IL2230 HADL
Lab1: Handwritten Digits Recognition from MLP to CNN

Irene Merino, Zhao Linghan, Wang Jiayang, Tu Han

November 14, 2023

Part 1: CNN in Pytorch

1.1 Variance in accuracy based on the number of feature maps in layer 1 and 2.

To study the variance in accuracy in the CNN we create a Python code that iterates through the different values that we want to test in each convolutional layer. In order to achieve the best possible results we have studied which number of epoch is optimal to use for the training.In Figure 1, we study the change in accuracy for different number of feature maps after 10 epochs :

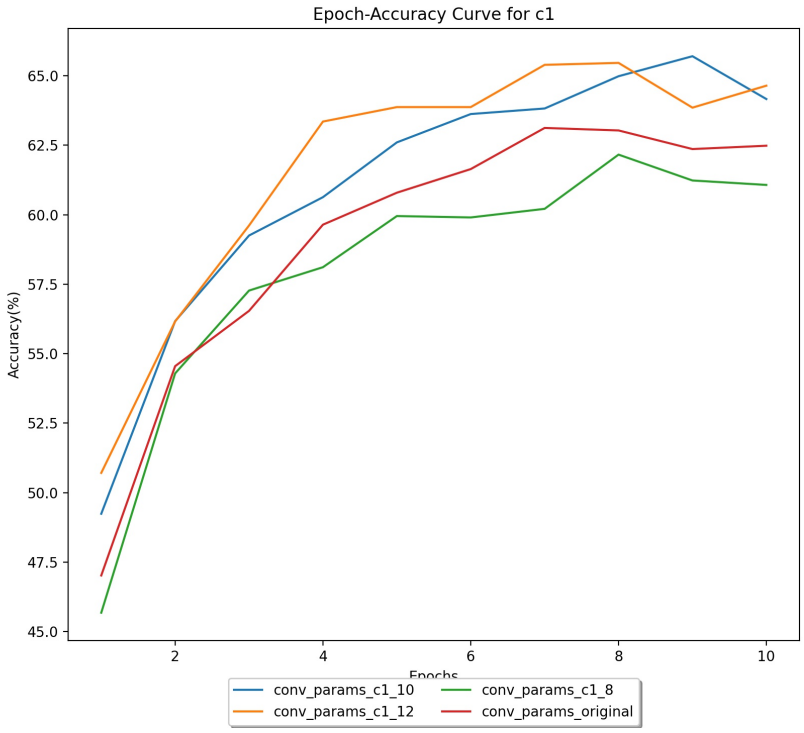

Figure 1: Epoch-Accuracy for Convolutional Layer 1

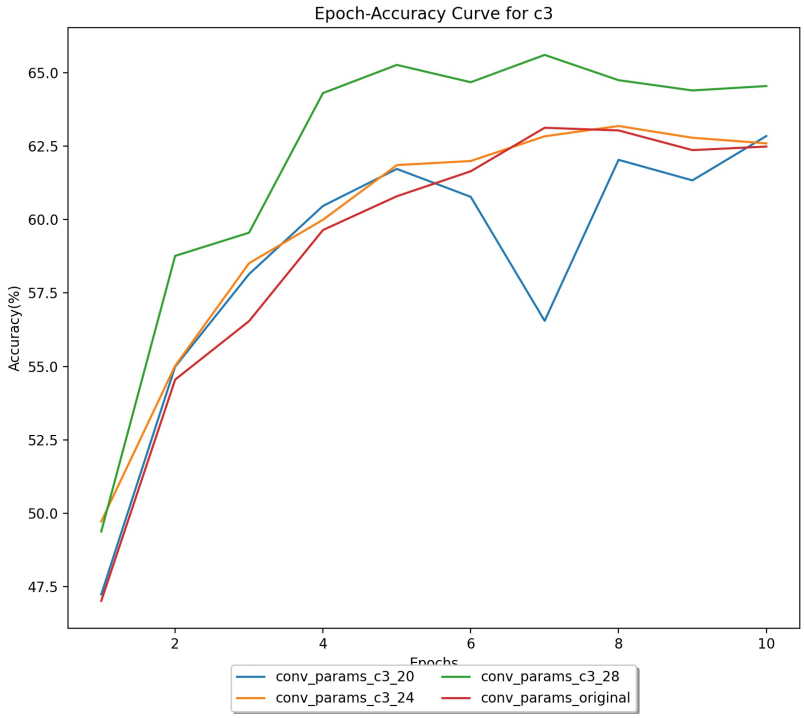

Figure 2: Epoch-Accuracy for Convolutional Layer 2

Now we do the same for the number of feature maps in the second convolutional layer of the network. As the value for the accuracy grows higher when we increase the number of epochs we have expanded the range of our study to 20-100 epochs. As can be seen in Figure 3 and Figure 4, this expansion doesn't result in a higher value of accuracy for the higher values of epochs. This can be due to a problem of overfitting.

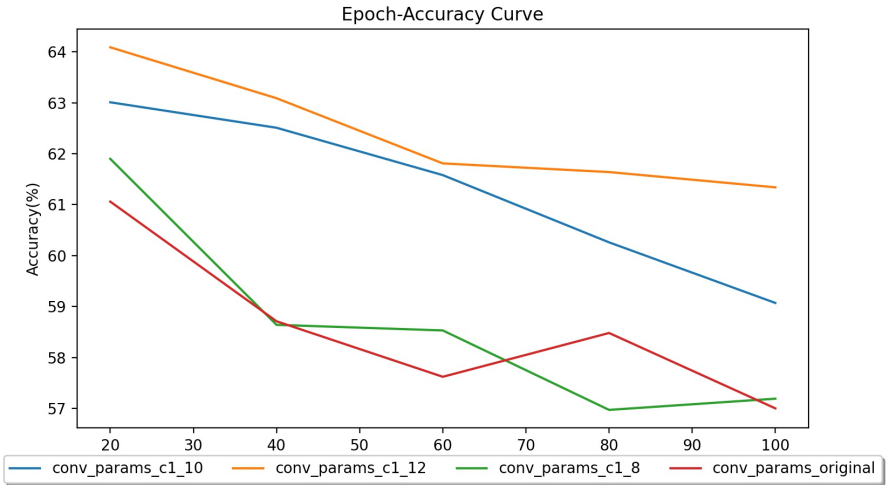

Figure 3: Epoch-Accuracy for Convolutional Layer 1

Analyzing this results we concluded that the best value to use would be 20 epochs. In Figure 5, we can see the difference in variance in accuracy for the different number of feature maps in layer 1 and 2 using 20 epochs.

As can be seen in Figure 5 the difference in accuracy is not of big significance (5%). However, we can conclude that the optimal values of accuracy comes from increasing the number of feature maps in the first layer. Changes in the accuracy based on the number of feature maps in the second layer

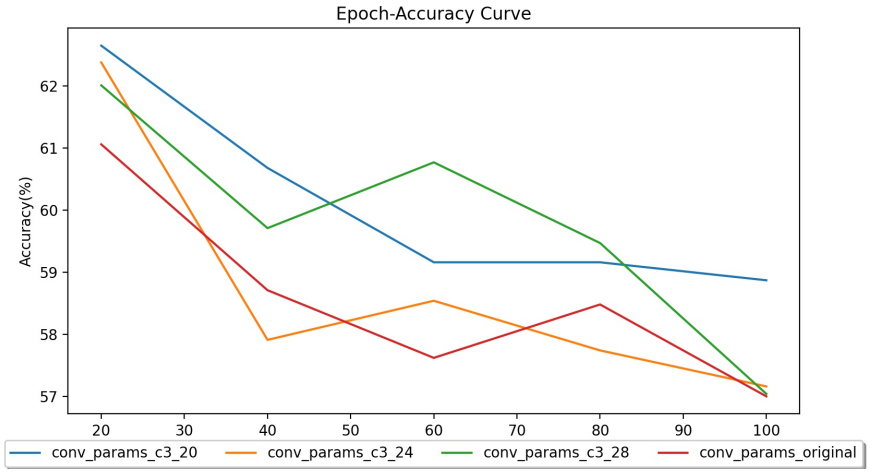

Figure 4: Epoch-Accuracy for Convolutional Layer 2

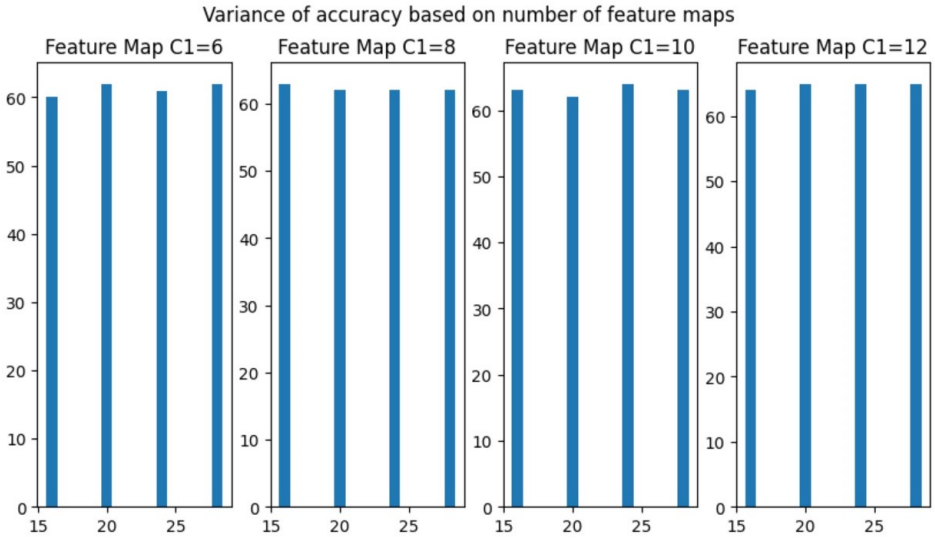

Figure 5: Variance in accuracy based on number of feature maps

of the network are more variable and become constant when the number of feature maps in the first layer is higher.

1.2 Variance in accuracy based on the number of layers in the network.

We use a similar approach than in the previous section to study the difference in variance now for a variation in the number of layers of the CNN. In order to do this we increase the number of convolutional + pooling layers in steps of one. To design these networks we have to consider the size of the output matrix for each filter. This size responds to the following equations.

For the convolutional layer:

$$Output = \frac{(W - K + 2P)}{S} + 1 \tag{1}$$

For the pooling layer:

$$Output = \frac{(W - K)}{S} + 1 \tag{2}$$

Based on this equations we created an excel file to calculate the size of the filters and the values of padding and stride that was needed to achieve the dimensions that we aimed for:

	7 Layer CNN							
	Input image	conv1	pool1	conv2	pool2	fc1	fc2	fc3
Filter size	32	5	2	5	2	120	84	10
Stride		1	2	1	2			
Padding		0	0	0	0			
Result matrix		28	14	10	5			

Figure 6: Parameters for the 7 layer CNN

	9 Layer CNN									
	Input image	conv1	pool1	conv2	pool2	conv3	pool3	fc1	fc2	fc3
Filter size	32	5	2	5	2	5	2	120	84	10
Stride		1	2	1	2	1	2			
Padding		2	0	2	0	2				
Result matrix		32	16	16	8	8	4			

Figure 7: Parameters for the 9 layer CNN

	11 Layer CNN											
	Input image	conv1	pool1	conv2	pool2	conv3	pool3	conv4	pool4	fc1	fc2	fc3
Filter size	32	3	2	3	2	3	2	3	2	120	84	10
Stride		1	2	1	2	1	2	1	2			
Padding		1	0	1	0	1		1				
Result matrix		32	16	16	8	8	4	4	2			

Figure 8: Parameters for the 11 layer CNN

	13 Layer CNN													
	Input image	conv1	pool1	conv2	pool2	conv3	pool3	conv4	pool4	conv5	pool5	fc1	fc2	fc3
Filter size	32	3	1	3	2	3	2	3	2	3	2	120	84	10
Stride		1	1	1	2	1	2	1	2	1	2			
Padding		1	0	1	0	1		1		1				
Result matrix		32	32	32	16	16	8	8	4	4	2			

Figure 9: Parameters for the 13 layer CNN

The number of neurons in each layer is 6 for the first one and then either 16 or 20 for the following ones, we maintain them constant in the different networks as we do not want to measure this parameter in this section. Training this networks with 20 epochs we achieved the following results:

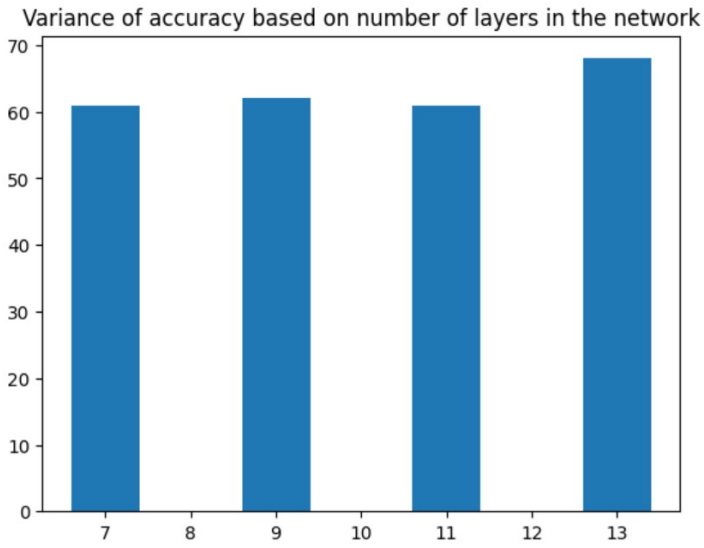

Figure 10: Variance of accuracy based on number of layers in the CNN

As we can see the variance of accuracy based on the number of layers in the network is significant, being optimal for 13 layers in this case.

1.3 Variance in accuracy based on type of activation layer.

Finally, we repeat the same procedure but now we vary the type of activation function that is performed in the output of the convolutional filters and MLP neurons. We will be studying the difference between ReLu, Tanh and Sigmoid, first using 10 epochs to train the networks. We achieved the following results:

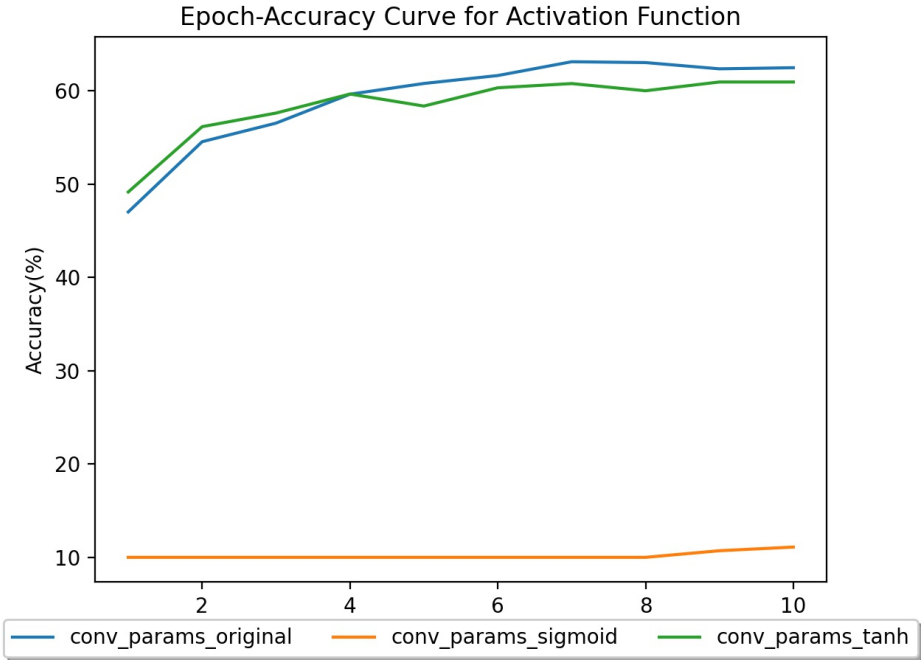

Figure 11: Variance of accuracy based on type of activation function

As we can see ReLu(original) and Tanh present positive results and get a high accuracy in small epochs, on the other hand the sigmoid activation function makes the network perform poorly. However, when we continue increasing the epochs, the model with ReLu or Tanh shows the characteristic of overfitting. Meanwhile, the model with sigmoid reached a high accuracy and reaches the stage of overfitting later than other models:

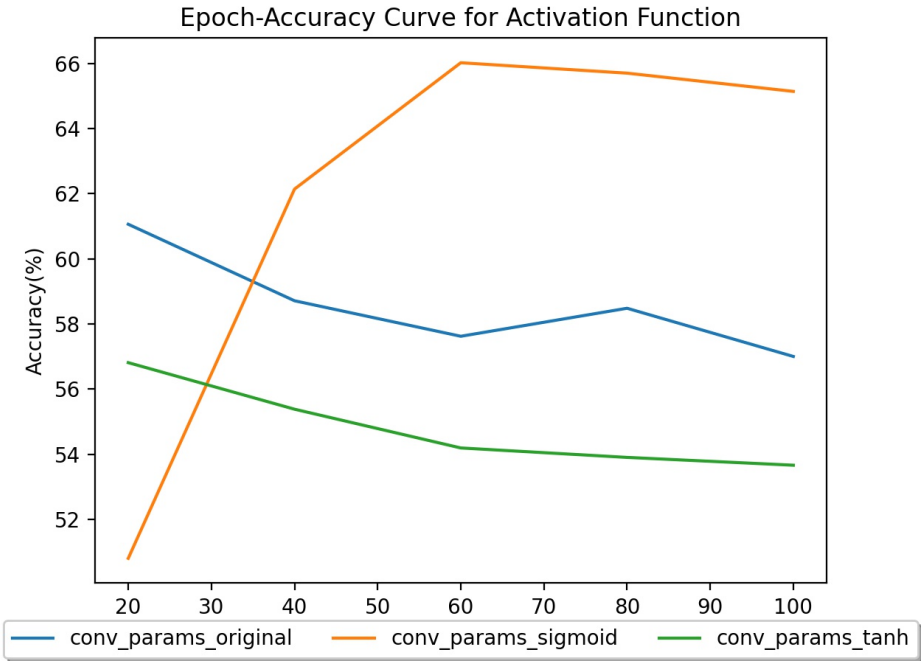

Figure 12: Overfitting accuracy based on type of activation function

1.4 Summary

Based on the analysis above, it's evident that larger feature maps and additional layers typically enhance the performance of a model. And model with Relu will converge quicker than model with tanh or sigmoid. It's also important to keep the training epochs moderate to avoid overfitting. Then by comparing the best performance model in each parameter, we get the benchmark as below:

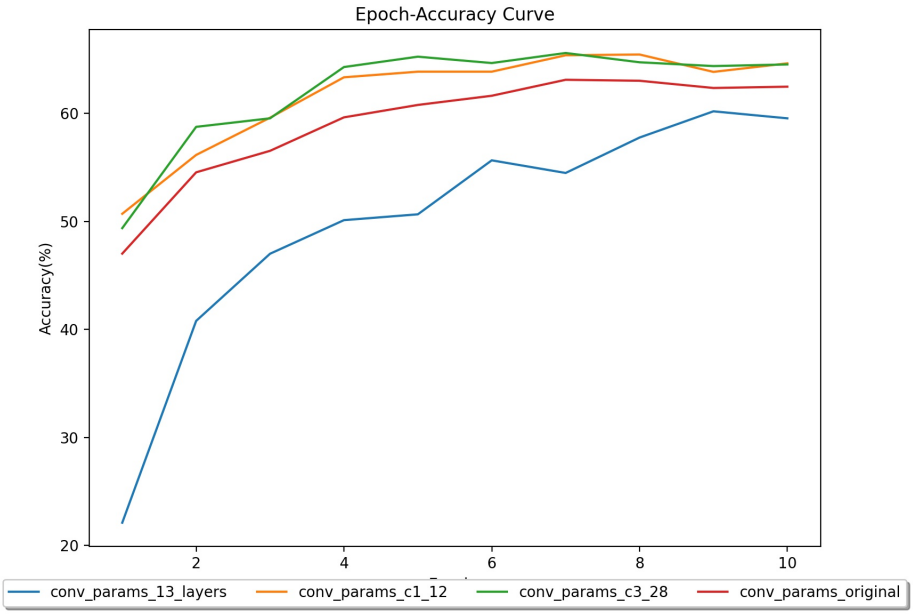

Figure 13: Best accuracy when model is underfitting

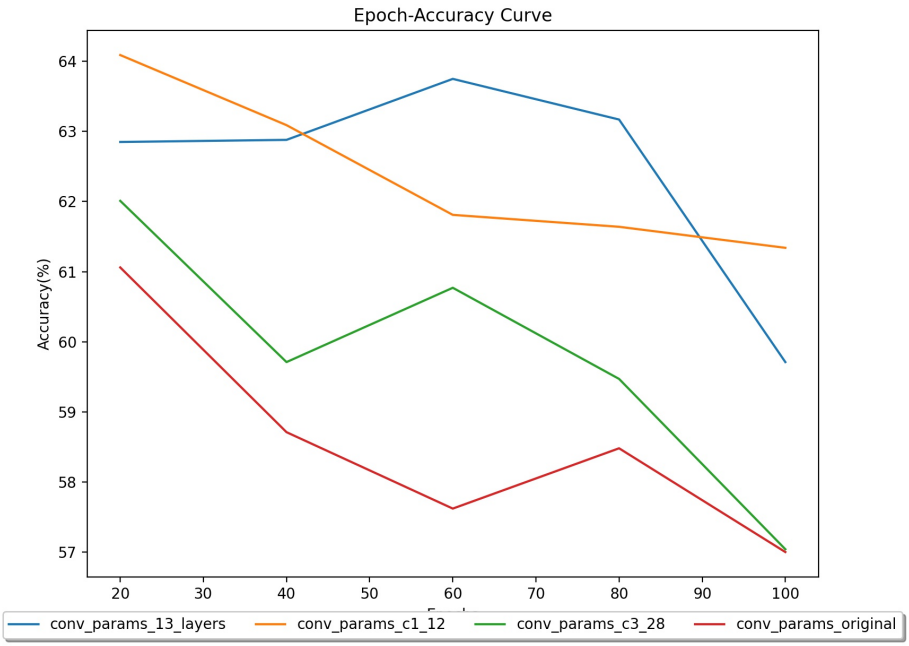

Figure 14: Best accuracy when model is overfitting

The benchmark shows that the feature map size has the most significant impact to the accuracy when model is underfitting. And the model with higher layer will converge slower in the early stage of training so shows a worse accuracy. But when all models have reached its best performance, it's clear that the model with more layer play much better than other model. And larger feature map in the first convolutional layer performs better than a larger feature map in the second convolutional layer.

Part 2: Handwritten Digits Recognition

Part 2 of this lab intends to solve handwritten digits recognition. The data set is the MNIST, which consists of only 10 classes from 0,1, 2, ... to 9.

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image.

2.1 Variance in accuracy of the approaches.

2.1.1 Design and train an MLP

For MLP1 with 1 hidden layer, we try different numbers of neurons for the hidden layer, from 30 to 300 neurons with a step size of 30. And we draw the plot of Accuracy for different neurons here. The best classification accuracy is 96 % , with 240 neurons and 2 training epoches. The graph indicates that the accuracy significantly increases with the number of neurons, reaching a peak before declining as the number continues to grow. This trend can be attributed to the increase in model capacity with more neurons, which enhances the model's learning ability and, consequently, its accuracy. However, when the neuron count becomes excessively high, overfitting may occur, leading to a decrease in accuracy.

The observations suggest that for neuron counts ranging from 30 to 300, the accuracy consistently remains above 93 %. This phenomenon can likely be attributed to the relatively simplistic nature of the MNIST dataset, which comprises solely grayscale images of handwritten digits. For such tasks, even simpler networks with fewer neurons are capable of achieving relatively high accuracy.

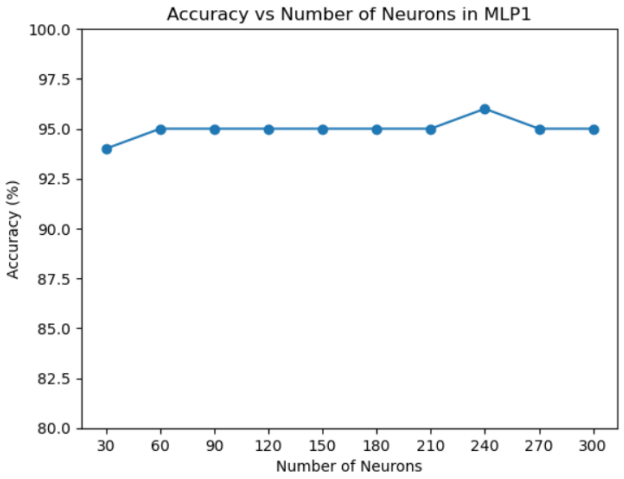

Figure 15: Accuracy-Neurons of MLP1

For MLP2 with 2 hidden layers, we try different numbers of neurons for the two hidden layers. The original numbers are 300 neurons for the first hidden layer and 100 neurons for the second hidden layer.

In our experiment, we established an identical range of neuron counts (10, 20, 50, 100, and 200) for both hidden layers and observed their impact on accuracy. When one layer's neurons changed, in the first layer there is a constant number of neurons equal to 300 and in the second layer the number of neurons is 100. After two training epochs, we noted that as the number of neurons increased, the accuracy for the second hidden layer peaked around 20 neurons, then gradually decreased, whereas the accuracy for the first hidden layer consistently rose. This phenomenon might suggest that the second hidden layer is more prone to overfitting.

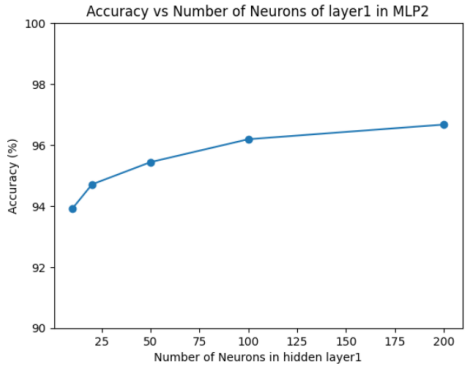

Figure 16: First Hidden Layer

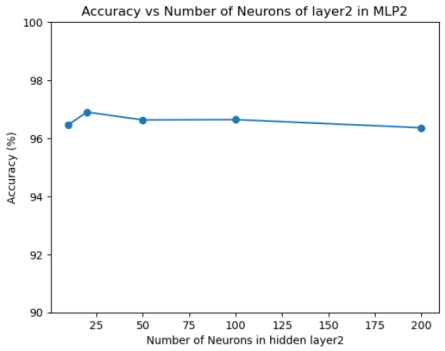

Figure 17: Second Hidden Layer

2.1.2 Train a neural network implementing the LeNet-5 mode

The original LeNet-5 model was designed to process images of size 32x32. Given that our chosen image size is 28x28, adjustments to the code are necessary to accommodate this discrepancy. Common methods for such adaptation include zero padding and altering the parameters of convolutional and pooling layers. In this instance, we opt for the latter approach. After recalculating, we adjust the parameters to 16x4x4 instead of the original 16x5x5. The classification accuracy is 98 % for the network on 10000 test images with 2 training epoches, which is larger than the MLP.

2.2 Practical information and comparison using pytorch profiler

We can evaluate the inference performance of the two types of models by comparing parameters such as classification accuracy, runtime, and memory consumption. And here are the results.

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	CPU Mem	Self CPU Mem	# of Calls
model_inference	72.14%	844.000us	100.00%	1.170ms	1.170ms	0 b	-9.53 Kb	1
aten::linear	1.28%	15.000us	23.93%	280.000us	140.000us	4.84 Kb	0 b	2
aten::addmm	16.41%	192.000us	18.89%	221.000us	110.500us	4.84 Kb	4.84 Kb	2
aten::t	2.22%	26.000us	3.76%	44.000us	22.000us	0 b	0 b	2
aten::relu	1.11%	13.000us	2.14%	25.000us	25.000us	4.69 Kb	0 b	1
aten::reshape	0.68%	8.000us	1.79%	21.000us	21.000us	0 b	0 b	1
aten::transpose	1.11%	13.000us	1.54%	18.000us	9.000us	0 b	0 b	2
aten::copy_	1.54%	18.000us	1.54%	18.000us	9.000us	0 b	0 b	2
aten::view	1.11%	13.000us	1.11%	13.000us	13.000us	0 b	0 b	1
aten::clamp_min	1.03%	12.000us	1.03%	12.000us	12.000us	4.69 Kb	4.69 Kb	1

Self CPU time total: 1.170ms

Figure 18: Performance of MLP1

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	CPU Mem	Self CPU Mem	# of Calls
model_inference	39.66%	397.000us	100.00%	1.001ms	1.001ms	0 b	-15.78 Kb	1
aten::linear	2.00%	20.000us	46.45%	465.000us	155.000us	7.97 Kb	0 b	3
aten::addmm	32.97%	330.000us	37.26%	373.000us	124.333us	7.97 Kb	7.97 Kb	3
aten::relu	8.39%	84.000us	11.49%	115.000us	57.500us	7.81 Kb	0 b	2
aten::t	4.70%	47.000us	7.19%	72.000us	24.000us	0 b	0 b	3
aten::clamp_min	3.10%	31.000us	3.10%	31.000us	15.500us	7.81 Kb	7.81 Kb	2
aten::copy_	2.90%	29.000us	2.90%	29.000us	9.667us	0 b	0 b	3
aten::transpose	1.90%	19.000us	2.50%	25.000us	8.333us	0 b	0 b	3
aten::reshape	0.40%	4.000us	2.40%	24.000us	24.000us	0 b	0 b	1
aten::view	2.00%	20.000us	2.00%	20.000us	20.000us	0 b	0 b	1

Self CPU time total: 1.001ms

Figure 19: Performance of MLP2

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	CPU Mem	Self CPU Mem	# of Calls
model_inference	31.13%	503.000us	100.00%	1.616ms	1.616ms	0 b	-199.03 Kb	1
aten::conv2d	0.99%	16.000us	37.75%	610.000us	305.000us	70.00 Kb	0 b	2
aten::convolution	3.90%	63.000us	36.76%	594.000us	297.000us	70.00 Kb	0 b	2
aten::_convolution	1.67%	27.000us	32.86%	531.000us	265.500us	70.00 Kb	0 b	2
aten::mkldnn_convolution	28.59%	462.000us	31.19%	504.000us	252.000us	70.00 Kb	0 b	2
aten::linear	1.24%	20.000us	14.79%	239.000us	79.667us	3.34 Kb	0 b	3
aten::addmm	8.66%	140.000us	10.89%	176.000us	58.667us	3.34 Kb	3.34 Kb	3
aten::max_pool2d	0.56%	9.000us	9.34%	151.000us	75.500us	52.50 Kb	0 b	2
aten::max_pool2d_with_indices	8.79%	142.000us	8.79%	142.000us	71.000us	52.50 Kb	52.50 Kb	2
aten::relu	2.66%	43.000us	5.94%	96.000us	24.000us	73.19 Kb	0 b	4
aten::clamp_min	3.28%	53.000us	3.28%	53.000us	13.250us	73.19 Kb	73.19 Kb	4
aten::t	1.49%	24.000us	2.66%	43.000us	14.333us	0 b	0 b	3
aten::empty	1.61%	26.000us	1.61%	26.000us	6.500us	70.00 Kb	70.00 Kb	4
aten::copy_	1.42%	23.000us	1.42%	23.000us	7.667us	0 b	0 b	3
aten::transpose	0.87%	14.000us	1.18%	19.000us	6.333us	0 b	0 b	3
aten::flatten	0.31%	5.000us	1.05%	17.000us	17.000us	0 b	0 b	1
aten::as_strided_	0.80%	13.000us	0.80%	13.000us	6.500us	0 b	0 b	2
aten::expand	0.74%	12.000us	0.80%	13.000us	4.333us	0 b	0 b	3
aten::view	0.74%	12.000us	0.74%	12.000us	12.000us	0 b	0 b	1
aten::as_strided	0.37%	6.000us	0.37%	6.000us	1.000us	0 b	0 b	6
aten::resize_	0.19%	3.000us	0.19%	3.000us	1.500us	0 b	0 b	2
aten::resolve_conj	0.00%	0.000us	0.00%	0.000us	0.000us	0 b	0 b	6

Self CPU time total: 1.616ms

Figure 20: Performance of LeNet-5

For execution time:
LeNet-5's model inference accounts for 33.20% of CPU time, totaling 504 microseconds. MLP's model inference occupies 39.66% of CPU time, totaling 397 microseconds. Although MLP has a higher percentage of CPU usage during inference, its actual time taken is slightly shorter, which may indicate that MLP is faster in single inference operations compared to LeNet-5.

For memory consumption:
In the LeNet-5 data, model inference reduced memory usage by about 199.03KB. In the MLP data, model inference reduced memory usage by 15.78KB. This may indicate that LeNet-5 manages memory

more effectively during inference, or it processes more intermediate data, resulting in more memory being freed up after inference is complete.

For inference performance:
Through image comparison, we can also see that convolutional operations constitute a significant portion of the processing time in LeNet-5 including time for "aten::conv2d", "aten::convolution", etc. In contrast, for MLP, the primary operations are associated with fully connected layers including time for "aten::linear", " aten::addmm", etc.

For classification accuracy:
Based on the previous experimental results, it is evident that after the same 2 training epochs, LeNet-5 exhibits higher accuracy (98%), whereas MLP achieves only 96%.

2.3 Summary

Based on the previous analysis, it can be deduced that in terms of graphical processing, the LeNet-5 model exhibits superior accuracy and memory management capabilities, albeit at the expense of increased running time. Moreover, augmenting the training epochs can effectively enhance the inferential accuracy for both MLP and LeNet-5. Additionally, for MLP, while an increment in neurons may bolster accuracy to some extent, considerations must be given to potential overfitting issues and neuron saturation.

TA's question1: Why is your self CPU memory negative? Answer: Because it's the memory released after the code executed.

TA's question2: Why does the memory usage of model inference not equal the sum of all the parameters listed below? Answer: We have updated the graph. In the previous one, the table only shows the first 10 rows due to the code "row limit=10". And this removes some of the usage of memory usage.