Samwoo Seong
7953-6137-66
samwoose@usc.edu

EE569 Homework #5
April 05, 2020

1.

1.1 Abstract and Motivation

In field of image processing, object detection task has been one of most interesting and important topics. Once LeCun et al. published a paper on LeNet-5 which is a convolutional neural net also known as CNN used to recognize hand-written digits images with machine (e.g., computer and digital camera) and it outperformed any other existing methods, it became popular quickly. Briefly speaking, CNN is able to create powerful match filters based on collected data depending on tasks. Furthermore, performance of a CNN can be improved by design of its system architecture. For those reasons, many people from different fields are interested in CNN and there have been a lot of researches.

1.2 Approach and Procedures

(b)

1. I have used a given model structure which is LeNet-5. To observe role of each parameter such as filter weights, learning rate, weight decay and momentum, I only give a change to one parameter and fix the rest of values. For example, If I want to find out the role of learning rate, I choose 5 different values for learning rate while I use the same values for weight decay, and momentum and the same method for filter weights initialization. Chosen values and methods for each parameter are given below.

	Learning Rate	Momentum	Weight Decay	Initialization Method
1st	0.001	0.0	0	Initialize filter weight with normal distribution
2nd	0.020	0.3	0.01	Initialize filter weight with uniform distribution
3rd	0.250	0.5	0.03	Initialize filter weight with constant
4th	0.500	0.7	0.09	Initialize filter weight with ones

5th	1.000	0.9	0.27	Initialize filter weight with zeros

Table 1. Chosen values and methods of parameters for the experiments

2.
I draw possible values for pairs of parameters based on the results from the experiment at problem 1 (b)-1. To be specific, 2 values for momentum and weight decay respectively, 3 values for learning rate, and random initialization method for filter weights. The reason for this selection is explained at discussion category.

	Pair (Learning rate, Momentum, Weight decay)
1st pair	(0.001, 0.7, 0.01)
2nd pair	(0.001, 0.7, 0.03)
3rd pair	(0.001, 0.9, 0.01)
4th pair	(0.001, 0.9, 0.03)
5th pair	(0.02, 0.7, 0.01)
6th pair	(0.02, 0.7, 0.03)
7th pair	(0.02, 0.9, 0.01)
8th pair	(0.02, 0.9, 0.03)
9th pair	(0.25, 0.7, 0.01)
10th pair	(0.25, 0.7, 0.03)
11th pair	(0.25, 0.9, 0.01)
12th pair	(0.25, 0.9, 0.03)

Table 2. Chosen pairs for searching the best pair.

After training model for each pair, I calculate accuracy on test data set and find a pair that gives the best accuracy. With the pair, I train the whole model one more time from the beginning with increased number of epochs to see shape of learning curve with wide view.

1.3 Experimental Results (Partially open source (e.g., Pytorch) is used)

(b)

<Legend>
Green triangle: accuracy on train data set for each epoch
Blue square: accuracy on test data set for each epoch
x-axis: current epoch
y-axis: accuracy within range 0~1

Note: when accuracy on train data set and accuracy on test data set are almost same, blue square overlap the green triangle. i.e. Hard to see green triangles. Need to be zoomed.

1.
Number of epochs = 20
<5 different values of learning rate & weight decay = 0 & momentum = 0.0>
(1) learning rate = 0.001

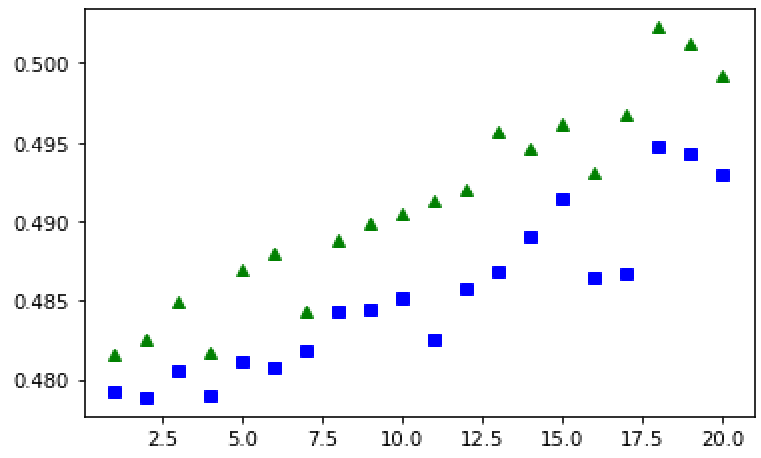

Fig 1.
(2) learning rate = 0.02

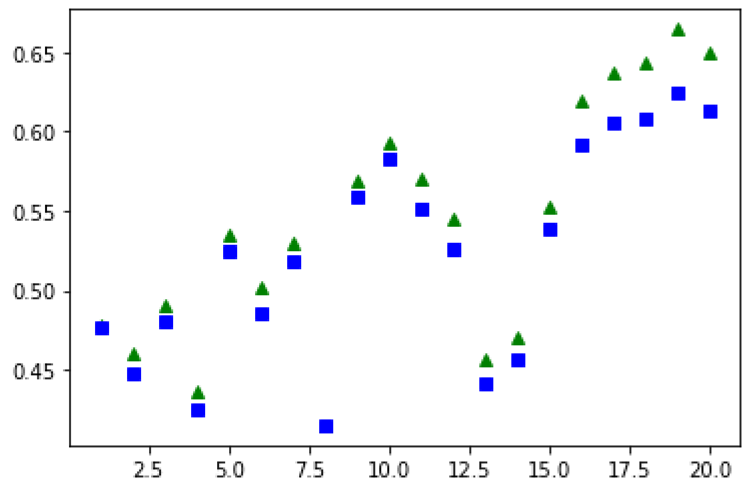

Fig 2.

(3) learning rate = 0.25

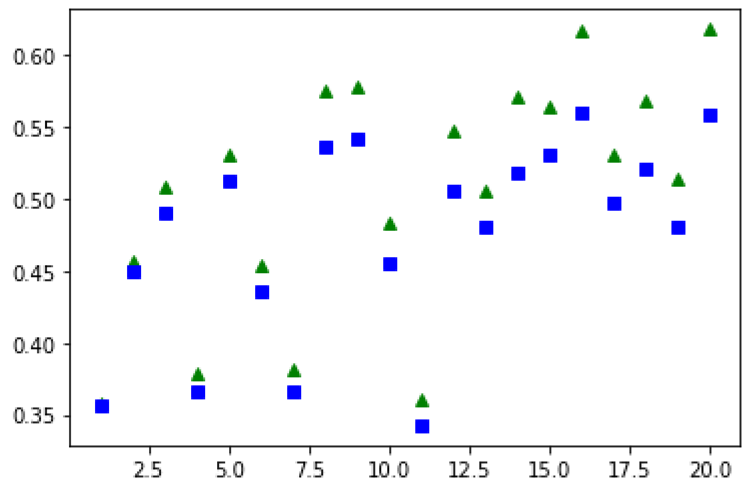

Fig 3.

(4) learning rate = 0.5

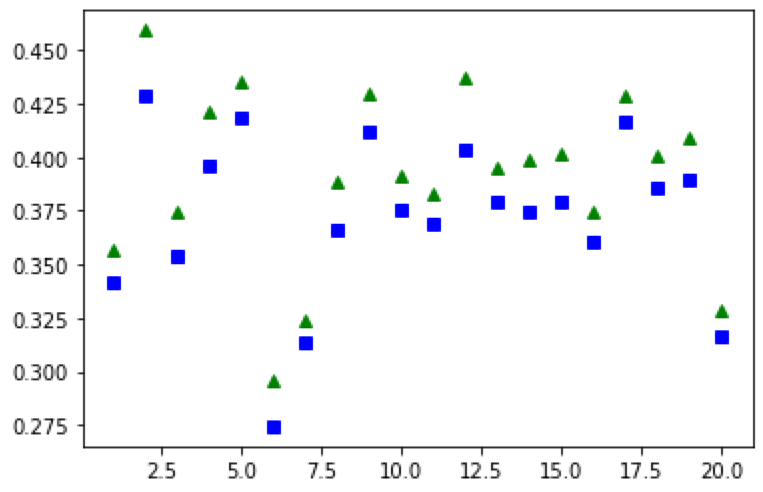

Fig 4.

(5) learning rate = 1.0

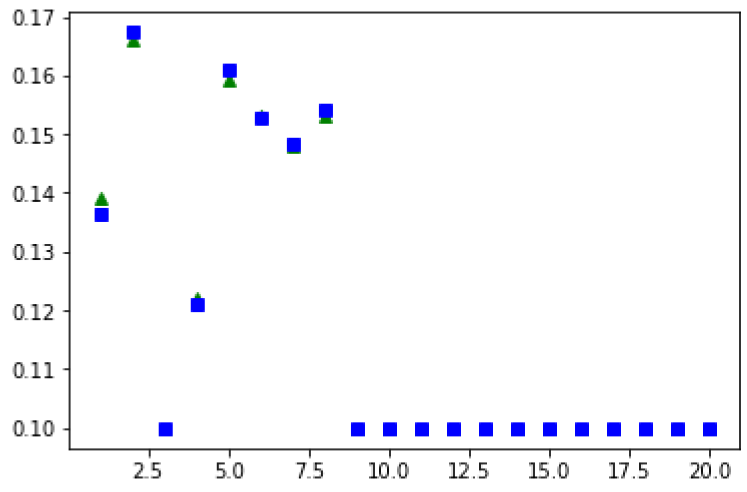

Fig 5.

<5 different values of momentum & learning rate = 0.001 & weight decay = 0>
(1) momentum = 0.0

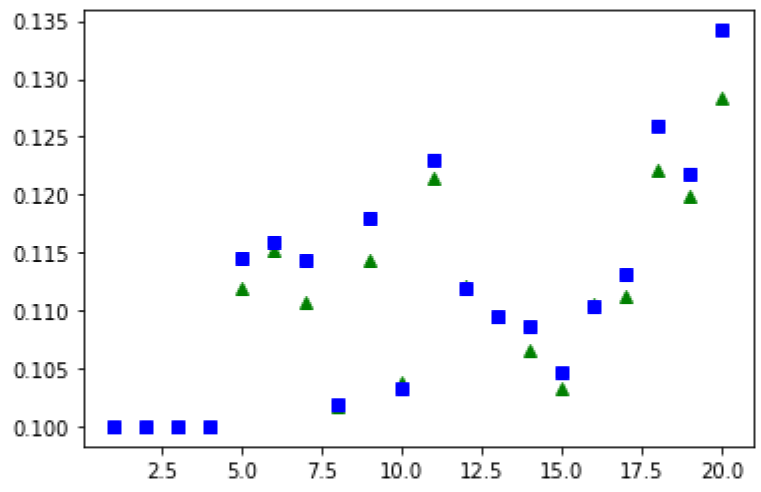

Fig 6.

(2) momentum = 0.3

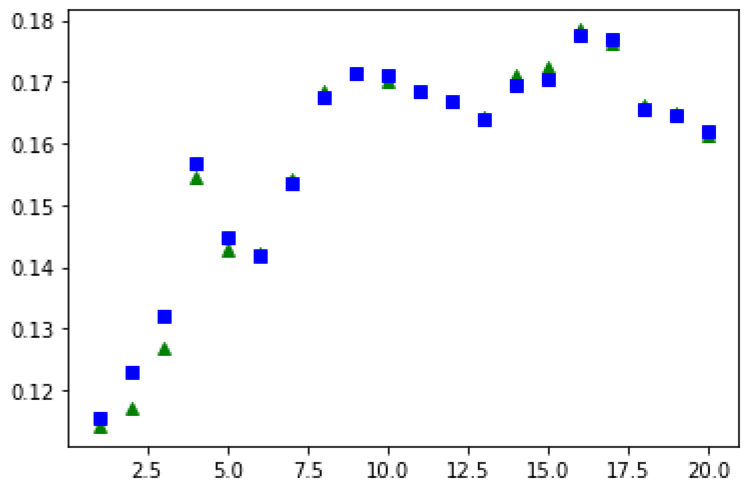

Fig 7.

(3) momentum = 0.5

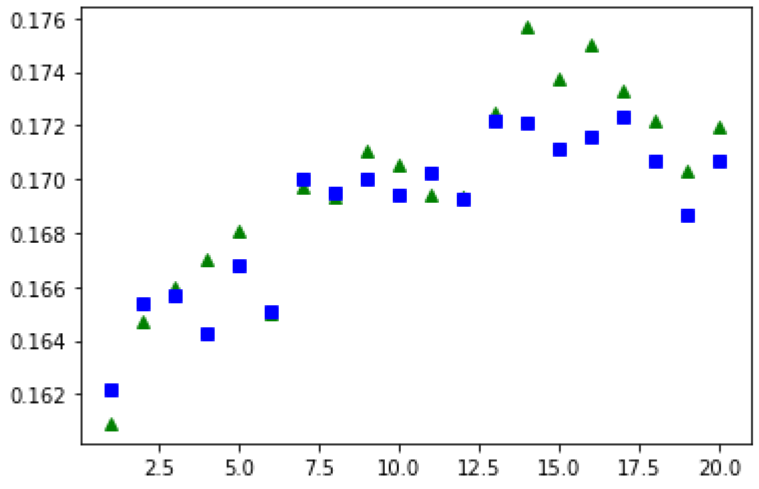

Fig 8.

(4) momentum = 0.7

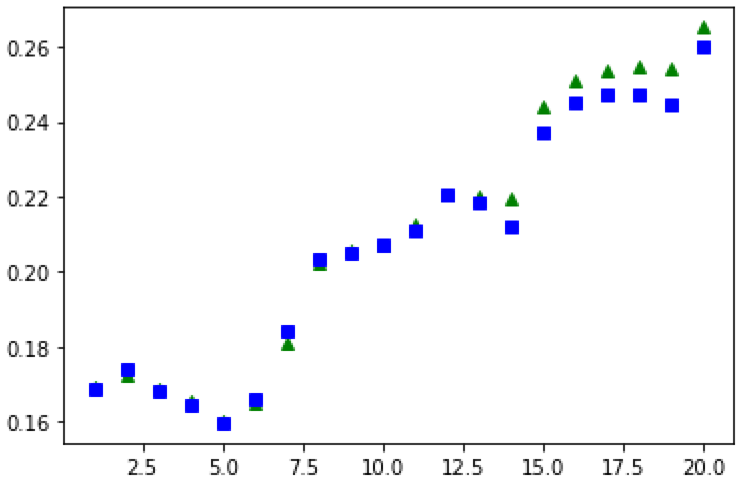

Fig 9.

(5) momentum = 0.9

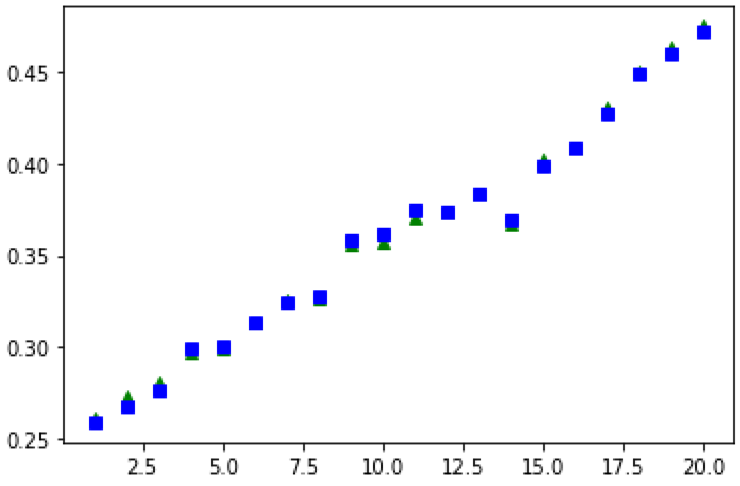

Fig 10.

<5 different values of weight decay & learning rate = 0.001 & momentum = 0.0 >

(1) weight decay = 0

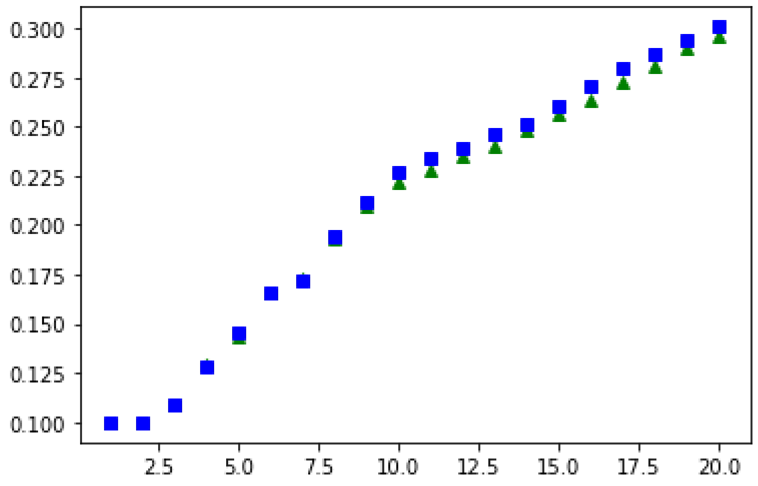

Fig 11.

(2) weight decay = 0.01

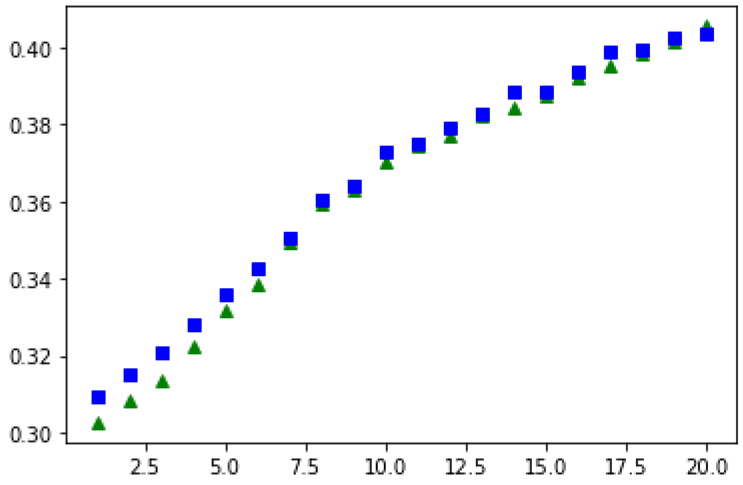

Fig 12.

(3) weight decay = 0.03

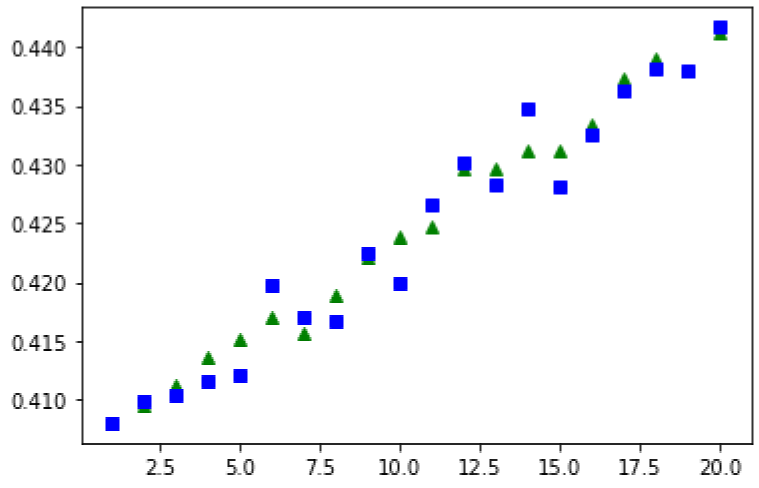

Fig 13.

(4) weight decay = 0.09

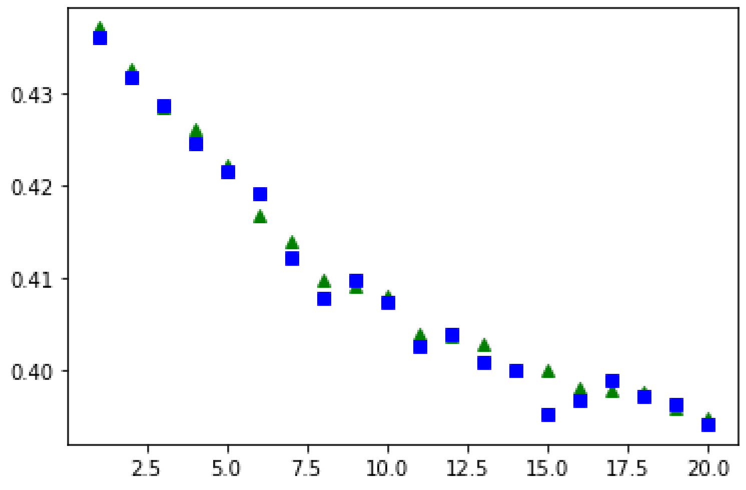

Fig 14.
(5) weight decay = 0.27

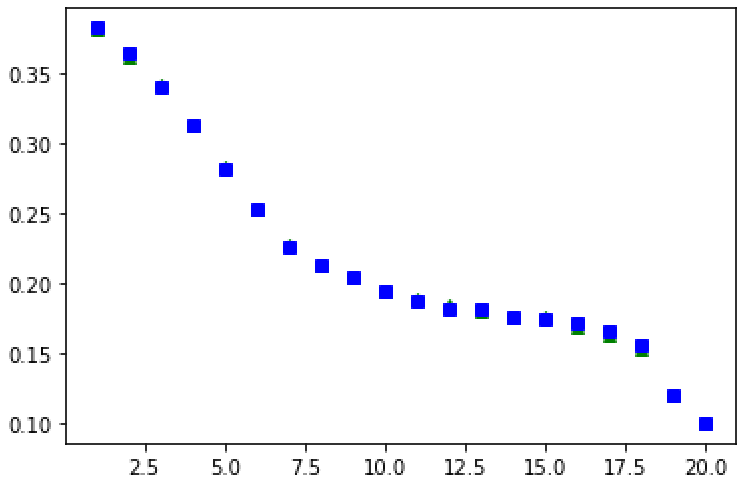

Fig 15.

<Different filter weight initialization methods>
(1) Initialization by normal distribution
(learning rate = 0.001, momentum = 0.0, weight decay = 0)

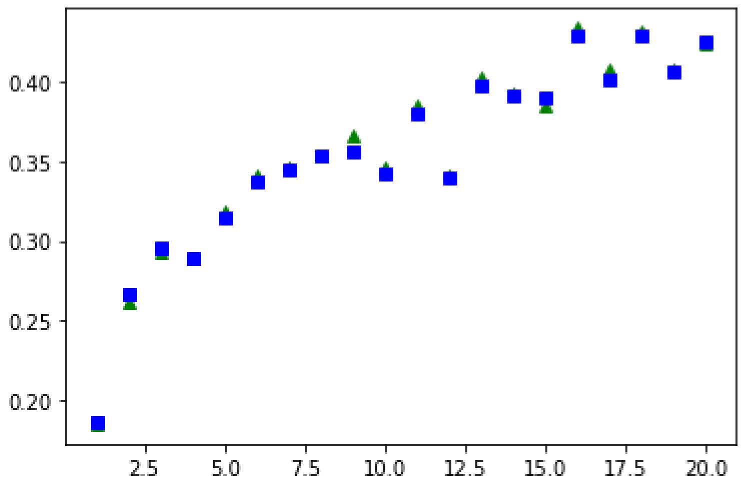

Fig 16.

(2) Initialization by uniform distribution
(learning rate = 0.001, momentum = 0.0, weight decay = 0)

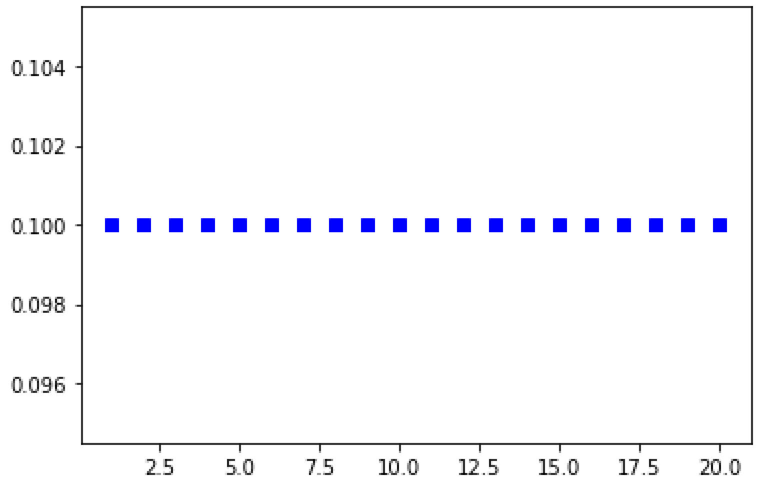

Fig 17.

(3) Initialization by constant 0.2
(learning rate = 0.001, momentum = 0.0, weight decay = 0)

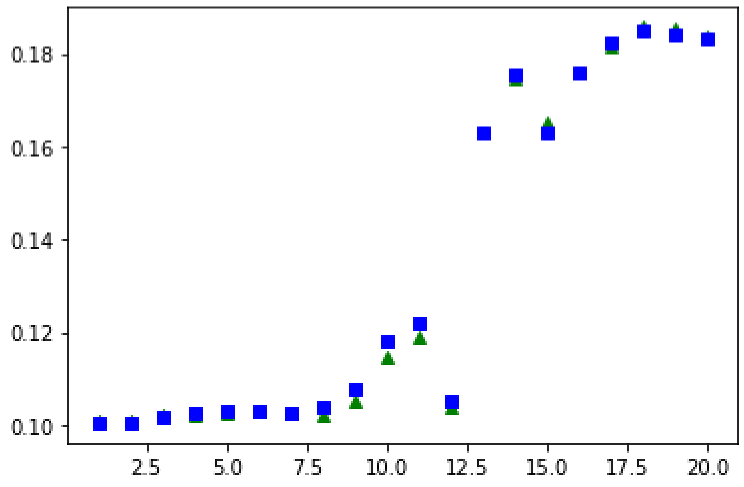

Fig 18.

(4) Initialization by ones
(learning rate = 0.001, momentum = 0.0, weight decay = 0)

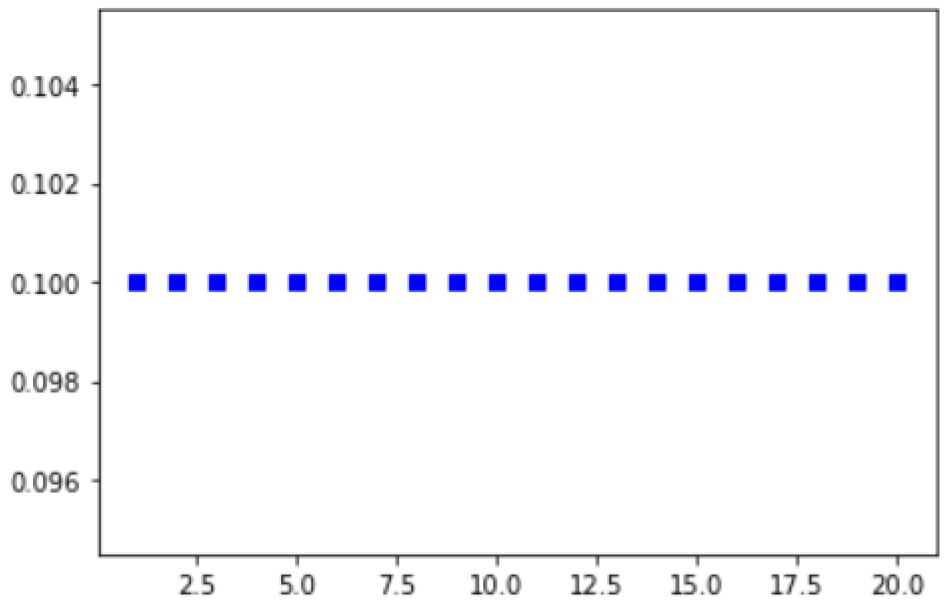

Fig 19.

(5) Initialization by zeros
(learning rate = 0.001, momentum = 0.0, weight decay = 0)

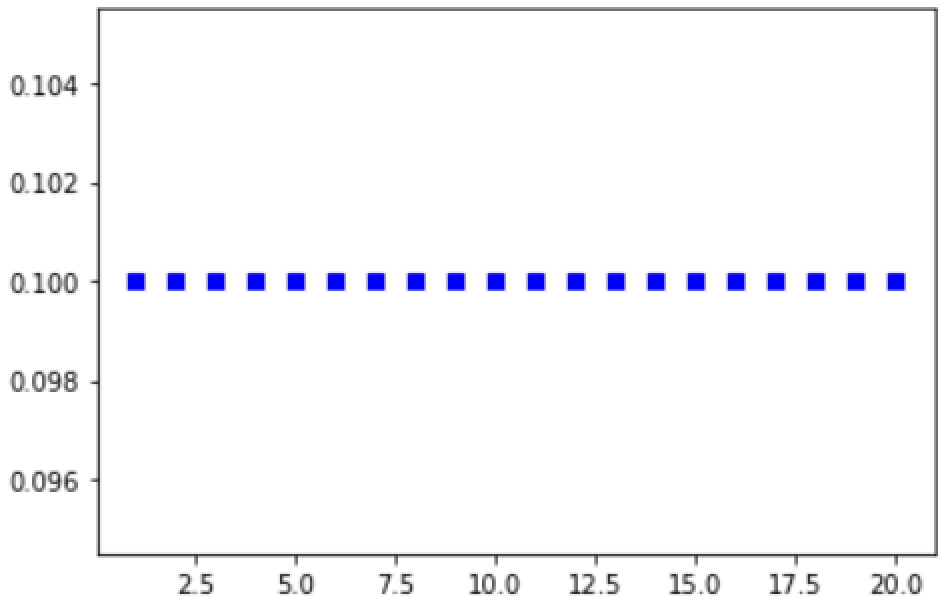

Fig 20.

2.

	Pair (Learning rate, Momentum, Weight decay)	Accuracy on test set
1st pair	(0.001, 0.7, 0.01)	43.2%
2nd pair	(0.001, 0.7, 0.03)	10.0%
3rd pair	(0.001, 0.9, 0.01)	53.8%
4th pair	(0.001, 0.9, 0.03)	31.99%
5th pair	(0.02, 0.7, 0.01)	59.86%
6th pair	(0.02, 0.7, 0.03)	10%
7th pair	(0.02, 0.9, 0.01)	51.74%
8th pair	(0.02, 0.9, 0.03)	10%
9th pair	(0.25, 0.7, 0.01)	10%
10th pair	(0.25, 0.7, 0.03)	10%
11th pair	(0.25, 0.9, 0.01)	10%
12th pair	(0.25, 0.9, 0.03)	10%

Table 3. Accuracy on test data set with 12 different pairs of parameters

=>Chosen best pair of parameters
(Learning rate, momentum, weight decay) = (0.02, 0.7, 0.01)
Number of epochs = 80
<Legend>
Green triangle: accuracy on train data set for each epoch
Blue square: accuracy on test data set for each epoch
x-axis: current epoch
y-axis: accuracy within range 0~1
=>Final accuracy on train set = 65%
Final accuracy on test set = 63%

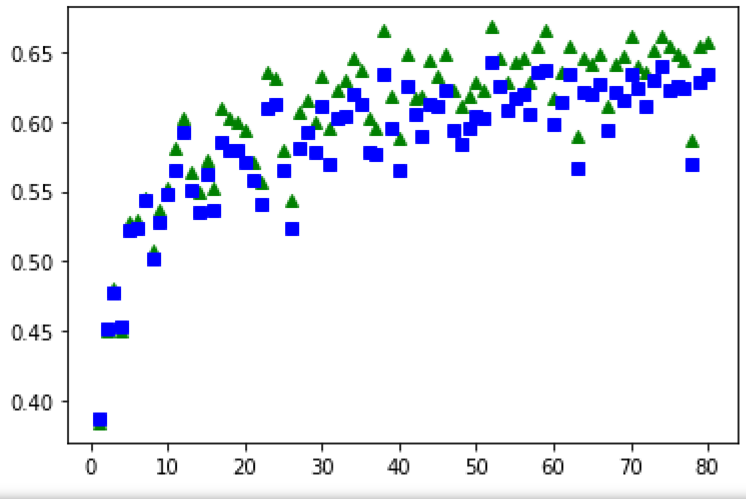

Fig 21. Learning curve with the best pair of parameters

1.4 Discussion
(a)
1.
1)
The function of fully connected layer is to make the extracted data by convolution and maximum pooling layers a feature vector which will be used to classification technique. This can be done flattening the values from the previous layer to 1-D vector. Thus, each image will be represented by a feature vector. This role is called feature extraction. [1] Furthermore, the last fully connected layer plays another important role which is to decide what class the input image belongs to. It is also known as decision subnet. [1]

2)
The role of convolution layer is to detect patterns. In convolution layers, they have many filters with learnable weights. These filters can capture patterns from images, for instance cat's face by learning filter weights from the data by combination of backpropagation and optimization techniques. By adding several convolution layers properly, the CNN will be able to capture more complex patterns from images. However, it will increase the model size meaning time to train the model will be longer. [2]

3)
The function of max pooling is similar to role of down sampling. It lowers information extracted from an input image by filters and decrease dimension while it maintains its important components by pooling maximum value among values in certain size window. This is used after every convolution layer. [3]

4)
Non-linear activation function has two important roles. First one is to introduce non-linearity to the recognition system. The other role is to handle the sign confusion situation caused by several convolution layers. For instance, let's assume we have a CNN made of 2 convolution layers in cascade. Now, negative correlations are suppressed by non-linear activation function. Therefore, the system can avoid sign confusion. [1]

5)
Softmax function maps index to real value which represent probability to be a certain class. It is useful for classification task. That's why it is used at the last fully connected layer.

2.
Overfitting is that since model is over trained by train data set, it performs significantly well on train data set while performance on test data set is relatively poor. It happens when the model is too powerful to learn from train data (i.e., machine learning model has a lot of flexibility) and

data set itself is a very noisy data set. To overcome this issue, there are two methods that can be used in CNN. One is a general solution that people can use a lot of data. Generally speaking, a lot of data tends to suppress effect by noise. Second method is to use appropriate weight decay. It is also known as regularization factor. It prevents CNN from learn too much by train data set. A simple equation is shown below.[4]

$$L'(w) = L(w) + \frac{1}{2}\lambda\|w\|^2$$

$$\frac{\partial L'(w)}{\partial w} = \frac{\partial L(w)}{\partial w} + \lambda w$$

Eq 1. Regularization lambda at optimization process.

Also, we can use data augmentation, and noise injection method. Their principles are the same as the first method using a lot of data to overcome overfitting caused by possible noise in the data set.

3.
In object detection problem in images, CNN can create a very sophisticated match filters by multiple convolution layers with learnable filter weights and multiple optimization steps. This optimization step tunes the filter weights well to capture specific patterns in given images which is very difficult to be achieved by fixed filter weights. Also, the more convolution layers exist, the more complex and wider patterns filter can capture. That's why CNN outperforms many other object recognition methods. Example is given below.

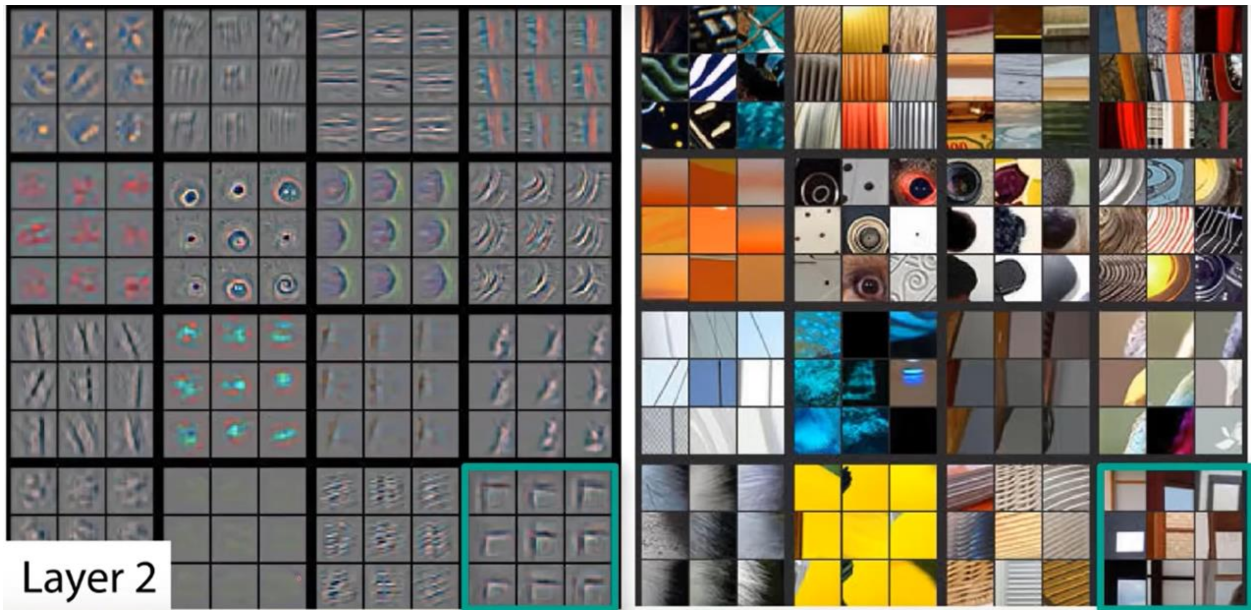

Fig 22. At lower level layer, filter can capture pattern in small space.

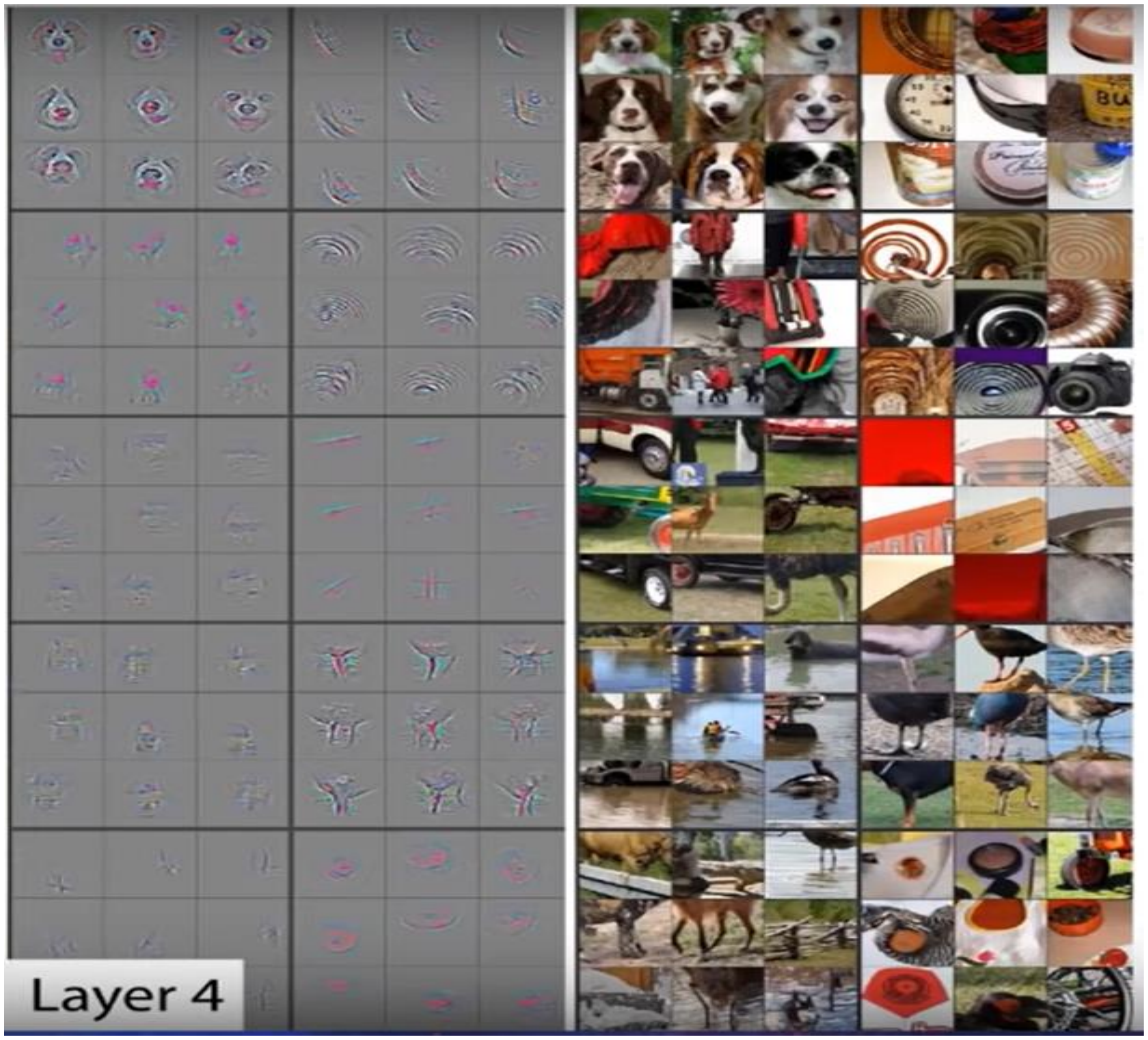

Fig 23. At higher level layer, filter can capture pattern in bigger space.

4.

Loss function is a tool to measure difference between outputs by current trained model and wanted output by labels. For the loss function, we can use one of cross entropy, mean squared error, and much more. In almost every case, since we don't know exact filter weight that can capture patterns from input images, value of loss function is relatively high at the very beginning. Therefore, we need a method to reduce the value of the loss function. This is where the back propagation comes in. Once we calculate the value of the loss function, we can decrease this value by taking derivatives the loss function with respect to weights of previous layers. Since there are no unique solution, we need to introduce gradient descent method which searches the direction to the local optimum. To handle the weights at several layers, we can apply chain rules. By repeating forward propagation and backward propagation, we can reduce the value of the loss function and it will lead us to better accuracy. One of famous optimization methods is stochastic gradient descent. [11]

(b)

1.

(1) learning rate

As we can observe at the experiment results, it contributes to speed of convergence. In general, the bigger value of a learning rate is the faster accuracy converge to certain level. However, if the value is too big, it will cause the zigzag pattern because the large learning rate update the location too much and it misses the local optimum. Therefore, we need to find a value that provides enough speed to converge fast and doesn't lead to the zigzag pattern as well. Illustration of this phenomenon is given below.

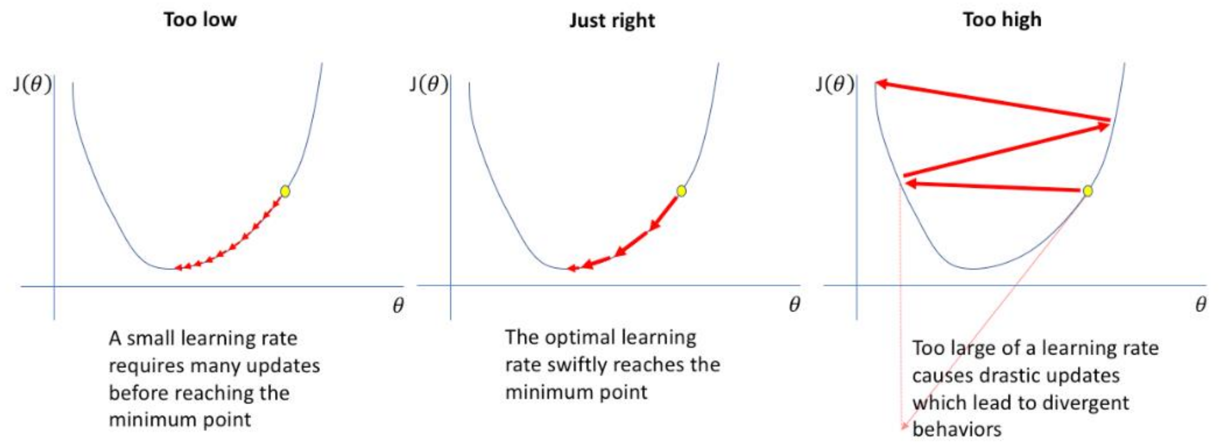
Too low
Just right
Too high
J(θ)
J(θ)
J(θ)
θ
θ
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Fig 24. Illustration to show effect of value of learning rate. [5]

(2) Momentum.

Momentum is a method that is used to avoid for CNN to be stuck in local optimum that is not necessarily providing global optimum. The principle is that when CNN gets staying at some points it makes a "jump" to near location by the value of momentum. We can observe it reduce zig zag pattern caused by many local optima during optimization process through Fig 6 to Fig 10 as the value of momentum becomes bigger.

(3) Weight Decay

As we explained earlier, weight decay is a method to combat overfitting issue in classification task. Generally speaking, when the value of weight decay is relatively big, it prevents the model (hear CNN) from overlearning from the train data. Therefore, difference between accuracy on train data set and accuracy on test data set is very small. However, as we can observe, if this value gets too big like in Fig 14, and Fig15. The model loses the ability to learn filter weights from the data due to too much regularization (i.e., too high value of weight decay). Thus, we need to find a value that can prevent the overfitting and still have ability to learn from the train data as well.

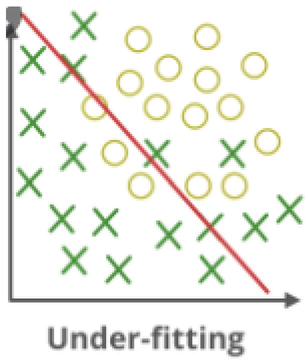

Fig 25. trained model by too high regularization [6]

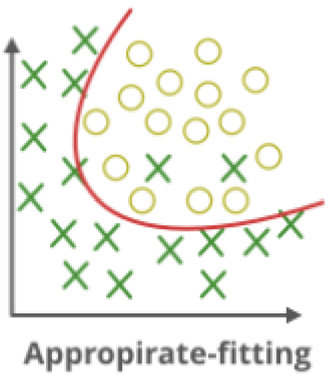

Fig 26. trained model by appropriate regularization [6]

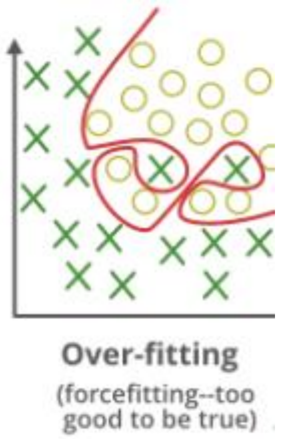

Fig 27. trained model by too powerful learning ability (i.e. too low regularization) [6]

(4) Filter Weights Initialization.
I set momentum to 0 to see importance of filter weights initialization. As we can observe in Fig. 16, and 18, some initializations provide us well behaved learning curves. However, other initializations such as Fig. 17, 19, 20, the optimization process get stuck at local optima because of poor choice of initialization and no momentum to escape. Also, properly chosen initial filter weights can boost up the learning speed because it can shorten the convergence check process in case the learning reaches the maximum accuracy based on the given learning rate, weight

decay, and momentum. However, as I have experienced through experiments, momentum, learning rate, and weight decay play relatively more significant roles in CNN.

2.
Through the experiments in problem 1 (b)-1, we can narrow down the values of parameters such as learning rate, momentum, and weight decay, and filter weights initialization methods. To be specific, the value of learning rate should be large enough to obtain proper speed of learning, but small enough to prevent zigzag pattern on learning curve. In my situation, to be safe, I choose low value for learning rate even though it slows down the speed of learning. Second, for the value of momentum, it should be able to escape from the local optima when the model gets stuck. I chose some large values for the momentum because large values tend to result in better learning curves. Third, value for weight decay needs to be small enough to maintain ability for model to learn filter weight from the data and big enough for model to overcome overfitting. In this practice, I chose low value for the weight decay because I can use a lot of data samples which can be a method to fight against the overfitting.
Chosen parameters are provided as follows

	1st value	2nd value	3rd value
Learning Rate	0.001	0.02	0.25
Momentum	0.7	0.9	-
Weight Decay	0.01	0.03	-

Table 3. Chosen values of parameters for selecting the best pair of parameters

=>Chosen best pair of parameters are given below.
(Learning rate, momentum, weight decay) = (0.02, 0.7, 0.01)
As we expect when learning rate is small, momentum is relatively big, and weight decay is low, it shows best performance on the test data set.
As increasing number of epochs, I could gain slightly better accuracy on test set, 63%. It seems there is not very sever overfitting issue because the gap between accuracy on train set and test set is only 2 percent.

(c)
1.
I picked up four papers to learn how authors could achieve high performance with CIFAR-10 data set. These papers' titles are "All You Need Is A Good Init", "Fractional Max-Pooling", "Striving for Simplicity: The All Convolutional Net", and "Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree". I will call those papers first paper, second paper, third paper, and fourth paper respectively. The first paper suggests a new weight initialization method driven by data set to improve performance. Briefly speaking, it uses variance of the output from each layer and orthonormal matrices that can be derived by given data set. Principle used in the second, third, and fourth paper is quite similar to each other. All these three papers are paying attention to the role of max pooling. In LeNet-5 of our structure, it uses 2x2 size window size with stride. However, in those papers, they come up with different approach. For example, the second

paper suggests non integer window size max pooling. Specifically, it conducts experiments with $\sqrt{2}$ size max pooling. In the third paper it applies a technique that learns the pooling operation from the input data. Lastly, the fourth paper suggests three pooling functions such as mixed max-average pooling, gated max-average pooling, and tree pooling.
All those papers could achieve significant improvement. Their accuracies are 94.16%, 96.53%, 95.59%, and 93.95% respectively.

2.
There are a lot of researches that modifies structure of LeNet-5. For instance, they use a lot of filters, multiple convolution layers, etc. However, this type of modification significantly increases model size to get good prediction results meaning it will be computationally very expensive. For those solutions conducted on the papers mentioned earlier it slightly increases model size due to learnable max-pooling operations. However, this change does not need intensive computation requirements.
Pros and cons of LeNet-5 and other methods on the papers can be summarized as follows

	Pros	Cons
LeNet-5	-Originally simple structure -Original structure works well for simple task such as hand-written digits recognition.	-Need to make a lot of changes in its structure to achieve high performance in more complex task -Computationally expensive in case the structure becomes deeper.
Other methods with modification in Max-pooling operation	-Simply make a modification in max-pooling method -Does not create a lot of burdens in terms of computation.	-Theoretically max-pooling method becomes more complicated. -It can lead to implementation complexity

Table 4. Pros and cons of LeNet-5 and other methods on the paper

References

[1] Kuo, C., 2016. Understanding convolutional neural networks with a mathematical model. *Journal of Visual Communication and Image Representation*, 41, pp.406-413.

[2] deeplizard., 2017. Convolutional Neural Networks (CNNs) explained. YouTube video. [Online]

Available: https://www.youtube.com/watch?v=YRhxdVk_sIs

[3] missinglink.ai. Fully Connected Layers in Convolutional Neural Networks: The Complete Guide. [Online] Available: https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/

[4] Min Zhang, 2020, EE 569 Discussion 11. Course material [Online] Available:

https://courses.uscden.net/d2l/le/content/17205/viewContent/304327/View?ou=17205

[5] Jeremy Jordan, 2018, Setting The Learning Rate Of Your Neural Network. [Online] Available:

https://www.jeremyjordan.me/nn-learning-rate/

[6] Amod Kolwalkar, 2019, Regularization in Machine Learning and Deep Learning. [Online] Available:

https://medium.com/analytics-vidhya/regularization-in-machine-learning-and-deep-learning-f5fa06a3e58a

[7] Benjamin Graham, 2015, Fractional Max-Pooling. [Online] Available:

https://arxiv.org/abs/1412.6071

[8] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015 Striving for Simplicity: The All Convolutional Net. [Online] Available:

https://arxiv.org/pdf/1412.6806.pdf

[9] Dmytro Mishkin, Jiri Matas, 2015, All You Need Is a Good Init. [Online] Available:

https://arxiv.org/abs/1511.06422

[10] Chen-Yu Lee, Patrick W. Gallagher, Zhuowen Tu, 2015, Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree. [Online] Available:

https://arxiv.org/abs/1509.08985

[11] deeplizard, 2018, Backpropagation explained. YouTube video. [Online] Available:

https://www.youtube.com/watch?v=XE3krf3CQls&list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU&index=23