**Homework 2 Instruction**

This assignment needs to be done in Palmetto. Details on how to log in to Palmetto and install PyTorch in Palmetto are available here <https://github.com/yongkaiwu/palemtto> .

Please make sure that PyTorch is installed correctly in Palmetto. The correct sign is when you log back into Palmetto and get the following screen:

Text

Description automatically generated

The assignment is divided into the following sections.

1. Download the CIFAR-10 dataset and preprocess the CIFAR-10 dataset using the tools in PyTorch (optional).

2. Define the network structure.

3. Define the loss function.

4. Define the optimization function.

5. Train the neural network model and obtain the accuracy of training and testing data.

**Section 1 Prepare Data and Perform Pre-processing**

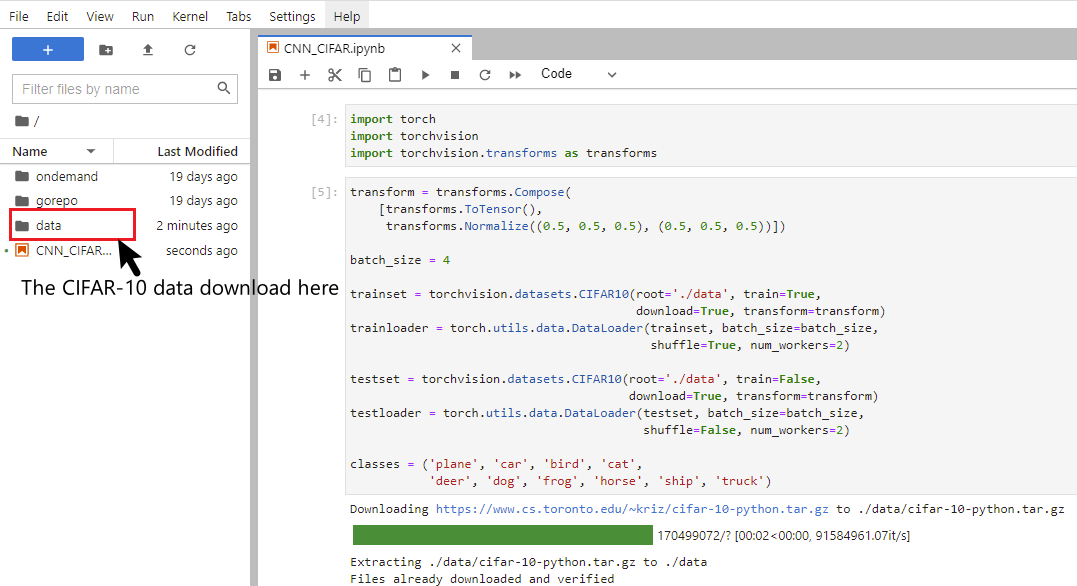
CIFAR-10 (Canadian Institute for Advanced Research) is a dataset collected by Alex Krizhevsky, Vinod Nair and Geoffrey Hinton for image recognition, 60,000 32\*32 color images, 50,000 training data, 10000 test data There are 10 categories, airplane, car, bird, cat, deer, dog, frog, horse, boat, truck, 6000 images in each category. Compared with MNIST, there are more color and color noise, different size, angle and color of objects in the same category, as shown in the following figure.

PyTorch has perfect handling of CIFAR-10 datasets, and we can use torchvision in torch, that has data loaders for common datasets such as ImageNet, CIFAR10, MNIST, etc. and data transformers for images, to load and preprocess CIFAR-10 datasets. The details can be found here <http://pytorch.org/vision/main/generated/torchvision.datasets.CIFAR10.html> .

A picture containing text, screenshot

Description automatically generated

In Palmetto, after we log into Jupyter Lab, we download and load data using the methods in torchvision and utils, and the result is shown in the figure below:



As you can see below, we can print some of the images and the corresponding labels. I would like to emphasize that the Batch size here is 4 (you can define the size of the Batch\_size), the size of each Image is (32\*32), and the input channel of each Image is 3 (R, G, B). These details will be used later in the definition of the network model.

Graphical user interface, application

Description automatically generated

**Section 2 Defining the Network Structure**

In general, after we load data, we use PyTorch to learn the network model using the following steps:

Diagram

Description automatically generated

In defining the network structure, we will likely use the following methods:

(1). Containers (such as nn.Module(), nn.Sequential()…)

(2). Convolution Layers (such as nn.Conv1d, nn.Conv2d …)

(3). Pooling layers (such as nn.MaxPool2d, nn.AvgPool2d…)

(4). Padding Layers

(5). Non-linear Activations (such as nn.ReLu(), nn.Tanh…)

(6). Normalization Layers (such as nn.BatchNorm1d, nn.BatchNorm2d…)

The details are here <https://pytorch.org/docs/stable/nn.html> .

These methods will be learned by yourselves and used in this assignment. Here I will build a simple network model of MINIST image classification to help you understand how these methods are used.

class NeuralNetwork(nn.Module):

# Create a network container using nn.Module()

def \_\_init\_\_(self):

super(NeuralNetwork, self).\_\_init\_\_()

# note: for each neural network, the size of the output of this layer

# should be exactly the same as the size of the input of next layer

self.model = nn.Sequential(

# use nn.Conv2d to insert the first convolutional layer

# the input channel =1, output channel = 6,

# kernel size =3, stride =1

# Note: if input x = [28,28], after the first convol layer

# the output will be [6, 26,26].

# rounding down((x.size - kernel\_size + 2\* padding)/2)+1

# here we don't have the padding,

# rounding down((28 - 3 + 2\* 0)/2)+1 = 26

nn.Conv2d(1, 6, kernel\_size=3, stride = 1),

# the number in the BatchNorm should be the same as

# the number of the output channel

nn.BatchNorm2d(6),

# here we have (6, 26, 26)

# pooling with the height and width reduced by half each

# after maxpooling, we have (6, 13, 13)

nn.MaxPool2d(kernel\_size=2, stride =2),

# the activation function doesn't change the size of channel

nn.ReLU(),

# here, the input is (6, 13, 13) and insert it into

# the second convolutional layer

# the second convol layer requires the input channel = 6,

# output channel = 16, and kernel\_size=3, stride =1

# insert this (6, 13, 13) input and get (16, 11, 11) output

# rounding down((13-3 + 2\*0)/1)+1 = 11

nn.Conv2d(6, 16, kernel\_size=3, stride =1),

nn.BatchNorm2d(16),

# here we have (16, 11, 11)

# pooling with the height and width reduced by half each

# after maxpooling, we have (16, 5, 5)

nn.MaxPool2d(kernel\_size=2, stride=2),

# the activation function

nn.ReLU(),

# flattening layers to facilitate fully connected layer input

nn.Flatten(),

# here we have the input (16, 5, 5), after flattening,

# we have 16\*5\*5 = 400 (the size of input)

nn.Linear(400,120)

nn.ReLU(),

nn.Linear(120, 84),

nn.ReLU(),

# The MNIST data is 10- categories of questions

# the final output should be 10.

nn.Linear(84,10)

)

def forward(self, x):

x = self.model(x)

return x

Note that the selection of parameters can be rather challenge. In PyTorch, the nn.Conv2d function mainly uses the following formula to accept parameters <https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html> .

Graphical user interface, text, application

Description automatically generated

**Section 3: Defining the loss function**

Since this is an image classification problem, it is important to choose a suitable loss function. PyTorch provides a number of tools for calculating the loss function: <https://pytorch.org/docs/stable/nn.html#loss-functions> .

Here I briefly describe the use of nn.CrossEntropyLoss().

<https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>

# Example of target with class probabilities

loss = nn.CrossEntropyLoss()

input = torch.randn(3, 5, requires\_grad=True)

target = torch.randn(3, 5).softmax(dim=1)

output = loss(input, target)

output.backward()

**Section 4: Define optimization functions**

The torch.optim is a package that implements various optimization algorithms. Most of the common methods are already supported and the interface is generic enough so that more complex methods can be easily integrated in the future as well <https://pytorch.org/docs/stable/optim.html> . As an example,

optimizer = optim.SGD(model.parameters(), lr=0.01)

Here the optimizer uses the SGD algorithm, and the learning rate is set to 0.01 after the model parameters are put in.

When combining with the loss function of the model, the initial gradient of each epoch is guaranteed to be 0. After the loss function backward, the step() method should be used for the gradient of the optimizer.

for input, target in dataset:

optimizer.zero\_grad()

output = model(input)

loss = loss\_fn(output, target)

loss.backward()

optimizer.step()

**Section5: Train the neural network model and get the accuracy of training and testing.**

Follow the python file hints.