense(10))
model.add(Activation('softmax'))
```



This is the result of locally connected. The training locally connected is slower than fully. And the test accuracy is not good as fully connected. And the curve is not stable. The train accuracy tends to 0.85. the test accuracy tends to 0.96. I found the reason why the speed of training is slow. I think it's because there is no weight share, every time the network need to get new weight that occupy a lot of space.

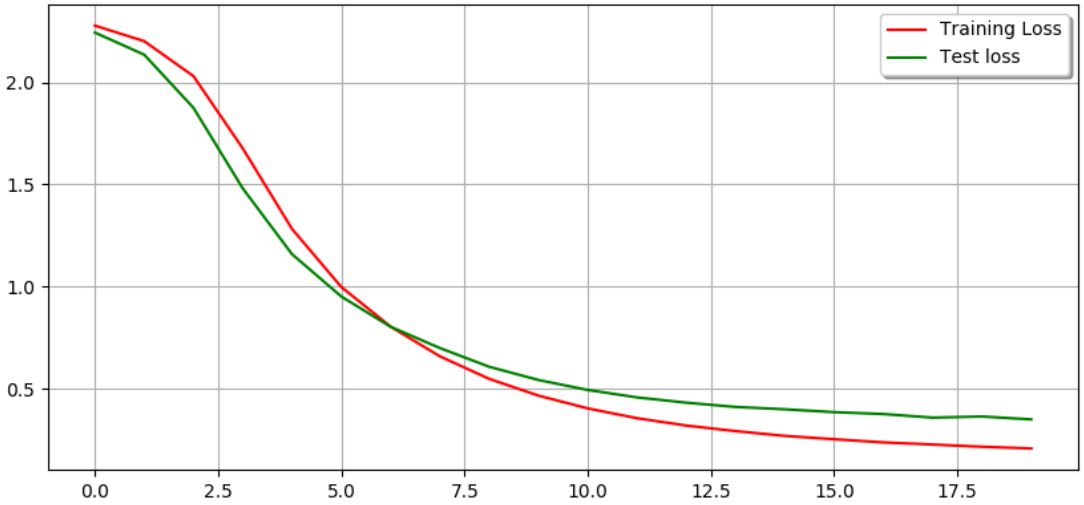
3. CNN, that has two Convolution layers, two polling layers, and two fully layers.
Features in a direction can be extracted by sharing the same parameter, so the amount of training will be much smaller than that of a fully connected neural network. I think the speed of training is not slow like locally connected because there is weight share. And it has convolution layers that can extract the different characteristics of the input and pooling layers that can reduce the characteristics and fully connected layer that can combine all local features into global features. And I think the accuracy is better than locally and fully.

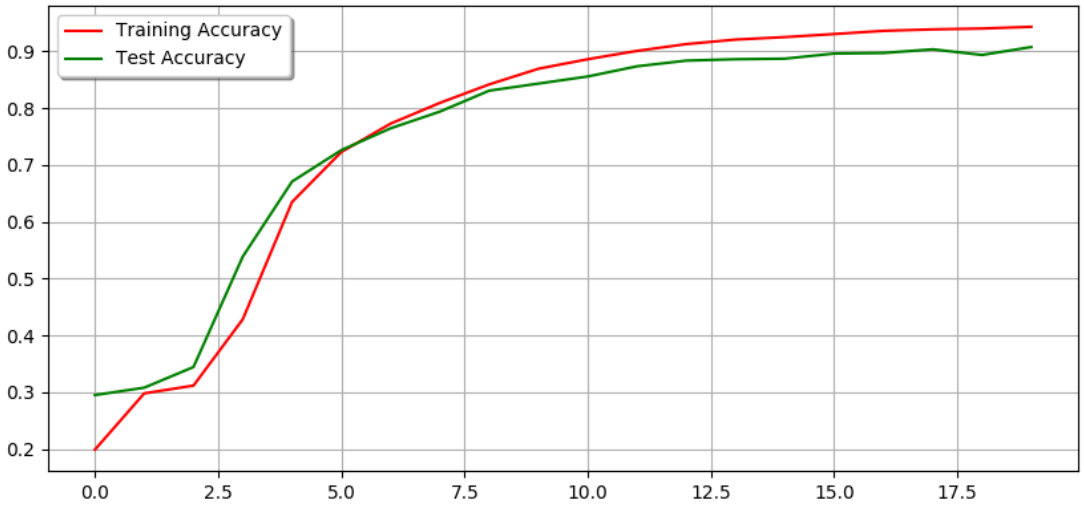
```
model.add(Convolution2D(
 nb_filter=32,
nb_row=5,
nb_col=5,
border_mode='same',
input_shape=(1,16,16),
kernel_initializer=initializers,
bias_initializer='zeros'))
model.add(Activation('relu'))
model.add(MaxPooling2D(
        pool_size=(2,2),
        strides=(2,2),
        border_mode='same',#padding method
        ))
model.add(Convolution2D(64,5,5,border_mode='same',
        kernel_initializer=initializers,
        bias_initializer ='zero'
        ))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2,2),border_mode='same'))

#fully
model.add(Flatten())
model.add(Dense(1024,kernel_initializer=initializers,bias_initializer='zeros'))
model.add(Activation('relu'))

#fully
model.add(Dense(10,))
model.add(Activation('softmax'))
```





This is the result of CNN, the accuracy is stable. The train accuracy tends to 0.95 and the test accuracy tends to 0.91. The accuracy is better than fully connected but not good as locally but running is faster than locally.

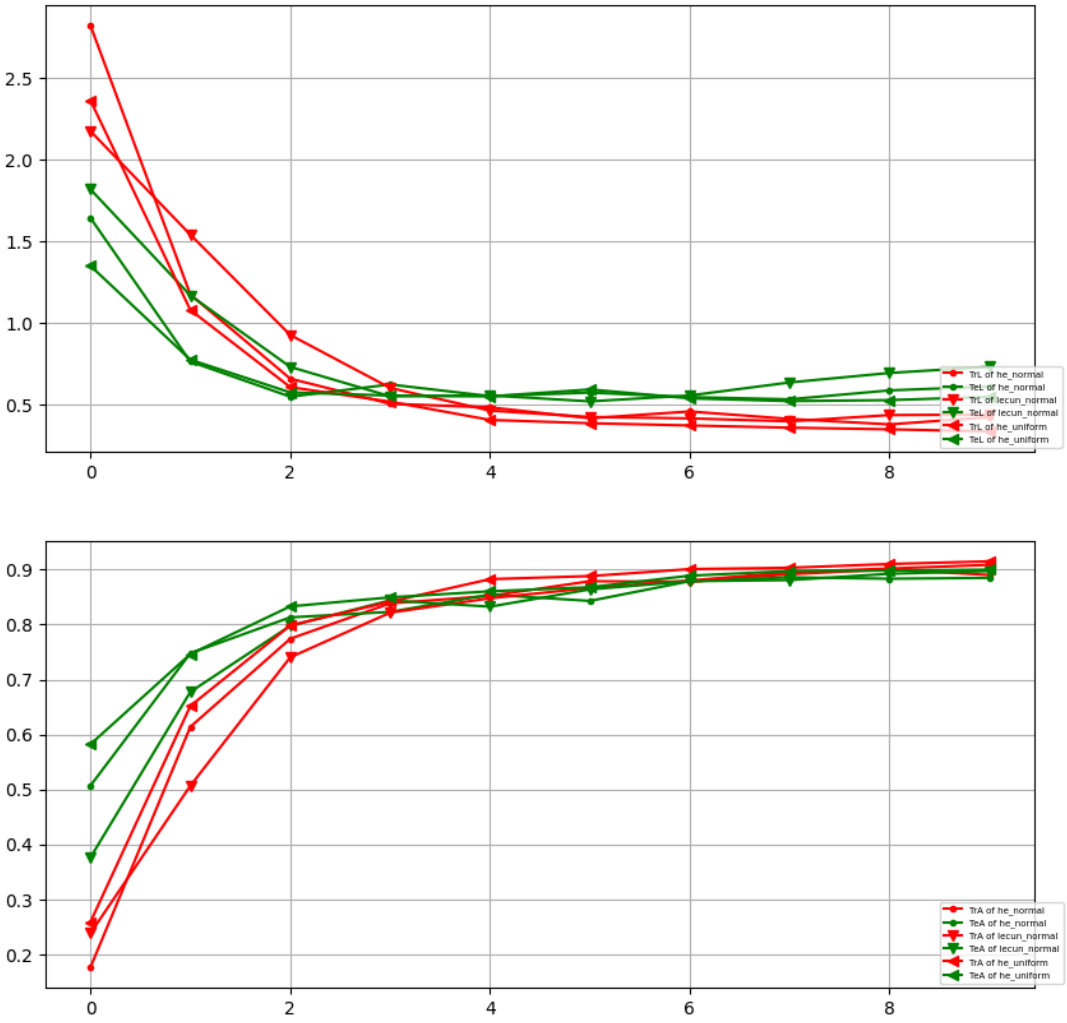
Task 2:

After that, I did some optimization in three networks: parameter, learning rate, batch size, and momentum. And do many experiments to find best one.

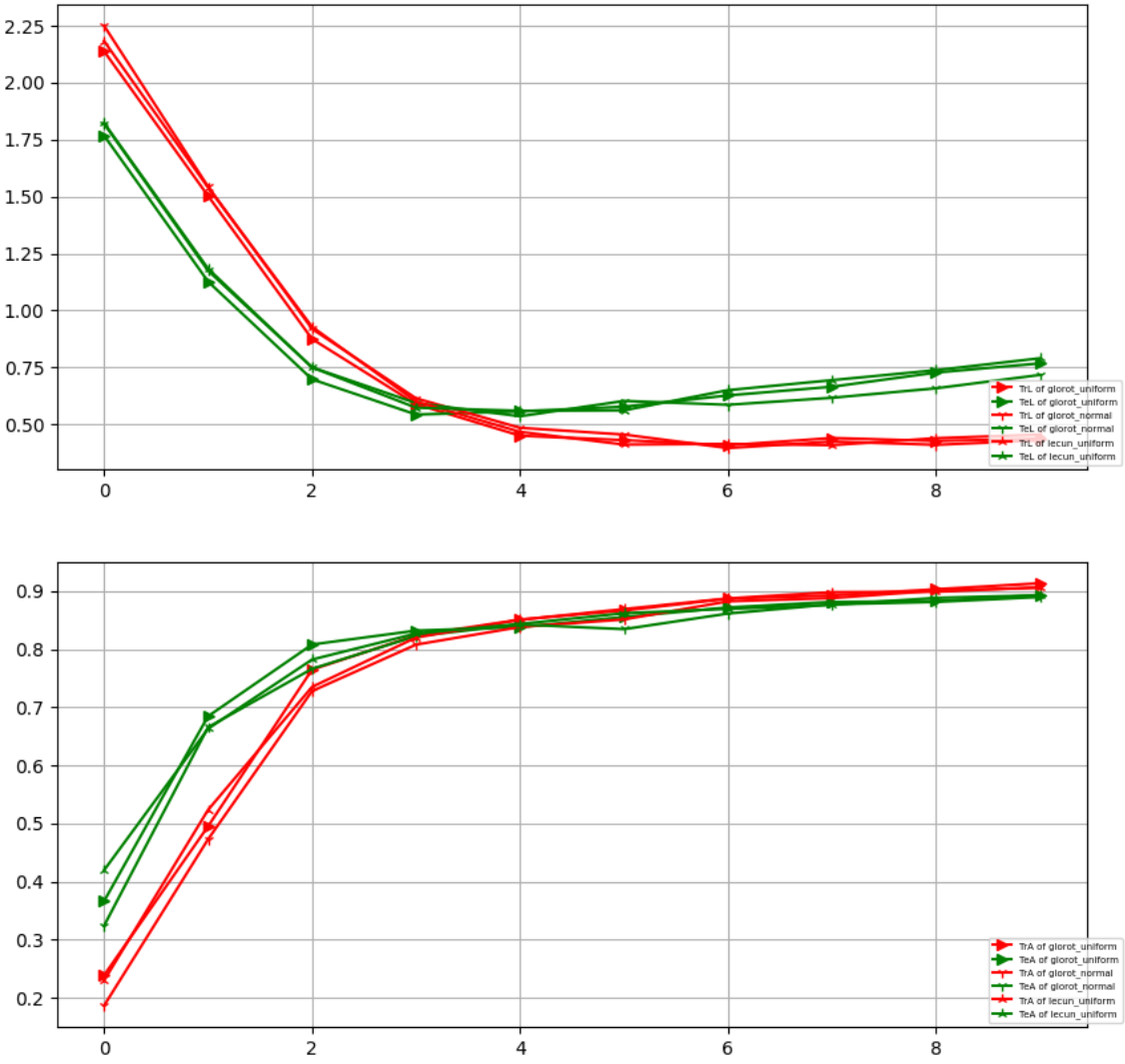
The first is fully. I choose 13 kind of different initializes, and finally I found the he_normal is best parameter. I get the result of (1)he_normal, (2)lecun_normal, (3)he_uniform, (4)glorot_uniform, (5)glorot_normal, (6)lecun_uniform,(7) orthogonal, (8)VarianceScaling, (9)TruncatedNormal, (10)RandomUniform, (11)RandomNormal, (12)Constant, (13)ones. And I compare three by three and find some better and compare again.

Before I did experiment to find best one. I searched some information about parameter initializers. There are some requirements: 1. Parameters cannot all be initialized to 0, nor can they all be initialized to the same value because of symmetry failure. So I think the (12) constant, and (13) ones will get bad accuracy. 2. It is best to ensure that the mean value of the parameters initialized is 0, plus or minus is staggered, and the number of positive and negative parameters is roughly equal. He_normal and lecun_normal goes from being centered at 0, so I think these two has good accuracy. 3. Initialization parameters should not be too large or too small.

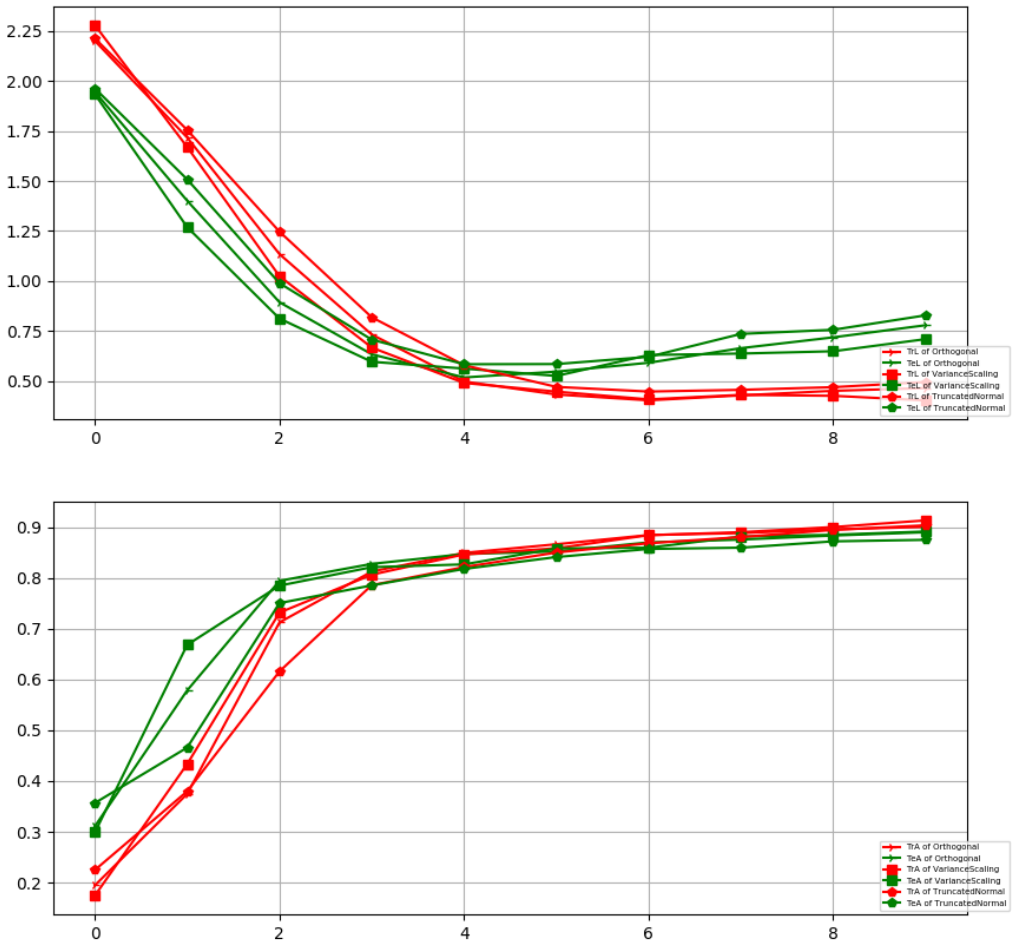
And l learned some initialization ways. Glorot is not adapt to relu, sigmoid and not linear activation function. So, I think (4) glorot_uniform and (5) glorot_normal is not good accuracy. He initializer is good for linear activation function, so (1) (3) will be good. The random way is not stable. And varianceScaling can adapt the shape of the target tensor, so it will also be good. So, let’s do it.



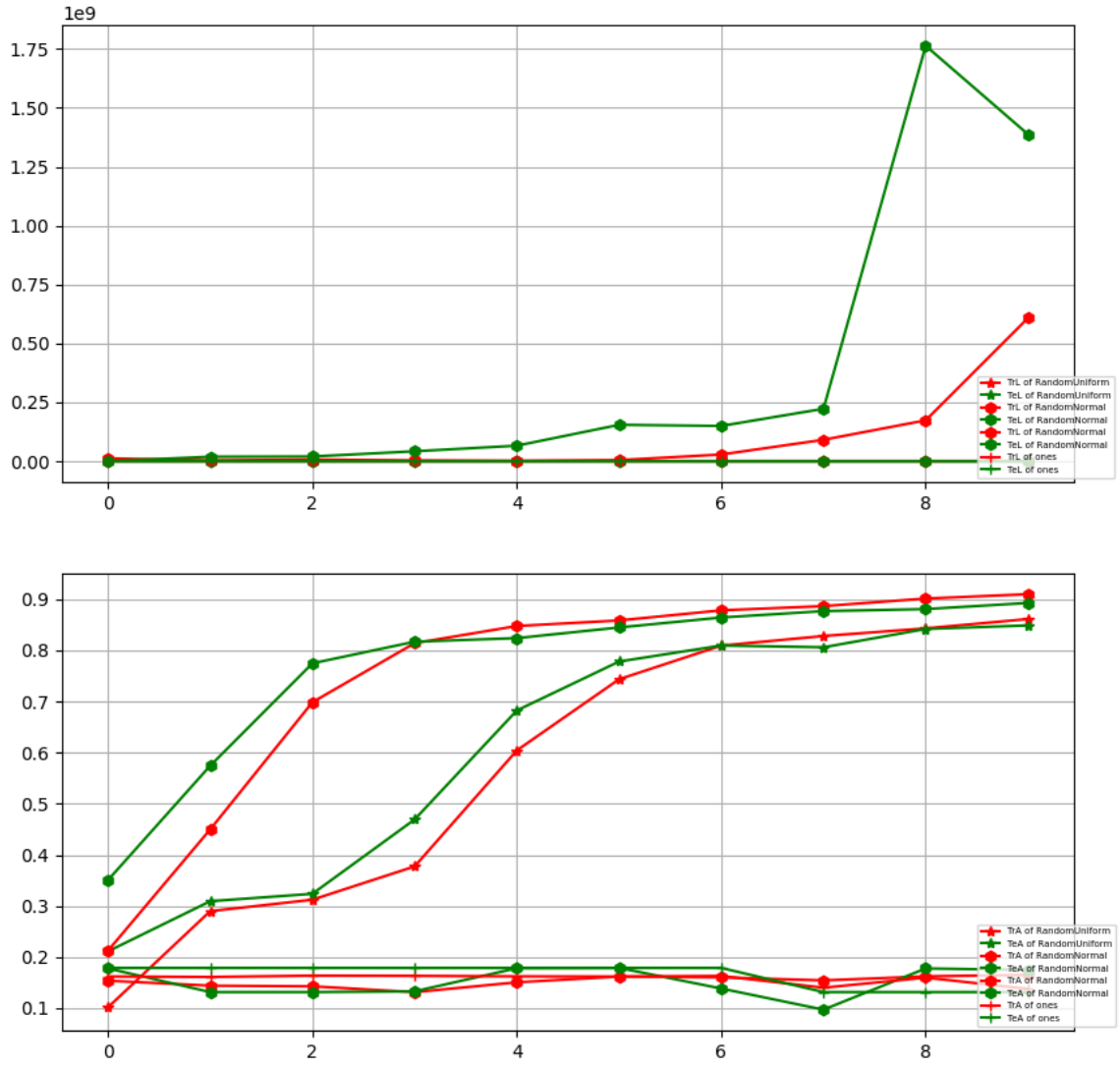
This is the result of (1) (2) (3). We can see the test accuracy is better than train accuracy. The lecun_normal learn fast and the result is good. He_uniform learn slow, but the accuracy is best. The he_normal learn as fast as lecun_normal, but the accuracy is bad in these three. I think the he_uniform is effective in these three. It prove he function is good.



This is the result of (4) (5) (6). We can see the accuracy of them is similar. The glorot_normal learn a little faster. So, I choose glorot_normal. It proves the normal is good than others because it goes from zero. But research shows the glorot is not good for not linear activation. I don't know why the glorot also has good accuracy.

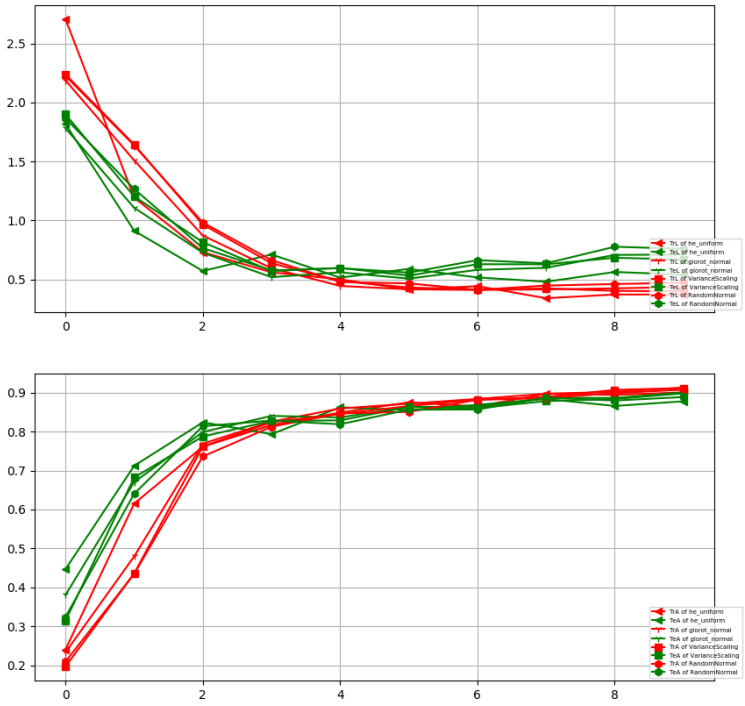


This is the result of (7) (8) (9). The VarianceScaling learn faster than others, and the performance is also good as others. The performance of TruncatedNormal is a little bad in these three and it learn slowest. We choose VarianceScaling. Orthogonal's accuracy is also good but it learn slower than VarianceScaling. So I choose VarianceScaling.



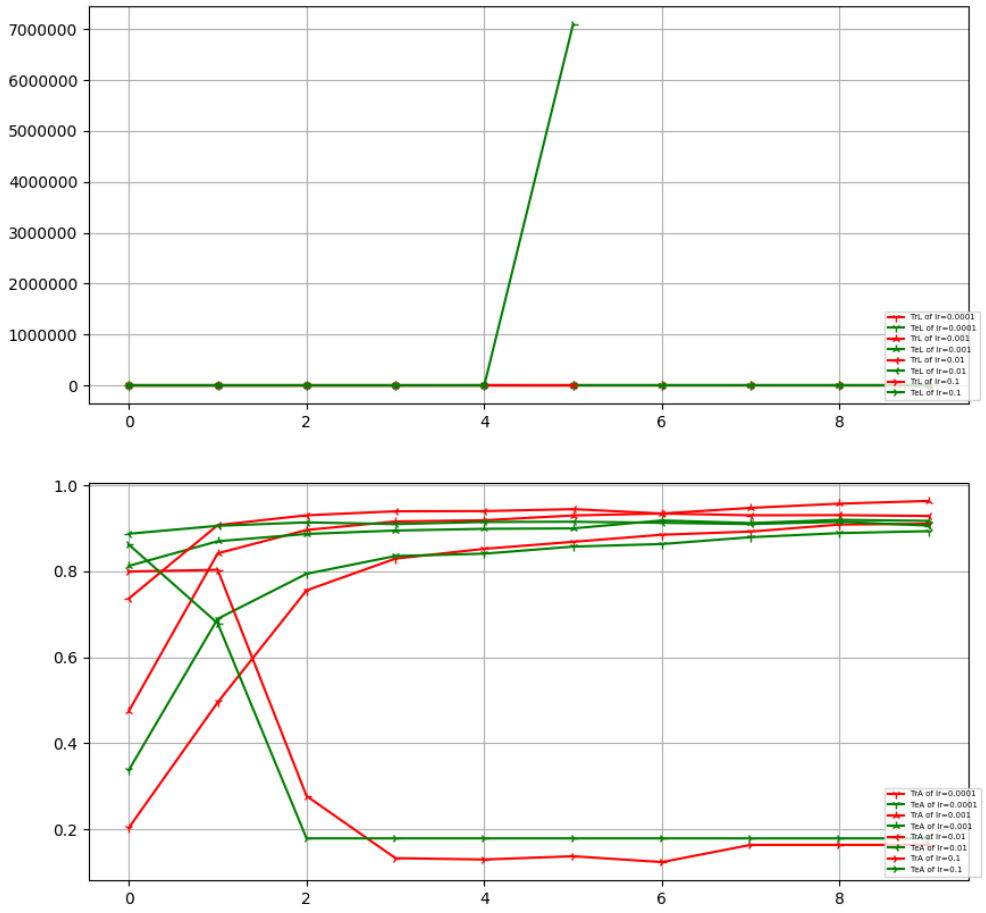
This is the result of (10) (11) (12) (13). Constants(value=10) and ones is bad performance. It's obvious that the RandomNormal is the best one. It is effective. It proves that the constant value is not good.

The last, I compared the (3) (5) (8) (11).



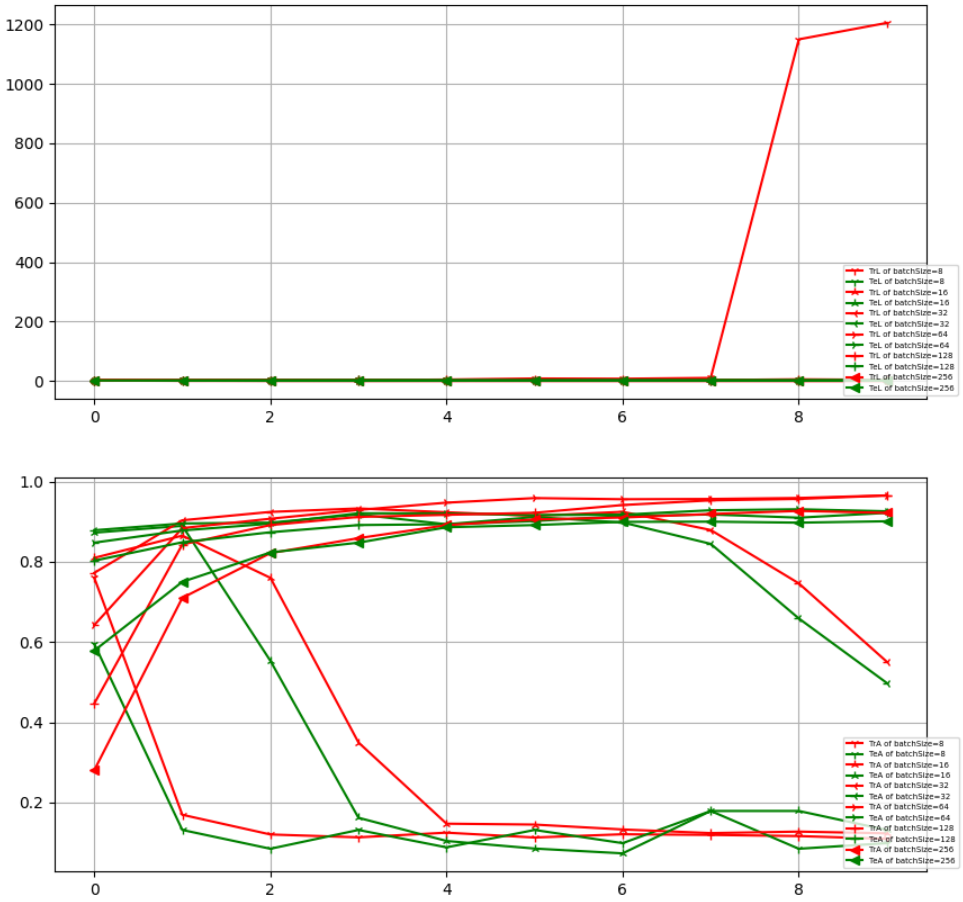
This is the result of (3) (5) (8) (11). We can see the VarianceScaling learn fast and its performance is good, so the VarianceScaling is effective. Others are good performance, but others learn slower compared with VaricanceScaling.

Next, I compared different learning rate: 0.1, 0.01, 0.001, 0.0001 using VarianceScaling initializers. If the learning rate is low, training becomes more reliable, but optimization takes longer because each step toward the loss function minimum is small. If the learning rate is high, the training may not converge at all and may even diverge. The change in weight can be so large that the optimization oversteps the minimum, making the loss function worse. So, I think when the learning rate is 0.1 and 0.01, it learns fast but has not good accuracy. However, when learning rate is 0.0001, it learns slow but has good accuracy. Let's see.



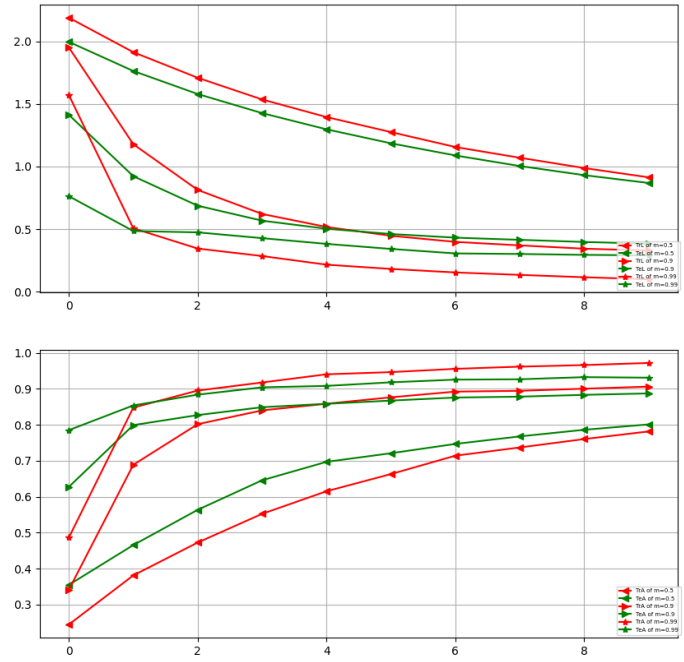
This is the result of different learning rate. When lr=0.1, it has very bad performance because it changes too much every time. When lr=0.0001, it learns fast, but it has not very good performance. When lr = 0.001, it is effective. It proves the assume above.

The next I compared different batch size: 8, 16, 32, 64, 128, 256 using VarianceScaling initializer and lr =0.001.
I did some research at first.
1. There is no Batch Size, the gradient is accurate, and it is only applicable to small sample database
2. Batch Size=1, the gradient changes from one to the other, which is very inaccurate, making it difficult for the network to converge.
3. Batch Size increases and the gradient becomes accurate.
4. Batch Size is increased, the gradient is already very accurate, and it is useless to increase the Batch Size again
Note: Batch Size has increased. To reach the same accuracy, the epoch must be increased.
Because my default batch size is 128, it is good performance. So I think when batch size is greater than 128, it also is good. Let's see.



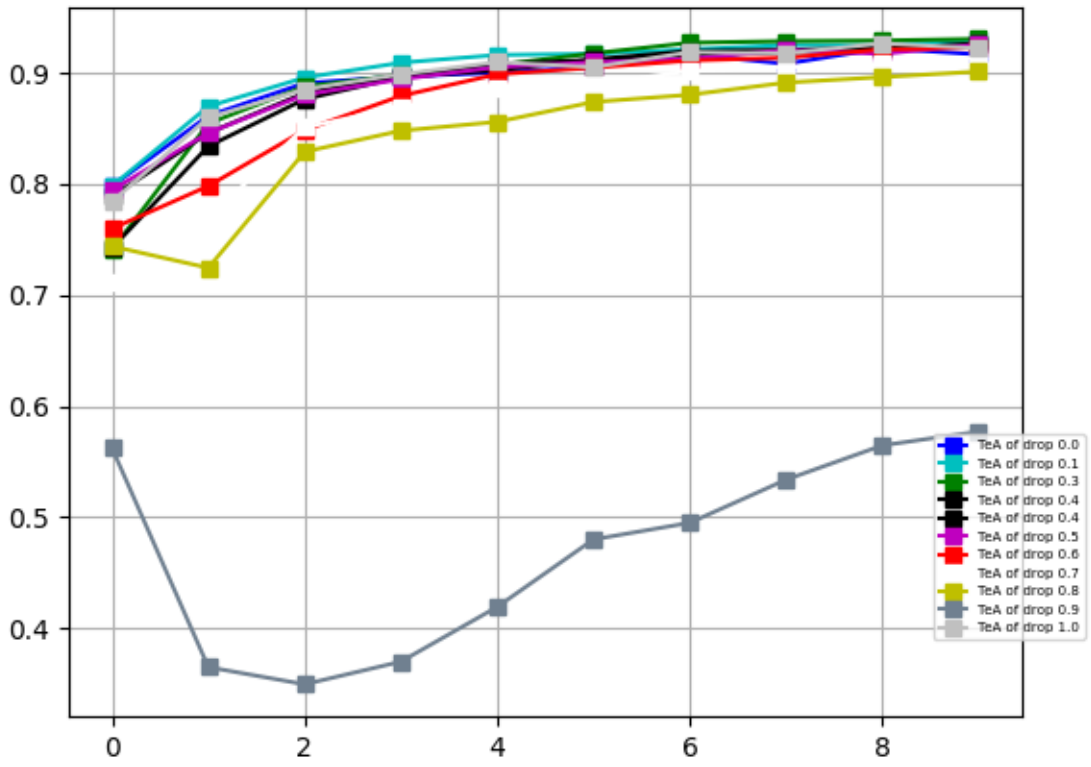
This is the result of different batch size. When batch size is 8 and 16, they learn fast, but they are bad performance. When batch size is 128 and 256, they are good. But when batch size is 32, the performance is also good, and the performance is a little better than others. It is weird result.

The next, I compared the different momentum. When the front and rear gradients are in the same direction, learning can be accelerated. When the front and rear gradient direction is not consistent, the shock can be suppressed. I guess everyone can be good performance, but the speeds are different.



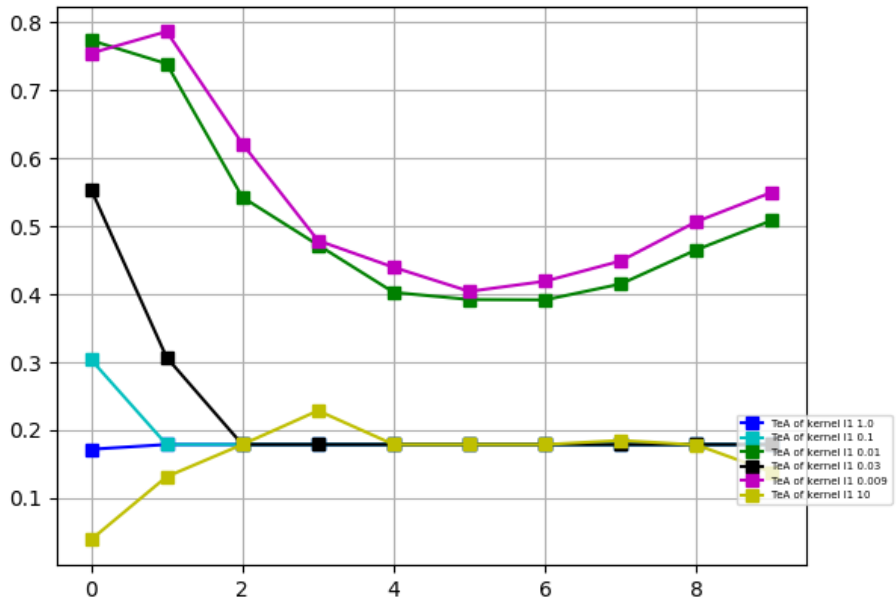
This is the result of different momentum. When momentum is 0.5, the performance is bad than others. When momentum is 0.99, the performance is good and learn fast. When momentum is 0.9, it also learns fast but not good performance as 0.99. So, I choose 0.99. I think there is no best momentum, because it may cause accelerated to good direction and bad direction.

Next. I tried the dropout's influence. Dropout, like L1 and L2, is a way to solve overfitting, while gradient test is a way to check whether the "back propagation" calculation is accurate. I guess more dropout, not good accuracy because it reduces many neutrons.

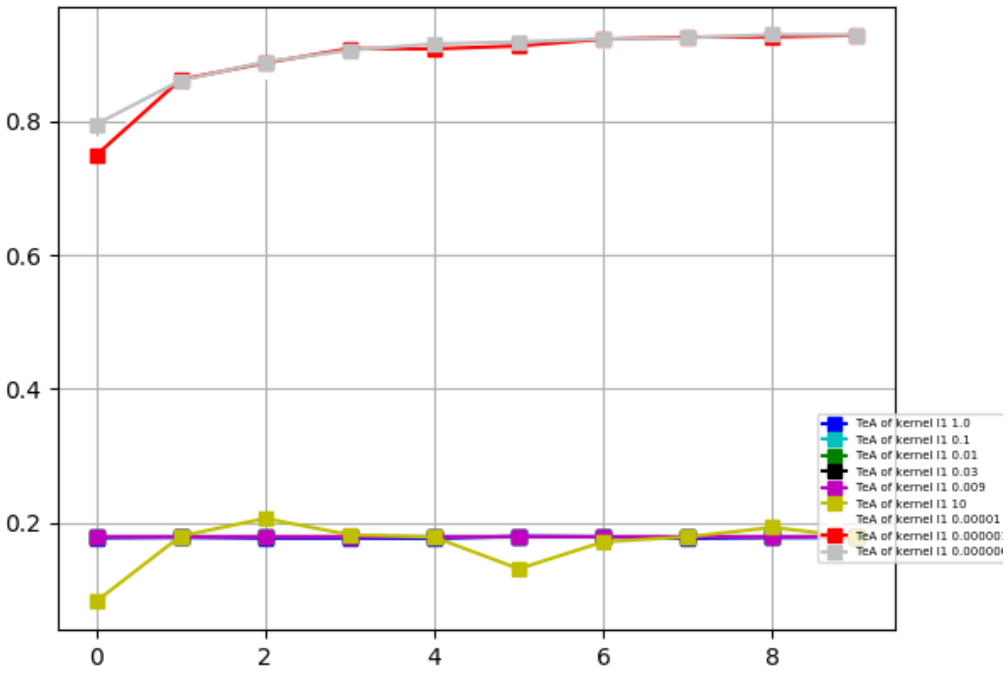


This is the result. When I set the dropout is 0.9, totally dropout, the accuracy is very bad. It effective. When dropout is 0.3, it is very effective. Actually, when the parameter of dropout is smaller, the accuracy is better except the 1.

Next, I compared the different L1 regularization.
We usually use this equation to calculate real data error, and L1 L2 is just behind the error formula with a thing, let error depends not only on the stand or fall of fitting data fitting, and depend on just like c d that the size of the value of the parameter. If this is the square of each parameter, so we call it L2 regularization, if is the absolute value of each parameter, we called the L1 regularization.
I think when you add something in the error, if this number is big, the error is big, so it is not accurate. So, I think the L1 is less, the performance is good.

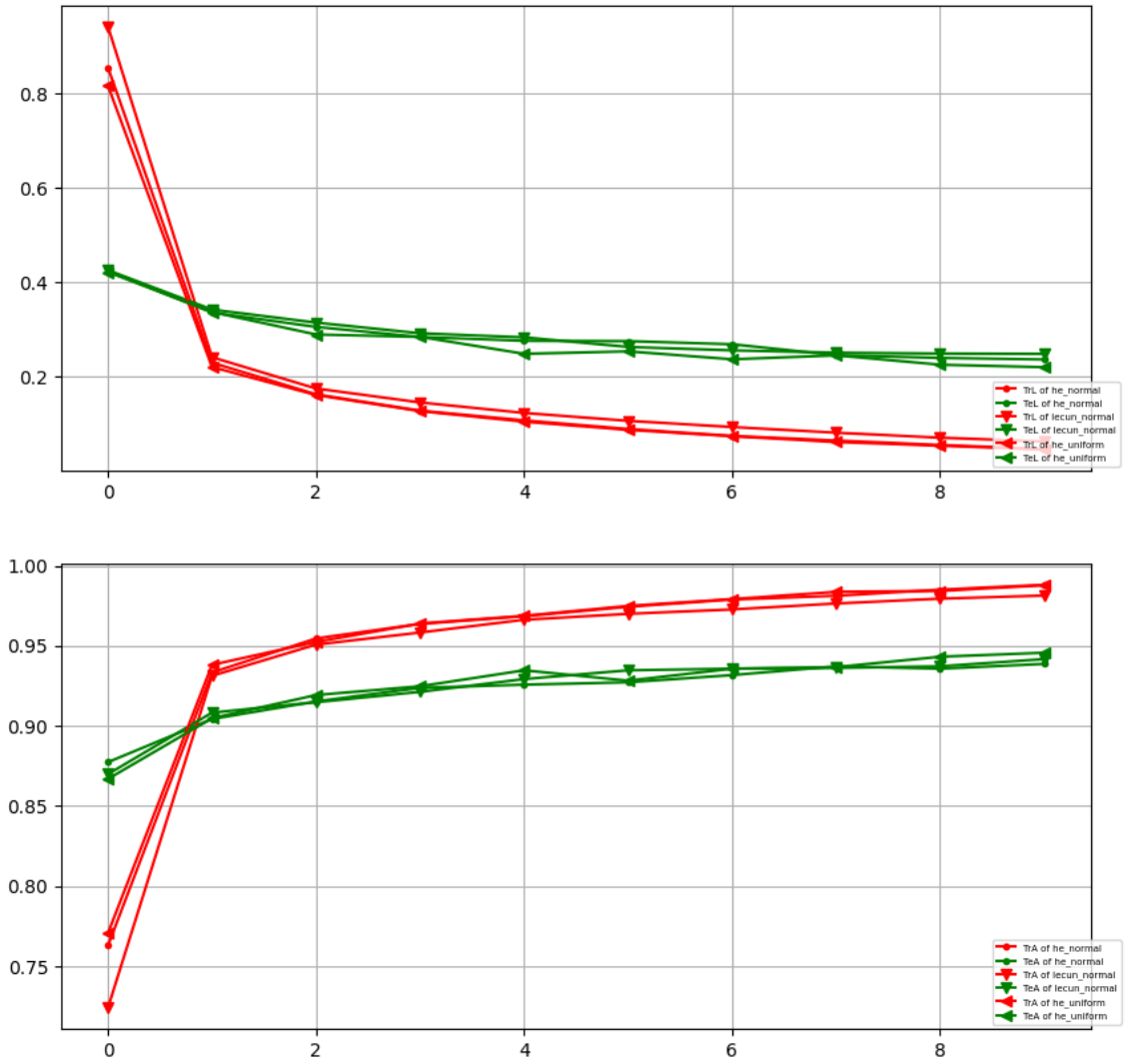


Next, I add kernel L1 regularization: 1.0, 0.1, 0.01, 0.03, 0.009, 10. Only 0.03 and 0.009 is a little better. So, next I try a lot more parameter and activity L1 regularization

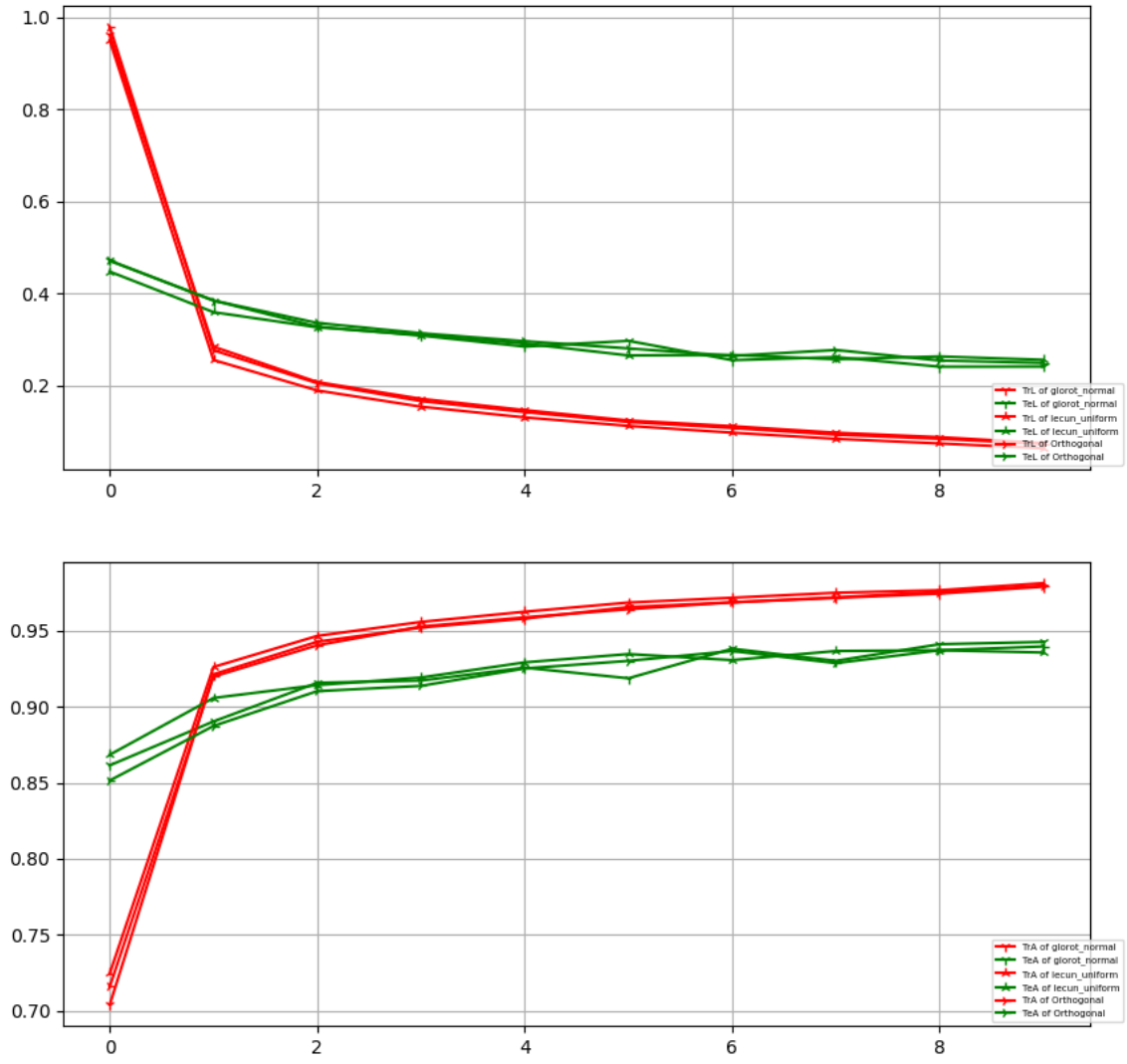


When parameter of L1 is 0.000001 and 0.0000001, the accuracy is better. They are effective. Others is very bad, and the accuracy is below 0.2. So, when regularization is high, the performance is not good.

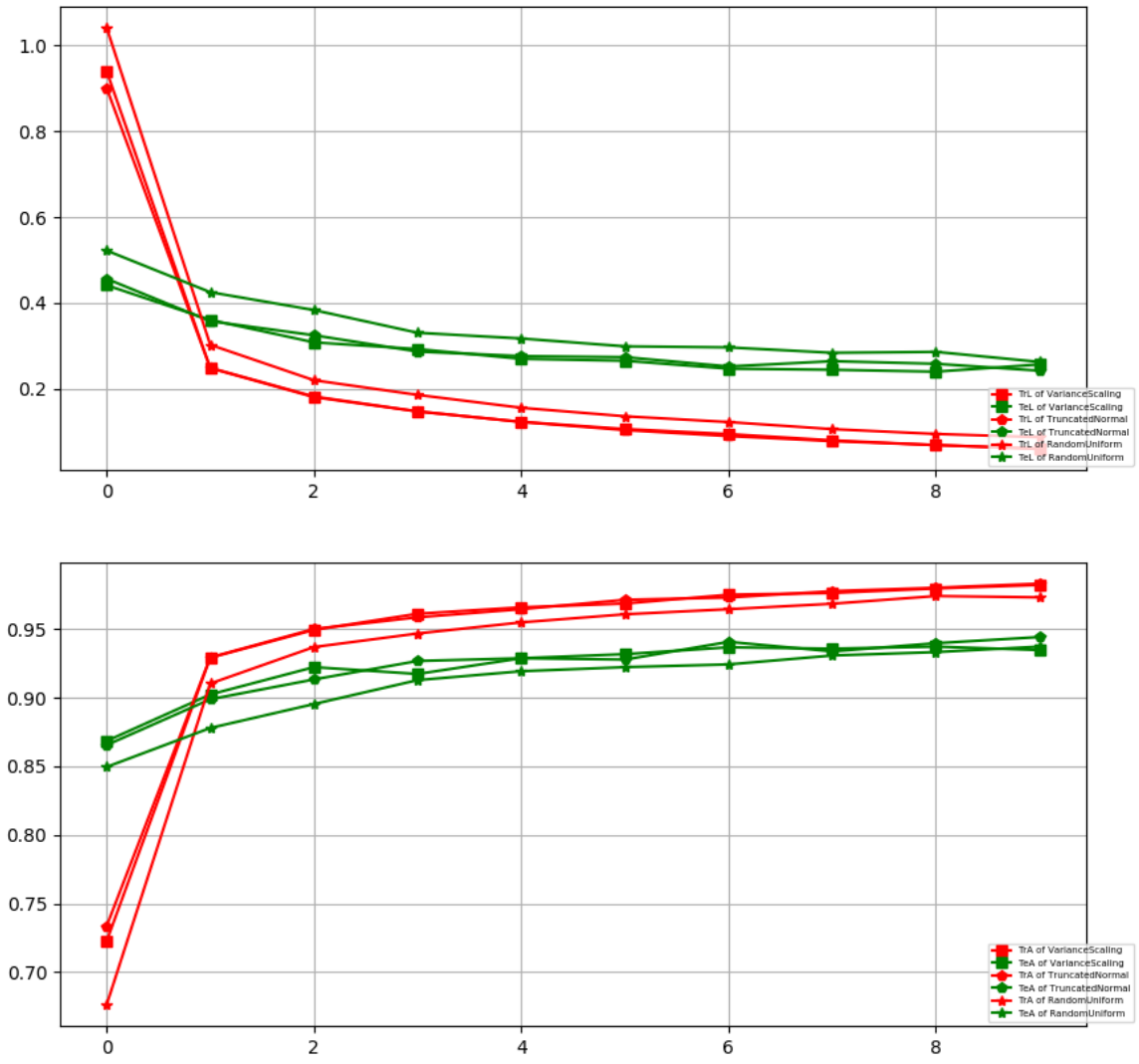
The second is CNN. I choose 13 kind of different initializes, and finally I found the he_normal is best parameter. I get the result of (1)he_normal, (2)lecun_normal, (3)he_uniform, (4)glorot_uniform, (5)glorot_normal, (6)lecun_uniform,(7) orthogonal, (8)VarianceScaling, (9)TruncatedNormal, (10)RandomUniform, (11)RandomNormal, (12)Constant, (13)ones. And I compare three by three and find some better and compare again.



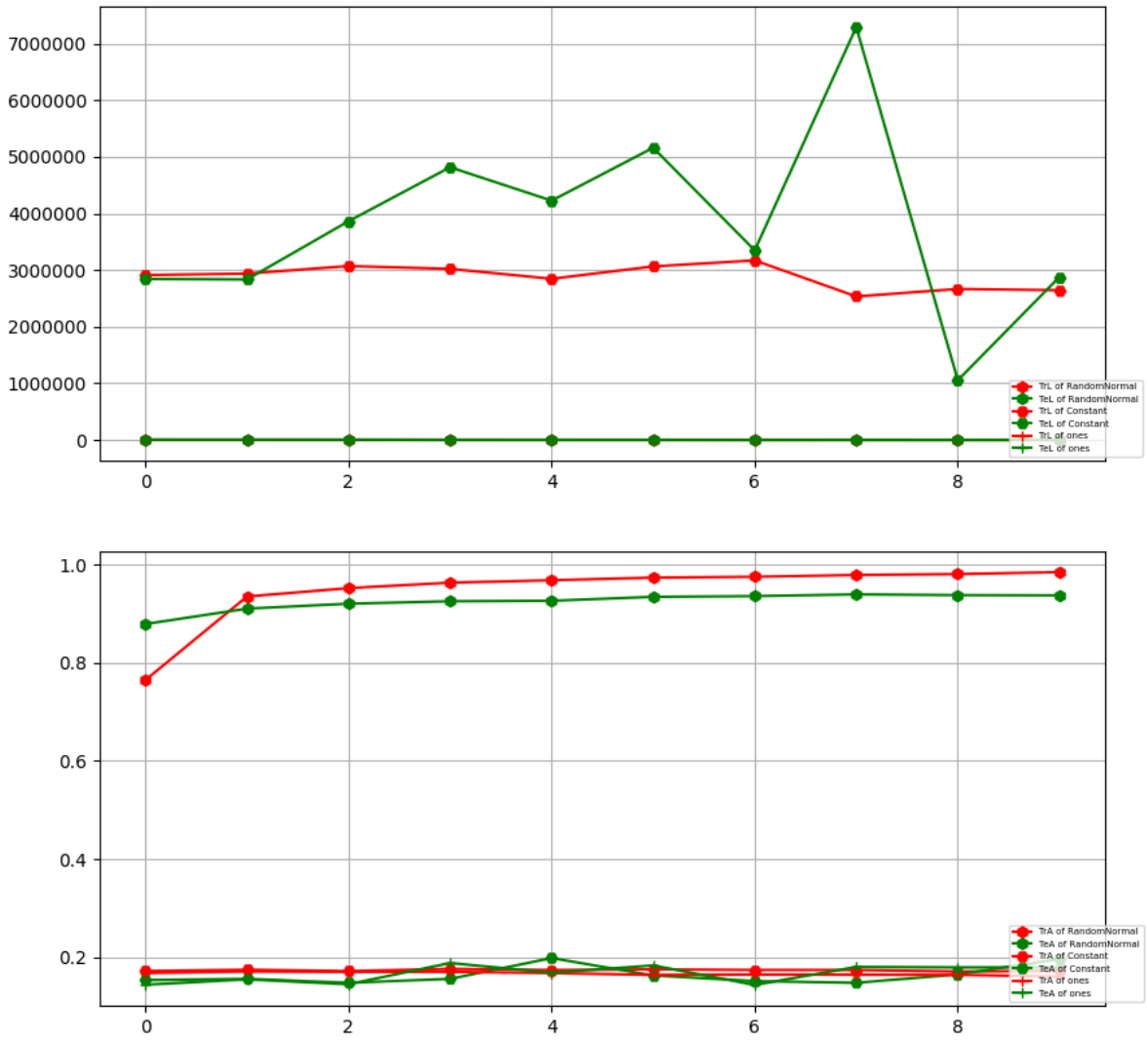
This is the result of (1) (2) (3), according to this, we can see the three models have similar accuracy, but he_normal is a little better, lecun_normal is only learn a little faster than others. The he_uniform is in the middle. I think the he_normal is efficiency.



This is the result of (4) (5) (6). According to this, although the accuracy of them are similar, we can see the lecun_uniform is learn faster and the test accuracy is better than others. So, I choose the lecun_ uniform.

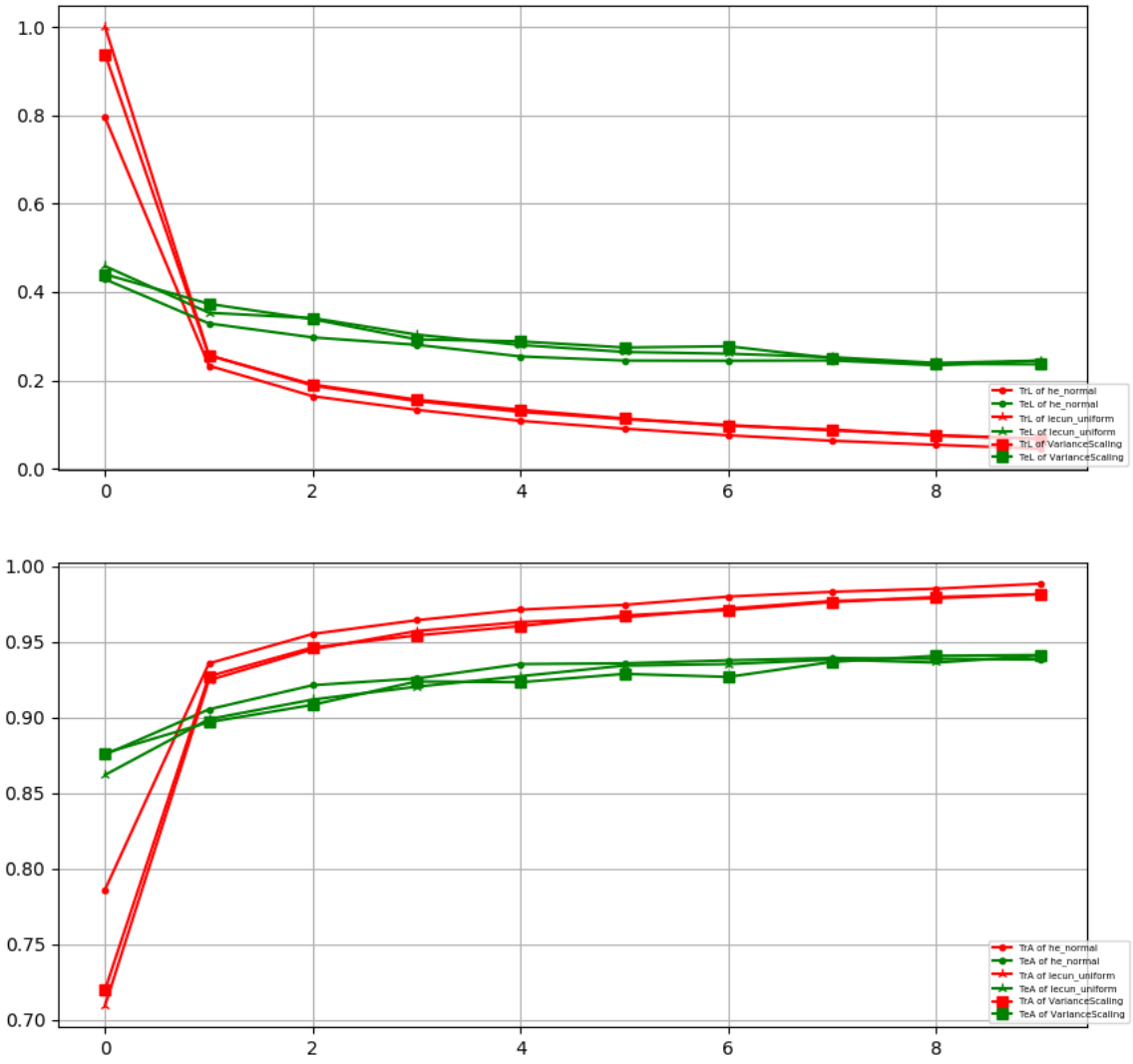


This is the result of (7) (8) (9). This three performances are different. it's obvious VarianceScalling is better because it has higher train accuracy, test accuracy, and the loss is lower than others. Although the RandomUniform is learn faster than others, it does not best performance.



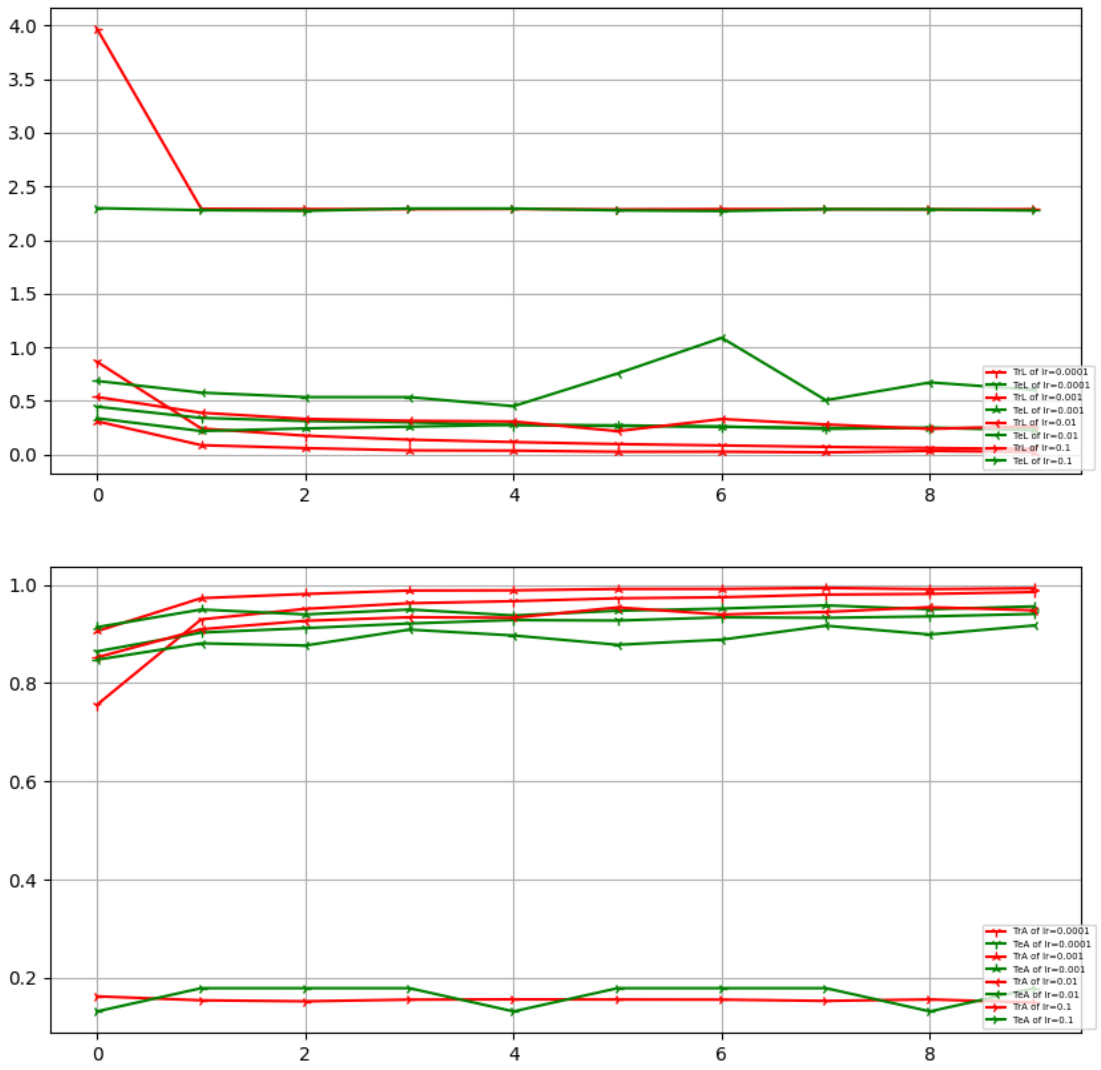
This is the result of last three. It is obvious that the constant and ones are bad performance. And the loss of randomNormal is not stable. So, I choose nothing in this three.

And then I compared the three-better model: he_normal, lecun_uniform, and VarianceScaling.



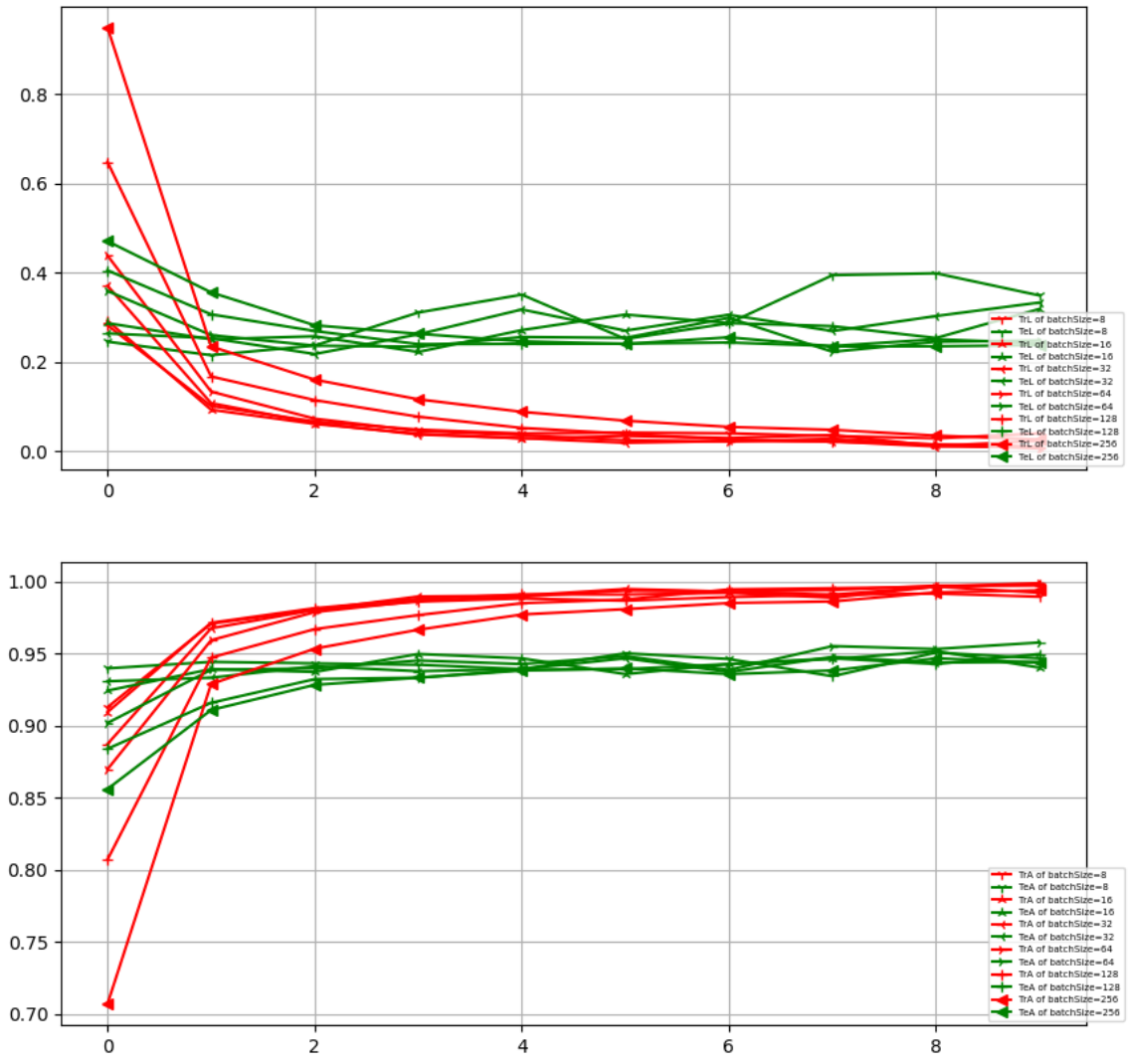
This is the result. We can see lecun_uniform and VarianceScaling are learn faster than he_normal, but in he_normal, it starts at higher accuracy and final accuracy is higher than others. So, he_normal is efficient I think.

So, next I choose he_normal as initialization parameter and compare learn rate. I choose learning rate of 0.0001, 0.001, 0.01, and 0.1.

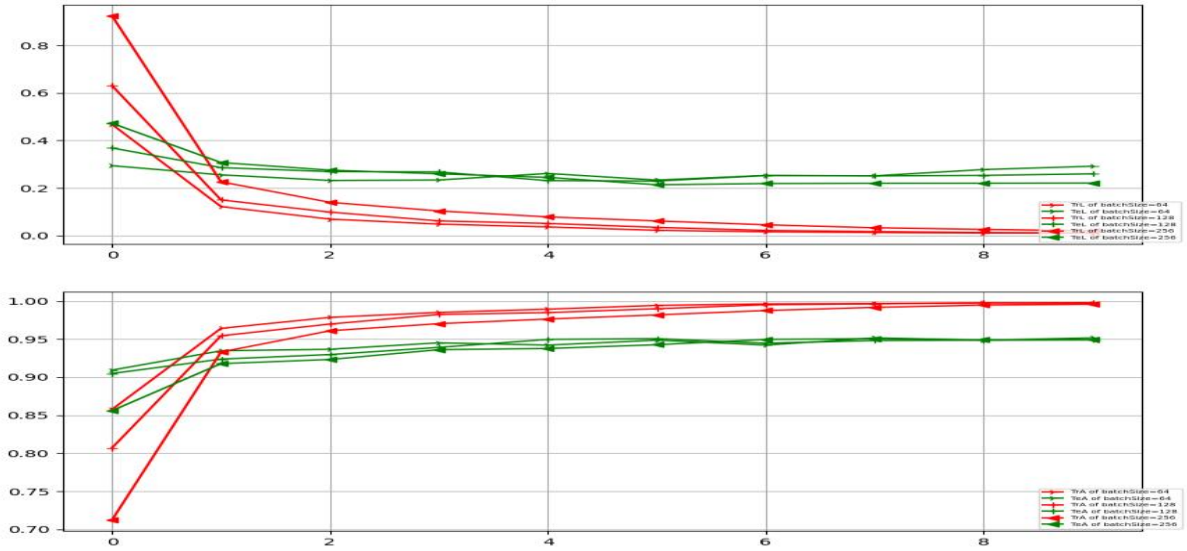


this is the result, when learning rate is 0.1, it learns very slow, and the performance is very bad. When the learning rate is 0.001, the performance is best. When learning rate is 0.0001, it learn faster than others. When learning rate is 0.01, the loss is not stable. So, I think learning rate = 0.001 is effective.

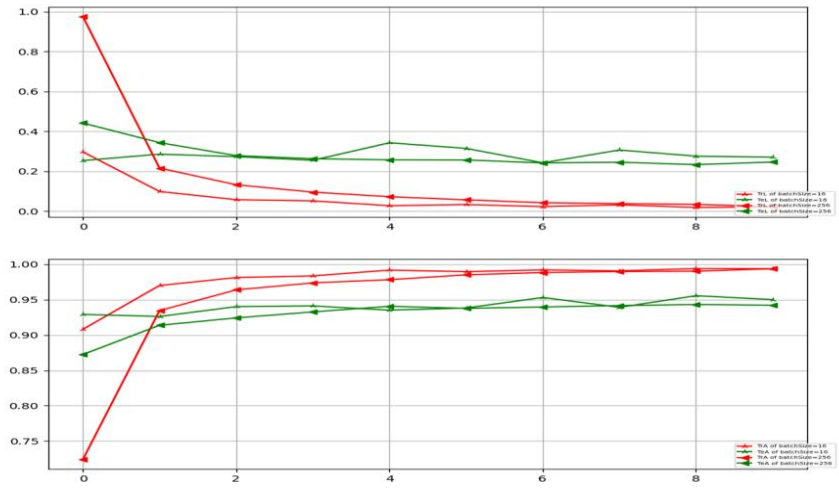
The next step, I tested the batch size. I set the batch size is 8, 16, 32, 64, 128, 256.



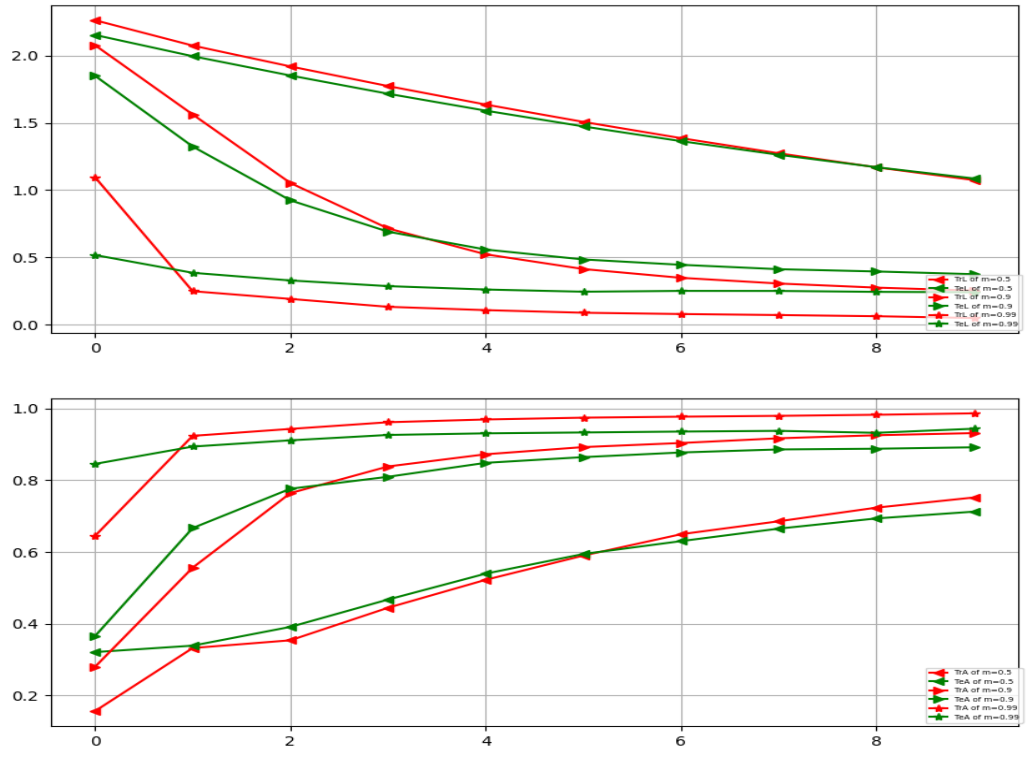
when batch size is 256, it learns faster than others. And the final performance is similar. When batch size is 128, it also learns faster. When batch size is 64, it starts at high performance. When batch size is 16, the performance is a little better than others.



So, I compare 64,128,256 again. We can see when the batch size is 256, it learn faster and the final accuracy is high.

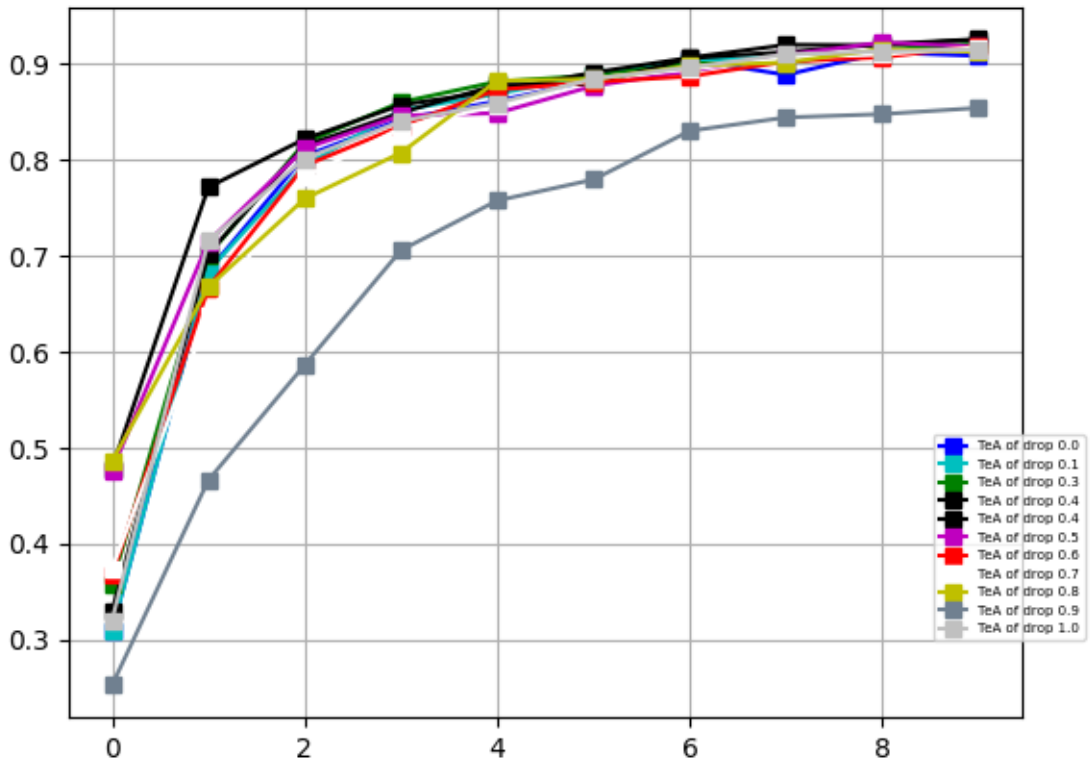


And then I compared the 16 and 256. I found the batch size is 16, the performance is a little better, but it is not stable. When batch size is 256, it learns fast, and the performance is also good. So, I choose 256.



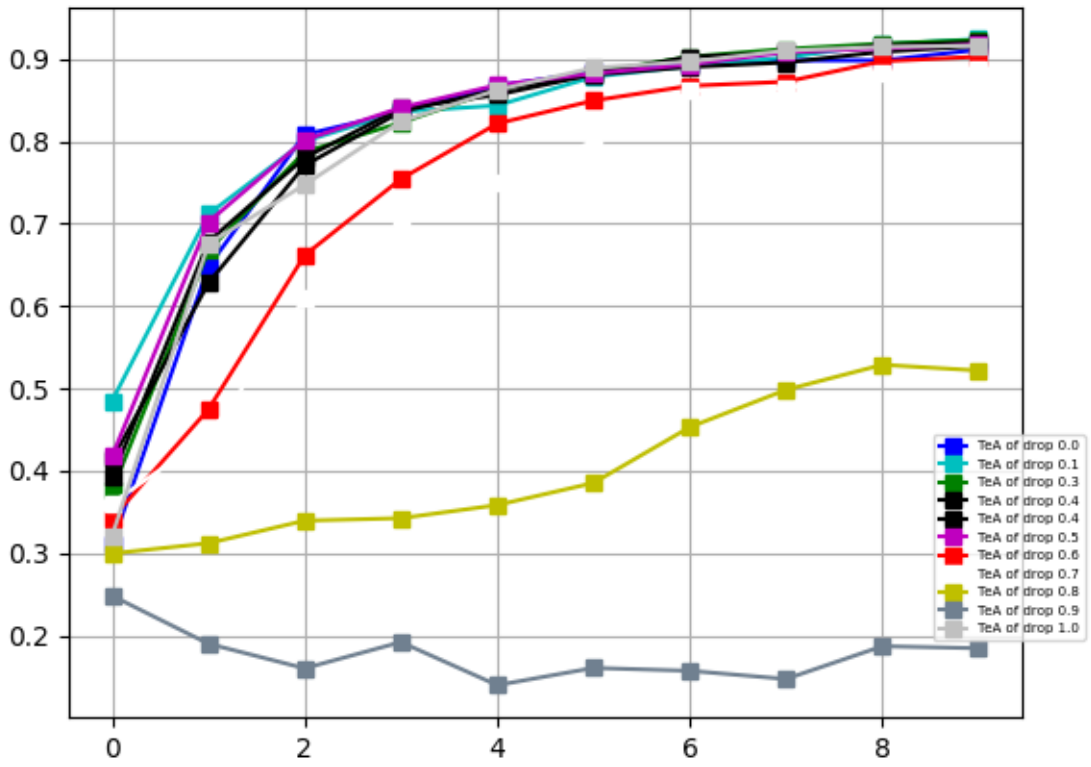
Next I compare different momentum: 0.5, 0.9, 0.99.
It is obvious that when momentum is 0.99, the performance is best.

Next, I test the dropout influence.



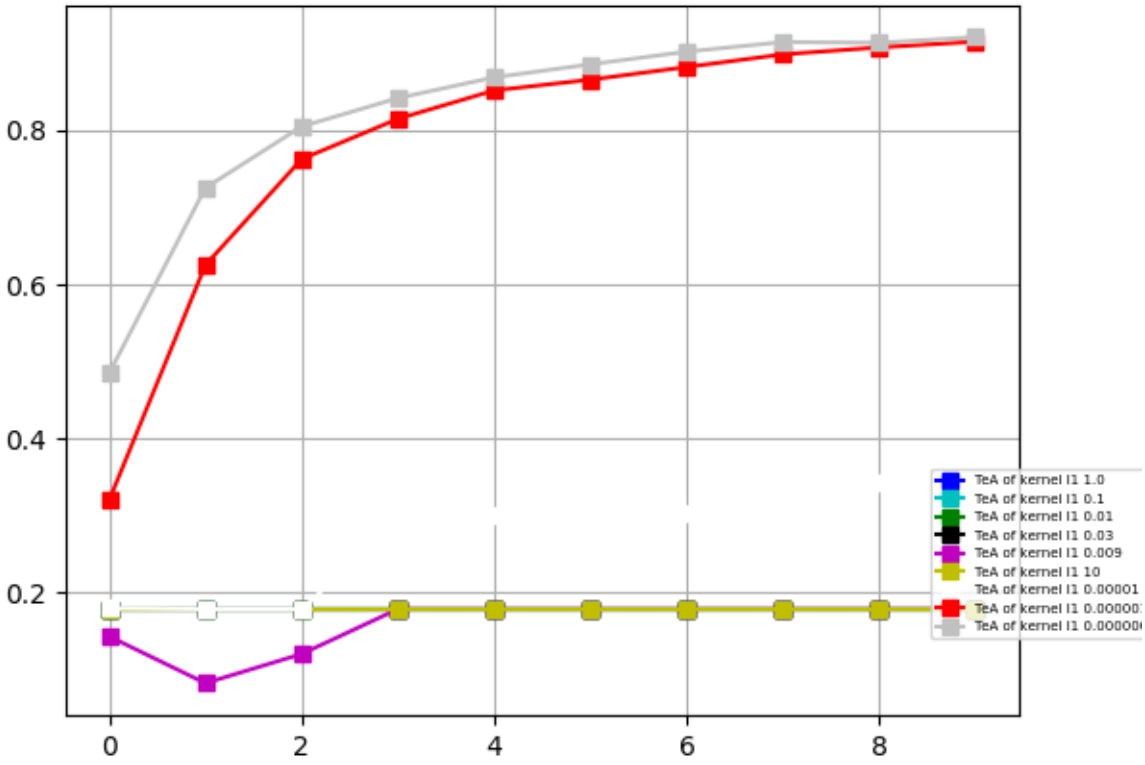
This is the result of adding the dropout layer in second layer (after conv2D). We can see when dropout is 0.9, the accuracy is not good but it's ok. When dropout is 0.4, it is best.

After that, I add two dropout layers in cnn.



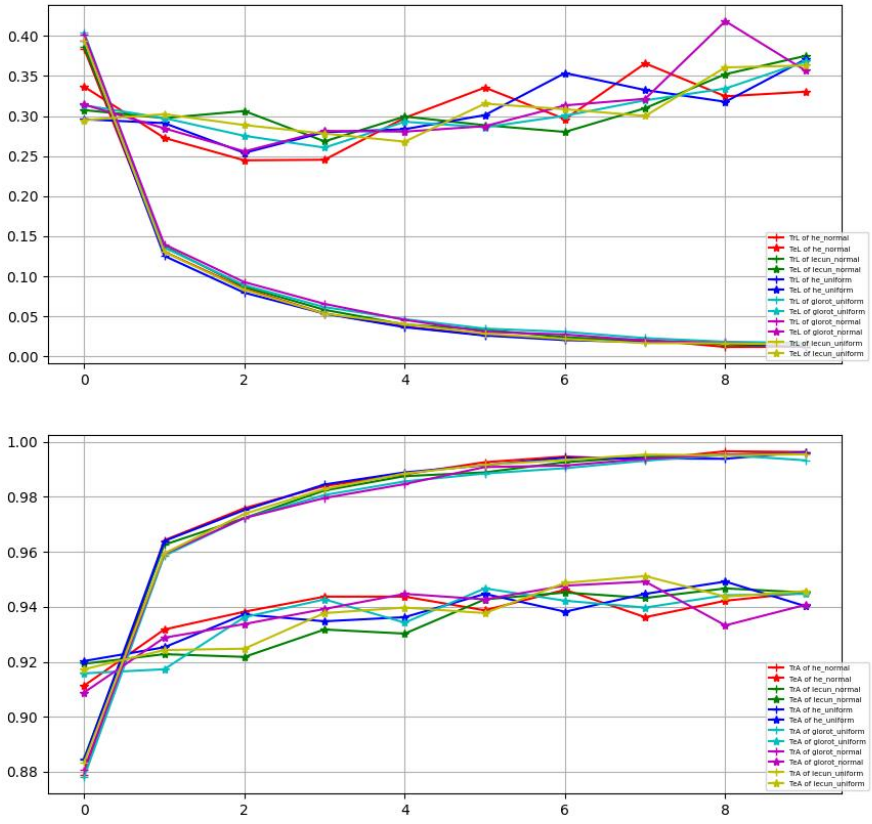
This is the result. When dropout is 0.9 and 0.8, the accuracy is very bad. It is not effective. When dropout is 0.3 and 0.4, they are effective.

Next, I compared different l1.

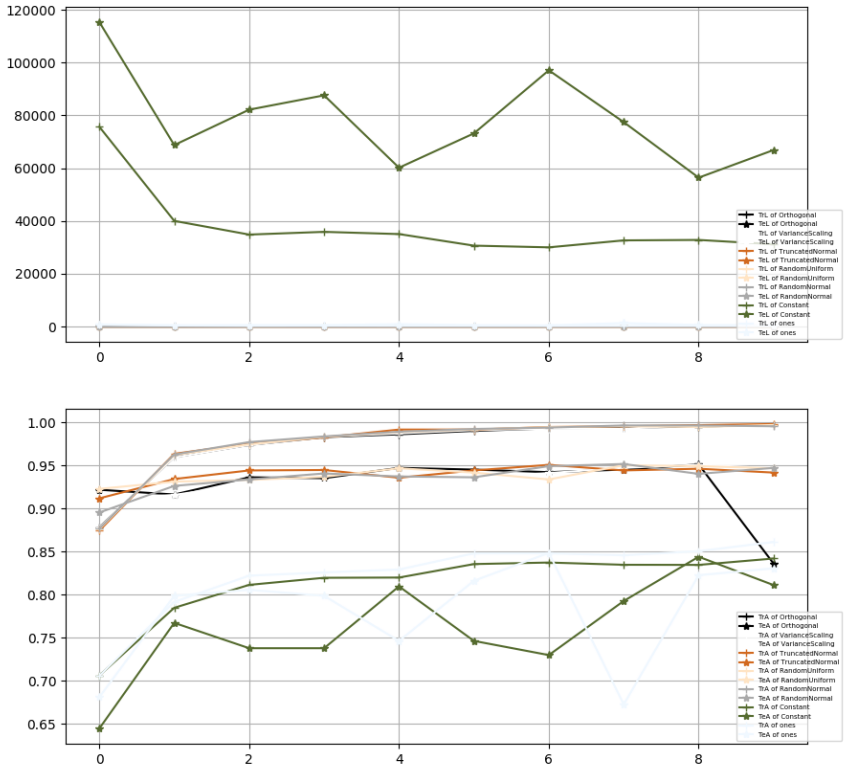


The result is similar with fully connected. When regularization is high, the performance is not good. So 0.000001 and 0.0000001 are effective.

The third one is Locally Connect network. I compared all the parameters six by six using different color to separate.

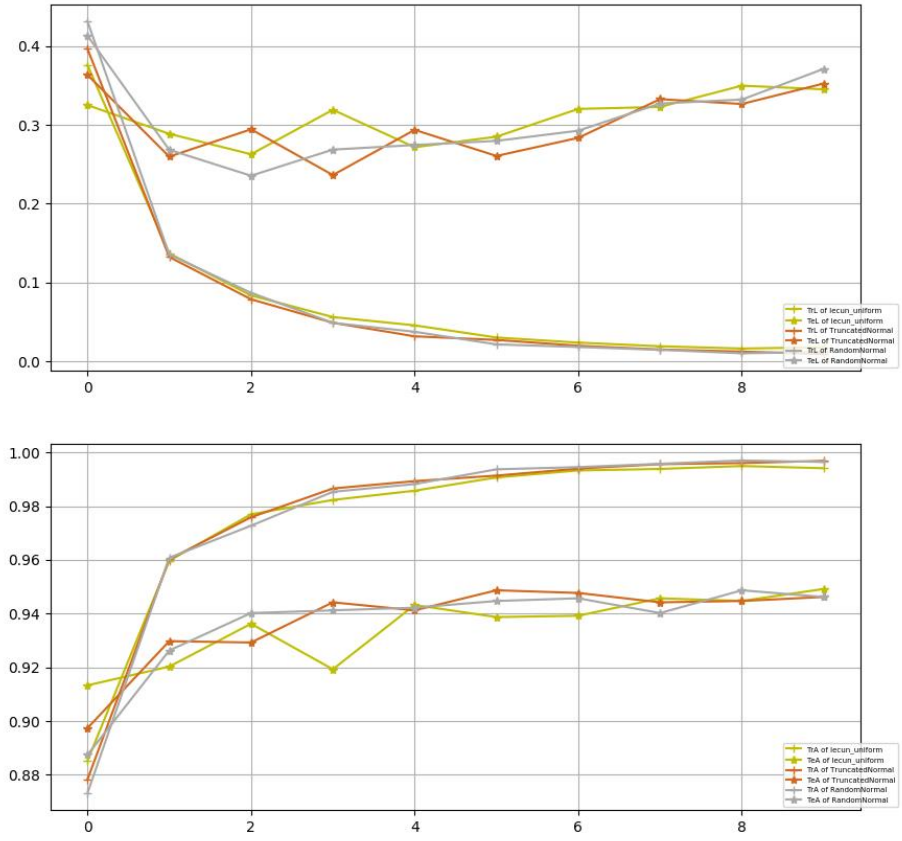


This is the result of (1) (2) (3) (4) (5) (6). Lecun_uniform learns fast because it get high point quickly. Lecun_uniform is effective. At same time, it has the better performance. Glorot_normal and he_uniform learn fast too, but the final performance is not good as lecun_uniform. Glorot_uniform and lecun_normal learn a little slow.



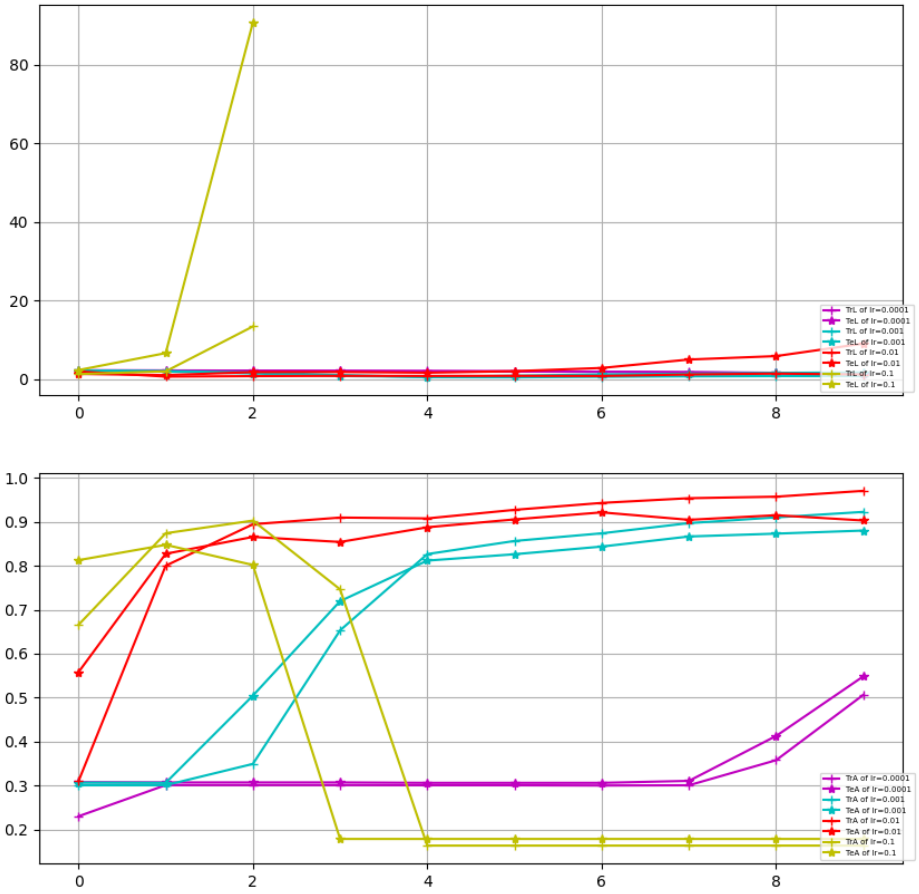
This is the result of (7) (8) (9) (10) (11) (12). We can see the constants and ones learn slow and the performance is bad. Othogonal learns fast, but the final result is not good. And RandomNormal and TruncatedNormal are effective.

Next I compare the RandomNormal, TruncatedNormal and lecun_uniform.



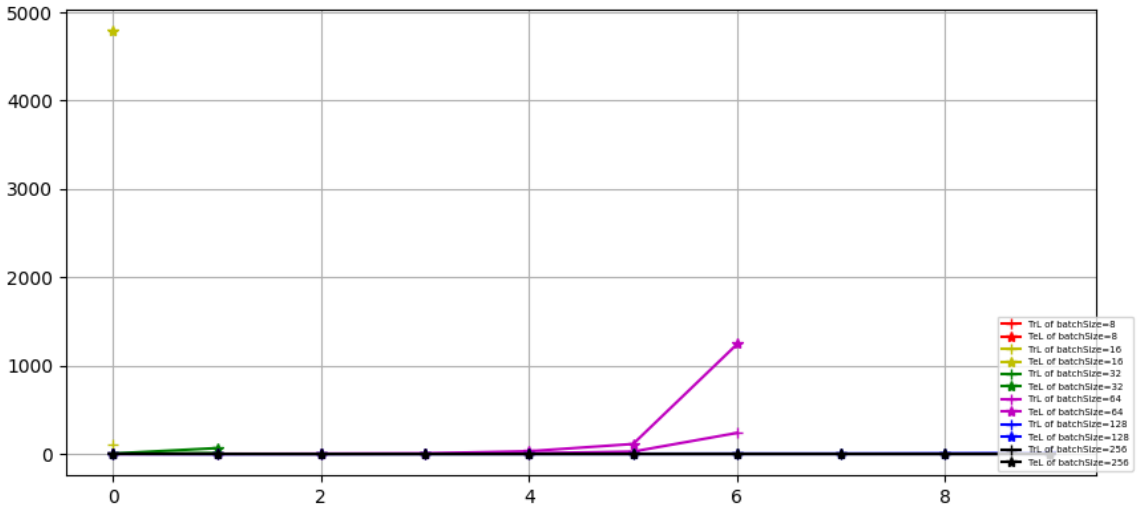
This is the result. TruncatedNormal learns fast but the final accuracy is not best. Lecun_uniform learns a little slower, but the accuracy is best. RandomNormal learn fast at the beginning, the final accuracy is same as TruncatedNormal. So lecun_uniform is effective.

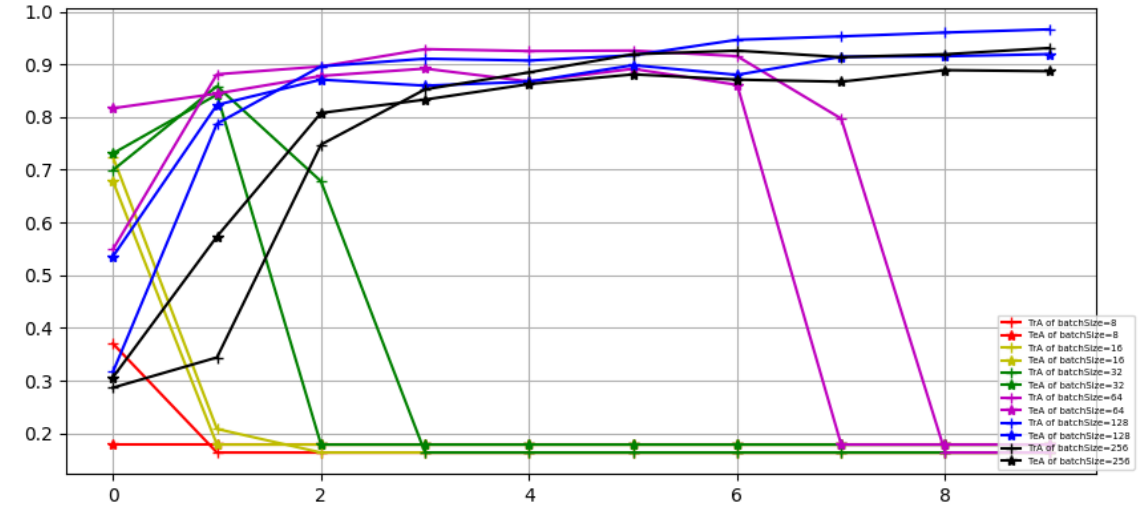
Next I compared the different learning rate



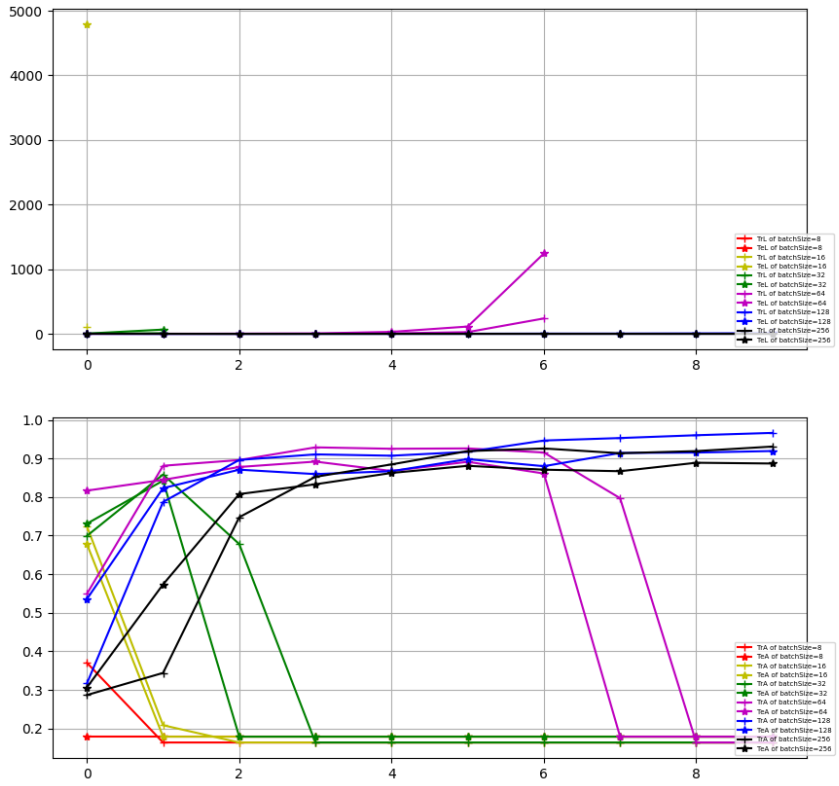
This is the result. When lr is 0.1, it learns fast, but the performance is very bad. When lr is 0.0001, it learns slow and the performance is also bad. When lr is 0.001 and 0.01, their performance is similar, and 0.01 is a little better. And when lr is 0.01, it learns faster than 0.001. so I choose 0.01 as learning

rate.


5000
4000
3000
2000
1000
0
0
2
4
6
8
TrL of batchSize=8
TeL of batchSize=8
TrL of batchSize=16
TeL of batchSize=16
TrL of batchSize=32
TeL of batchSize=32
TrL of batchSize=64
TeL of batchSize=64
TrL of batchSize=128
TeL of batchSize=128
TrL of batchSize=256
TeL of batchSize=256

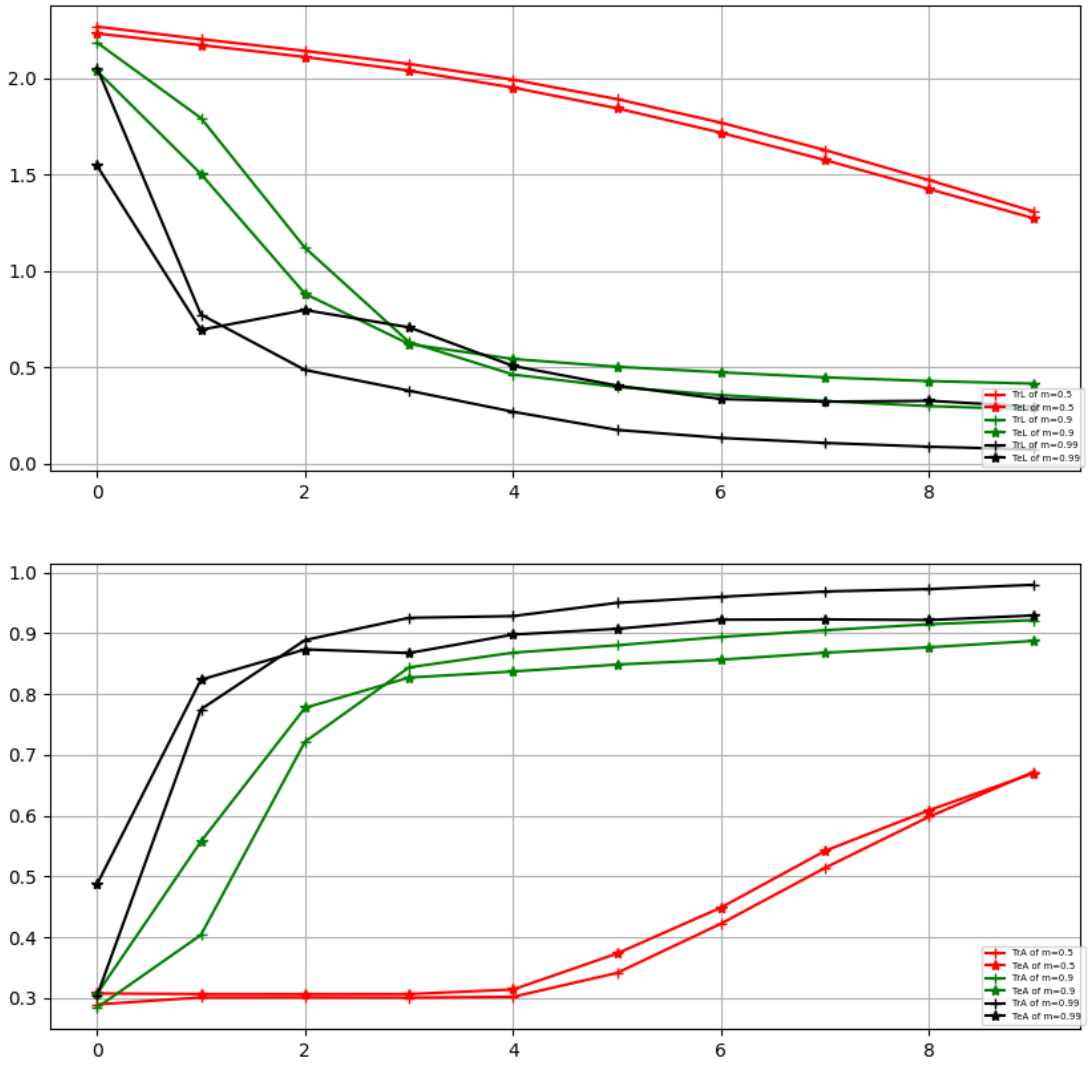

1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
2
4
6
8
TrA of batchSize=8
TeA of batchSize=8
TrA of batchSize=16
TeA of batchSize=16
TrA of batchSize=32
TeA of batchSize=32
TrA of batchSize=64
TeA of batchSize=64
TrA of batchSize=128
TeA of batchSize=128
TrA of batchSize=256
TeA of batchSize=256

Next, I compared the different batch size.



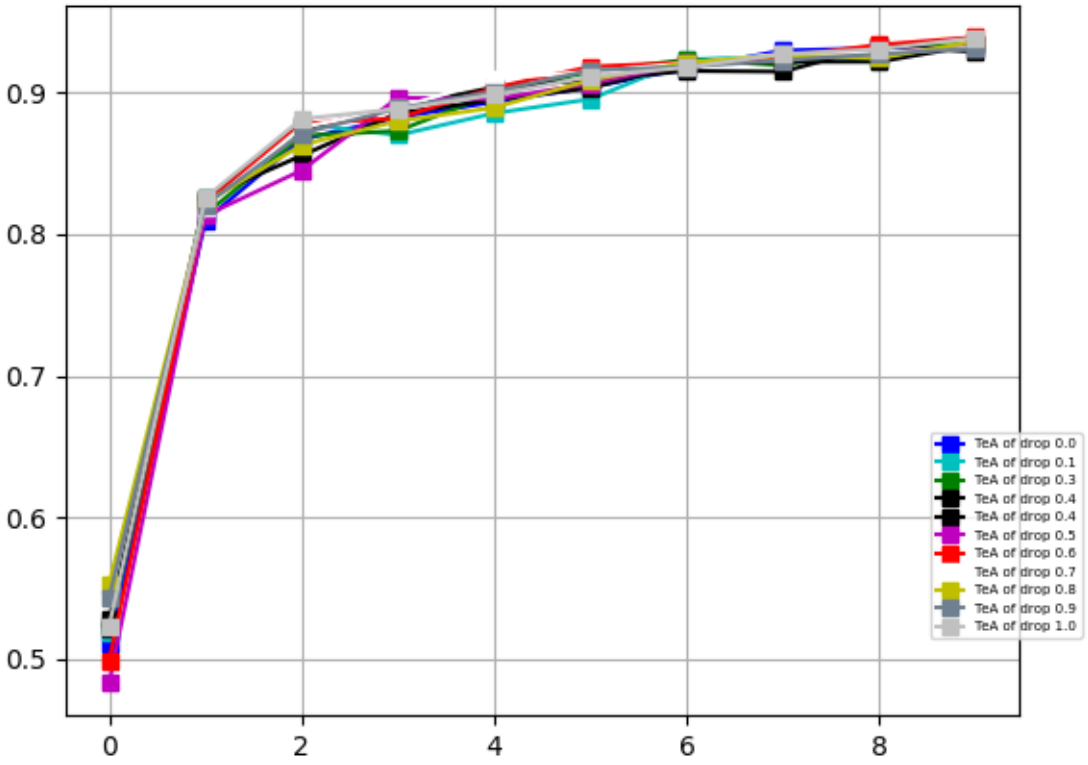
This is the result of different batch size. We can see if batch size is 64, it learns very fast but gets low accuracy finally. If batch size is 32, 16, and 8, the performances are very bad. If batch size is 128 and

256, they also learn fast and get good accuracy. Batch size is 128 is better than batch size is 256.

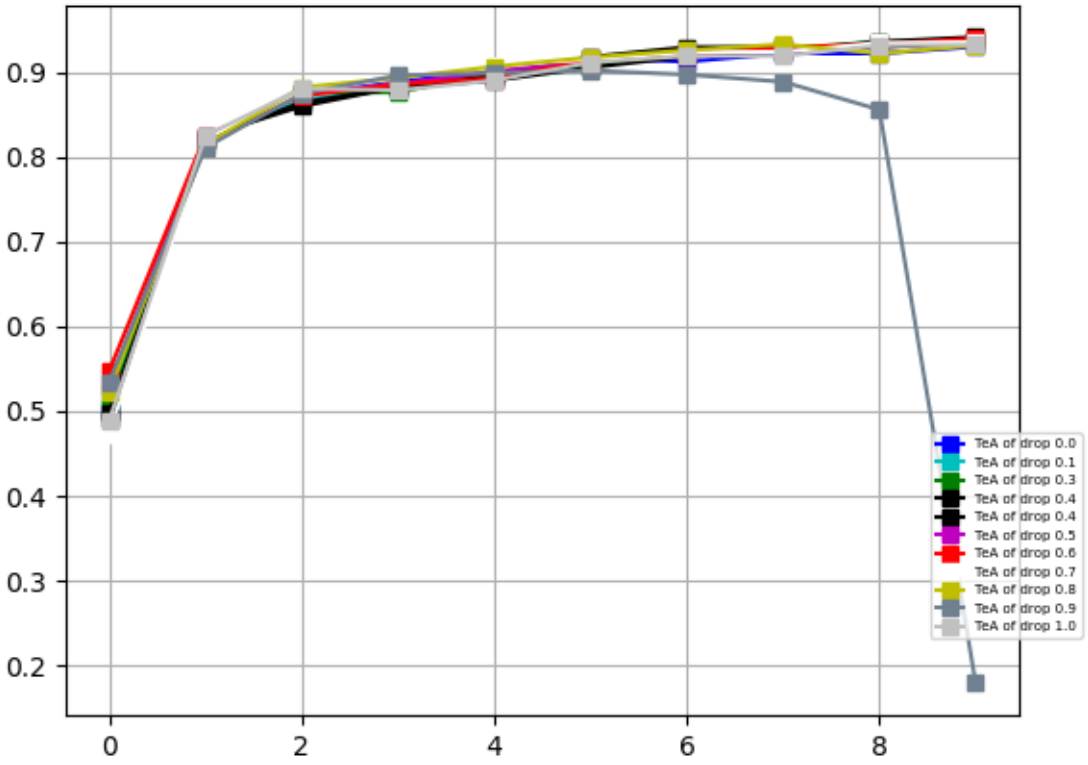


This is the result of different momentum. When momentum is 0.99, it learns fast and accuracy is best. It is effective. If momentum is 0.9, it also learns fast, but the accuracy is not good as 0.99 of momentum. If the momentum is 0.5, it learns slow and has bad accuracy. It is ineffective.

Next, I add one dropout in second layer to see the influence.

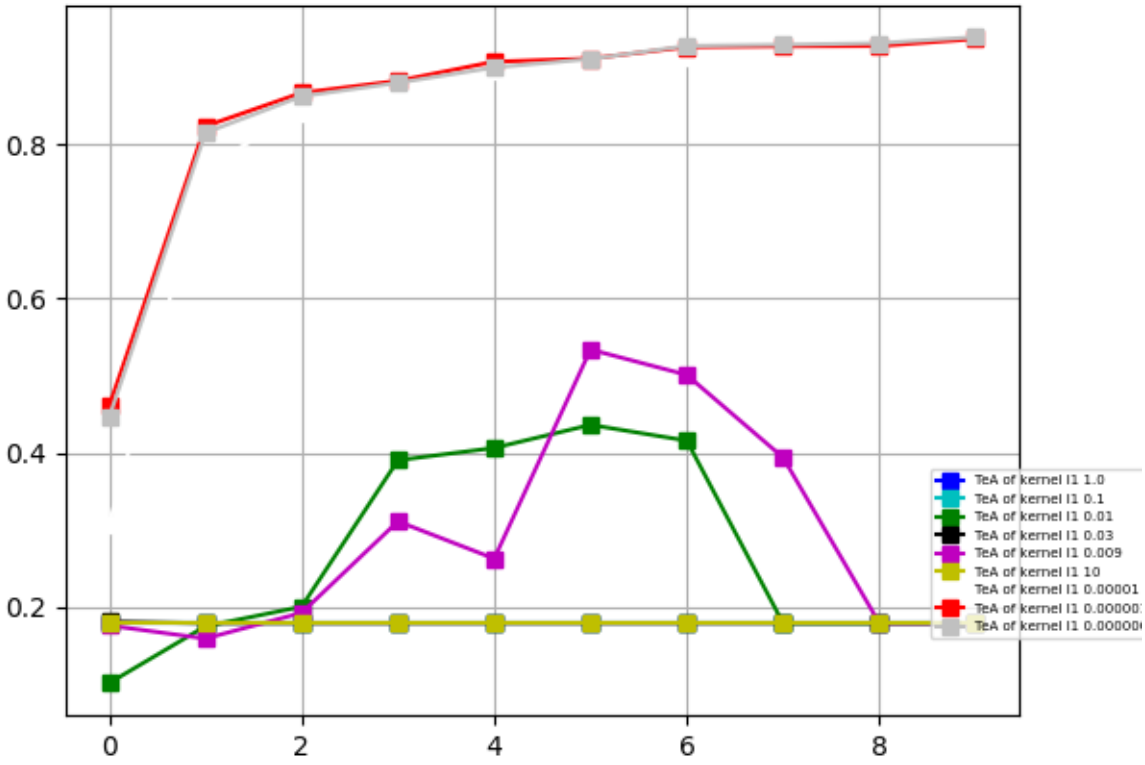


This is the result. we can see the result is similar. So, I tried the two dropout layers.



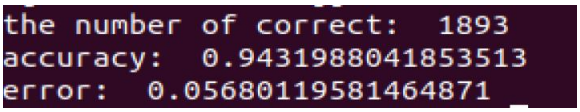
We can see only when dropout is 0.9, the accuracy is not good. And others are similar. I think it is because of locally network that has already reduce many neutrons.

Next, I compared different L1 regularizations.



The result is similar with fully and cnn. When the number is less, the accuracy is good. L1: 0.0000001 is effective. L1: 0.1, 0.01 is ineffective.

Next, I used the ensemble to improve the generalization. This is the result.



Training multiple models and combining predictions together can reduce variance. Multi-model integration works only if each model has its own characteristics and the errors predicted by each model are different. The simple integration approach is to average the predictions, which works because different models do not usually produce the same errors on the test set.
So, it is obvious that the test accuracy is better than single model. I check the single cnn, single locally, and single fully that using best parameters. The best one trend to 0.94, but others cannot exceed 0.94.