HW5-Coding

January 3, 2024

1 Homework 5: Convolutional neural network (30 points)

In this part, you need to implement and train a convolutional neural network on the CIFAR-10 dataset with PyTorch. ### What is PyTorch?

PyTorch is a system for executing dynamic computational graphs over Tensor objects that behave similarly as numpy ndarray. It comes with a powerful automatic differentiation engine that removes the need for manual back-propagation.

1.0.1 Why?

- Our code will now run on GPUs! Much faster training. When using a framework like PyTorch or TensorFlow you can harness the power of the GPU for your own custom neural network architectures without having to write CUDA code directly (which is beyond the scope of this class).
- We want you to be ready to use one of these frameworks for your project so you can experiment more efficiently than if you were writing every feature you want to use by hand.
- We want you to stand on the shoulders of giants! TensorFlow and PyTorch are both excellent frameworks that will make your lives a lot easier, and now that you understand their guts, you are free to use them :)
- We want you to be exposed to the sort of deep learning code you might run into in academia or industry. ## How can I learn PyTorch?

Justin Johnson has made an excellent tutorial for PyTorch.

You can also find the detailed API doc here. If you have other questions that are not addressed by the API docs, the PyTorch forum is a much better place to ask than StackOverflow.

Install PyTorch and Skorch.

[ ]:
```
!pip install -q torch skorch torchvision torchtext
```

[ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import skorch
```

```python
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

1.1 0. Tensor Operations (5 points)

Tensor operations are important in deep learning models. In this part, you are required to get famaliar to some common tensor operations in PyTorch.

1.1.1 1) Tensor squeezing, unsqueezing and viewing

Tensor squeezing, unsqueezing and viewing are important methods to change the dimension of a Tensor, and the corresponding functions are torch.squeeze, torch.unsqueeze and torch.Tensor.view. Please read the documents of the functions, and finish the following practice.

[ ]:
```python
# x is a tensor with size being (3, 2)
x = torch.Tensor([[1, 2],
                  [3, 4],
                  [5, 6]])
x.shape

# Add two new dimensions to x by using the function torch.unsqueeze, so that␣
↪the size of x becomes (3, 1, 2, 1).
x = torch.unsqueeze(x, 1)
x = torch.unsqueeze(x, 3)
print(x.shape)

# Remove the two dimensions justed added by using the function torch.squeeze,␣
↪and change the size of x back to (3, 2).
x = torch.squeeze(x, 3)
x = torch.squeeze(x, 1)
# x = torch.squeeze(x, (0, 2)) # in torch 2.0 this will work
print(x.shape)

# x is now a two-dimensional tensor, or in other words a matrix. Now use the␣
↪function torch.Tensor.view and change x to a one-dimensional vector with␣
↪size being (6).
x = x.view(6)
print(x.shape)
```

```
torch.Size([3, 1, 2, 1])
torch.Size([3, 2])
torch.Size([6])
```

1.1.2 2) Tensor concatenation and stack

Tensor concatenation and stack are operations to combine small tensors into big tensors. The corresponding functions are torch.cat and torch.stack. Please read the documents of the functions, and finish the following practice.

[ ]:
```
# x is a tensor with size being (3, 2)
x = torch.Tensor([[1, 2], [3, 4], [5, 6]])

# y is a tensor with size being (3, 2)
y = torch.Tensor([[-1, -2], [-3, -4], [-5, -6]])

# Our goal is to generate a tensor z with size as (2, 3, 2), and z[0,:,:] = x,␣
↪z[1,:,:] = y.

# Use torch.stack to generate such a z
z = torch.stack((x, y))
# print(z)
print(z[0,:,:])

# Use torch.cat and torch.unsqueeze to generate such a z
# print('torch.unsqueeze(x, 0) = ', torch.unsqueeze(x, 0))
z = torch.cat((torch.unsqueeze(x, 0), torch.unsqueeze(y, 0)))
# print(z)
print(z[1,:,:])
```

```
tensor([[1., 2.],
        [3., 4.],
        [5., 6.]])
tensor([[-1., -2.],
        [-3., -4.],
        [-5., -6.]])
```

1.1.3 3) Tensor expansion

Tensor expansion is to expand a tensor into a larger tensor along singleton dimensions. The corresponding functions are torch.Tensor.expand and torch.Tensor.expand_as. Please read the documents of the functions, and finish the following practice.

[ ]:
```
# x is a tensor with size being (3)
x = torch.Tensor([1, 2, 3])

# Our goal is to generate a tensor z with size (2, 3), so that z[0,:,:] = x,␣
↪z[1,:,:] = x.

# [TO DO]
# Change the size of x into (1, 3) by using torch.unsqueeze.
```

```
x = torch.unsqueeze(x, 0)
print(x.shape)

# [TO DO]
# Then expand the new tensor to the target tensor by using torch.Tensor.expand.
z = x.expand(2, -1)
# print(z)
print(z.shape)
```

```
torch.Size([1, 3])
torch.Size([2, 3])
```

1.1.4 4) Tensor reduction in a given dimension

In deep learning, we often need to compute the mean/sum/max/min value in a given dimension of a tensor. Please read the document of torch.mean, torch.sum, torch.max, torch.min, torch.topk, and finish the following practice.

[ ]:
```
# x is a random tensor with size being (10, 50)
x = torch.randn(10, 50)

# Compute the mean value for each row of x.
# You need to generate a tensor x_mean of size (10), and x_mean[k, :] is the␣
↪mean value of the k-th row of x.

# dim = 1: eliminate the second(1)'s dimension

x_mean = torch.mean(x, dim=1)
# print(x_mean)
print(x_mean[3, ])

# Compute the sum value for each row of x.
# You need to generate a tensor x_sum of size (10).
x_sum = torch.sum(x, dim=1)
print(x_sum.shape)

# Compute the max value for each row of x.
# You need to generate a tensor x_max of size (10).
(x_max, indices) = torch.max(x, dim=1)
# print(x_max, indices)
print(x_max.shape)

# Compute the min value for each row of x.
# You need to generate a tensor x_min of size (10).
(x_min, indices) = torch.min(x, dim=1)
print(x_min.shape)
```

```
# Compute the top-5 values for each row of x.
# (wrong) You need to generate a tensor x_mean of size (10, 5), and x_top[k, :]
 ↪is the top-5 values of each row in x.
# (right) You need to generate a tensor, top-5 values of each row
(x_xtop, indices) = torch.topk(x, k=5, dim=1)
print((x_xtop.shape))
```

```
tensor(0.2702)
torch.Size([10])
torch.Size([10])
torch.Size([10])
torch.Size([10, 5])
```

1.2 Convolutional Neural Networks

Implement a convolutional neural network for image classification on CIFAR-10 dataset.

CIFAR-10 is an image dataset of 10 categories. Each image has a size of 32x32 pixels. The following code will download the dataset, and split it into `train` and `test`. For this question, we use the default validation split generated by Skorch.

[ ]:
```
train = torchvision.datasets.CIFAR10("./data", train=True, download=True)
test = torchvision.datasets.CIFAR10("./data", train=False, download=True)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

The following code visualizes some samples in the dataset. You may use it to debug your model if necessary.

[ ]:
```
def plot(data, labels=None, num_sample=5):
  n = min(len(data), num_sample)
  for i in range(n):
    plt.subplot(1, n, i+1)
    plt.imshow(data[i], cmap="gray")
    plt.xticks([])
    plt.yticks([])
    if labels is not None:
      plt.title(labels[i])

train.labels = [train.classes[target] for target in train.targets]
plot(train.data, train.labels)
```

1.2.1 1) Basic CNN implementation

Consider a basic CNN model

- It has 3 convolutional layers, followed by a linear layer.
- Each convolutional layer has a kernel size of 3, a padding of 1.
- ReLU activation is applied on every hidden layer.

Please implement this model in the following section. The hyperparameters is then be tuned and you need to fill the results in the table.

a) Implement convolutional layers (10 Points) Implement the initialization function and the forward function of the CNN.

[ ]:
```
class CNN(nn.Module):
  def __init__(self, channels):
    super(CNN, self).__init__()
    # implement parameter definitions here
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    # print('channels = ', channels)
    self.channels = channels

    self.backbone = nn.Sequential(
        nn.Conv2d(3, channels, kernel_size=3, padding=1),
        nn.ReLU(),

        nn.Conv2d(channels, channels, kernel_size=3, padding=1),
        nn.ReLU(),

        nn.Conv2d(channels, channels, kernel_size=3, padding=1),
        nn.ReLU(),
    )

    # regard the image has the same size 32 * 32
    self.f1 = nn.Linear(channels * 32 * 32, 10)
```

```
    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

  def forward(self, images):
    # implement the forward function here
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    images = self.backbone(images)

    # flatten the image
    images = images.view(images.shape[0], -1)
    images = self.f1(images)

    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    return images
```

b) Tune hyperparameters Train the CNN model on CIFAR-10 dataset. We can tune the number of channels, optimizer, learning rate and the number of epochs for best validation accuracy.

[ ]:
```
# implement hyperparameters, you can select and modify the hyperparameters by␣
↪yourself here.

optimizer = [torch.optim.SGD, torch.optim.Adam]
learning_rate = [1e-3, 1e-2]
channel = [128, 256, 512]

train_data_normalized = torch.Tensor(train.data/255)
train_data_normalized = train_data_normalized.permute(0,3,1,2)

for l in learning_rate:
  for o in optimizer:
    for c in channel:
      print(f'The channel was {c}, the learning rate was {l} and the optimizer␣
↪was {str(o)}')

      cnn = CNN(channels = c)

      model = skorch.NeuralNetClassifier(cnn, criterion=torch.nn.
↪CrossEntropyLoss,
                                         device="cuda",
                                         optimizer=o,
                                        #  optimizer__momentum=0.90,
                                         lr=l,
                                         max_epochs=50,
                                         batch_size=32,
                                         callbacks=[skorch.callbacks.
↪EarlyStopping(lower_is_better=True)])
```

```python
# implement input normalization & type cast here
model.fit(train_data_normalized, np.asarray(train.targets))
```

```
The channel was 128, the learning rate was 0.001 and the optimizer was <class
'torch.optim.sgd.SGD'>
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        2.1158       0.3160        1.9293
8.0129
      2        1.8676       0.3645        1.8060
7.0036
      3        1.7704       0.3984        1.7152
7.0300
      4        1.6917       0.4253        1.6462
7.0520
      5        1.6336       0.4405        1.5969
7.0703
      6        1.5857       0.4537        1.5550
7.0006
      7        1.5434       0.4653        1.5178
7.0303
      8        1.5062       0.4777        1.4855
6.9867
      9        1.4740       0.4851        1.4584
7.0018
     10        1.4458       0.4927        1.4353
7.0111
     11        1.4201       0.4999        1.4149
7.0238
     12        1.3961       0.5051        1.3962
7.0131
     13        1.3729       0.5109        1.3787
7.0701
     14        1.3496       0.5166        1.3605
7.0898
     15        1.3254       0.5228        1.3404
7.0641
     16        1.3003       0.5288        1.3189
7.0448
     17        1.2751       0.5329        1.2981
7.0350
     18        1.2510       0.5411        1.2792
7.0506
     19        1.2286       0.5472        1.2625
7.0371
     20        1.2082       0.5523        1.2481
7.0452
```

21	1.1893	0.5558	1.2354	7.0610
22	1.1714	0.5615	1.2241	7.0859
23	1.1543	0.5647	1.2139	7.0981
24	1.1376	0.5670	1.2042	7.1177
25	1.1213	0.5722	1.1951	7.1225
26	1.1052	0.5741	1.1863	7.1420
27	1.0892	0.5781	1.1781	7.1389
28	1.0733	0.5832	1.1701	7.1626
29	1.0574	0.5866	1.1624	7.1693
30	1.0415	0.5895	1.1553	7.1629
31	1.0258	0.5915	1.1485	7.1485
32	1.0102	0.5954	1.1422	7.1612
33	0.9948	0.5981	1.1362	7.1532
34	0.9796	0.6016	1.1305	7.1491
35	0.9645	0.6052	1.1253	7.1466
36	0.9495	0.6078	1.1203	7.1506
37	0.9347	0.6089	1.1159	7.1595
38	0.9200	0.6109	1.1117	7.1716
39	0.9054	0.6129	1.1079	7.1662
40	0.8907	0.6141	1.1043	7.1629
41	0.8761	0.6160	1.1011	7.1744
42	0.8615	0.6161	1.0983	7.1671
43	0.8469	0.6193	1.0959	7.1680
44	0.8323	0.6205	1.0939	7.1720

```
     45        0.8176       0.6211        1.0921
7.1611
     46        0.8030       0.6235        1.0907
7.1709
     47        0.7884       0.6242        1.0898
7.1742
     48        0.7738       0.6250        1.0895
7.1943
     49        0.7593       0.6255        1.0895  7.1965
     50        0.7448       0.6266        1.0902  7.1881
The channel was 256, the learning rate was 0.001 and the optimizer was <class
'torch.optim.sgd.SGD'>
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
      1        2.0442       0.3434        1.8783
16.4868
      2        1.8054       0.3976        1.7304
16.5860
      3        1.7004       0.4273        1.6449
16.6578
      4        1.6279       0.4455        1.5829
16.6932
      5        1.5640       0.4653        1.5236
16.7557
      6        1.5044       0.4785        1.4709
16.7104
      7        1.4544       0.4940        1.4321
16.7861
      8        1.4143       0.5052        1.4030
16.7697
      9        1.3789       0.5118        1.3771
16.7234
     10        1.3442       0.5204        1.3506
16.7258
     11        1.3091       0.5285        1.3241
16.5999
     12        1.2755       0.5356        1.3007
16.5341
     13        1.2457       0.5426        1.2806
16.4736
     14        1.2194       0.5506        1.2631
16.4122
     15        1.1957       0.5565        1.2472
16.3709
     16        1.1736       0.5612        1.2330
16.4181
     17        1.1525       0.5658        1.2197
16.4600
```

```
     18        1.1320       0.5706         1.2072
16.4435
     19        1.1117       0.5745         1.1960
16.4278
     20        1.0917       0.5785         1.1856
16.4144
     21        1.0720       0.5834         1.1760
16.4010
     22        1.0525       0.5882         1.1671
16.3734
     23        1.0335       0.5924         1.1589
16.3465
     24        1.0148       0.5960         1.1515
16.3348
     25        0.9964       0.5974         1.1447
16.3804
     26        0.9784       0.5993         1.1385
16.3272
     27        0.9607       0.6022         1.1328
16.3608
     28        0.9431       0.6049         1.1276
16.3947
     29        0.9255       0.6067         1.1225
16.4668
     30        0.9080       0.6086         1.1179
16.5375
     31        0.8904       0.6086         1.1137  16.5316
     32        0.8726       0.6107         1.1097
16.6019
     33        0.8547       0.6129         1.1059
16.6545
     34        0.8366       0.6152         1.1026
16.6232
     35        0.8184       0.6159         1.0996
16.6570
     36        0.8000       0.6172         1.0973
16.5988
     37        0.7814       0.6189         1.0957
16.5260
     38        0.7628       0.6214         1.0950
16.5240
     39        0.7441       0.6227         1.0952  16.4719
     40        0.7254       0.6224         1.0965  16.4138
     41        0.7067       0.6222         1.0988  16.4675
     42        0.6880       0.6223         1.1025  16.4783
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 512, the learning rate was 0.001 and the optimizer was <class
'torch.optim.sgd.SGD'>
```

epoch	train_loss	valid_acc	valid_loss	dur
1	1.9867	0.3623	1.8191	46.1175
2	1.7513	0.4171	1.6693	46.1142
3	1.6400	0.4468	1.5838	45.7091
4	1.5603	0.4680	1.5144	45.3723
5	1.4935	0.4853	1.4582	45.4629
6	1.4396	0.5000	1.4170	45.4543
7	1.3952	0.5077	1.3844	45.3653
8	1.3541	0.5188	1.3529	45.4058
9	1.3127	0.5287	1.3208	45.8955
10	1.2729	0.5375	1.2915	46.3144
11	1.2376	0.5454	1.2668	46.4242
12	1.2072	0.5545	1.2464	46.3143
13	1.1800	0.5601	1.2289	45.7325
14	1.1545	0.5658	1.2133	45.9819
15	1.1300	0.5724	1.1989	45.9304
16	1.1059	0.5773	1.1857	45.4933
17	1.0821	0.5817	1.1732	45.3715
18	1.0587	0.5876	1.1618	45.4426
19	1.0358	0.5915	1.1513	45.5602
20	1.0134	0.5973	1.1418	46.1363
21	0.9915	0.6005	1.1330	46.3644
22	0.9700	0.6039	1.1250	45.7330
23	0.9487	0.6072	1.1174	45.8170

```
     24        0.9277        0.6107        1.1103
45.4220
     25        0.9067        0.6116        1.1037
45.4124
     26        0.8858        0.6138        1.0975
45.3856
     27        0.8651        0.6175        1.0919
45.7772
     28        0.8444        0.6187        1.0870
46.3898
     29        0.8239        0.6195        1.0828
46.3944
     30        0.8035        0.6216        1.0797
46.1950
     31        0.7833        0.6235        1.0776
45.4395
     32        0.7632        0.6250        1.0765
45.5423
     33        0.7433        0.6255        1.0766  45.4350
     34        0.7235        0.6264        1.0779  45.3818
     35        0.7037        0.6283        1.0803  45.3848
     36        0.6839        0.6295        1.0839  45.7210
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 128, the learning rate was 0.001 and the optimizer was <class
'torch.optim.adam.Adam'>
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        1.4839        0.5829        1.1889
7.5850
      2        1.0473        0.6502        1.0072
7.5615
      3        0.8172        0.6484        1.0422  7.5612
      4        0.6176        0.6250        1.2363  7.5515
      5        0.4350        0.5978        1.5414  7.5729
      6        0.3217        0.6011        1.8818  7.5781
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 256, the learning rate was 0.001 and the optimizer was <class
'torch.optim.adam.Adam'>
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
      1        1.7182        0.5077        1.3801
16.8513
      2        1.2644        0.5565        1.2567
16.8689
      3        1.0805        0.5825        1.1950
16.9014
      4        0.8733        0.5819        1.2774  16.8458
      5        0.6655        0.5767        1.5078  16.8153
```

```
      6        0.5032       0.5598        1.8750  16.8164
      7        0.3858       0.5273        2.4469  16.8811
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 512, the learning rate was 0.001 and the optimizer was <class
'torch.optim.adam.Adam'>
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
      1        2.3566       0.1000        2.3026
46.3657
      2        2.3028       0.1000        2.3026  46.2071
      3        2.3028       0.1000        2.3026  45.8301
      4        2.3028       0.1000        2.3026  46.2545
      5        2.3028       0.1000        2.3026  46.7461
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 128, the learning rate was 0.01 and the optimizer was <class
'torch.optim.sgd.SGD'>
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        1.7802       0.4473        1.5509
7.2227
      2        1.4604       0.4953        1.3926
7.1891
      3        1.3259       0.5312        1.2938
7.1699
      4        1.2092       0.5520        1.2376
7.1683
      5        1.1174       0.5713        1.1887
7.1695
      6        1.0313       0.5887        1.1468
7.1558
      7        0.9441       0.6095        1.1049
7.1596
      8        0.8574       0.6212        1.0799
7.1328
      9        0.7744       0.6294        1.0816  7.1242
     10        0.6936       0.6324        1.1068  7.1195
     11        0.6121       0.6323        1.1574  7.1237
     12        0.5278       0.6290        1.2431  7.1241
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 256, the learning rate was 0.01 and the optimizer was <class
'torch.optim.sgd.SGD'>
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
      1        1.7304       0.4675        1.4906
16.3829
      2        1.3863       0.5328        1.3044
16.5476
      3        1.2254       0.5659        1.2175
```

```
16.5300
      4        1.1000       0.5942        1.1371
16.5410
      5        0.9843       0.6187        1.0781
16.5368
      6        0.8811       0.6341        1.0504
16.4840
      7        0.7862       0.6431        1.0436
16.4648
      8        0.6937       0.6450        1.0633  16.3988
      9        0.5998       0.6420        1.1201  16.3038
     10        0.5008       0.6338        1.2165  16.3112
     11        0.3968       0.6297        1.3476  16.3209
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 512, the learning rate was 0.01 and the optimizer was <class
'torch.optim.sgd.SGD'>
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
      1        1.7122       0.4814        1.4455
45.7170
      2        1.3445       0.5414        1.2749
45.8045
      3        1.1590       0.5824        1.1645
46.0595
      4        1.0109       0.6085        1.0983
46.3534
      5        0.8835       0.6283        1.0726
46.3809
      6        0.7679       0.6391        1.0765  46.4801
      7        0.6554       0.6344        1.1211  46.4627
      8        0.5380       0.6302        1.2184  45.9412
      9        0.4127       0.6206        1.3871  45.4630
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 128, the learning rate was 0.01 and the optimizer was <class
'torch.optim.adam.Adam'>
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        2.9942       0.1000        2.3039
7.4479
      2        2.3042       0.1000        2.3039  7.4428
      3        2.3042       0.1000        2.3039  7.4505
      4        2.3042       0.1000        2.3039  7.4814
      5        2.3042       0.1000        2.3039  7.4697
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 256, the learning rate was 0.01 and the optimizer was <class
'torch.optim.adam.Adam'>
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
```

```
      1        8.8825       0.1000        2.3039
16.5144
      2        2.3042       0.1000        2.3039  16.7190
      3        2.3042       0.1000        2.3039  16.8494
      4        2.3042       0.1000        2.3039  16.9722
      5        2.3042       0.1000        2.3039  17.0036
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 512, the learning rate was 0.01 and the optimizer was <class
'torch.optim.adam.Adam'>
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
      1       55.4451       0.1000        2.3038
46.9036
      2        2.3042       0.1000        2.3039  46.8506
      3        2.3042       0.1000        2.3039  46.5957
      4        2.3042       0.1000        2.3039  45.6649
      5        2.3042       0.1000        2.3039  45.5824
Stopping since valid_loss has not improved in the last 5 epochs.
```

Write down **validation accuracy** of your model under different hyperparameter settings. Note the validation set is automatically split by Skorch during `model.fit()`.

learning rate = 10^{-3}: | #channel for each layer optimizer | SGD | Adam | |:———————— ———:|:——:|:——:| | 128 |0.6266|0.6502| | 256 |0.6227|0.5825| | 512 |0.6295|0.1000|

learning rate = 10^{-2}: | #channel for each layer optimizer | SGD | Adam | |:———————— ———:|:——:|:——:| | 128 |0.6324|0.1000| | 256 |0.6450|0.1000| | 512 |0.6391|0.1000|

1.2.2 2) Full CNN implementation (10 points)

Based on the CNN in the previous question, implement a full CNN model with max pooling layer.

- Add a max pooling layer after each convolutional layer.
- Each max pooling layer has a kernel size of 2 and a stride of 2.

Please implement this model in the following section. The hyperparameters is then be tuned and fill the results in the table. You are also required to complete the questions.

a) Implement max pooling layers Similar to the CNN implementation in previous question, implement max pooling layers.

[ ]:
```python
class CNN_MaxPool(nn.Module):
  def __init__(self, channels):
    super(CNN_MaxPool, self).__init__()
    # implement parameter definitions here
    # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    self.channels = channels
```

```python
    self.backbone = nn.Sequential(
        nn.Conv2d(3, channels, kernel_size=3, padding=1),
        nn.BatchNorm2d(channels),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Dropout2d(p=0.2),

        nn.Conv2d(channels, channels, kernel_size=3, padding=1),
        nn.BatchNorm2d(channels),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Dropout2d(p=0.2),

        nn.Conv2d(channels, channels, kernel_size=3, padding=1),
        nn.BatchNorm2d(channels),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
        nn.Dropout2d(p=0.2)
    )

    # regard the image has the same size 32 * 32
    # after 3 max pooling, the size is 32 / 2 / 2 / 2 = 4
    self.f1 = nn.Linear(channels * 4 * 4, 10)

    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

  def forward(self, images):
    # implement the forward function here

    images = self.backbone(images)

    # flatten the image
    images = images.view(images.shape[0], -1)

    images = self.f1(images)

    # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    return images
```

b) Tune hyperparameters Based on the better optimizer found in the previous problem, we can tune the number of channels and learning rate for best validation accuracy.

[ ]:
```python
# implement hyperparameters, you can select and modify the hyperparameters by␣
↪yourself here.
learning_rate = [1e-4, 1e-3, 1e-2]
channel = [128, 256, 512]
```

```
# Select the better optimizer by the result shown in the previous problem, you␣
↪can select and modify it by yourself here.

better_optimizer = torch.optim.Adam

train_data_normalized = torch.Tensor(train.data/255)
train_data_normalized = train_data_normalized.permute(0,3,1,2)

for l in learning_rate:
    for c in channel:
      print(f'The channel was {c}, the learning rate was {l}')

      cnn = CNN_MaxPool(channels = c)

      model = skorch.NeuralNetClassifier(cnn, criterion=torch.nn.
↪CrossEntropyLoss,
                                   device="cuda",
                                   optimizer=better_optimizer,
                                   lr=l,
                                   max_epochs=50,
                                   batch_size=32 * 8,
                                   callbacks=[skorch.callbacks.
↪EarlyStopping(lower_is_better=True)])
      # implement input normalization & type cast here
      model.fit(train_data_normalized, np.asarray(train.targets))
```

```
The channel was 128, the learning rate was 0.0001
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        1.8045       0.4908        1.4717
3.2317
      2        1.4935       0.5442        1.3015
3.1580
      3        1.3585       0.5886        1.1966
3.1443
      4        1.2596       0.6146        1.1193
3.1602
      5        1.1847       0.6342        1.0625
3.1622
      6        1.1234       0.6518        1.0135
3.1547
      7        1.0764       0.6625        0.9778
3.1650
      8        1.0358       0.6719        0.9484
3.1638
      9        0.9967       0.6813        0.9210
3.1614
```

10	0.9698	0.6923	0.8992	3.1662
11	0.9418	0.6989	0.8795	3.1710
12	0.9159	0.7008	0.8661	3.1699
13	0.8987	0.7088	0.8503	3.1751
14	0.8736	0.7155	0.8327	3.1795
15	0.8550	0.7192	0.8209	3.1766
16	0.8351	0.7245	0.8086	3.1782
17	0.8160	0.7282	0.7994	3.1833
18	0.8006	0.7312	0.7842	3.1876
19	0.7876	0.7328	0.7803	3.1829
20	0.7678	0.7339	0.7716	3.1953
21	0.7583	0.7398	0.7576	3.1882
22	0.7469	0.7415	0.7519	3.1923
23	0.7304	0.7401	0.7504	3.1926
24	0.7186	0.7418	0.7470	3.1946
25	0.7094	0.7458	0.7346	3.1994
26	0.6931	0.7453	0.7383	3.1950
27	0.6847	0.7467	0.7313	3.1931
28	0.6752	0.7515	0.7170	3.1928
29	0.6627	0.7544	0.7119	3.1875
30	0.6524	0.7511	0.7180	3.1824
31	0.6418	0.7569	0.7052	3.1839
32	0.6332	0.7589	0.6995	3.1801
33	0.6226	0.7579	0.6945	3.1792
34	0.6115	0.7575	0.6946	3.1858
35	0.6090	0.7609	0.6869	3.1795
36	0.5965	0.7622	0.6868	

```
3.1880
     37        0.5886       0.7613        0.6831  3.1852
     38        0.5806       0.7669        0.6744
3.1756
     39        0.5691       0.7656        0.6816  3.1742
     40        0.5664       0.7697        0.6742
3.1783
     41        0.5545       0.7657        0.6778  3.1825
     42        0.5465       0.7708        0.6708
3.1714
     43        0.5400       0.7692        0.6658  3.1728
     44        0.5343       0.7663        0.6744  3.1602
     45        0.5261       0.7737        0.6608
3.1719
     46        0.5252       0.7717        0.6618  3.1673
     47        0.5109       0.7700        0.6638  3.1695
     48        0.5050       0.7716        0.6618  3.1599
     49        0.4969       0.7755        0.6571
3.1588
     50        0.4920       0.7733        0.6619  3.1607
The channel was 256, the learning rate was 0.0001
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        1.6336       0.5527        1.2881
6.1969
      2        1.2999       0.6045        1.1220
6.1984
      3        1.1574       0.6433        1.0264
6.2124
      4        1.0544       0.6692        0.9555
6.2015
      5        0.9872       0.6815        0.9150
6.2078
      6        0.9240       0.6962        0.8753
6.1982
      7        0.8772       0.7051        0.8505
6.2226
      8        0.8395       0.7134        0.8215
6.1924
      9        0.8040       0.7197        0.7986
6.1769
     10        0.7724       0.7305        0.7737
6.1587
     11        0.7434       0.7366        0.7681
6.1682
     12        0.7170       0.7445        0.7461
6.1463
     13        0.6899       0.7460        0.7380
```

```
6.1383
     14      0.6647      0.7519       0.7267
6.1318
     15      0.6392      0.7543       0.7104
6.1295
     16      0.6187      0.7574       0.7044
6.1260
     17      0.5990      0.7591       0.6985
6.1007
     18      0.5774      0.7591       0.6911  6.0768
     19      0.5607      0.7651       0.6901
6.0741
     20      0.5382      0.7668       0.6811
6.1156
     21      0.5218      0.7682       0.6737
6.1021
     22      0.5011      0.7710       0.6630
6.1216
     23      0.4857      0.7717       0.6631  6.0929
     24      0.4669      0.7751       0.6545
6.1075
     25      0.4572      0.7752       0.6540
6.1212
     26      0.4396      0.7743       0.6600  6.1145
     27      0.4239      0.7756       0.6529
6.1094
     28      0.4048      0.7788       0.6474
6.1126
     29      0.3948      0.7783       0.6507  6.1339
     30      0.3807      0.7789       0.6526  6.1389
     31      0.3655      0.7767       0.6542  6.1480
     32      0.3515      0.7814       0.6444
6.1751
     33      0.3393      0.7840       0.6455  6.1548
     34      0.3247      0.7816       0.6497  6.1603
     35      0.3153      0.7823       0.6487  6.1681
     36      0.3007      0.7860       0.6388
6.1728
     37      0.2920      0.7851       0.6449  6.1701
     38      0.2808      0.7826       0.6554  6.1807
     39      0.2743      0.7845       0.6524  6.1956
     40      0.2562      0.7842       0.6490  6.2041
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 512, the learning rate was 0.0001
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
      1        1.4867       0.5875        1.1599
13.7566
```

```
      2        1.1323       0.6544        0.9950
13.8768
      3        0.9813       0.6778        0.9199
13.9526
      4        0.8851       0.7036        0.8458
14.0137
      5        0.8155       0.7077        0.8244
14.0539
      6        0.7579       0.7197        0.7897
14.0974
      7        0.7093       0.7321        0.7623
14.0722
      8        0.6648       0.7351        0.7490
14.0787
      9        0.6232       0.7519        0.7108
14.0801
     10        0.5813       0.7579        0.6961
13.9528
     11        0.5446       0.7622        0.6848
13.9407
     12        0.5114       0.7663        0.6737
14.0248
     13        0.4788       0.7650        0.6776  13.9431
     14        0.4443       0.7624        0.6877  14.0306
     15        0.4155       0.7715        0.6587
14.0328
     16        0.3874       0.7740        0.6575
13.8163
     17        0.3581       0.7736        0.6607  13.7884
     18        0.3337       0.7801        0.6459
13.7710
     19        0.3078       0.7799        0.6515  13.7418
     20        0.2844       0.7740        0.6669  13.6890
     21        0.2644       0.7758        0.6733  13.6630
     22        0.2433       0.7715        0.6883  13.8098
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 128, the learning rate was 0.001
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        1.5918       0.5432        1.2453
3.1035
      2        1.1911       0.6467        1.0016
3.1075
      3        1.0245       0.6690        0.9421
3.1208
      4        0.9245       0.7108        0.8356
3.1018
      5        0.8580       0.7191        0.8196
```

```
3.1075
      6       0.8062      0.7294      0.7835
3.1001
      7       0.7627      0.7342      0.7764
3.0934
      8       0.7159      0.7441      0.7422
3.0959
      9       0.6845      0.7535      0.7172
3.1002
     10       0.6500      0.7504      0.7301  3.1135
     11       0.6233      0.7222      0.8134  3.1064
     12       0.5967      0.7500      0.7301  3.1062
     13       0.5679      0.7382      0.7676  3.0977
     14       0.5419      0.7573      0.7070
3.1099
     15       0.5189      0.7275      0.8109  3.1219
     16       0.5012      0.7383      0.7763  3.1253
     17       0.4856      0.7759      0.6584
3.1117
     18       0.4610      0.7671      0.6920  3.1101
     19       0.4503      0.7484      0.7565  3.1313
     20       0.4330      0.7603      0.7146  3.1280
     21       0.4164      0.7584      0.7378  3.1179
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 256, the learning rate was 0.001
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1       1.6579      0.5450      1.2561
6.1184
      2       1.1583      0.6407      1.0060
6.1430
      3       0.9712      0.6851      0.9020
6.1335
      4       0.8689      0.7178      0.8076
6.1475
      5       0.7908      0.7155      0.8156  6.1657
      6       0.7173      0.7254      0.7869
6.1685
      7       0.6691      0.7162      0.8146  6.1811
      8       0.6200      0.7231      0.8026  6.2037
      9       0.5697      0.7593      0.6993
6.1986
     10       0.5247      0.7511      0.7574  6.2029
     11       0.4784      0.7777      0.6565
6.2012
     12       0.4481      0.7608      0.7093  6.2085
     13       0.4193      0.7678      0.7088  6.2271
     14       0.3876      0.7664      0.7174  6.2404
```

```
     15        0.3628       0.7653        0.7162  6.2230
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 512, the learning rate was 0.001
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
      1        1.8660       0.5239        1.3267
13.8909
      2        1.2144       0.6288        1.0492
14.0318
      3        0.9933       0.6585        1.0033
14.0499
      4        0.8716       0.7019        0.8538
14.0592
      5        0.7827       0.7338        0.7637
13.9973
      6        0.7105       0.7051        0.8631  13.9783
      7        0.6455       0.7081        0.8460  14.0149
      8        0.5830       0.7151        0.8477  14.0355
      9        0.5239       0.7465        0.7621
14.0760
     10        0.4676       0.7337        0.8000  14.0717
     11        0.4258       0.7354        0.8330  13.8616
     12        0.4056       0.7492        0.8092  13.8587
     13        0.3547       0.7633        0.7739  13.9999
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 128, the learning rate was 0.01
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        2.4661       0.3902        1.6711
3.1283
      2        1.4921       0.5048        1.3910
3.1077
      3        1.2807       0.5838        1.1837
3.1125
      4        1.1125       0.6630        0.9687
3.0927
      5        0.9888       0.6790        0.9151
3.0898
      6        0.9001       0.6382        1.0577  3.0995
      7        0.8429       0.6774        0.9419  3.0966
      8        0.7930       0.7120        0.8200
3.0852
      9        0.7398       0.6705        0.9908  3.0919
     10        0.7032       0.7199        0.8102
3.0905
     11        0.6733       0.7302        0.7912
3.0858
     12        0.6391       0.7149        0.8442  3.0781
```

```
     13        0.6147       0.7170        0.8561  3.0809
     14        0.5882       0.7029        0.8850  3.0770
     15        0.5654       0.7282        0.8245  3.0779
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 256, the learning rate was 0.01
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        3.5919       0.3983        1.6982
6.0360
      2        1.4422       0.4942        1.4063
6.0587
      3        1.2272       0.6151        1.1065
6.0509
      4        1.0562       0.6300        1.0532
6.0635
      5        0.9361       0.6477        1.0331
6.0767
      6        0.8409       0.6235        1.1104  6.0766
      7        0.7679       0.7008        0.8644
6.0790
      8        0.6919       0.6667        0.9843  6.0795
      9        0.6367       0.6870        0.9418  6.0814
     10        0.5896       0.7083        0.8764  6.0957
     11        0.5484       0.7123        0.8610
6.1184
     12        0.4997       0.7515        0.7285
6.1226
     13        0.4751       0.7501        0.7545  6.1230
     14        0.4319       0.7270        0.8634  6.1406
     15        0.4018       0.7519        0.7678  6.1402
     16        0.3827       0.7382        0.8527  6.1434
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 512, the learning rate was 0.01
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
      1        6.0569       0.4086        1.6634
13.7351
      2        1.5010       0.4979        1.4120
13.8113
      3        1.2651       0.5863        1.1882
13.7868
      4        1.0987       0.6096        1.1351
13.8378
      5        0.9611       0.6584        0.9870
13.8555
      6        0.8590       0.6933        0.8747
13.9215
      7        0.7620       0.7215        0.8052
```

```
13.9019
      8        0.6876       0.7403        0.7532
13.9872
      9        0.6161       0.7516        0.7395
13.9732
     10        0.5579       0.6949        0.9249  13.9310
     11        0.4903       0.6445        1.1340  13.9594
     12        0.4353       0.7367        0.7914  13.9665
     13        0.3780       0.7526        0.7753  13.9434
Stopping since valid_loss has not improved in the last 5 epochs.
```

Write down the **validation accuracy** of the model under different hyperparameter settings.

learning rate = 10^{-4}: | #channel for each layer | validation accuracy | |:——————:|:————-:| | 128 | 0.7755 | | 256 | 0.7860 | | 512 | 0.7801 |

learning rate = 10^{-3}: | #channel for each layer | validation accuracy | |:——————:|:————-:| | 128 | 0.7759 | | 256 | 0.7777 | | 512 | 0.7633 |

learning rate = 10^{-2}: | #channel for each layer | validation accuracy | |:——————:|:————-:| | 128 | 0.7302 | | 256 | 0.7519 | | 512 | 0.7526 |

For the best model you have, test it on the test set.

[ ]:
```python
# implement the same input normalization & type cast here

optimizer = torch.optim.Adam
learning_rate = 1e-4
channel = 256

cnn = CNN_MaxPool(channels = channel)
model = skorch.NeuralNetClassifier(cnn, criterion=torch.nn.CrossEntropyLoss,
                                   device="cuda",
                                   optimizer=optimizer,
                                   lr=learning_rate,
                                   max_epochs=50,
                                   batch_size=32 * 8,
                                   callbacks=[skorch.callbacks.
 ↪EarlyStopping(lower_is_better=True)])

# implement input normalization & type cast here
model.fit(train_data_normalized, np.asarray(train.targets))


test_data_normalized = torch.Tensor(test.data/255)
test_data_normalized = test_data_normalized.permute(0,3,1,2)
test.predictions = model.predict(test_data_normalized)
sklearn.metrics.accuracy_score(test.targets, test.predictions)
```

```
  epoch    train_loss    valid_acc    valid_loss     dur
```

1	1.6229	0.5552	1.2808	5.7887
2	1.2905	0.6147	1.1130	5.7904
3	1.1429	0.6455	1.0174	5.7758
4	1.0483	0.6692	0.9528	5.7884
5	0.9770	0.6930	0.8981	5.7653
6	0.9212	0.7024	0.8618	5.7668
7	0.8761	0.7114	0.8418	5.8012
8	0.8357	0.7177	0.8171	5.7866
9	0.8008	0.7259	0.7928	5.7409
10	0.7699	0.7294	0.7760	5.7591
11	0.7416	0.7389	0.7623	5.7847
12	0.7144	0.7436	0.7479	5.7757
13	0.6867	0.7459	0.7379	5.7796
14	0.6621	0.7522	0.7261	5.7849
15	0.6387	0.7562	0.7117	5.7848
16	0.6178	0.7578	0.7046	5.7691
17	0.5950	0.7620	0.6940	5.7612
18	0.5779	0.7624	0.6915	5.7960
19	0.5589	0.7671	0.6826	5.7776
20	0.5397	0.7683	0.6823	5.7812
21	0.5170	0.7703	0.6769	5.7648
22	0.5039	0.7711	0.6676	5.7785
23	0.4839	0.7740	0.6576	5.7657
24	0.4651	0.7744	0.6569	

```
5.7748
      25        0.4493        0.7770        0.6514
5.7814
      26        0.4353        0.7772        0.6568  5.7899
      27        0.4183        0.7785        0.6526  5.7753
      28        0.4051        0.7792        0.6506
5.7794
      29        0.3886        0.7831        0.6463
5.7860
      30        0.3738        0.7800        0.6461  5.7802
      31        0.3578        0.7841        0.6455
5.7607
      32        0.3497        0.7830        0.6408  5.7976
      33        0.3381        0.7836        0.6397  5.7657
      34        0.3251        0.7826        0.6426  5.7709
      35        0.3099        0.7844        0.6464  5.7790
      36        0.2998        0.7850        0.6435  5.7733
      37        0.2838        0.7862        0.6367
5.7886
      38        0.2772        0.7858        0.6401  5.7790
      39        0.2692        0.7871        0.6405  5.8318
      40        0.2595        0.7866        0.6365  5.7907
      41        0.2484        0.7872        0.6409  5.7713
      42        0.2385        0.7827        0.6520  5.7901
      43        0.2300        0.7860        0.6646  5.7787
      44        0.2202        0.7845        0.6562  5.7917
Stopping since valid_loss has not improved in the last 5 epochs.
```

[ ]: 0.7804

How much **test accuracy** do you get? What can you conclude for the design of CNN structure and tuning of hyperparameters? (5 points)

Your Answer:

1. The test accuracy we got is 0.7804.
   The best model is the one with 256 channels for each layer and learning rate 10^{-4}.
2. From the results above, we can conclude that the number of channels, learning rate and the optimizer are important hyperparameters, they could greatly influence the training speed and evaluation effects. Also the structure of CNN is also important, the max pooling layer could improve the performance of the model. And with dropout layer, the model could be more robust to against overfitting.
   The number of channels which also influence the net structure, could effect performence as hyperparameter. It should not be too small or too large. The learning rate should not be too large, otherwise the model will not converge.