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.

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 (https://github.com/jcjohnson/pytorch-examples) for PyTorch.

You can also find the detailed API doc (http://pytorch.org/docs/stable/index.html) here. If you have other questions that are not addressed by the API docs, the PyTorch forum (https://discuss.pytorch.org/) is a much better place to ask than StackOverflow.

Install PyTorch and Skorch.

```
In [2]:  !pip install -q torch skorch torchvision torchtext
```

```
In [1]: 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
```

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) 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 (https://pytorch.org/docs/stable/torch.html#torch.squeeze), torch.unsqueeze (https://pytorch.org/docs/stable/torch.html#torch.unsqueeze) and torch.Tensor.view (https://pytorch.org/docs/stable/tensors.html#torch.Tensor.view). Please read the documents of the functions, and finish the following practice.

```
In [2]: # 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 siz
        x = torch.unsqueeze(torch.unsqueeze(x, 1), 3)
        print(x.shape)
        # Remove the two dimensions justed added by using the function torch.squeeze, and ch
        x = torch.squeeze(x)
        print(x.shape)
        # x is now a two-dimensional tensor, or in other words a matrix. Now use the functio
        x = x.view(6)
        print(x.shape)
```

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

2) Tensor concatenation and stack

Tensor concatenation and stack are operations to combine small tensors into big tensors. The corresponding functions are torch.cat (https://pytorch.org/docs/stable/torch.html#torch.cat) and torch.stack (https://pytorch.org/docs/stable/torch.html#torch.stack). Please read the documents of the functions, and finish the following practice.

In [3]:
```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,:

# Use torch.stack to generate such a z
z_stack = torch.stack([x, y], dim=0)
print("Using torch.stack:")
print(z_stack[0,:,:])
# Use torch.cat and torch.unsqueeze to generate such a z
z_cat = torch.cat([x.unsqueeze(0), y.unsqueeze(0)], dim=0)
print("Using torch.cat and torch.unsqueeze:")
print(z_cat[1,:,:])
```

```
Using torch.stack:
tensor([[1., 2.],
        [3., 4.],
        [5., 6.]])
Using torch.cat and torch.unsqueeze:
tensor([[-1., -2.],
        [-3., -4.],
        [-5., -6.]])
```

3) Tensor expansion

Tensor expansion is to expand a tensor into a larger tensor along singleton dimensions. The corresponding functions are torch.Tensor.expand (https://pytorch.org/docs/stable/tensors.html#torch.Tensor.expand) and torch.Tensor.expand_as (https://pytorch.org/docs/stable/tensors.html#torch.Tensor.expand_as). Please read the documents of the functions, and finish the following practice.

```
In [4]: # 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, :, :
        
        # [TO DO]
        # Change the size of x into (1, 3) by using torch.unsqueeze.
        pass
        x_expanded = x.unsqueeze(0)
        print(x_expanded.shape)

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

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

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 (https://pytorch.org/docs/stable/torch.html#torch.mean), torch.sum (https://pytorch.org/docs/stable/torch.html#torch.sum), torch.max (https://pytorch.org/docs/stable/torch.html#torch.max), torch.min (https://pytorch.org/docs/stable/torch.html#torch.min), torch.topk (https://pytorch.org/docs/stable/torch.html#torch.topk), and finish the following practice.

```
In [5]:  # 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 va
         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, _ = 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, _ = 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
         _, indices = torch.topk(x, k=5, dim=1, largest=True, sorted=True)
         x_mean_xtop = torch.gather(x, dim=1, index=indices)
         print((x_mean_xtop.shape))
```

```
tensor(-0.0340)
torch.Size([10])
torch.Size([10])
torch.Size([10])
torch.Size([10, 5])
```

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.

```
In [6]:  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.

```
In [7]: 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)
```


frog   truck   truck   deer   automobile

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.

```python
In [8]: 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)*****
        # 第一个卷积层
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=channels, kernel_size=3, padd
        self.relu1 = nn.ReLU()

        # 第二个卷积层
        self.conv2 = nn.Conv2d(in_channels=channels, out_channels=channels, kernel_size=
        self.relu2 = nn.ReLU()

        # 第三个卷积层
        self.conv3 = nn.Conv2d(in_channels=channels, out_channels=channels, kernel_size=
        self.relu3 = nn.ReLU()

        # 全连接层
        self.fc = 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)*****
        x = self.relu1(self.conv1(images))
        x = self.relu2(self.conv2(x))
        x = self.relu3(self.conv3(x))
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        images = x
        # *****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.

```
In [9]:  # implement hyperparameters, you can select and modify the hyperparameters by yourse

         optimize = [torch.optim.SGD, torch.optim.Adam]
         learning_rate = [1e-5, 1e-3, 1e-1]
         channel = [4, 8, 16, 32]


         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 {

               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=256,
                                           callbacks=[skorch.callbacks.EarlyStopping(lower_i
               # implement input normalization & type cast here
               model.fit(train_data_normalized, np.asarray(train.targets, dtype=np.int64))
```

```
The channel was 4, the learning rate was 1e-05 and the optimizer was <class 't
orch.optim.sgd.SGD'>
  epoch    train_loss    valid_acc    valid_loss    dur
-------  ------------  -----------  ------------  ------
      1        2.3031       0.1027        2.3031  1.8668
      2        2.3031       0.1026        2.3031  1.3916
      3        2.3031       0.1026        2.3031  1.3521
      4        2.3031       0.1027        2.3031  1.3282
      5        2.3030       0.1026        2.3031  1.2006
Stopping since valid_loss has not improved in the last 5 epochs.
The channel was 8, the learning rate was 1e-05 and the optimizer was <class 't
orch.optim.sgd.SGD'>
  epoch    train_loss    valid_acc    valid_loss    dur
-------  ------------  -----------  ------------  ------
      1        2.3033       0.0999        2.3033  1.5563
      2        2.3033       0.1001        2.3033  1.5063
      3        2.3033       0.1001        2.3033  1.4978
      4        2.3032       0.1000        2.3032  1.5326
      5        2.3032       0.1000        2.3032  1.5401
```

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
32	0.57	0.58
16	0.53	0.59
8	0.50	0.55

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.

```
In [10]: 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)*****
        # Convolutional layer 1 with max pooling
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=channels, kernel_size=3, padd
        self.relu1 = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Convolutional layer 2 with max pooling
        self.conv2 = nn.Conv2d(in_channels=channels, out_channels=channels, kernel_size=
        self.relu2 = nn.ReLU()
        self.maxpool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Convolutional layer 3 with max pooling
        self.conv3 = nn.Conv2d(in_channels=channels, out_channels=channels, kernel_size=
        self.relu3 = nn.ReLU()
        self.maxpool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Fully connected layer
        self.fc = nn.Linear(channels * 4 * 4, 10)  # Assuming input image size is (3, 32
        # *****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)*****
        x = self.maxpool1(self.relu1(self.conv1(images)))
        x = self.maxpool2(self.relu2(self.conv2(x)))
        x = self.maxpool3(self.relu3(self.conv3(x)))
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        images = x
        # *****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.

```
In [15]: # implement hyperparameters, you can select and modify the hyperparameters by yourse
         learning_rate = [1e-5, 1e-3]
         channel = [4, 8, 16, 32]
         # Select the better optimizer by the result shown in the previous problem, you can s
         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=1,
                                        max_epochs=50,
                                        batch_size=256,
                                        callbacks=[skorch.callbacks.EarlyStopping(lower_i
                 # implement input normalization & type cast here
                 model.fit(train_data_normalized, np.asarray(train.targets, dtype=np.int64))
```

```
The channel was 4, the learning rate was 1e-05
  epoch    train_loss    valid_acc    valid_loss      dur
-------  ------------  -----------  ------------  ------
      1        2.3051       0.1000        2.3049  1.9930
      2        2.3048       0.1000        2.3047  1.8580
      3        2.3046       0.1000        2.3044  2.2488
      4        2.3043       0.1000        2.3042  2.1639
      5        2.3040       0.1000        2.3038  2.1984
      6        2.3036       0.1000        2.3033  2.1384
      7        2.3031       0.1000        2.3028  1.4617
      8        2.3025       0.1000        2.3022  1.0359
      9        2.3019       0.1000        2.3016  1.0914
     10        2.3013       0.1000        2.3009  1.0241
     11        2.3006       0.1007        2.3003  0.9990
     12        2.2999       0.1026        2.2995  1.0059
     13        2.2991       0.1101        2.2987  1.0007
     14        2.2983       0.1205        2.2978  0.9751
     15        2.2974       0.1344        2.2968  1.0091
     16        2.2963       0.1412        2.2957  1.0143
     17        2.2952       0.1466        2.2945  1.0288
```

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

#channel for each layer	validation accuracy
32	0.69
16	0.63
8	0.57

#channel for each layer　validation accuracy

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

```
In [16]: # 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)
```

Out[16]: 0.6826

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: 0.683

as the channel rises, we can get a better accuracy. Also we should not make the learning rate too small since this will make the system change very little at each round, it might Stop since valid_loss has not improved in the last 5 epochs. And if we make the learning rate too big it might change too much.