IL2230 Lab 1

Kevin Pettersson (kevinpet@kth.se)
Ruolan Wang (ruolanw@kth.se)
Mathieu Leconte (mlec@kth.se)
Xinyi Zhang (xinyz@kth.se)

November 14th 2023

1 Note

All settings used for training and validation can be seen in the attached code with this hand in.

2 Tutorial

- What is the accuracy of the image classifier?
  - 56%
- What is the accuracy for each image class?
  - Accuracy for class: plane is 67.5 %
  - Accuracy for class: car is 65.1 %
  - Accuracy for class: bird is 49.3 %
  - Accuracy for class: cat is 28.7 %
  - Accuracy for class: deer is 36.6 %
  - Accuracy for class: dog is 55.2 %
  - Accuracy for class: frog is 71.4 %
  - Accuracy for class: horse is 63.2 %
  - Accuracy for class: ship is 58.8 %
  - Accuracy for class: truck is 66.6 %

3 Exploration

Results were obtained with an epochs of 2 unless specified otherwise.

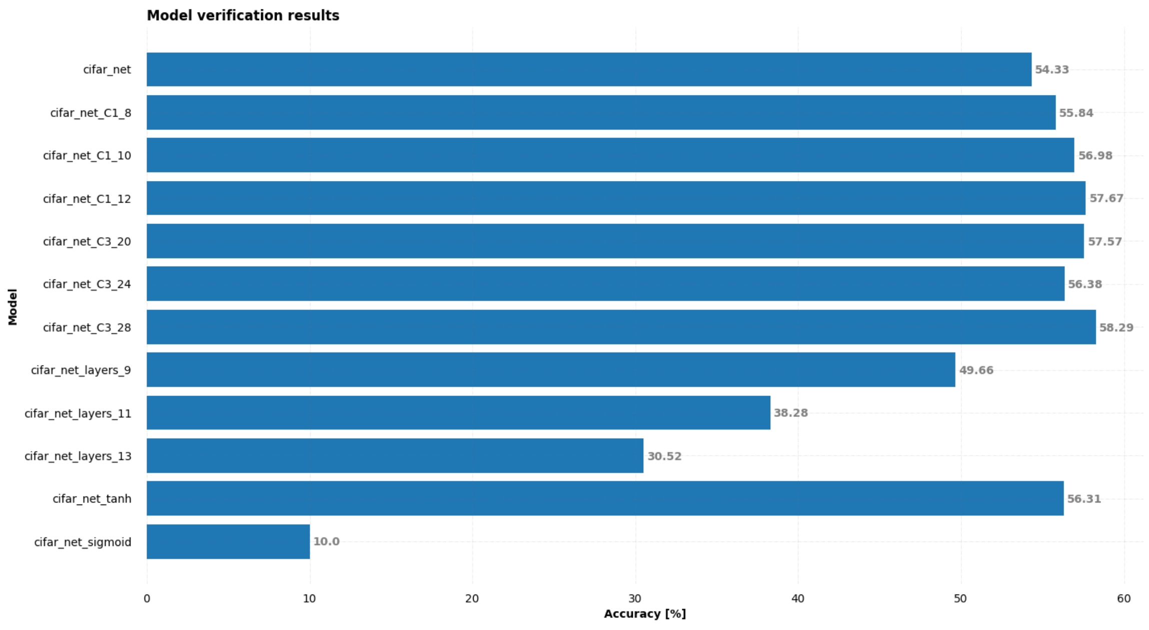

Figure 1: Models versus interference accuracy on the CIFAR10 dataset.

3.1 Impact of feature map

3.1.1 Structure

First we increase the number of C1's feature map, from 6 feature maps to 8,10,12.

structure of CNN with C1's feature map=6, C2's feature map=16(original):

- (conv1): Conv2d(3, 6, kernel_ size=(5, 5), stride=(1, 1))
- (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
- (fc1): Linear(in_features=400, out_features=120, bias=True)
- (fc2): Linear(in_features=120, out_features=84, bias=True)
- (fc3): Linear(in_features=84, out_features=10, bias=True)

structure of CNN with C1's feature map=8, C2's feature map=16:

- (conv1): Conv2d(3, 8, kernel_ size=(5, 5), stride=(1, 1))
- (conv2): Conv2d(8, 16, kernel_size=(5, 5), stride=(1, 1))
- (fc1): Linear(in_features=400, out_features=120, bias=True)
- (fc2): Linear(in_features=120, out_features=84, bias=True)
- (fc3): Linear(in_features=84, out_features=10, bias=True)

structure of CNN with C1's feature map=10, C2's feature map=16:

- (conv1): Conv2d(3, 10, kernel_ size=(5, 5), stride=(1, 1))
- (conv2): Conv2d(10, 16, kernel_size=(5, 5), stride=(1, 1))
- (fc1): Linear(in_features=400, out_features=120, bias=True)
- (fc2): Linear(in_features=120, out_features=84, bias=True)
- (fc3): Linear(in_features=84, out_features=10, bias=True)

structure of CNN with C1's feature map=12, C2's feature map=16:

- (conv1): Conv2d(3, 12, kernel_ size=(5, 5), stride=(1, 1))
- (conv2): Conv2d(12, 16, kernel_size=(5, 5), stride=(1, 1))
- (fc1): Linear(in_features=400, out_features=120, bias=True)
- (fc2): Linear(in_features=120, out_features=84, bias=True)
- (fc3): Linear(in_features=84, out_features=10, bias=True)

Second we increase the number of C2's feature map, from 16 feature maps to 20,24,28

structure of CNN with C1's feature map=6, C2's feature map=20:

- (conv1): Conv2d(3, 6, kernel_ size=(5, 5), stride=(1, 1))
- (conv2): Conv2d(6, 20, kernel_size=(5, 5), stride=(1, 1))
- (fc1): Linear(in_features=500, out_features=120, bias=True)
- (fc2): Linear(in_features=120, out_features=84, bias=True)
- (fc3): Linear(in_features=84, out_features=10, bias=True)

structure of CNN with C1's feature map=6, C2's feature map=24:

- (conv1): Conv2d(3, 6, kernel_ size=(5, 5), stride=(1, 1))
- (conv2): Conv2d(6, 24, kernel_size=(5, 5), stride=(1, 1))
- (fc1): Linear(in_features=600, out_features=120, bias=True)
- (fc2): Linear(in_features=120, out_features=84, bias=True)
- (fc3): Linear(in_features=84, out_features=10, bias=True)

structure of CNN with C1's feature map=6, C2's feature map=28:

- (conv1): Conv2d(3, 6, kernel_ size=(5, 5), stride=(1, 1))
- (conv2): Conv2d(6, 28, kernel_size=(5, 5), stride=(1, 1))
- (fc1): Linear(in_features=700, out_features=120, bias=True)
- (fc2): Linear(in_features=120, out_features=84, bias=True)
- (fc3): Linear(in_features=84, out_features=10, bias=True)

3.1.2 Performance

We estimate each CNN's performance by recording the total accuracy.

Table 1: C1's feature maps increases,C2's feature maps =16

C1	6	8	10	12	14
total accuracy	54%	55%	56%	57%	59%

Table 2: C2's feature maps increases,C1's feature maps =6

C2	16	20	24	28	32
total accuracy	54%	54%	56%	57%	58%

3.1.3 Discussion

We can observe a slight but positive impact of the increase in number of feature maps for this network, meaning that this factor is non-negligible but also not critical in terms of accuracy. In Figure 1 we see the increasing accuracy with the increasing, however, since we used such low epochs we don't see the clear increase in accuracy as expected when increasing C3.

3.2 Impact of layer depth

3.2.1 Structure

We modify the structure by adding each time a convolution layer followed by a pooling layer in order to reach 9, 11 and 13 layers. From 11 layers onward we add a padding of 1 to the pooling layer. The filter size is also decreased in order to keep the data size to a minimum.
9 layers:
- (conv1): Conv2d(3, 6, kernel_ size=(3, 3), stride=(1, 1))
- (conv2): Conv2d(6, 12, kernel_ size=(3, 3), stride=(1, 1))
- (conv3): Conv2d(12, 16, kernel_ size=(3, 3), stride=(1, 1))

11 layers:
- (conv1): Conv2d(3, 6, kernel_ size=(2, 2), stride=(1, 1))
- (conv2): Conv2d(6, 8, kernel_ size=(2, 2), stride=(1, 1))
- (conv3): Conv2d(8, 12, kernel_ size=(2, 2), stride=(1, 1))
- (conv4): Conv2d(12, 16, kernel_size(2, 2), stride=(1, 1))

13 layers:
- (conv1): Conv2d(3, 6, kernel_ size=(2, 2), stride=(1, 1))
- (conv2): Conv2d(6, 8, kernel_ size=(2, 2), stride=(1, 1))
- (conv3): Conv2d(8, 12, kernel_ size=(2, 2), stride=(1, 1))
- (conv4): Conv2d(12, 14, kernel_size(2, 2), stride=(1, 1))
- (conv5): Conv2d(14, 16, kernel_size(2, 2), stride=(1, 1))

3.2.2 Performance

Table 3: Performance with different layers, epoch=10

Layer	7	9	11	13
total accuracy	54%	57%	54%	50%

3.2.3 Discussion

The accuracy drops significantly when adding convolution layers to the network. This can be explained by a too high compression rate of the subsequent layers which doesn't leave enough data in order to have correct guesses. The number and parameters of convolution/sub-sampling layers should be calculated carefully in order to avoid this.

3.3 Impact of nonlinear function

3.3.1 Structure

The structure is the base model structure, we only change the activation function from rectified linear unit to sigmoid and tanh.

3.3.2 Performance

Total accuracy is recorded for each activation function.

Table 4: Different activation functions

Non-linear function	ReLu	Sigmoid	Tanh
total accuracy	54%	10%	58%

3.3.3 Discussion

The tanh function seems to achieve slightly better result than ReLu, but the sigmoid function doesn't seem to work, as the network achieve only 10% accuracy which is equal to random guess. This indicate that the choice of activation function is an extremely important factor in the architecture of a network.

3.4 Summary

The parameter which seem to have the most impact on accuracy is the activation function, although negative, since the sigmoid basically renders the network inaccurate. The layer depth comes after this, if the number and parameters of convolution layers aren't tuned for the size of the data then the compression rate is too high and accuracy drops. Finally, the number of feature maps has a small but positive impact on accuracy.

4 Handwritten Digits Recognition

4.1 MLP

4.1.1 Procedure

- We set MLP1 with the following parameters:

```
params_mlp1 = {"n_hidden_layers": 1, "hidden_neurons": [],"input_size": trainset[0][0].shape[-1]**2, "n_classes": 10}
```

- We use the for loop to implement MLP1 with different numbers of neurons:

```
for neurons in range(30, 301, 30):
    params_mlp1["hidden_neurons"] = [neurons]
    name_mlp1 = "MLP1" + str(neurons)
    train_mlp(params_mlp1, name_mlp1, device, 20)
```

- For MLP2, we can try 300 neurons for the first hidden layer and 100 neurons for thesecond hidden layer:

```
params_mlp2 = {"n_hidden_layers": 2, "hidden_neurons": [300, 100], "input_size": trainset[0][0].shape[-1]**2, "n_classes": 10}
```

4.1.2 Result

We train MLP2_300_100 with epochs ranging from 2 to 20 in order to draw the evolution of the loss and training accuracy relative to training length. We can then determine the optimal number of epochs and if we begin to overfits. We can see in Figure 2, 3 that after 10-15 epochs the testing accuracy is negligible, therefore, to save time we elected to use 10 epochs for all our testing in this part.

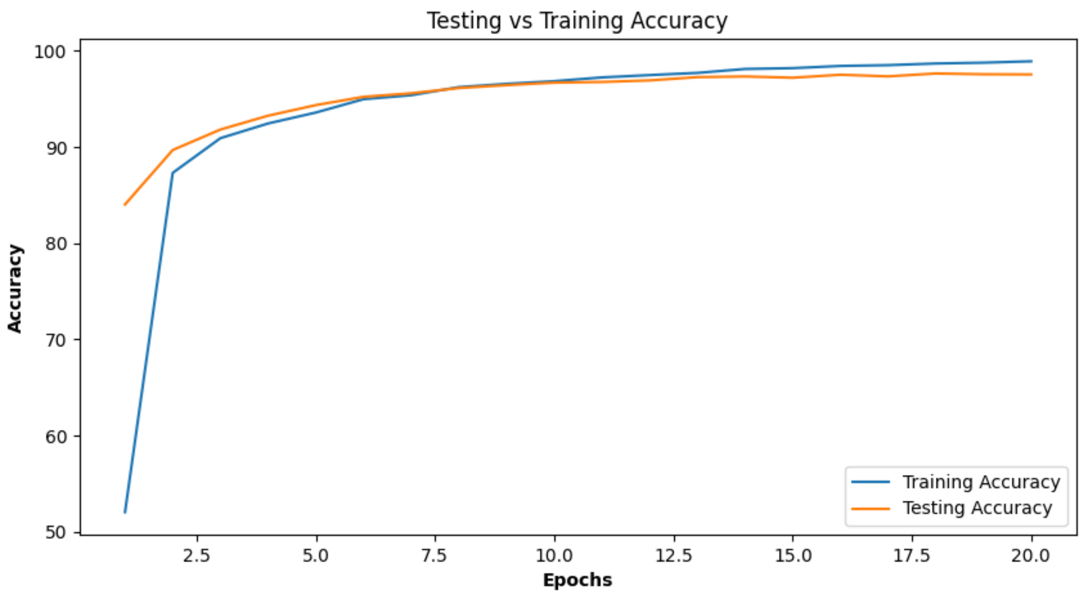

Figure 2: Testing vs training accuracy for MLP2 with 300 neurons for the first hidden layer and 100 for the second hidden layer.

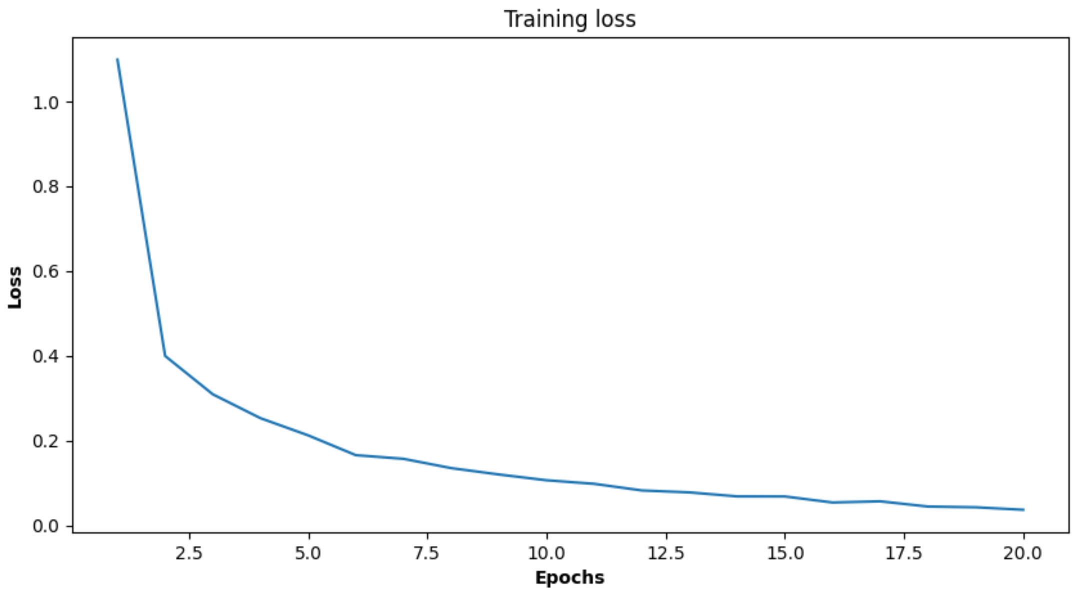

Figure 3: Training loss for MLP2 with 300 neurons for the first hidden layer and 100 for the second hidden layer.

4.1.3 Discussion

For each MLP, 1 or 2 hidden layer, we see a slight increase in accuracy and run time when increasing the number of neurons. The MLP2 has slightly better performance at the cost of more memory usage, around 100 Mb for MLP2 and 40Mb for MLP1, and very slightly increased run time. We also see that the training length in epochs is an important factor to determine correctly. LeNet5 has the best performance as can be seen in Figure 4, but uses the most memory 1.25Gb a massive increase of the MLPs and a slight longer CPU time.

4.2 LeNet-5

4.2.1 Procedure

- First We have to normalize the image from 28*28 to 32*32 in order to adapt in the LeNet-5 model.

- We write and train a neural network implementing the LeNet-5 model:

params_lenet5 = {'n_conv': 2, 'n_fc': 3, 'conv_ch': [6, 16], 'filter_size': [5, 5], 'fc_size': [120, 84], 'pooling_size': 2, 'input_size': trainset[0][0].shape[-1], 'input_channels': 1, 'n_classes': len(classes), 'activation_fn': F.relu}

4.2.2 Result

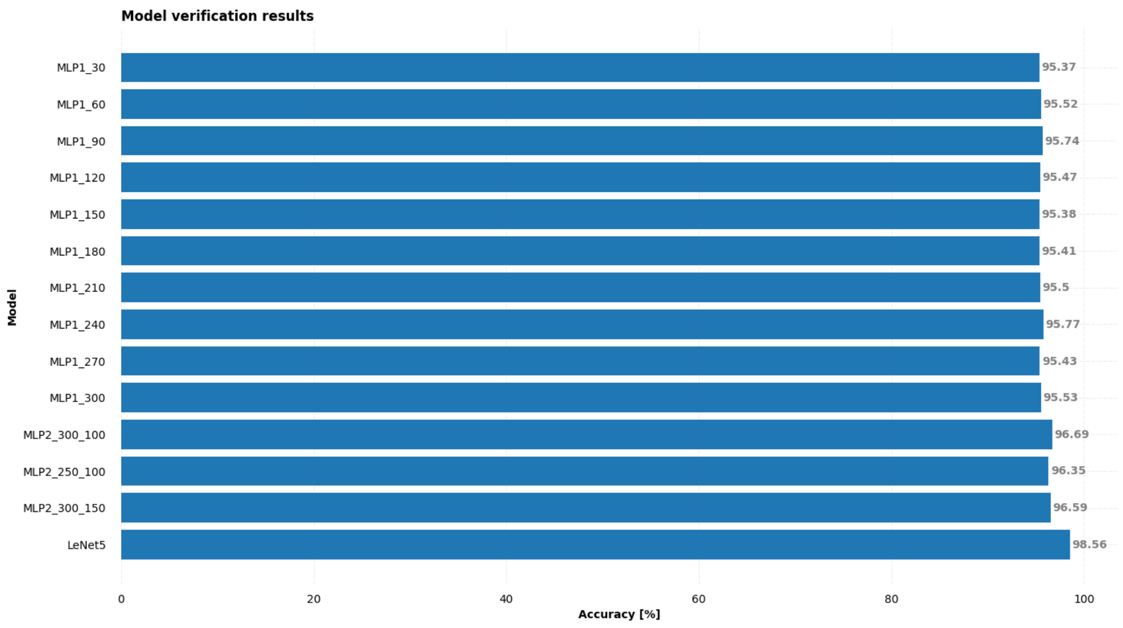

Figure 4: Models versus interference accuracy on the MNIST dataset.

Table 5: CPU time and memory usage for validation of models.

Model	Total CPU time [s]	Total memory usage [Gb]
MLP1_30	2.190	0.040
MLP1_60	2.193	0.040
MLP1_90	2.087	0.040
MLP1_120	2.162	0.040
MLP1_150	2.153	0.040
MLP1_180	2.191	0.040
MLP1_210	2.461	0.040
MLP1_240	2.187	0.040
MLP1_270	2.171	0.040
MLP1_300	2.190	0.040
MLP2_300_100	2.260	0.107
MLP2_250_100	2.192	0.096
MLP2_300_150	2.266	0.114
LeNet5	2.668	1.253

4.2.3 Discussion

LeNet-5 architecture performs about as well in accuracy as MLPs but is slightly slower and is way more inefficient in regard to memory since is uses about 1.5Gb of memory so more than an order of magnitude of difference.