Introduction to Machine Learning: Homework V

Due on Dec 20th, 2022 at 11:59pm

Professor Ziping Zhao

Bingnan Li
2020533092

1. [Deep Learning Models]

   (a) Consider a 3D convolution layer. Suppose the input size is $32 \times 32 \times 3$ (width, height, depth) and we use ten 5×5 (width, height) kernels to convolve with it. Set stride = 1 and pad = 2. What is the output size? Let the bias for each kernel be a scalar, how many parameters do we have in this layer?

   Solution:

   The input data shape is

$$[C_{in}, D_{in}, H_{in}, W_{in}] = [1, 3, 32, 32]$$

   and the kernel size is

$$[C_{out}, D_{kernel}, H_{kernel}, w_{kernel}] = [10, 3, 5, 5]$$

   Then by the formula, we have output data with a shape

$$[C_{out}, D_{out}, H_{out}, W_{out}]$$

   and

$$\begin{aligned} D_{out} &= \frac{D_{in} + 2 \times pad - (D_{kernel} - 1) - 1}{stride} + 1 = 1 \\ H_{out} &= \frac{H_{in} + 2 \times pad - (H_{kernel} - 1) - 1}{stride} + 1 = 32 \\ W_{out} &= \frac{W_{in} + 2 \times pad - (W_{kernel} - 1) - 1}{stride} + 1 = 32 \end{aligned}$$

   Thus, the output size is $[C_{out}, D_{out}, H_{out}, W_{out}] = [10, 1, 32, 32]$
   Moreover, the total number of parameters is

$$\#\{parameters\} = \#\{parameters\ in\ kernel\} + \#\{biases\} = 5 \times 5 \times 3 \times 10 + 10 = 760$$

   (b) The convolution layer is followed by a max pooling layer with 2×2 (width, height) filter and stride = 2. What is the output size of the pooling layer? How many parameters do we have in the pooling layer?

   Solution:

   The input data shape is

$$[C_{in}, D_{in}, H_{in}, W_{in}] = [10, 1, 32, 32]$$

   and the kernel shape is

$$[C_{out}, D_{kernel}, H_{kernel}, W_{kernel}] = [10, 1, 2, 2]$$

   Then by the formula, we have output data with a shape

$$[C_{out}, D_{out}, H_{out}, W_{out}]$$

   and

$$\begin{aligned} D_{out} &= \frac{D_{in} + 2 \times pad - (D_{kernel} - 1) - 1}{stride} + 1 = 1 \\ H_{out} &= \frac{H_{in} + 2 \times pad - (H_{kernel} - 1) - 1}{stride} + 1 = 16 \\ W_{out} &= \frac{W_{in} + 2 \times pad - (W_{kernel} - 1) - 1}{stride} + 1 = 16 \end{aligned}$$

Thus, the output size is $[C_{out}, D_{out}, H_{out}, W_{out}] = [10, 1, 16, 16]$.

Moreover, given that the max pooling layer only performs maximizing, there are no parameters in the pooling layer. Thus, the total number of parameters of the pooling layer is 0.

2. [Deep Learning Models]

(a) (i.) When $r = 1$, $\boldsymbol{W}$ is exactly the eigenvector of $\boldsymbol{X}\boldsymbol{X}^T$ corresponding to its largest eigenvalue.

Proof:

When $r = 1$, we know that $\boldsymbol{W}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}$ is a scalar, which means

$$Tr(\boldsymbol{W}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}) = \boldsymbol{W}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}$$

Hence, the optimization problem can be rewritten as follows:

$$\underset{\boldsymbol{W}}{\text{maximize}} \quad \boldsymbol{W}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W} \quad s.t.\ \boldsymbol{W}^T\boldsymbol{W} = 1$$

Then, the Lagrangian is:

$$\mathcal{L}(\boldsymbol{W}, \lambda) = \boldsymbol{W}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W} - \lambda\boldsymbol{W}^T\boldsymbol{W}$$

Taking the derivative of $\mathcal{L}$ w.r.t $\boldsymbol{W}$ and setting it to 0, we get

$$2\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}^\star - 2\lambda^\star\boldsymbol{W}^\star = 0$$
$$\Rightarrow$$
$$\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}^\star = \lambda^\star\boldsymbol{W}^\star$$

Thus, $\lambda^\star$ is eigenvalue of $\boldsymbol{X}\boldsymbol{X}^T$ and $\boldsymbol{W}^\star$ is the corresponding eigenvector. Moreover, since

$$\boldsymbol{W}^{\star T}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}^\star = \lambda^\star$$

Therefore, we need to get the largest eigenvalue $\lambda^\star$ to maximize $\boldsymbol{W}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}$, which means $\boldsymbol{W}^\star$ is the eigenvector corresponding to the largest eigenvalue.

(ii.) $\boldsymbol{W}$ is also the solution to

$$\underset{\boldsymbol{W}}{\text{minimize}} \quad ||\boldsymbol{X} - \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{X}||_F^2 \quad s.t. \quad \boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}_r$$

Proof:

Given that $|\boldsymbol{P}|_F = \sqrt{Tr(\boldsymbol{P}^T\boldsymbol{P})}$, we get

$$\begin{aligned}
||\boldsymbol{X} - \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{X}||_F^2 &= Tr[(\boldsymbol{X} - \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{X})^T(\boldsymbol{X} - \boldsymbol{W}\boldsymbol{W}^T\boldsymbol{X})] \\
&= Tr[\boldsymbol{X}^T\boldsymbol{X} - \boldsymbol{X}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{X} - \boldsymbol{X}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{X} + \boldsymbol{X}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{X}] \\
&= Tr[\boldsymbol{X}^T\boldsymbol{X} - \boldsymbol{X}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{X}] \\
&= Tr(\boldsymbol{X}^T\boldsymbol{X}) - Tr(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{W}^T\boldsymbol{X}) \\
&= Tr(\boldsymbol{X}^T\boldsymbol{X}) - Tr(\boldsymbol{W}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W})
\end{aligned}$$

Since the first term of the equation has nothing to do with $\boldsymbol{W}$, the optimization problem can be rewritten as follows:

$$\underset{\boldsymbol{W}}{\text{minimize}} \quad -Tr(\boldsymbol{W}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{W}) \quad s.t.\ \boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}_r$$

which is equivalent to the PCA problem.

(b) (i.) $\boldsymbol{A}_2^\star$ can be solved from:

$$\underset{\boldsymbol{A}_2}{\text{minimize}} \quad ||\boldsymbol{X} - \boldsymbol{A}_2\boldsymbol{A}_2^\dagger\boldsymbol{X}||_F^2$$

Proof:
First, we consider the original problem:

$$||\boldsymbol{X} - \hat{\boldsymbol{X}}||_F^2 = ||\boldsymbol{X} - \boldsymbol{A}_2\boldsymbol{H} - \boldsymbol{b}_2\boldsymbol{1}^T||_F^2$$

By setting the derivative w.r.t. $\boldsymbol{b}_2$ to 0, we get

$$\boldsymbol{b}_2 = \frac{1}{M}(\boldsymbol{X} - \boldsymbol{A}_2\boldsymbol{H})\boldsymbol{1}$$

Then, insert the result back into the original problem and we get

$$\begin{aligned}
||\boldsymbol{X} - \boldsymbol{A}_2\boldsymbol{H} - \boldsymbol{b}_2\boldsymbol{1}^T||_F^2 &= ||\boldsymbol{X} - \boldsymbol{A}_2\boldsymbol{H} - \frac{1}{M}(\boldsymbol{X} - \boldsymbol{A}_2\boldsymbol{H})\boldsymbol{1}\boldsymbol{1}^T||_F^2 \\
&= ||\boldsymbol{X}(\boldsymbol{I} - \frac{\boldsymbol{1}\boldsymbol{1}^T}{M}) - \boldsymbol{A}_2\boldsymbol{H}(\boldsymbol{I} - \frac{\boldsymbol{1}\boldsymbol{1}^T}{M})||_F^2 \\
&\equiv ||\boldsymbol{X'} - \boldsymbol{A}_2\boldsymbol{H'}||_F^2 \\
&= ||\boldsymbol{X'} - \boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{X'} - \boldsymbol{A}_2\boldsymbol{b}_1\boldsymbol{1}^T\left(\boldsymbol{I} - \frac{\boldsymbol{1}\boldsymbol{1}^T}{M}\right)||_F^2 \\
&= ||\boldsymbol{X'} - \boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{X'} - \boldsymbol{A}_2\boldsymbol{b}_1\left(\boldsymbol{1}^T - \frac{\boldsymbol{1}^T\boldsymbol{1}\boldsymbol{1}^T}{M}\right)||_F^2 \\
&= ||\boldsymbol{X'} - \boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{X'} - \boldsymbol{A}_2\boldsymbol{b}_1\left(\boldsymbol{1}^T - \frac{M\boldsymbol{1}^T}{M}\right)||_F^2 \\
&= ||\boldsymbol{X'} - \boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{X'}||_F^2
\end{aligned}$$

Thus, the original optimization problem is equivalent to the following problem:

$$\underset{\boldsymbol{A}_1, \boldsymbol{A}_2}{\text{minimize}} \quad ||\boldsymbol{X'} - \boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{X'}||_F^2$$

and is also equivalent to

$$\underset{\boldsymbol{A}_1, \boldsymbol{A}_2}{\text{minimize}} \quad ||\boldsymbol{X} - \boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{X}||_F^2$$

Given that $\boldsymbol{A}^\dagger = \arg\min_{\boldsymbol{A}_1} ||\boldsymbol{X} - \boldsymbol{A}_2\boldsymbol{A}_1\boldsymbol{X}||_F^2$, we know that the problem can be further converted as follows:

$$\underset{\boldsymbol{A}_2}{\text{minimize}} \quad ||\boldsymbol{X} - \boldsymbol{A}_2\boldsymbol{A}_2^\dagger\boldsymbol{X}||_F^2$$

(ii.) The solution $\boldsymbol{W}$ from (a) can be taken as the same as $\boldsymbol{A}_2^\star$.

3. [Ensemble Learning] Suppose there are L independent two-class classifiers used for simple voting and the output of classifier j ($j = 1 \cdots L$) is denoted as d_j. From the point of view that the mean squared error of an estimator can be decomposed into the bias part and the variance part, explain why increasing L can lead to an increase in classification accuracy.
Proof:

By the definition of MSE, we get

$$\begin{aligned}
MSE(\hat{y}) &= \mathbb{E}[(\hat{y}-y)^2] = \mathbb{E}[(\hat{y}-\mathbb{E}[\hat{y}]+\mathbb{E}[\hat{y}]-y)^2] \\
&= \mathbb{E}[(\hat{y}-\mathbb{E}[\hat{y}])^2 + 2(\hat{y}-\mathbb{E}[\hat{y}])(\mathbb{E}[\hat{y}]-y) + (\mathbb{E}[\hat{y}]-y)^2] \\
&= \mathbb{E}[(\hat{y}-\mathbb{E}[\hat{y}])^2] + 2\mathbb{E}[(\hat{y}-\mathbb{E}[\hat{y}])]\mathbb{E}[(\mathbb{E}[\hat{y}]-y)] + \mathbb{E}[(\mathbb{E}[\hat{y}]-y)^2] \\
&= \mathbb{E}[(\hat{y}-\mathbb{E}[\hat{y}])^2] + 2(\mathbb{E}[\hat{y}]-\mathbb{E}[\hat{y}])\mathbb{E}[(\mathbb{E}[\hat{y}]-y)] + (\mathbb{E}[\hat{y}]-y)^2 \\
&= \mathbb{E}[(\hat{y}-\mathbb{E}[\hat{y}])^2] + (\mathbb{E}[\hat{y}]-y)^2 \\
&= \mathrm{Var}(\hat{y}) + \mathrm{Bias}^2(\hat{y}, y)
\end{aligned}$$

Then, for the second term of the equation above, we get

$$\begin{aligned}
\mathrm{Bias}^2(\hat{y}, y) &\propto \mathrm{Bias}(\hat{y}, y) \\
&= \mathbb{E}[\hat{y}] - y \\
&\propto \mathbb{E}[\hat{y}] \\
&= \mathbb{E}\left[\frac{1}{L}\sum_j d_j\right] \\
&\geq \frac{1}{L} \times L \min_j\{\mathbb{E}[d_j]\} \\
&= \min_j\{\mathbb{E}[d_j]\}
\end{aligned}$$

which means that the Bias term won't change as L gets larger.

As for the first term, we get

$$\begin{aligned}
\mathrm{Var}(\hat{y}) &= \mathrm{Var}\left(\frac{1}{L}\sum_j d_j\right) \\
&= \frac{1}{L^2}\mathrm{Var}\left(\sum_j d_j\right) \\
&\leq \frac{1}{L^2} \times L \max_j\left(\mathrm{Var}(d_j)\right) \\
&= \frac{1}{L}\max_j\{\mathrm{Var}(d_j)\}
\end{aligned}$$

which means that the Variance term will get smaller when L gets larger.

In conclusion, $MSE(\hat{y})$ will get smaller as L gets larger, so the classification will be more accurate.

4. [Model Assessment and Selection] Suppose we carry out a K-fold cross-validation on a dataset and obtain the classification error rates $\{p_i\}_{i=1}^K$, describe the steps of a one-sided t test on testing the null hypothesis H_0 that the classifier has error percentage p_0 or less at a significance level α.
Solution:

Assume that $\{p_i\}_{i=1}^K$ follow normal distribution and define

$$q_i = p_i - p_0$$

where q_i also follows the normal distribution but with mean μ. After that, we define the null hypothesis and alternative hypothesis as follows:

$$\begin{aligned}
H_0 &: \ \mu \leq \mu_0 = 0 \\
H_1 &: \ \mu > \mu_0 = 0
\end{aligned}$$

Then, under the null hypothesis, we have

$$\sqrt{K}\frac{m-\mu_0}{S} = \frac{\sqrt{K}m}{S} \sim \tau_{K-1}$$

where $m = \frac{\sum_{i=1}^{K} q_i}{K}$ and $S^2 = \frac{\sum_{i=1}^{K}(q_i-m)^2}{K-1}$.

Then, we failed to reject H_0 at significant level α if

$$\frac{\sqrt{K}m}{S} \in (-\infty, t_{\alpha,K-1})$$

HW5-Coding

December 18, 2022

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 numpy as np
import sklearn
import skorch
import torch
import torch.nn as nn
import torchvision
from matplotlib import pyplot as plt
```

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]:
```python
# 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(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)

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

# [TO DO]
# Then expand the new tensor to the target tensor by using torch.Tensor.expand.
z = torch.Tensor.expand(x, 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]: # 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, 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, 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, _ = torch.max(x, 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, _ = torch.min(x, 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_mean_xtop, _ = torch.topk(x, 5, 1)
print(x_mean_xtop.shape)
```

```
tensor(0.0931)
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]: 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.conv_1 = torch.nn.Conv2d(in_channels=3, out_channels=channels,␣
 ↪kernel_size=3, padding=1)
```

```
        self.conv_2 = torch.nn.Conv2d(in_channels=channels,␣
↪out_channels=channels, kernel_size=3, padding=1)
        self.conv_3 = torch.nn.Conv2d(in_channels=channels,␣
↪out_channels=channels, kernel_size=3, padding=1)
        self.relu = torch.nn.ReLU()
        self.linear = torch.nn.Linear(in_features=32 * 32 * channels,␣
↪out_features=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)*****
        out = self.relu(self.conv_1(images))
        out = self.relu(self.conv_2(out))
        out = self.relu(self.conv_3(out))
        out = torch.nn.Flatten(1, -1)(out)
        out = self.linear(out)

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

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 here

learning_rate = 1e-4
optimize =  torch.optim.Adam
channel = 64

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

print(f'The channel was {channel}, the learning rate was {learning_rate} and the␣
↪optimizer was {str(optimize)}')

cnn = CNN(channels=channel)
model = skorch.NeuralNetClassifier(cnn, criterion=torch.nn.CrossEntropyLoss,
                                   device="cuda",
                                   optimizer=optimize,
                                   # optimizer__momentum=0.90,
                                   lr=learning_rate,
                                   max_epochs=15,
                                   batch_size=64,
                                   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 64, the learning rate was 0.0001 and the optimizer was <class
'torch.optim.adam.Adam'>
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  -------
      1        1.6819       0.5020        1.4174
10.9766
      2        1.3244       0.5650        1.2413
9.9119
      3        1.1825       0.5860        1.1763
9.9063
      4        1.0709       0.6101        1.1131
9.9199
      5        0.9765       0.6264        1.0658
9.9347
      6        0.8994       0.6411        1.0356
10.7903
      7        0.8355       0.6520        1.0182
9.8990
      8        0.7797       0.6573        1.0091
9.8756
      9        0.7293       0.6609        1.0072
9.9047
     10        0.6823       0.6622        1.0101  10.0021
     11        0.6373       0.6630        1.0206  9.8481
     12        0.5937       0.6618        1.0378  9.8932
     13        0.5516       0.6587        1.0656  9.8847
Stopping since valid_loss has not improved in the last 5 epochs.
```

[10]:

```
<class 'skorch.classifier.NeuralNetClassifier'>[initialized](
  module_=CNN(
    (conv_1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv_2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv_3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (relu): ReLU()
    (linear): Linear(in_features=65536, out_features=10, bias=True)
  ),
)
```

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	Adam
(64, 64, 64)		0.6630

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.

a) Implement max pooling layers Copy 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)*****
        assert len(channels) == 1 or len(
            channels) == 3, f"invalid channels number, expect 1 or 3 but get␣
↪{len(channels)} instead."
        if len(channels) == 1:
            CHANNELS = channels * 3
        else:
            CHANNELS = channels
        self.conv_1 = torch.nn.Conv2d(in_channels=3, out_channels=CHANNELS[0],␣
↪kernel_size=3, padding=1)
        self.conv_2 = torch.nn.Conv2d(in_channels=CHANNELS[0],␣
↪out_channels=CHANNELS[1], kernel_size=3, padding=1)
        self.conv_3 = torch.nn.Conv2d(in_channels=CHANNELS[1],␣
↪out_channels=CHANNELS[2], kernel_size=3, padding=1)
        self.relu = torch.nn.ReLU()
        self.max_pool = torch.nn.MaxPool2d(kernel_size=2, stride=2)
        self.linear = torch.nn.Linear(in_features=4 * 4 * CHANNELS[2],␣
↪out_features=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)*****
        out = self.relu(self.max_pool(self.conv_1(images)))
        out = self.relu(self.max_pool(self.conv_2(out)))
        out = self.relu(self.max_pool(self.conv_3(out)))
        out = torch.nn.Flatten(1, -1)(out)
        out = self.linear(out)
        # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
        return out
```

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

[12]:
```
channel = 64
train_data_normalized = torch.Tensor(train.data / 255)
train_data_normalized = train_data_normalized.permute(0, 3, 1, 2)
cnn_max_pool_same_channels = CNN_MaxPool(channels=[channel])
# cnn_max_pool_diff_channels = CNN_MaxPool(channels=[c, 2 * c, 4 * c])
model = skorch.NeuralNetClassifier(cnn_max_pool_same_channels, criterion=torch.
 ↪nn.CrossEntropyLoss,
                                                                   device="cuda",
                                                                   optimizer=torch.optim.Adam,
                                                                   lr=0.0001,
                                                                   max_epochs=25,
                                                                   batch_size=64,
                                                                   callbacks=[skorch.callbacks.
 ↪EarlyStopping(lower_is_better=True)], )
# implement input normalization & type cast here
model.fit(train_data_normalized, np.asarray(train.targets))
```

```
  epoch    train_loss    valid_acc    valid_loss     dur
-------  ------------  -----------  ------------  ------
      1        1.9269       0.4119        1.6712
3.4668
      2        1.6067       0.4523        1.5397
3.5056
      3        1.4981       0.4770        1.4602
3.6588
      4        1.4211       0.5048        1.3998
3.5595
      5        1.3675       0.5217        1.3588
4.0562
      6        1.3265       0.5365        1.3244
3.6377
      7        1.2920       0.5464        1.2912
3.7853
      8        1.2612       0.5584        1.2617
3.8943
      9        1.2331       0.5683        1.2352
3.5465
     10        1.2072       0.5769        1.2106
3.6510
     11        1.1830       0.5853        1.1882
3.5429
     12        1.1600       0.5929        1.1679
3.4975
     13        1.1383       0.5988        1.1495
3.4832
     14        1.1177       0.6024        1.1318
3.4758
     15        1.0980       0.6076        1.1161
```

```
3.4716
     16        1.0791       0.6131        1.1012
3.4695
     17        1.0612       0.6193        1.0876
3.4775
     18        1.0441       0.6215        1.0742
4.2113
     19        1.0279       0.6265        1.0615
4.5485
     20        1.0126       0.6287        1.0497
3.6776
     21        0.9979       0.6339        1.0389
4.7526
     22        0.9839       0.6370        1.0286
4.3440
     23        0.9706       0.6396        1.0188
4.2747
     24        0.9578       0.6434        1.0091
4.2250
     25        0.9456       0.6470        1.0014
4.2377
```

[12]:
```
<class 'skorch.classifier.NeuralNetClassifier'>[initialized](
  module_=CNN_MaxPool(
    (conv_1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv_2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv_3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (relu): ReLU()
    (max_pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (linear): Linear(in_features=1024, out_features=10, bias=True)
  ),
)
```

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

#channel for each layer	validation accuracy
(64, 64, 64)	0.6470

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

[13]:
```
# implement the same input normalization & type cast here
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)
```

[13]: 0.654

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: 0.654