CSCI3230 / ESTR3108: Fundamentals of Artificial Intelligence 2023-2024 Term 1
The Chinese University of Hong Kong

Assignment 4

Due date: 10 Dec 2023 (Sun) 23:59

Full mark: 100
Expected time spent: 8-10 hours

Aims: 1. Understand knowledge about neural networks and hands-on programming of convolutional neural networks, including training and testing.
2. Enhance the understanding of uninformed and informed search, and hands-on programming of search algorithms such as DFS, BFS, UCS and A*.

Assignment 4 has two programming parts. The first part is about CNN, the second part is about search.

- For Part 1, please put all your plots and answers into a <ID>_part1.pdf file.
- For Part 2, please submit your code search.py, by renaming the file as <ID>_search.py
  Our TA will run your submitted code with his testing script for assessment.

Overall, please put <ID>_part1.pdf and <ID>_part2_search.py into the zipped folder <ID>_asmt4.zip and submit the overall .zip file through Blackboard.

Description Part 1:

In the first part, you will practice on essentials of convolutional neural networks, including network building, improving model training, and network evaluation, using PyTorch.

Questions:

1. In tutorial 7, we have introduced the dataset of MNIST and visualized some samples from it. We also set up a simple training framework to learn a MLP classifier. In this question, we will try to practice the training of networks on a more complex image dataset, i.e., CIFAR10. (50%)

   We provide a codebase (part1_codebase.py) for this question with which you can revise and complete the missing parts as required. To efficiently train the network, you are suggested to use GPU. If you do not have GPU card on your own PC, you can use the free GPUs from Google Colab. Here is a Google Colab CIFAR10 tutorial on how to use it.

   To submit the answers for this question, you don't need to submit the actual code file. Please screenshoot your lines of key code that you revised/added to the provided codebase, and show the figure of curves that is output from your model.

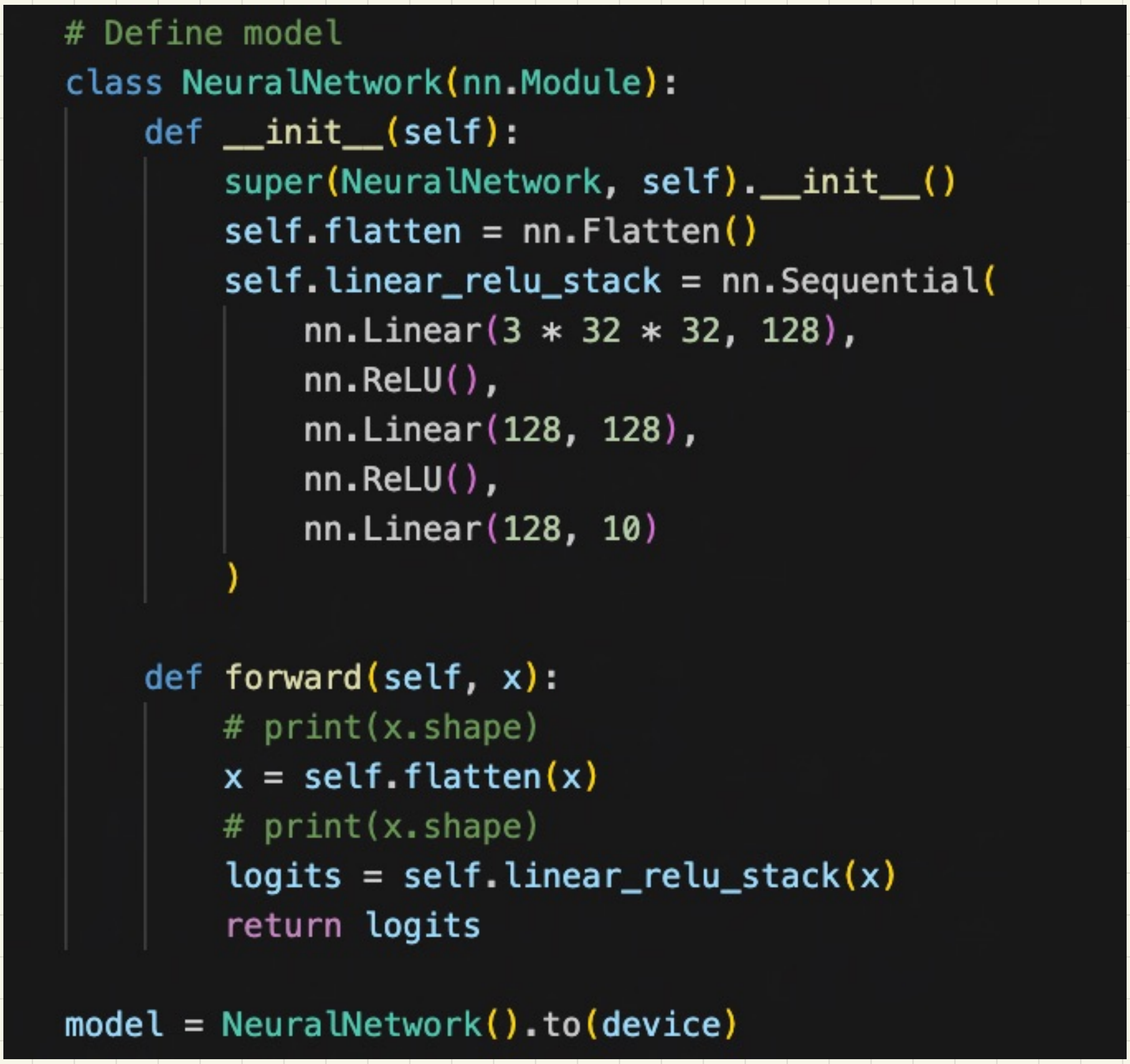
   a) In tutorial 7, it is seen that if we only train a MLP for just a few epochs, the network can not converge well. So a solution for improving the performance of the classifier is to train for more epochs. Please train the fully-connected network provided in codebase for 100 epochs and present the followings:

      i) plot the curves of training loss;
      ii) plot the curves of training accuracy and test accuracy;
      iii) report the best test accuracy that you can achieve.

      (Hint: With proper implementation, the best test accuracy is expected to be larger than 50%).

1a) goal : $\geq 50\%$ on test accuracy

architecture :



```
# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(3 * 32 * 32, 128),
            nn.ReLU(),
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        # print(x.shape)
        x = self.flatten(x)
        # print(x.shape)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
```

hyper-parameters :

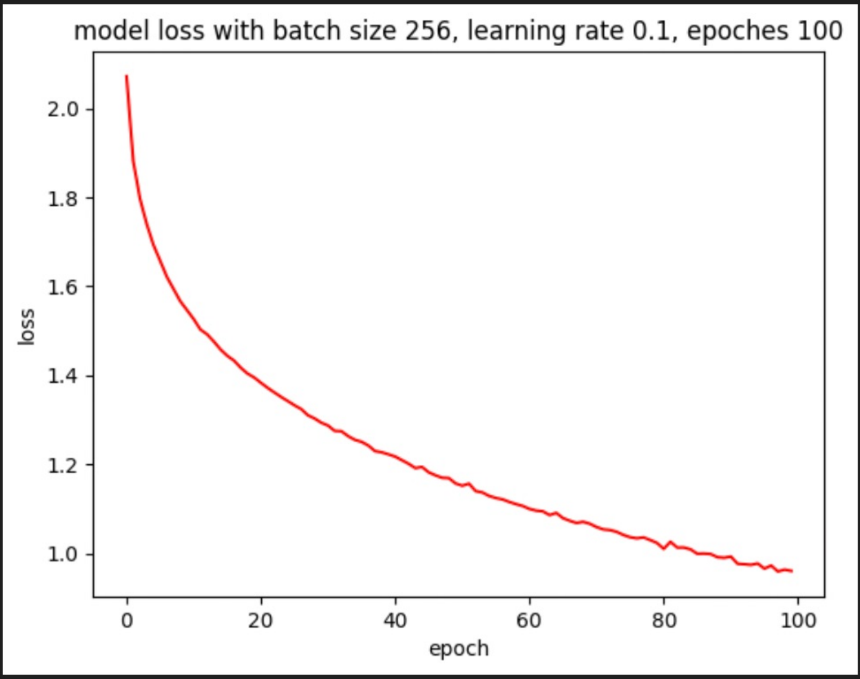
batch size bs	: 256
learning rate α	: 0.1
epochs ep	: 100

```
---------------- Epoch 100 ----------------
100%|██████████| 196/196 [00:02<00:00, 83.99it/s]
 Train accuracy: 65.9%, Avg loss: 0.960182
100%|██████████| 40/40 [00:00<00:00, 112.96it/s]
 Test accuracy: 51.8%, Avg loss: 1.484396
```

i)

```
plt.plot(training_losses, color="r")
plt.title('model loss with batch size '+ str(bs) + ', learning rate ' + str(alpha) + ', epoches ' + str(ep))
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
```

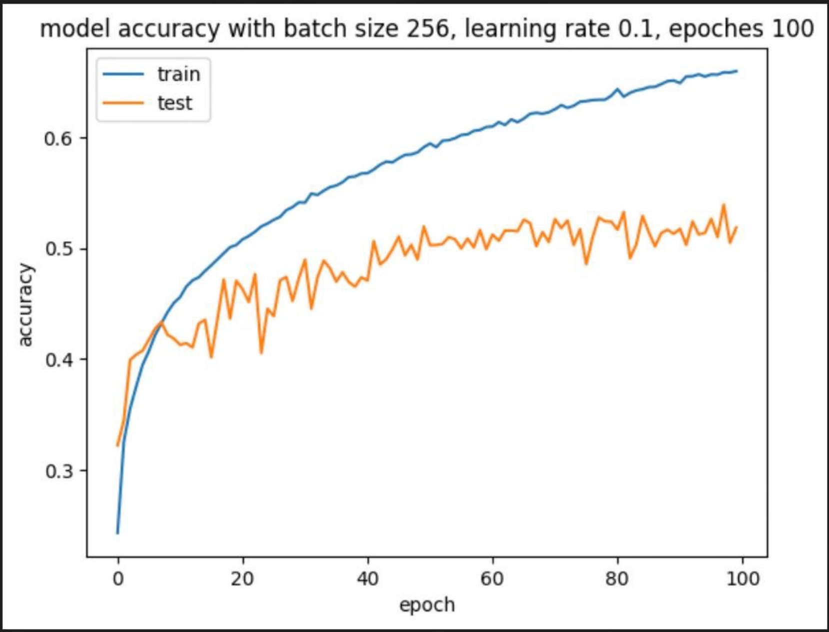
```
[34]  ✓ 0.0s
```



ii)

```
plt.plot(training_accuracy)
plt.plot(testing_accuracy)
plt.title('model accuracy with batch size '+ str(bs) + ', learning rate ' + str(alpha) + ', epoches ' + str(ep))
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
print(max(testing_accuracy))
```

```
[35]  ✓ 0.0s
```



```
0.539
```

iii) best test acc : 0.539

(10%)

b) Further, we can make the neural network deeper for improving the performance. In tutorial 9, we just build a neural network composed with 3 linear layers, which is rather simple for image classification task. Here, you can build a CNN consisting of 3 convolutional layers following with ReLU activation function, 1 pooling layer, 1 flatten layer and 1 linear layer (*Please do not add/delete any other layers for this sub-question*). Please design your network, train your network for 100 epochs and present the followings:

i) plot the curves of training loss;
ii) plot the curves of training accuracy and test accuracy;
iii) report the best test accuracy that you can achieve.

(Hint: The best test accuracy is expected to be larger than 60%).

(15%)

c) Based on CNN architecture, you can further use data augmention tricks to enrich training data, such as RandomCrop(32, padding=4). More operations for data augment in PyTorch are here. Based on the CNN architecture in (b), retrain your network (with data augmentations) for 100 epochs and present the followings:

i) plot the curves of training loss;
ii) plot the curves of training accuracy and test accuracy;
iii) report the best test accuracy that you can achieve.

(Hint: The best test accuracy is expected to be larger than 65%).

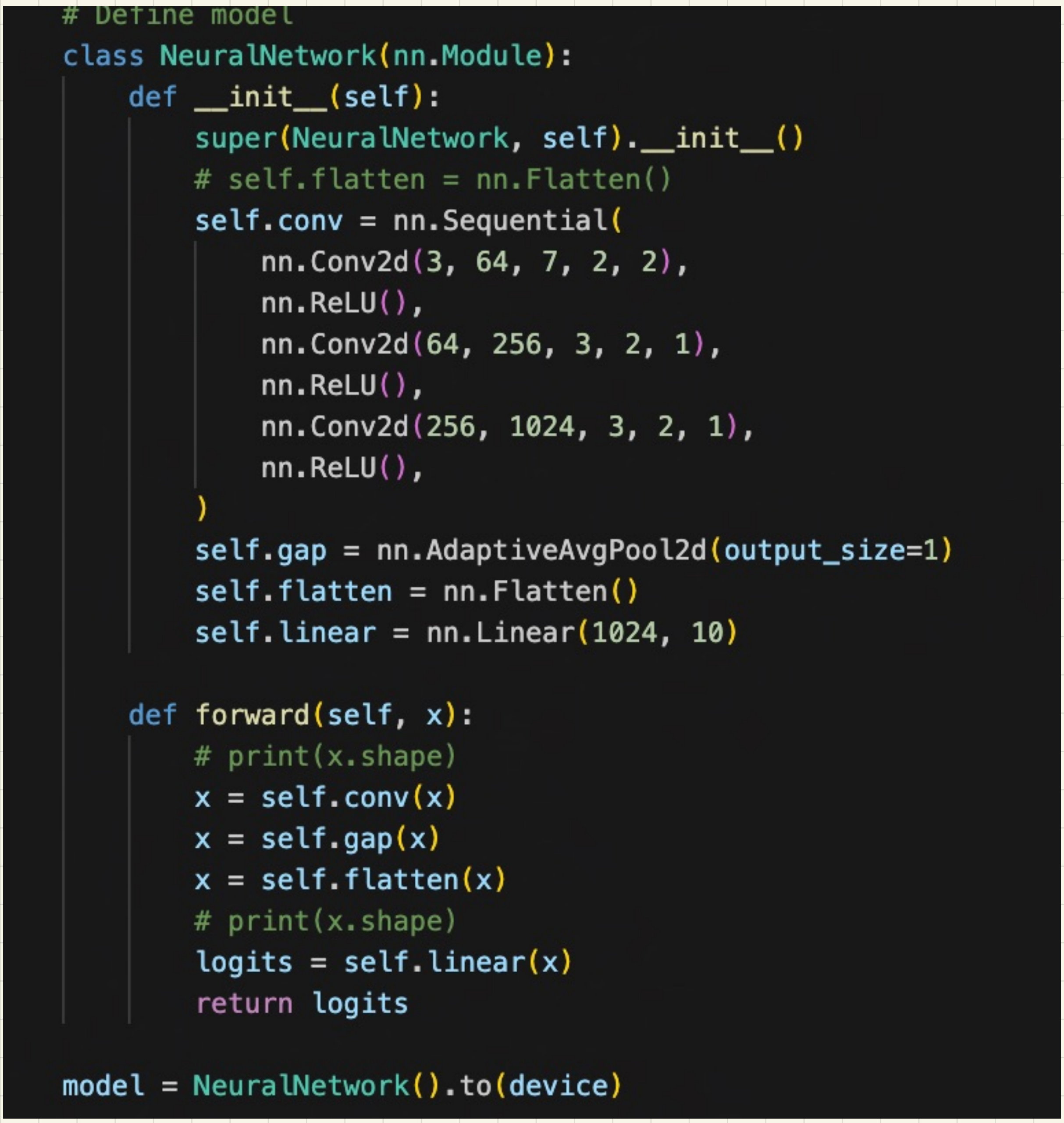
(10%)

d) Congratulations, you have successfully built up a CNN model for CIFAR10 classification. Now, it is time for brainstorms! You can use any reasonable techniques to improve the model performance, such as using using deeper neural networks, transferring pre-trained parameters, exploring other optimizers and so on. Try your best to improve the test accuracy and present the followings:
i) State what specific tricks you are using, and show screenshot of your code corresponding to your added training tricks.
ii) plot the curves of training loss and testing loss in a single figure;
iii) plot the curve of training accuracy and testing accuracy in a single figure;
iv) report the best test accuracy you can achieve.
(Hint: The expected best test accuracy should be larger than 80% to get full marks here. An extra bonus mark of 10% will be awarded if achieving an accuracy of larger than 90%).

(15%)

b) goal : ≥ 60 % on test accuracy

architecture :



```
# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        # self.flatten = nn.Flatten()
        self.conv = nn.Sequential(
            nn.Conv2d(3, 64, 7, 2, 2),
            nn.ReLU(),
            nn.Conv2d(64, 256, 3, 2, 1),
            nn.ReLU(),
            nn.Conv2d(256, 1024, 3, 2, 1),
            nn.ReLU(),
        )
        self.gap = nn.AdaptiveAvgPool2d(output_size=1)
        self.flatten = nn.Flatten()
        self.linear = nn.Linear(1024, 10)

    def forward(self, x):
        # print(x.shape)
        x = self.conv(x)
        x = self.gap(x)
        x = self.flatten(x)
        # print(x.shape)
        logits = self.linear(x)
        return logits

model = NeuralNetwork().to(device)
```

hyper-parameters :

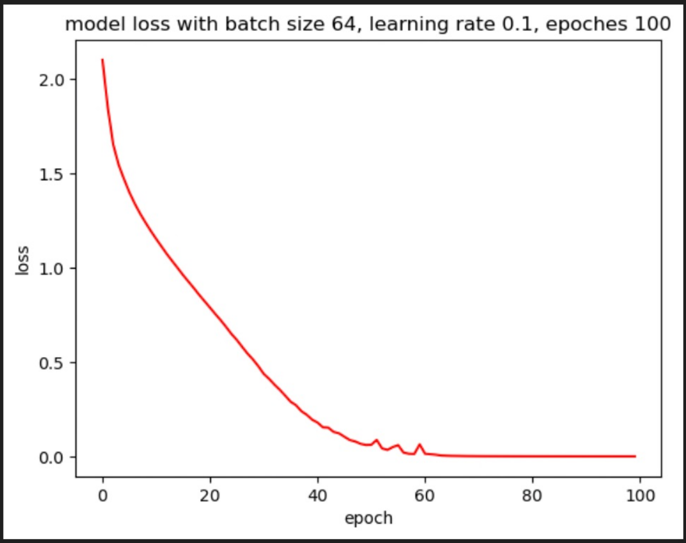
batch size	bs	: 64
learning rate	α	: 0.1
epochs	ep	: 100

i)

```
=============== Epoch 100 ===============
100%|██████████| 782/782 [00:16<00:00, 46.34it/s]
 Train accuracy: 100.0%, Avg loss: 0.000499
100%|██████████| 157/157 [00:01<00:00, 122.37it/s]
 Test accuracy: 70.7%, Avg loss: 2.226333
 Done!
```

```python
plt.plot(training_losses, color="r")
plt.title('model loss with batch size '+ str(bs) + ', learning rate ' + str(alpha) + ', epoches ' + str(ep))
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
```

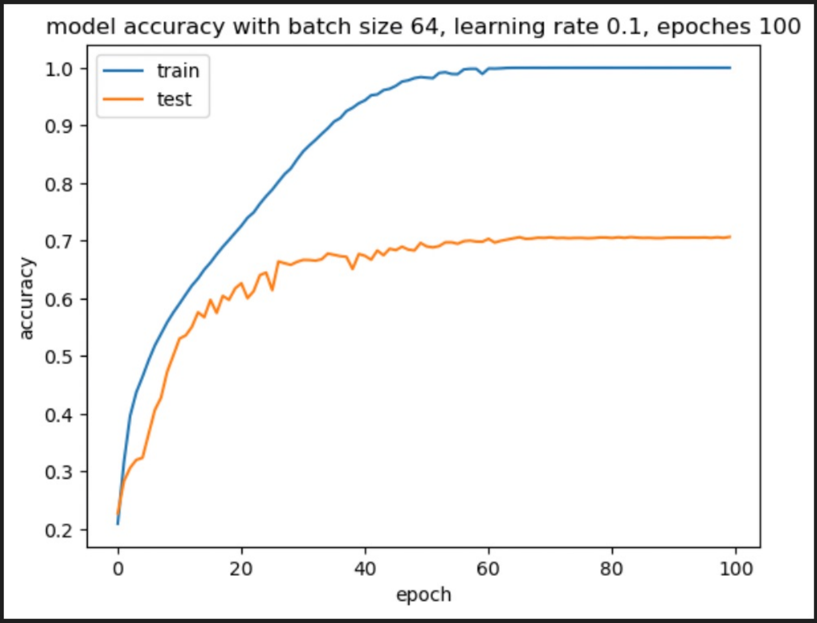
[124] ✓ 0.0s



ii)

```python
plt.plot(training_accuracy)
plt.plot(testing_accuracy)
plt.title('model accuracy with batch size '+ str(bs) + ', learning rate ' + str(alpha) + ', epoc
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
print(max(testing_accuracy))
```

125] ✓ 0.0s

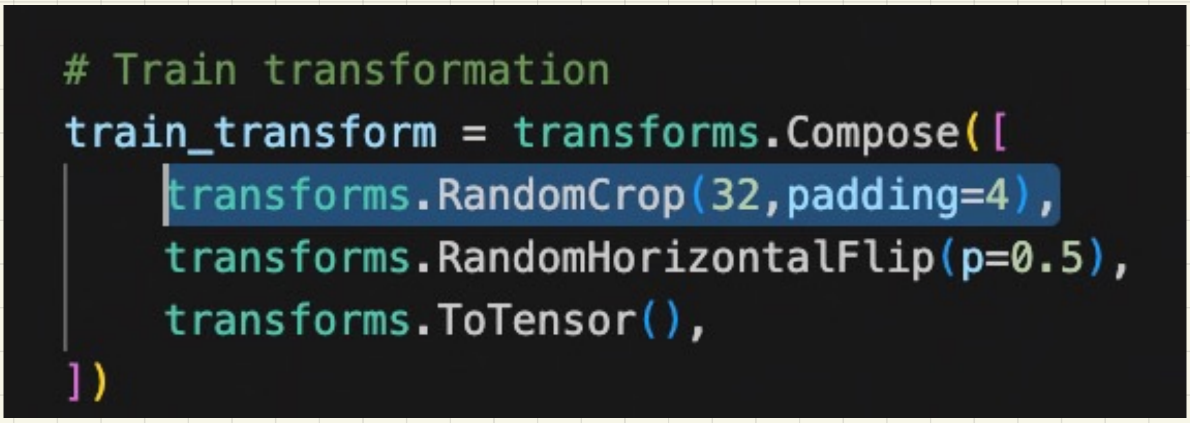


```
0.7068
```

iii) best test acc : 0,7068

c) goal : $\geq 65\%$ on test accuracy

architecture :
same as in (b) but with
RandomCrop added

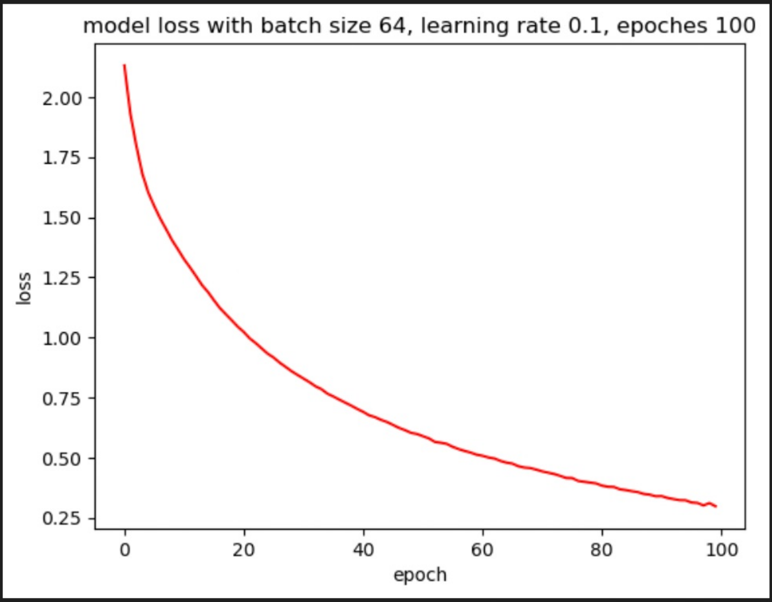


```
# Train transformation
train_transform = transforms.Compose([
    transforms.RandomCrop(32,padding=4),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
])
```

i)

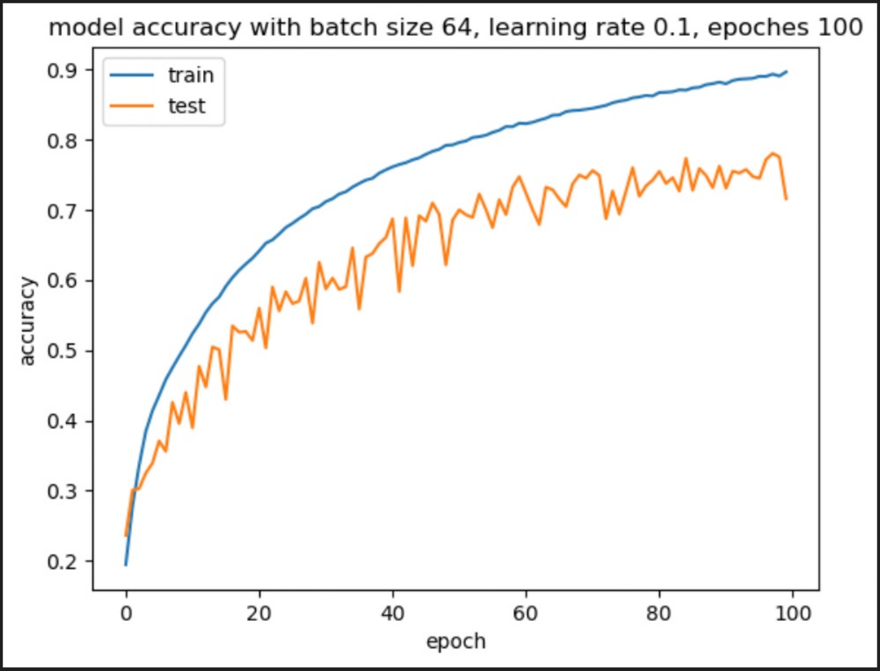
```
=============== Epoch 100 ===============
100%|██████████| 782/782 [00:17<00:00, 45.78it/s]
Train accuracy: 89.6%, Avg loss: 0.298347
100%|██████████| 157/157 [00:01<00:00, 117.26it/s]
Test accuracy: 71.6%, Avg loss: 1.121131
Done!
```

```
plt.plot(training_losses, color="r")
plt.title('model loss with batch size '+ str(bs) + ', learning rate ' + str(alpha) + ', epoches ' + str(ep))
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
[165]  ✓ 0.0s
```



ii)

```
plt.plot(training_accuracy)
plt.plot(testing_accuracy)
plt.title('model accuracy with batch size '+ str(bs) + ', learning rate ' + str(alpha) + ', epoches ' + str(ep
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
print(max(testing_accuracy))
✓ 0.0s
```



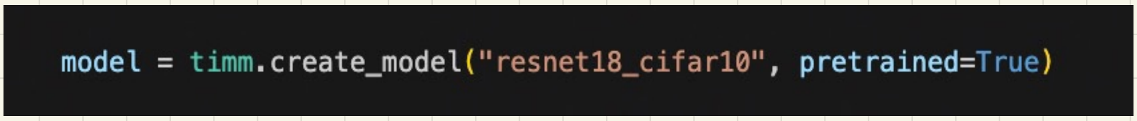
```
0.7801
```

iii) best test acc : 0.7801

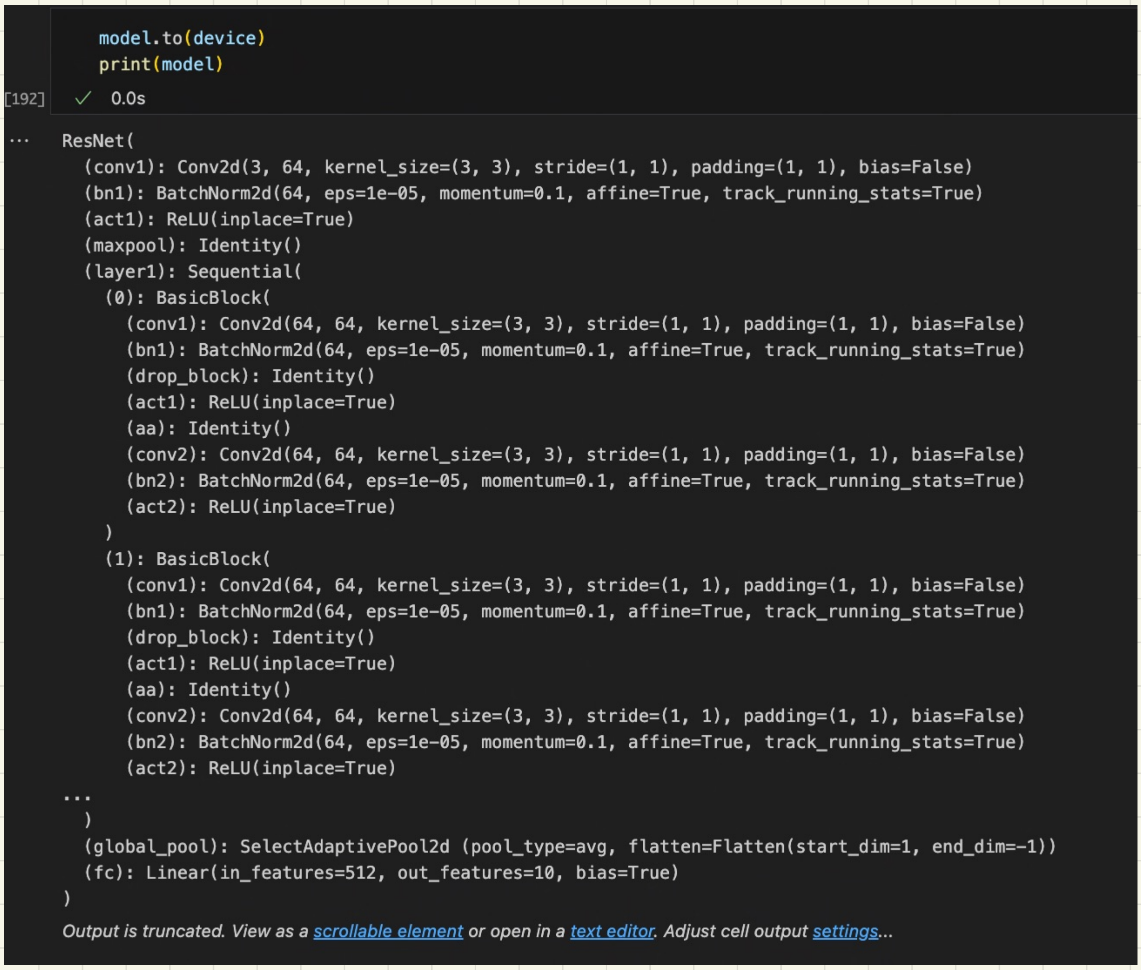
d) goal : ≥ 90 % on test accuracy

i) trick: using pretrained parameters (ResNet)

architecture :



```
model = timm.create_model("resnet18_cifar10", pretrained=True)
```



```
model.to(device)
print(model)
```

```
[192]  ✓ 0.0s
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (act1): ReLU(inplace=True)
  (maxpool): Identity()
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (drop_block): Identity()
      (act1): ReLU(inplace=True)
      (aa): Identity()
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (act2): ReLU(inplace=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (drop_block): Identity()
      (act1): ReLU(inplace=True)
      (aa): Identity()
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (act2): ReLU(inplace=True)
...
  )
  (global_pool): SelectAdaptivePool2d (pool_type=avg, flatten=Flatten(start_dim=1, end_dim=-1))
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

hyper-parameters :

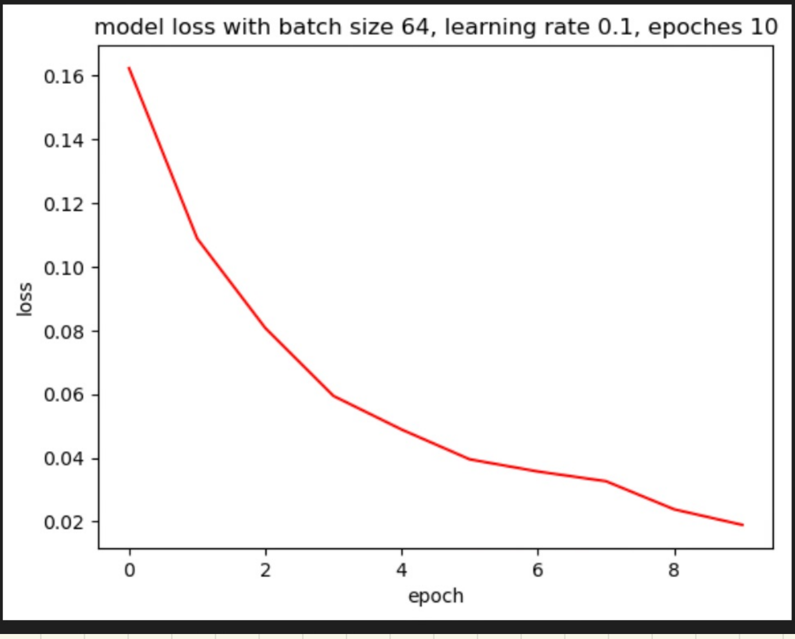
batch size	bs	: 64
learning rate	α	: 0.1
epochs	ep	: 10

ii)

```
 =============== Epoch 10 ===============
100%|██████████| 782/782 [01:02<00:00, 12.57it/s]
 Train accuracy: 99.4%, Avg loss: 0.018904
100%|██████████| 157/157 [00:03<00:00, 43.07it/s]
 Test accuracy: 92.4%, Avg loss: 0.310859
 Done!
```

```
plt.plot(training_losses, color="r")
plt.title('model loss with batch size '+ str(bs) + ', learning rate ' + str(alpha) + ', epoches ' +
plt.ylabel('loss')
plt.xlabel('epoch')
plt.show()
```

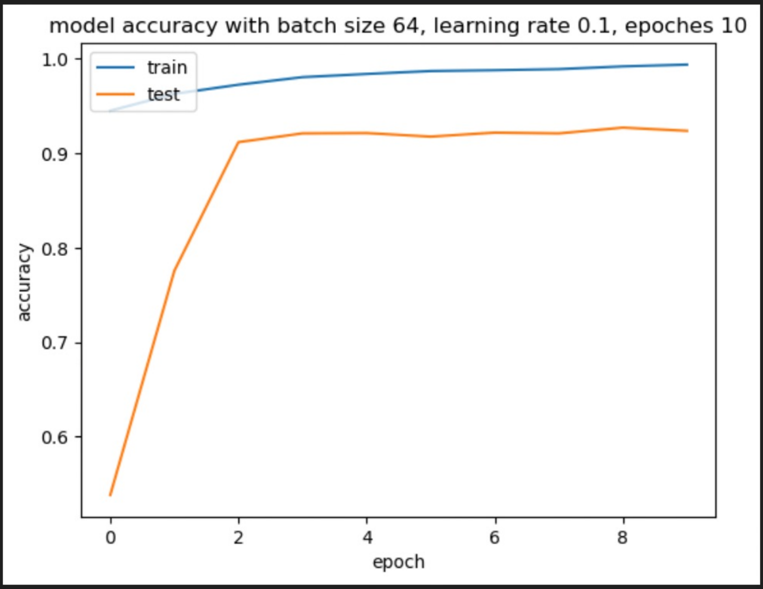
[197] ✓ 0.0s



iii)

```
plt.plot(training_accuracy)
plt.plot(testing_accuracy)
plt.title('model accuracy with batch size '+ str(bs) + ', learning rate ' + str(alpha) + ', epoches ' + str(ep))
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
print(max(testing_accuracy))
```

[198] ✓ 0.0s



```
0.9271
```

iv) best test acc : 0.9271