HW5-Coding

January 11, 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.

[1]:
```
!pip install -q torch skorch torchvision torchtext
```

[2]:
```
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
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.

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

[4]:
```
# 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[0,:,:])
# Use torch.cat and torch.unsqueeze to generate such a z
z = torch.cat((torch.unsqueeze(x, 0), torch.unsqueeze(y, 0)))
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.

[5]:
```
# 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, 3))
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.

[6]:

```python
# 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.
x_mean = torch.mean(x, dim=1)
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, x_max_indices = torch.max(x, dim=1)
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, x_min_indices = torch.min(x, dim=1)
print(x_min.shape)

# Compute the top-5 values for each row of x.
# 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.
x_xtop, x_xtop_indices = torch.topk(x, k=5, dim=1)
print(x_xtop.shape)
```

```
tensor(-0.0311)
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.

[7]:

```python
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.

[8]:
```
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.

[9]:
```
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)*****
    self.conv1 = nn.Conv2d(3, channels, kernel_size=3, padding=1)
    self.conv2 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)
    self.conv3 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)
    self.fc = nn.Linear(channels * 32 * 32, 10)
```

```
    self.relu = nn.ReLU()
    # *****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.relu(self.conv1(images))
    images = self.relu(self.conv2(images))
    images = self.relu(self.conv3(images))
    images = images.view(images.size(0), -1)
    images = self.fc(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.

[10]:
```
# implement hyperparameters, you can select and modify the hyperparameters by␣
↪yourself here.

optimize = [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 optimize:
    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,
                                   lr=l,
                                   max_epochs=50,
                                   batch_size=32,
                                   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 0.001 and the optimizer was <class
'torch.optim.sgd.SGD'>
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        2.1538       0.2984        1.9552
2.6352
      2        1.8782       0.3644        1.8057
2.5159
      3        1.7644       0.4022        1.7063
2.5077
      4        1.6868       0.4274        1.6442
2.5075
      5        1.6331       0.4393        1.5999
2.5342
      6        1.5890       0.4508        1.5622
2.5138
      7        1.5504       0.4631        1.5283
2.5257
      8        1.5158       0.4754        1.4974
2.4884
      9        1.4845       0.4845        1.4698
2.5094
     10        1.4560       0.4921        1.4452
2.5153
     11        1.4297       0.4997        1.4233
2.5087
     12        1.4052       0.5050        1.4040
2.5080
     13        1.3819       0.5106        1.3859
2.4973
     14        1.3589       0.5151        1.3680
2.5359
     15        1.3353       0.5195        1.3490
2.5063
     16        1.3106       0.5262        1.3286
2.5126
     17        1.2854       0.5358        1.3081
2.5261
     18        1.2608       0.5433        1.2885
2.5423
     19        1.2380       0.5490        1.2714
2.5132
     20        1.2174       0.5529        1.2568
2.5087
     21        1.1986       0.5548        1.2440
2.5285
     22        1.1813       0.5578        1.2328
2.5431
```

23	1.1651	0.5624	1.2229	2.5441
24	1.1498	0.5653	1.2139	2.5229
25	1.1349	0.5673	1.2058	2.5197
26	1.1205	0.5712	1.1980	2.5426
27	1.1063	0.5731	1.1907	2.5121
28	1.0922	0.5748	1.1839	2.5384
29	1.0782	0.5791	1.1773	2.5581
30	1.0641	0.5818	1.1709	2.5399
31	1.0499	0.5848	1.1647	2.5395
32	1.0356	0.5878	1.1586	2.5388
33	1.0213	0.5893	1.1528	2.6760
34	1.0068	0.5936	1.1473	2.5394
35	0.9923	0.5969	1.1414	2.5441
36	0.9779	0.5990	1.1361	2.5429
37	0.9634	0.6004	1.1311	2.5244
38	0.9489	0.6023	1.1261	2.5313
39	0.9345	0.6055	1.1215	2.5420
40	0.9201	0.6075	1.1168	2.5460
41	0.9057	0.6098	1.1127	2.5793
42	0.8912	0.6113	1.1087	2.5446
43	0.8767	0.6123	1.1049	2.5421
44	0.8621	0.6134	1.1013	2.5616
45	0.8474	0.6151	1.0979	2.5381
46	0.8326	0.6157	1.0951	2.5510

```
     47        0.8176       0.6175        1.0925
2.5599
     48        0.8026       0.6194        1.0902
2.5520
     49        0.7875       0.6209        1.0882
2.5809
     50        0.7722       0.6213        1.0867
2.5457
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        2.0563       0.3406        1.8789
7.1793
      2        1.8134       0.3898        1.7401
7.1448
      3        1.7079       0.4220        1.6553
7.1275
      4        1.6390       0.4405        1.5975
7.1238
      5        1.5826       0.4577        1.5463
7.1552
      6        1.5311       0.4734        1.4980
7.1274
      7        1.4837       0.4892        1.4555
7.1010
      8        1.4430       0.5009        1.4227
7.0573
      9        1.4095       0.5078        1.3980
7.0830
     10        1.3811       0.5110        1.3779
7.0372
     11        1.3549       0.5164        1.3582
7.0741
     12        1.3289       0.5213        1.3379
7.0458
     13        1.3020       0.5306        1.3161
7.1066
     14        1.2740       0.5382        1.2939
7.0709
     15        1.2460       0.5451        1.2722
7.2062
     16        1.2196       0.5524        1.2528
7.1100
     17        1.1956       0.5570        1.2357
7.1131
     18        1.1735       0.5639        1.2209
7.1158
```

19	1.1529	0.5683	1.2078	7.1141
20	1.1333	0.5725	1.1960	7.1172
21	1.1144	0.5777	1.1853	7.1421
22	1.0959	0.5817	1.1752	7.0767
23	1.0775	0.5858	1.1655	7.1139
24	1.0591	0.5900	1.1564	7.1460
25	1.0407	0.5954	1.1477	7.0990
26	1.0222	0.5977	1.1392	7.0885
27	1.0036	0.6006	1.1308	7.0724
28	0.9851	0.6029	1.1229	7.0989
29	0.9666	0.6045	1.1153	7.0775
30	0.9482	0.6071	1.1081	7.1163
31	0.9300	0.6106	1.1012	7.1367
32	0.9118	0.6126	1.0948	7.1540
33	0.8938	0.6158	1.0887	7.1020
34	0.8758	0.6178	1.0832	7.1123
35	0.8579	0.6211	1.0781	7.0862
36	0.8400	0.6228	1.0733	7.0589
37	0.8221	0.6255	1.0692	7.0586
38	0.8041	0.6254	1.0656	7.0663
39	0.7862	0.6277	1.0625	7.0586
40	0.7682	0.6290	1.0601	7.1094
41	0.7503	0.6310	1.0586	7.0598
42	0.7325	0.6326	1.0580	6.9988
43	0.7148	0.6352	1.0584	7.0550

```
     44        0.6971       0.6348        1.0597  7.0274
     45        0.6796       0.6364        1.0621  7.0591
     46        0.6622       0.6372        1.0657  7.0275
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        1.9754       0.3650        1.8107
19.2832
      2        1.7411       0.4187        1.6621
19.2475
      3        1.6337       0.4455        1.5773
19.2669
      4        1.5530       0.4695        1.5058
19.3221
      5        1.4831       0.4895        1.4450
19.3939
      6        1.4225       0.5102        1.3954
18.7571
      7        1.3679       0.5199        1.3532
18.7726
      8        1.3187       0.5309        1.3183
18.3529
      9        1.2777       0.5409        1.2904
19.2355
     10        1.2442       0.5488        1.2685
19.4449
     11        1.2156       0.5557        1.2501
19.3896
     12        1.1897       0.5616        1.2341
19.4934
     13        1.1651       0.5666        1.2193
19.5106
     14        1.1410       0.5718        1.2054
19.4175
     15        1.1172       0.5775        1.1923
19.5110
     16        1.0936       0.5811        1.1802
19.4205
     17        1.0702       0.5858        1.1688
19.4421
     18        1.0472       0.5896        1.1582
19.5591
     19        1.0246       0.5944        1.1484
19.5666
     20        1.0022       0.5985        1.1392
19.5558
```

```
     21       0.9801      0.6020      1.1307
19.5756
     22       0.9582      0.6051      1.1225
19.4179
     23       0.9364      0.6059      1.1148
19.3417
     24       0.9146      0.6090      1.1076
19.3634
     25       0.8930      0.6117      1.1009
19.2752
     26       0.8716      0.6145      1.0952
19.3319
     27       0.8503      0.6169      1.0905
19.3732
     28       0.8294      0.6185      1.0869
19.2923
     29       0.8087      0.6203      1.0845
19.3363
     30       0.7882      0.6225      1.0834
19.3682
     31       0.7679      0.6223      1.0835  19.3122
     32       0.7478      0.6247      1.0848  19.3185
     33       0.7277      0.6265      1.0873  19.3377
     34       0.7077      0.6262      1.0910  19.3369
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        1.5234       0.5851        1.1891
2.6840
      2        1.0571       0.6169        1.0892
2.6992
      3        0.8221       0.6141        1.1500  2.6971
      4        0.6047       0.6024        1.3418  2.6915
      5        0.4299       0.5875        1.6522  2.7243
      6        0.3110       0.5717        2.0032  2.7339
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        1.5872       0.5492        1.2596
7.2608
      2        1.1153       0.6067        1.1385
7.2793
      3        0.8639       0.6174        1.1836  7.2944
      4        0.6562       0.5973        1.4399  7.3466
```

```
      5        0.4779       0.5787        1.6401  7.3964
      6        0.3543       0.5912        1.8194  7.3584
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        2.0094       0.4524        1.5763
20.1288
      2        1.3341       0.5933        1.1718
20.0618
      3        1.0120       0.6028        1.1743  20.0439
      4        0.7524       0.5812        1.4462  20.0632
      5        0.5117       0.5493        1.8079  20.1398
      6        0.3501       0.5658        2.0140  19.8778
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        1.7794       0.4516        1.5389
2.5239
      2        1.4475       0.5071        1.3684
2.5350
      3        1.3014       0.5455        1.2685
2.5639
      4        1.1840       0.5665        1.2059
2.5228
      5        1.0889       0.5884        1.1525
2.5110
      6        0.9935       0.6106        1.0999
2.5434
      7        0.9005       0.6267        1.0665
2.4998
      8        0.8147       0.6328        1.0625
2.5299
      9        0.7331       0.6365        1.0841  2.5514
     10        0.6519       0.6339        1.1270  2.5452
     11        0.5681       0.6330        1.1972  2.5207
     12        0.4801       0.6275        1.3198  2.5524
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        1.7445       0.4624        1.4953
7.0353
      2        1.3844       0.5303        1.3078
```

```
7.0792
      3        1.2215       0.5637        1.2228
7.0867
      4        1.0985       0.5894        1.1513
7.0975
      5        0.9825       0.6145        1.0954
7.0483
      6        0.8741       0.6258        1.0755
7.0947
      7        0.7733       0.6318        1.0788  7.1635
      8        0.6754       0.6339        1.1130  7.1180
      9        0.5752       0.6342        1.1827  7.0919
     10        0.4693       0.6280        1.2978  7.1209
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        1.7083       0.4843        1.4394
19.4529
      2        1.3369       0.5464        1.2704
19.4450
      3        1.1600       0.5789        1.1729
19.5391
      4        1.0139       0.6071        1.1062
19.3637
      5        0.8858       0.6250        1.0830
19.4428
      6        0.7689       0.6330        1.0816
19.3910
      7        0.6552       0.6359        1.1100  19.2039
      8        0.5372       0.6317        1.1976  19.2425
      9        0.4117       0.6256        1.3515  19.2913
     10        0.2870       0.6163        1.5788  19.3785
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        3.0769       0.1000        2.3039
2.7074
      2        2.3042       0.1000        2.3039  2.6637
      3        2.3042       0.1000        2.3039  2.6802
      4        2.3042       0.1000        2.3039  2.6868
      5        2.3042       0.1000        2.3039  2.7117
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        6.5130       0.1000        2.3039
7.2951
      2        2.3042       0.1000        2.3039  7.3146
      3        2.3042       0.1000        2.3039  7.3648
      4        2.3042       0.1000        2.3039  7.2656
      5        2.3042       0.1000        2.3039  7.2715
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       42.1812       0.1000        2.3039
19.8690
      2        2.3042       0.1000        2.3039  19.7609
      3        2.3042       0.1000        2.3039  19.7402
      4        2.3042       0.1000        2.3039  19.7742
      5        2.3042       0.1000        2.3039  19.8196
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()`.

#channel for each layer	optimizer	SGD	Adam
128		0.6365	0.6169
256		0.6372	0.6147
512		0.6359	0.6028

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.

[11]:
```
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.conv1 = nn.Conv2d(3, channels, kernel_size=3, padding=1)
    self.conv2 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)
```

```python
        self.conv3 = nn.Conv2d(channels, channels, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc = nn.Linear(channels * 16, 10)
        self.relu = nn.ReLU()
        # *****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.pool(self.relu(self.conv1(images)))
        images = self.pool(self.relu(self.conv2(images)))
        images = self.pool(self.relu(self.conv3(images)))
        images = self.fc(images.view(images.size(0), -1))
        # *****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.

[12]:
```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.SGD

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,
                                   optimizer__momentum=0.90,
                                   lr=l,
                                   max_epochs=500,
                                   batch_size=32,
                                   callbacks=[skorch.callbacks.
 ↪EarlyStopping(lower_is_better=True)])
      # implement input normalization & type cast here
```

```python
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        2.2981       0.1008        2.2931
1.2544
      2        2.2873       0.1160        2.2783
1.2456
      3        2.2637       0.1913        2.2390
1.2406
      4        2.1914       0.2647        2.1200
1.2659
      5        2.0630       0.2773        2.0112
1.2407
      6        1.9991       0.2887        1.9726
1.2388
      7        1.9641       0.3045        1.9406
1.2428
      8        1.9280       0.3187        1.9043
1.2371
      9        1.8858       0.3386        1.8589
1.2442
     10        1.8381       0.3637        1.8082
1.2584
     11        1.7899       0.3852        1.7595
1.2423
     12        1.7442       0.4009        1.7142
1.2251
     13        1.7024       0.4138        1.6746
1.2344
     14        1.6661       0.4223        1.6420
1.2490
     15        1.6354       0.4299        1.6143
1.2432
     16        1.6084       0.4401        1.5900
1.2352
     17        1.5838       0.4467        1.5679
1.2908
     18        1.5611       0.4516        1.5477
1.2521
     19        1.5402       0.4585        1.5292
1.2573
     20        1.5211       0.4652        1.5121
1.2508
     21        1.5036       0.4728        1.4967
1.2525
     22        1.4875       0.4788        1.4822
```

```
1.2309
      23          1.4726          0.4852          1.4683
1.2362
      24          1.4586          0.4911          1.4552
1.2294
      25          1.4453          0.4969          1.4425
1.2294
      26          1.4325          0.5011          1.4299
1.2365
      27          1.4200          0.5061          1.4177
1.2458
      28          1.4078          0.5109          1.4060
1.2398
      29          1.3958          0.5138          1.3943
1.2350
      30          1.3839          0.5173          1.3820
1.2439
      31          1.3720          0.5226          1.3699
1.2308
      32          1.3602          0.5258          1.3580
1.2276
      33          1.3484          0.5283          1.3461
1.2491
      34          1.3367          0.5329          1.3342
1.2311
      35          1.3252          0.5367          1.3225
1.2333
      36          1.3139          0.5403          1.3114
1.2435
      37          1.3028          0.5437          1.3002
1.2431
      38          1.2918          0.5479          1.2895
1.2364
      39          1.2810          0.5519          1.2791
1.2408
      40          1.2705          0.5546          1.2688
1.2368
      41          1.2601          0.5595          1.2589
1.2404
      42          1.2499          0.5620          1.2493
1.2619
      43          1.2399          0.5665          1.2402
1.2810
      44          1.2301          0.5695          1.2310
1.2518
      45          1.2205          0.5723          1.2224
1.2632
      46          1.2111          0.5758          1.2136
```

1.2688			
47	1.2018	0.5788	1.2054
1.2762			
48	1.1927	0.5794	1.1972
1.3111			
49	1.1839	0.5825	1.1890
1.2514			
50	1.1752	0.5849	1.1816
1.2512			
51	1.1666	0.5883	1.1741
1.2772			
52	1.1583	0.5928	1.1668
1.2731			
53	1.1501	0.5940	1.1598
1.2501			
54	1.1420	0.5962	1.1530
1.2558			
55	1.1342	0.5984	1.1462
1.2485			
56	1.1264	0.6009	1.1398
1.2486			
57	1.1189	0.6030	1.1334
1.2641			
58	1.1116	0.6055	1.1269
1.2625			
59	1.1044	0.6076	1.1207
1.2528			
60	1.0973	0.6091	1.1148
1.2472			
61	1.0903	0.6111	1.1092
1.2445			
62	1.0835	0.6142	1.1034
1.2780			
63	1.0768	0.6174	1.0979
1.2460			
64	1.0702	0.6194	1.0930
1.2480			
65	1.0637	0.6210	1.0878
1.2386			
66	1.0573	0.6230	1.0830
1.2646			
67	1.0510	0.6253	1.0780
1.2486			
68	1.0448	0.6269	1.0734
1.2703			
69	1.0387	0.6283	1.0687
1.2576			
70	1.0326	0.6301	1.0642

1.2739
71 1.0266 0.6322 1.0599
1.2543
72 1.0206 0.6338 1.0559
1.2459
73 1.0147 0.6339 1.0516
1.2536
74 1.0089 0.6364 1.0477
1.2660
75 1.0031 0.6377 1.0436
1.2520
76 0.9974 0.6397 1.0397
1.2618
77 0.9917 0.6412 1.0360
1.2456
78 0.9861 0.6424 1.0323
1.2511
79 0.9805 0.6440 1.0291
1.2561
80 0.9750 0.6443 1.0255
1.2574
81 0.9696 0.6457 1.0218
1.2423
82 0.9641 0.6461 1.0180
1.2455
83 0.9587 0.6481 1.0143
1.2495
84 0.9534 0.6498 1.0108
1.2673
85 0.9481 0.6505 1.0075
1.2664
86 0.9428 0.6519 1.0042
1.2642
87 0.9376 0.6537 1.0012
1.2540
88 0.9325 0.6549 0.9985
1.2519
89 0.9274 0.6565 0.9954
1.2496
90 0.9224 0.6569 0.9925
1.2376
91 0.9174 0.6578 0.9894
1.2434
92 0.9124 0.6591 0.9864
1.2525
93 0.9074 0.6598 0.9833
1.2669
94 0.9025 0.6612 0.9807

1.2446
95 0.8977 0.6627 0.9778
1.2639
96 0.8929 0.6631 0.9749
1.2478
97 0.8882 0.6639 0.9721
1.2631
98 0.8835 0.6635 0.9698 1.2525
99 0.8788 0.6652 0.9671
1.2460
100 0.8741 0.6660 0.9646
1.2557
101 0.8695 0.6667 0.9622
1.2157
102 0.8649 0.6670 0.9600
1.2226
103 0.8604 0.6680 0.9579
1.2470
104 0.8560 0.6685 0.9556
1.2471
105 0.8515 0.6696 0.9531
1.2636
106 0.8471 0.6708 0.9509
1.2453
107 0.8427 0.6719 0.9488
1.2648
108 0.8384 0.6730 0.9467
1.2416
109 0.8341 0.6739 0.9449
1.2390
110 0.8299 0.6743 0.9427
1.2494
111 0.8256 0.6754 0.9408
1.2491
112 0.8214 0.6762 0.9387
1.3029
113 0.8171 0.6774 0.9368
1.2335
114 0.8130 0.6783 0.9350
1.2382
115 0.8088 0.6788 0.9337
1.2641
116 0.8047 0.6790 0.9319
1.2284
117 0.8006 0.6799 0.9305
1.2815
118 0.7965 0.6797 0.9289 1.2423
119 0.7924 0.6804 0.9277

1.2344				
120	0.7884	0.6814	0.9267	
1.2422				
121	0.7844	0.6825	0.9251	
1.2359				
122	0.7804	0.6836	0.9243	
1.2330				
123	0.7765	0.6839	0.9228	
1.2388				
124	0.7725	0.6838	0.9213	1.2402
125	0.7686	0.6843	0.9204	
1.2427				
126	0.7647	0.6852	0.9196	
1.2349				
127	0.7609	0.6853	0.9184	
1.2369				
128	0.7570	0.6855	0.9180	
1.2403				
129	0.7532	0.6864	0.9166	
1.2687				
130	0.7494	0.6865	0.9161	
1.2445				
131	0.7457	0.6863	0.9149	1.2484
132	0.7419	0.6866	0.9139	
1.2538				
133	0.7382	0.6870	0.9134	
1.2598				
134	0.7344	0.6867	0.9122	1.2445
135	0.7308	0.6870	0.9109	1.2340
136	0.7271	0.6877	0.9100	
1.2483				
137	0.7234	0.6882	0.9089	
1.2399				
138	0.7198	0.6890	0.9078	
1.2385				
139	0.7161	0.6893	0.9069	
1.2498				
140	0.7124	0.6896	0.9061	
1.2430				
141	0.7089	0.6899	0.9049	
1.2465				
142	0.7053	0.6908	0.9041	
1.2577				
143	0.7016	0.6909	0.9034	
1.2394				
144	0.6981	0.6913	0.9032	
1.2413				
145	0.6946	0.6922	0.9023	

```
1.2326
    146        0.6911        0.6923        0.9022
1.2295
    147        0.6875        0.6926        0.9014
1.2506
    148        0.6840        0.6930        0.9010
1.2510
    149        0.6805        0.6928        0.9002  1.2589
    150        0.6770        0.6929        0.9003  1.2646
    151        0.6736        0.6928        0.8998  1.2367
    152        0.6702        0.6926        0.9000  1.2473
    153        0.6667        0.6926        0.9000  1.2560
    154        0.6633        0.6929        0.8997  1.2398
    155        0.6599        0.6928        0.8996  1.2459
    156        0.6565        0.6930        0.8992  1.2429
    157        0.6531        0.6935        0.8989
1.2432
    158        0.6497        0.6941        0.8987
1.2375
    159        0.6464        0.6939        0.8989  1.2657
    160        0.6429        0.6943        0.8987
1.2539
    161        0.6396        0.6945        0.8985
1.2493
    162        0.6362        0.6949        0.8977
1.2395
    163        0.6329        0.6952        0.8974
1.2434
    164        0.6295        0.6964        0.8972
1.2459
    165        0.6262        0.6963        0.8976  1.2451
    166        0.6229        0.6966        0.8970
1.2489
    167        0.6196        0.6969        0.8970  1.2390
    168        0.6163        0.6969        0.8966  1.2390
    169        0.6130        0.6972        0.8965
1.2420
    170        0.6097        0.6971        0.8970  1.2501
    171        0.6064        0.6972        0.8972  1.2455
    172        0.6031        0.6973        0.8973  1.2587
    173        0.5999        0.6977        0.8974  1.2335
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 256, the learning rate was 0.0001
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        2.2903       0.1550        2.2764
2.0048
      2        2.2449       0.2557        2.1880
```

```
2.0367
     3        2.0923        0.2833        2.0082
1.9728
     4        1.9878        0.2994        1.9576
1.9713
     5        1.9433        0.3160        1.9156
1.9831
     6        1.8947        0.3414        1.8631
1.9908
     7        1.8377        0.3682        1.8015
1.9663
     8        1.7765        0.3975        1.7378
1.9583
     9        1.7172        0.4138        1.6813
1.9663
    10        1.6679        0.4275        1.6381
1.9672
    11        1.6291        0.4359        1.6038
1.9414
    12        1.5966        0.4453        1.5742
1.9511
    13        1.5674        0.4545        1.5478
1.9501
    14        1.5405        0.4639        1.5238
1.9752
    15        1.5154        0.4712        1.5013
1.9437
    16        1.4917        0.4786        1.4799
1.9782
    17        1.4694        0.4865        1.4592
1.9980
    18        1.4482        0.4944        1.4391
1.9624
    19        1.4282        0.5017        1.4202
2.0186
    20        1.4094        0.5087        1.4026
2.1429
    21        1.3914        0.5141        1.3858
1.9936
    22        1.3742        0.5193        1.3698
1.9731
    23        1.3577        0.5244        1.3539
1.9545
    24        1.3417        0.5297        1.3381
1.9676
    25        1.3262        0.5351        1.3228
2.0107
    26        1.3112        0.5422        1.3077
```

2.0154				
	27	1.2967	0.5489	1.2933
1.9544				
	28	1.2826	0.5542	1.2793
1.9334				
	29	1.2689	0.5590	1.2659
1.9792				
	30	1.2555	0.5633	1.2532
1.9778				
	31	1.2426	0.5684	1.2409
1.9689				
	32	1.2300	0.5723	1.2291
1.9635				
	33	1.2178	0.5752	1.2180
1.9777				
	34	1.2059	0.5789	1.2072
1.9589				
	35	1.1945	0.5826	1.1969
1.9786				
	36	1.1833	0.5852	1.1869
1.9523				
	37	1.1725	0.5895	1.1772
1.9688				
	38	1.1621	0.5923	1.1680
1.9473				
	39	1.1519	0.5945	1.1592
1.9135				
	40	1.1420	0.5976	1.1508
1.9208				
	41	1.1323	0.5991	1.1423
1.9732				
	42	1.1229	0.6027	1.1343
1.9566				
	43	1.1137	0.6052	1.1265
1.9981				
	44	1.1047	0.6077	1.1189
2.0796				
	45	1.0959	0.6095	1.1118
2.1442				
	46	1.0872	0.6112	1.1049
2.0307				
	47	1.0788	0.6140	1.0983
2.0678				
	48	1.0705	0.6178	1.0917
2.0187				
	49	1.0623	0.6202	1.0850
2.0403				
	50	1.0543	0.6222	1.0789

				2.0308
51	1.0464	0.6250	1.0724	2.1660
52	1.0387	0.6281	1.0665	2.0301
53	1.0310	0.6291	1.0608	2.0476
54	1.0235	0.6311	1.0552	2.0517
55	1.0161	0.6332	1.0496	2.0480
56	1.0088	0.6365	1.0441	2.0760
57	1.0016	0.6385	1.0388	2.0316
58	0.9944	0.6401	1.0335	2.0642
59	0.9872	0.6420	1.0284	2.0728
60	0.9802	0.6436	1.0236	2.0329
61	0.9732	0.6458	1.0188	2.0265
62	0.9663	0.6471	1.0141	2.0095
63	0.9595	0.6491	1.0092	2.0621
64	0.9527	0.6506	1.0044	2.0649
65	0.9459	0.6520	0.9998	2.0479
66	0.9392	0.6535	0.9956	2.0198
67	0.9326	0.6545	0.9911	2.0393
68	0.9260	0.6557	0.9865	2.0554
69	0.9195	0.6580	0.9822	2.0564
70	0.9130	0.6590	0.9779	2.0122
71	0.9065	0.6601	0.9738	2.0433
72	0.9001	0.6618	0.9698	2.0309
73	0.8937	0.6626	0.9659	2.0264
74	0.8874	0.6627	0.9620	

2.0271
75 0.8812 0.6650 0.9582
2.0270
76 0.8749 0.6660 0.9548
2.0442
77 0.8688 0.6680 0.9510
2.0258
78 0.8627 0.6696 0.9471
2.0518
79 0.8566 0.6712 0.9433
2.0563
80 0.8507 0.6721 0.9396
2.0658
81 0.8447 0.6743 0.9363
2.0697
82 0.8389 0.6755 0.9327
2.0295
83 0.8330 0.6771 0.9294
2.0627
84 0.8273 0.6777 0.9257
2.0321
85 0.8215 0.6795 0.9224
2.0290
86 0.8158 0.6799 0.9191
2.0629
87 0.8102 0.6807 0.9162
2.0277
88 0.8045 0.6818 0.9127
2.0242
89 0.7990 0.6822 0.9097
2.0278
90 0.7935 0.6830 0.9068
2.0422
91 0.7880 0.6843 0.9040
2.0201
92 0.7827 0.6844 0.9011
2.0660
93 0.7773 0.6862 0.8985
2.0853
94 0.7720 0.6863 0.8961
2.0685
95 0.7667 0.6868 0.8939
2.0514
96 0.7614 0.6883 0.8914
2.0493
97 0.7561 0.6895 0.8889
2.0512
98 0.7509 0.6906 0.8866

2.0296			
99	0.7457	0.6918	0.8843
2.0263			
100	0.7406	0.6923	0.8823
2.0315			
101	0.7355	0.6933	0.8796
2.0452			
102	0.7304	0.6952	0.8776
2.0468			
103	0.7253	0.6960	0.8756
2.0622			
104	0.7203	0.6962	0.8738
2.0462			
105	0.7152	0.6964	0.8720
2.0219			
106	0.7103	0.6968	0.8703
2.0615			
107	0.7053	0.6980	0.8687
2.0403			
108	0.7004	0.6997	0.8666
2.0554			
109	0.6955	0.7003	0.8650
2.0276			
110	0.6906	0.7005	0.8635
2.0263			
111	0.6857	0.7008	0.8621
2.0260			
112	0.6808	0.7010	0.8612
2.0502			
113	0.6760	0.7015	0.8599
2.0579			
114	0.6712	0.7023	0.8586
2.0614			
115	0.6663	0.7028	0.8574
2.0420			
116	0.6615	0.7039	0.8558
2.0442			
117	0.6567	0.7045	0.8546
2.0396			
118	0.6520	0.7050	0.8534
2.0352			
119	0.6472	0.7057	0.8526
2.0395			
120	0.6424	0.7058	0.8516
2.0554			
121	0.6377	0.7061	0.8511
2.0780			
122	0.6329	0.7070	0.8501

```
2.0524
    123         0.6282        0.7073        0.8498
2.0583
    124         0.6235        0.7080        0.8490
2.0361
    125         0.6188        0.7081        0.8485
2.0400
    126         0.6141        0.7086        0.8479
2.0471
    127         0.6094        0.7095        0.8476
2.0227
    128         0.6048        0.7098        0.8472
2.1048
    129         0.6001        0.7093        0.8468  2.0418
    130         0.5954        0.7100        0.8464
2.0521
    131         0.5908        0.7102        0.8463
2.0521
    132         0.5861        0.7097        0.8466  2.0439
    133         0.5815        0.7097        0.8466  2.0447
    134         0.5768        0.7098        0.8463  2.0570
    135         0.5722        0.7106        0.8466  2.0503
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        2.2784       0.2172        2.2421
5.8419
      2        2.1572       0.2795        2.0531
5.8371
      3        2.0003       0.2974        1.9655
5.8715
      4        1.9360       0.3221        1.9092
5.8313
      5        1.8735       0.3526        1.8395
5.8006
      6        1.8012       0.3877        1.7618
5.8270
      7        1.7293       0.4130        1.6916
5.8308
      8        1.6694       0.4266        1.6393
5.8585
      9        1.6218       0.4404        1.5964
5.8416
     10        1.5811       0.4551        1.5581
5.8279
     11        1.5449       0.4642        1.5240
5.8287
```

12	1.5128	0.4762	1.4937	5.8505
13	1.4844	0.4857	1.4668	5.8500
14	1.4590	0.4940	1.4427	5.8015
15	1.4358	0.5027	1.4208	5.8185
16	1.4143	0.5089	1.4003	5.7775
17	1.3938	0.5177	1.3807	5.7809
18	1.3741	0.5274	1.3618	5.8301
19	1.3551	0.5306	1.3440	5.8479
20	1.3369	0.5375	1.3267	5.8261
21	1.3193	0.5432	1.3101	5.8638
22	1.3023	0.5485	1.2939	5.8397
23	1.2859	0.5539	1.2783	5.8306
24	1.2701	0.5607	1.2636	5.8306
25	1.2547	0.5658	1.2493	5.8299
26	1.2398	0.5690	1.2356	5.8520
27	1.2255	0.5725	1.2227	5.8412
28	1.2115	0.5774	1.2100	5.8202
29	1.1980	0.5818	1.1977	5.8272
30	1.1848	0.5849	1.1859	5.8512
31	1.1721	0.5894	1.1747	5.8791
32	1.1597	0.5937	1.1642	5.8637
33	1.1477	0.5972	1.1539	5.8385
34	1.1360	0.6017	1.1440	5.8708
35	1.1246	0.6037	1.1344	5.8588

36	1.1135	0.6068	1.1251	5.8406
37	1.1027	0.6116	1.1162	5.8191
38	1.0921	0.6140	1.1076	5.8183
39	1.0818	0.6164	1.0994	5.8400
40	1.0717	0.6200	1.0912	5.8710
41	1.0618	0.6226	1.0833	5.8396
42	1.0522	0.6244	1.0758	5.8304
43	1.0427	0.6275	1.0682	5.8195
44	1.0335	0.6293	1.0610	5.8364
45	1.0244	0.6317	1.0538	5.8691
46	1.0154	0.6333	1.0469	5.8540
47	1.0066	0.6352	1.0401	5.8213
48	0.9980	0.6393	1.0336	5.8194
49	0.9894	0.6408	1.0274	5.8574
50	0.9810	0.6424	1.0211	5.8138
51	0.9727	0.6446	1.0148	5.8313
52	0.9645	0.6471	1.0088	5.8242
53	0.9564	0.6495	1.0032	5.8398
54	0.9484	0.6513	0.9975	5.9281
55	0.9405	0.6533	0.9920	5.8626
56	0.9327	0.6554	0.9866	5.8507
57	0.9249	0.6569	0.9812	5.8775
58	0.9172	0.6593	0.9760	5.8589
59	0.9096	0.6608	0.9710	5.8561

60	0.9021	0.6627	0.9661	5.8470
61	0.8945	0.6644	0.9611	5.8473
62	0.8872	0.6662	0.9564	5.8372
63	0.8798	0.6673	0.9519	5.8407
64	0.8725	0.6689	0.9474	5.8130
65	0.8653	0.6701	0.9432	5.7768
66	0.8581	0.6720	0.9390	5.8231
67	0.8509	0.6736	0.9348	5.8268
68	0.8439	0.6755	0.9311	5.7574
69	0.8368	0.6765	0.9273	5.7802
70	0.8299	0.6775	0.9238	5.7815
71	0.8229	0.6785	0.9204	5.7865
72	0.8161	0.6788	0.9170	5.7975
73	0.8092	0.6804	0.9135	5.8057
74	0.8024	0.6806	0.9106	5.7843
75	0.7957	0.6818	0.9073	5.7680
76	0.7890	0.6827	0.9045	5.8120
77	0.7823	0.6829	0.9018	5.8518
78	0.7757	0.6840	0.8989	5.8271
79	0.7691	0.6851	0.8959	5.8293
80	0.7626	0.6856	0.8934	5.8416
81	0.7560	0.6865	0.8910	5.8508
82	0.7496	0.6891	0.8888	5.8732
83	0.7431	0.6897	0.8866	5.8831

84	0.7367	0.6901	0.8844	5.8483
85	0.7303	0.6913	0.8825	5.8200
86	0.7239	0.6924	0.8802	5.8591
87	0.7176	0.6934	0.8783	5.8310
88	0.7113	0.6947	0.8765	5.8286
89	0.7050	0.6945	0.8747	5.8146
90	0.6987	0.6947	0.8729	5.8159
91	0.6925	0.6953	0.8716	5.8428
92	0.6862	0.6960	0.8702	5.8774
93	0.6800	0.6963	0.8690	5.8220
94	0.6738	0.6973	0.8673	5.8113
95	0.6676	0.6977	0.8658	5.8110
96	0.6615	0.6979	0.8650	5.8214
97	0.6552	0.6987	0.8638	5.8180
98	0.6491	0.6998	0.8623	5.8396
99	0.6429	0.6998	0.8611	5.8334
100	0.6368	0.7001	0.8600	5.8647
101	0.6306	0.7005	0.8597	5.8714
102	0.6245	0.7008	0.8586	5.8747
103	0.6183	0.7022	0.8580	5.8685
104	0.6122	0.7036	0.8573	5.8599
105	0.6061	0.7038	0.8566	5.8744
106	0.6000	0.7045	0.8560	5.8581
107	0.5939	0.7043	0.8556	5.8937
108	0.5877	0.7047	0.8554	5.8723
109	0.5816	0.7055	0.8550	5.8590

```
    110        0.5755       0.7062        0.8546
5.8729
    111        0.5694       0.7072        0.8542
5.8289
    112        0.5633       0.7078        0.8543  5.8658
    113        0.5572       0.7084        0.8542
5.8715
    114        0.5510       0.7092        0.8545  5.8243
    115        0.5449       0.7101        0.8547  5.8232
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        2.0880       0.3435        1.8522
1.2581
      2        1.7094       0.4222        1.6121
1.2551
      3        1.5293       0.4779        1.4590
1.2461
      4        1.4131       0.5166        1.3561
1.2641
      5        1.3185       0.5545        1.2578
1.2645
      6        1.2342       0.5851        1.1778
1.2813
      7        1.1595       0.6092        1.1174
1.2463
      8        1.0932       0.6240        1.0684
1.2586
      9        1.0339       0.6405        1.0294
1.2445
     10        0.9815       0.6539        0.9903
1.2962
     11        0.9356       0.6654        0.9619
1.2490
     12        0.8951       0.6706        0.9392
1.2587
     13        0.8587       0.6792        0.9227
1.2544
     14        0.8253       0.6838        0.9088
1.2355
     15        0.7949       0.6847        0.9001
1.2700
     16        0.7662       0.6857        0.8936
1.2282
     17        0.7392       0.6896        0.8875
1.2647
     18        0.7137       0.6924        0.8883  1.2491
```

```
     19        0.6892       0.6926        0.8917  1.2563
     20        0.6657       0.6920        0.8973  1.2501
     21        0.6426       0.6930        0.8992  1.2597
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        2.0294       0.3651        1.7795
2.0356
      2        1.6356       0.4571        1.5238
2.0602
      3        1.4573       0.5088        1.3801
2.0516
      4        1.3353       0.5480        1.2726
2.0518
      5        1.2342       0.5860        1.1688
2.0617
      6        1.1468       0.6158        1.0963
2.0561
      7        1.0718       0.6362        1.0431
2.0449
      8        1.0073       0.6504        0.9990
2.0351
      9        0.9512       0.6636        0.9614
2.0497
     10        0.9013       0.6760        0.9290
2.0321
     11        0.8565       0.6846        0.9053
2.0647
     12        0.8158       0.6907        0.8833
2.0046
     13        0.7780       0.6980        0.8638
2.0379
     14        0.7421       0.7061        0.8489
2.0325
     15        0.7077       0.7086        0.8370
2.0447
     16        0.6743       0.7126        0.8294
2.0551
     17        0.6419       0.7142        0.8287
2.0244
     18        0.6100       0.7141        0.8292  2.0451
     19        0.5787       0.7144        0.8338  2.0301
     20        0.5482       0.7154        0.8410  1.9690
     21        0.5178       0.7146        0.8501  2.0364
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        1.9412       0.4006        1.6886
5.8159
      2        1.5561       0.4832        1.4561
5.8241
      3        1.3814       0.5317        1.3202
5.7983
      4        1.2504       0.5825        1.1793
5.8271
      5        1.1435       0.6172        1.0891
5.7973
      6        1.0565       0.6367        1.0347
5.8330
      7        0.9844       0.6513        0.9924
5.8321
      8        0.9226       0.6649        0.9565
5.8528
      9        0.8678       0.6723        0.9307
5.8526
     10        0.8175       0.6816        0.9064
5.8205
     11        0.7708       0.6907        0.8843
5.8476
     12        0.7261       0.6977        0.8707
5.8333
     13        0.6835       0.7034        0.8592
5.8509
     14        0.6414       0.7077        0.8510
5.8561
     15        0.5999       0.7095        0.8512  5.8616
     16        0.5585       0.7137        0.8485
5.8777
     17        0.5169       0.7134        0.8533  5.8712
     18        0.4755       0.7154        0.8610  5.9445
     19        0.4344       0.7151        0.8835  5.8926
     20        0.3936       0.7166        0.9002  5.9293
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        1.6945       0.5278        1.3239
1.2576
      2        1.2017       0.6201        1.0778
1.2562
      3        0.9874       0.6588        0.9771
1.2651
      4        0.8541       0.6843        0.9079
1.2584
```

```
      5        0.7477       0.6944        0.9007
1.2542
      6        0.6615       0.6895        0.9414  1.2701
      7        0.5976       0.6989        0.9709  1.2705
      8        0.5369       0.7013        0.9904  1.2862
      9        0.4870       0.6831        1.1277  1.2548
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        1.6353       0.5433        1.2842
2.0251
      2        1.1358       0.6521        0.9950
2.0334
      3        0.9150       0.6876        0.9009
2.0270
      4        0.7748       0.7010        0.8741
1.9882
      5        0.6588       0.7116        0.8673
2.0708
      6        0.5528       0.7065        0.9097  2.0375
      7        0.4568       0.6892        1.0338  1.9990
      8        0.3879       0.6826        1.1266  2.0244
      9        0.3369       0.6901        1.2241  2.0188
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        1.6198       0.5535        1.2718
5.9230
      2        1.0995       0.6567        0.9795
5.8892
      3        0.8872       0.6842        0.8936
5.8864
      4        0.7348       0.7061        0.8603
5.8774
      5        0.6078       0.6981        0.9368  5.8177
      6        0.4952       0.7072        0.9728  5.8441
      7        0.3956       0.6971        1.1146  5.8306
      8        0.3316       0.7080        1.1339  5.8365
Stopping since valid_loss has not improved in the last 5 epochs.
```

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

#channel for each layer	validation accuracy
128	0.7013
256	0.7154
512	0.7166

#channel for each layer	validation accuracy

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

[14]:
```
# implement the same input normalization & type cast here
train_data_normalized = torch.Tensor(train.data / 255).permute(0, 3, 1, 2)
test_data_normalized = torch.Tensor(test.data/255)
test_data_normalized = test_data_normalized.permute(0,3,1,2)

optimizer = torch.optim.SGD
cnn = CNN_MaxPool(512)

model = skorch.NeuralNetClassifier(cnn, criterion=torch.nn.CrossEntropyLoss,
                                   device="cuda",
                                   optimizer=optimizer,
                                   optimizer__momentum=0.90,
                                   lr=1e-3,
                                   max_epochs=1000,
                                   batch_size=32,
                                   callbacks=[skorch.callbacks.
  ↪EarlyStopping(lower_is_better=True)])
model.fit(train_data_normalized, np.asarray(train.targets))

test.predictions = model.predict(test_data_normalized)
sklearn.metrics.accuracy_score(test.targets, test.predictions)
```

```
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        1.9479       0.3974        1.6958
5.8922
      2        1.5578       0.4850        1.4522
5.8591
      3        1.3842       0.5254        1.3217
5.8517
      4        1.2565       0.5781        1.1911
5.8742
      5        1.1525       0.6112        1.1068
5.8812
      6        1.0659       0.6318        1.0428
5.9010
      7        0.9925       0.6515        0.9954
5.8533
      8        0.9288       0.6682        0.9521
5.8652
      9        0.8725       0.6780        0.9187
5.8729
     10        0.8217       0.6885        0.8927
```

```
5.8721
      11         0.7745       0.6961        0.8742
5.8528
      12         0.7296       0.6975        0.8604
5.8843
      13         0.6868       0.7001        0.8505
5.8754
      14         0.6453       0.7029        0.8476
5.8646
      15         0.6037       0.7040        0.8464
5.8845
      16         0.5628       0.7057        0.8526  5.8861
      17         0.5219       0.7060        0.8667  5.8865
      18         0.4814       0.7047        0.8870  5.8785
      19         0.4410       0.7042        0.9110  5.8638
Stopping since valid_loss has not improved in the last 5 epochs.
```

[14]: 0.704

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. Adding pooling layers can improve computational speed and model accuracy.
2. Too high a learning rate may lead to not reaching the global optimum, but constantly oscillating.
3. The SGD optimizer is more perfect than the Adam optimizer.
4. If there are too few convolutional layers, it will lead to incomplete feature extraction; too many convolutional layers, on the other hand, can lead to worse results.