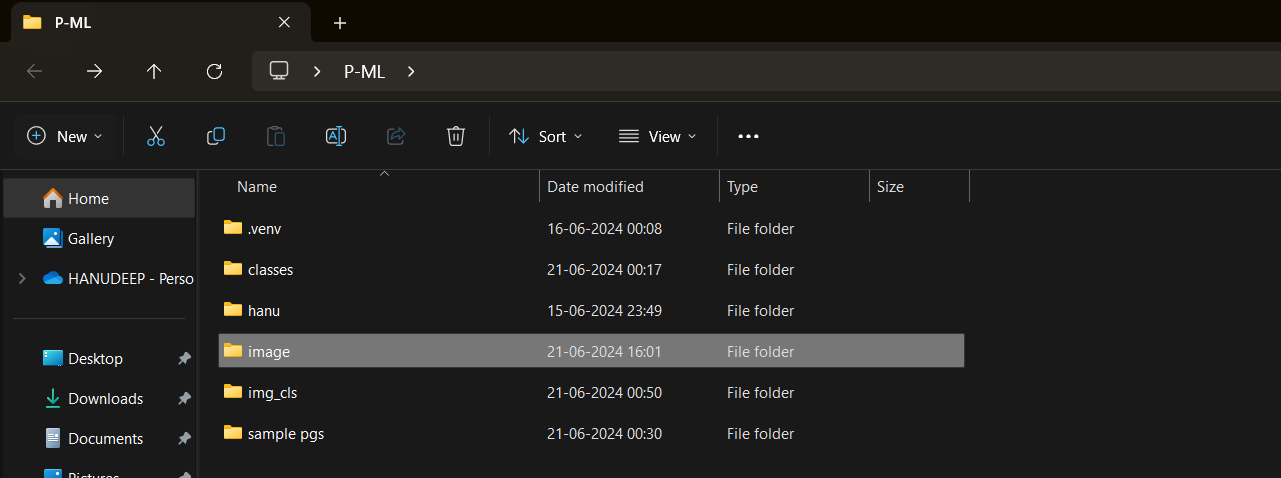
Image Classification and Detection using CIFAR10 dataset:

\*Create a Destination Folder to save datasets and program that to be executed

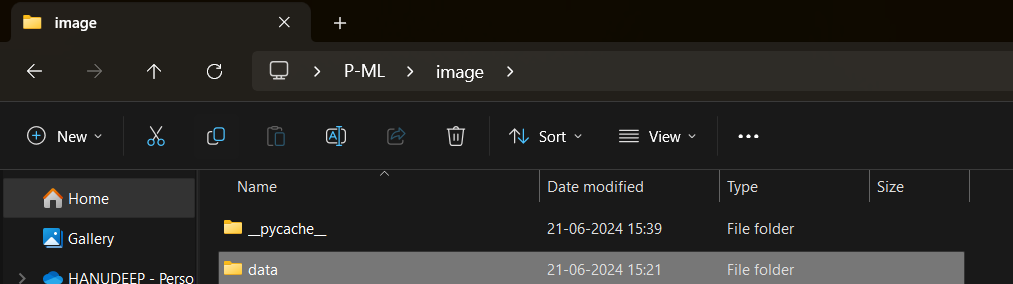


For the classification and detection we use image folder

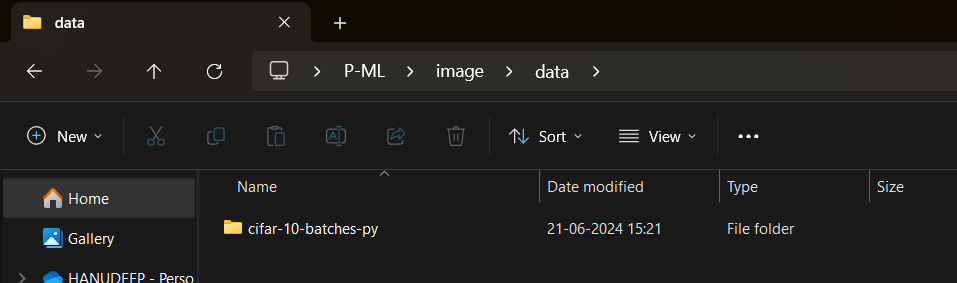
Now, Download dataset from google

https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

After downloading the dataset create a folder named data in images folder



In that copy the dataset to data folder and extract the zip file there..

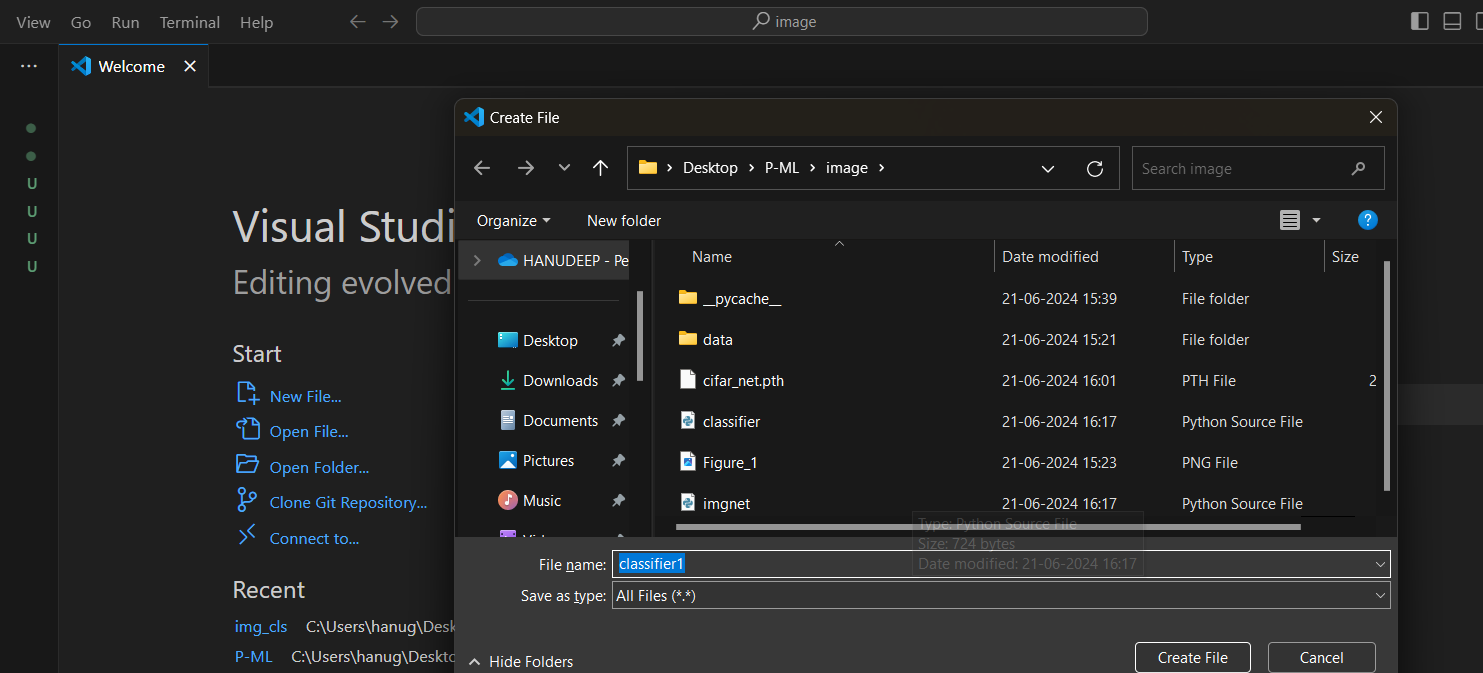


After Extraction delete the zip file…

………………………………………………………………………………………………………………………………………………………………………………..

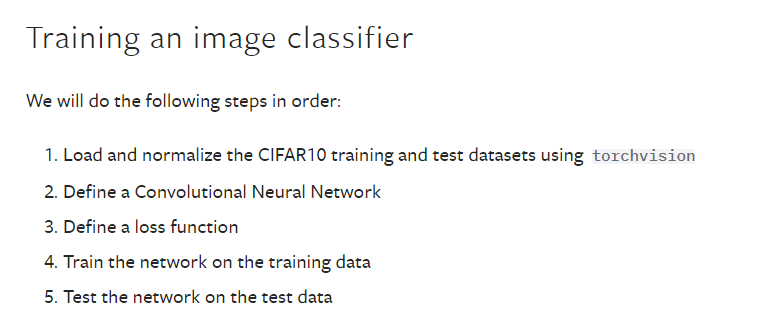
Now open vs code and click on new file and set the name of the file as classifier.py navigate to folder that u created for dataset and pgms

Here ie. Image folder



Then create ur file….

The total process we are going to perform here is



So first one..

1. Load and normalize the CIFAR10 training and test datasets…

For that required packages….

import torch

import torch.nn as nn

import torchvision

import torchvision.transforms as transforms

and to link the the datasets we use the following code…



Now ,



Up to this the code identifies the and displays the image names… let us combine and run the code in vs

The Final code for simple detection of images..

import torch

import torch.nn as nn

import torchvision

import torchvision.transforms as transforms

import matplotlib.pyplot as plt

import torch.optim as optim

import numpy as np

transform = transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10('./data', train=True,download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch\_size=4,shuffle=True, num\_workers=0)

testset = torchvision.datasets.CIFAR10('./data', train=False,download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch\_size=4,shuffle=True, num\_workers=0)

classes = ('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# functions to show an image

def imshow(img):

    img = img / 2 + 0.5     # unnormalize

    npimg = img.numpy()

    plt.imshow(np.transpose(npimg, (1, 2, 0)))

    plt.show()

# get some random training images

dataiter = iter(trainloader)

images, labels = next(dataiter)

# show images

imshow(torchvision.utils.make\_grid(images))

print(' '.join('%5s' % classes[labels[j]] for j in range(4)))

Save the file in image folder and run it

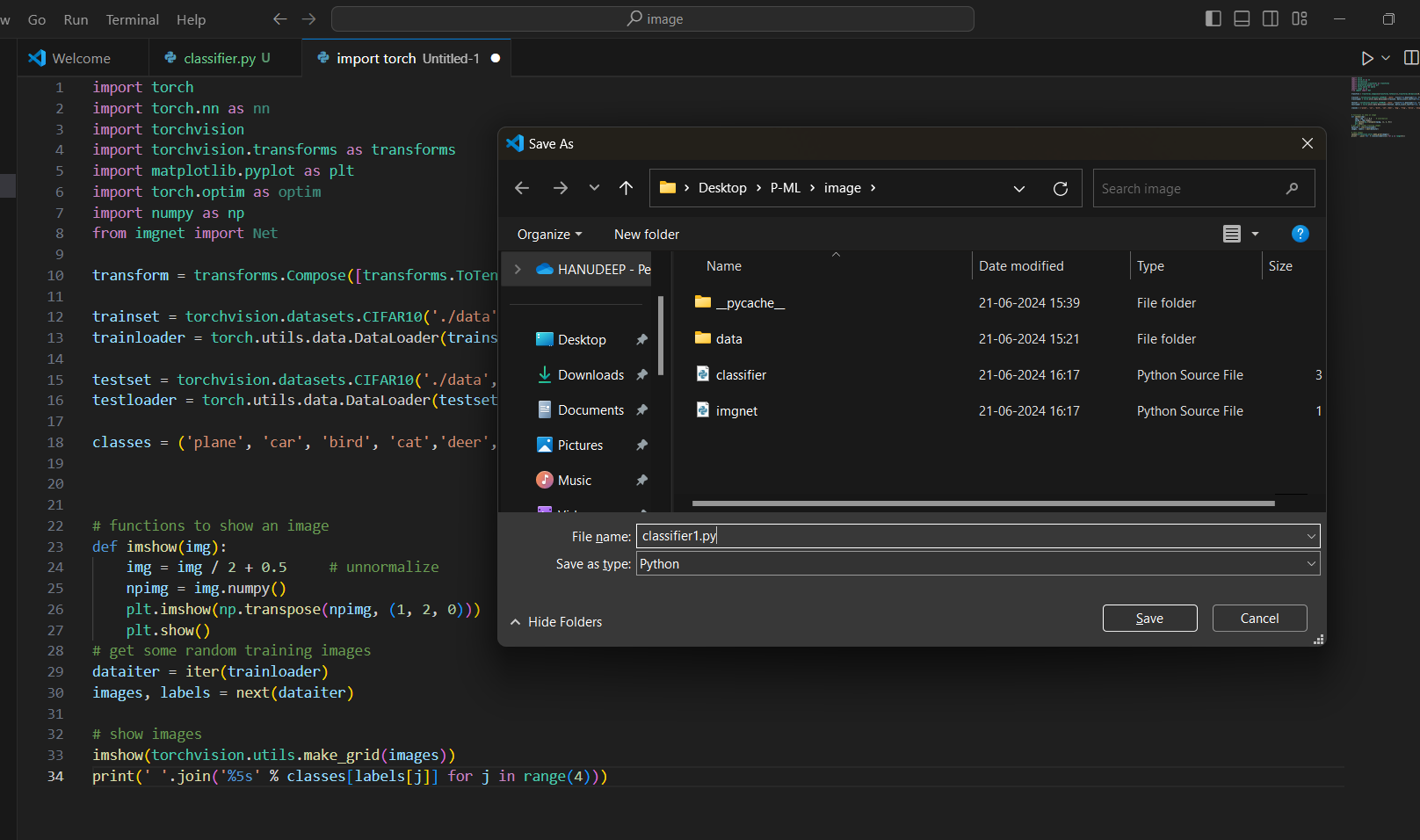
To run the code..

Method 1: click on the play button on top right

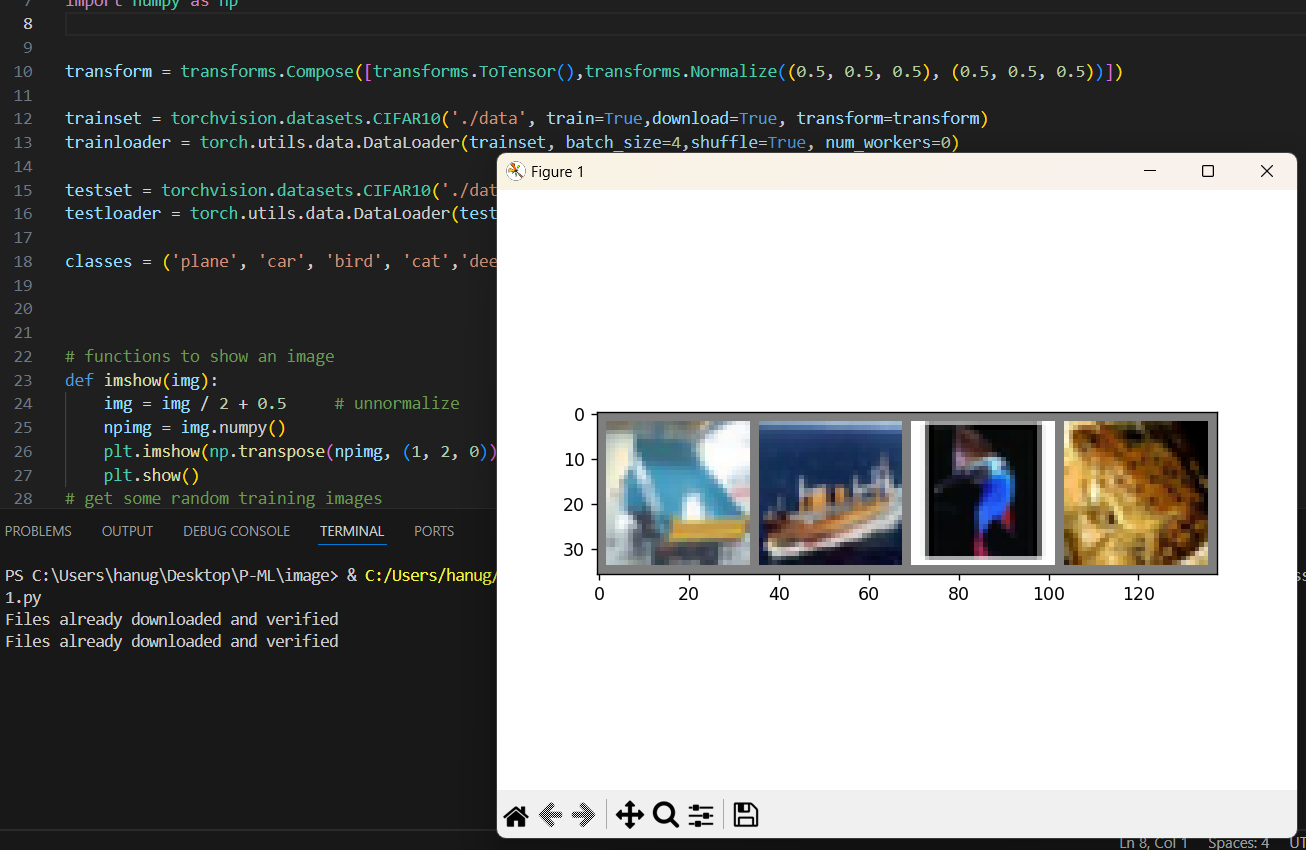
Method 2: goto view>Terminal

Type cmd : python filename.py

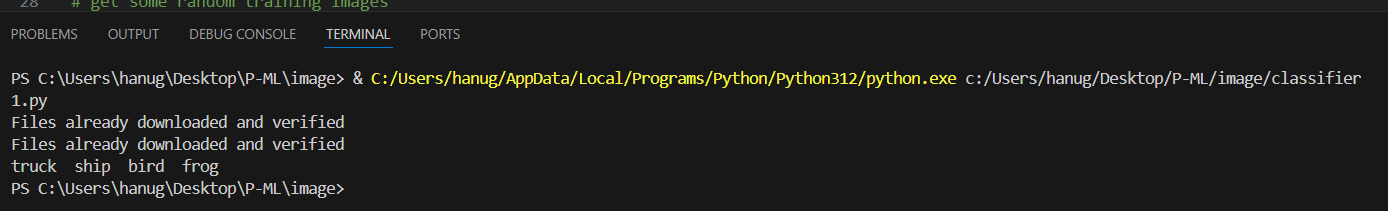
🡺python classifier.py



After running the program we get the test image like this…



Once close the image we get the result as… below



Now open a new file to create a neural network method

Code for the Neural network model

#Define a CNN

import torch

import torch.nn as nn

import torch.nn.functional as F

class Net(nn.Module):

    def \_\_init\_\_(self):

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

        self.conv1 = nn.Conv2d(3, 6, 5)

        self.pool = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(6, 16, 5)

        self.fc1 = nn.Linear(16 \* 5 \* 5, 120)

        self.fc2 = nn.Linear(120, 84)

        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):

        x = self.pool(F.relu(self.conv1(x)))

        x = self.pool(F.relu(self.conv2(x)))

        x = x.view(-1,16\*5\*5)# flatten all dimensions except batch

        x = F.relu(self.fc1(x))

        x = F.relu(self.fc2(x))

        x = self.fc3(x)

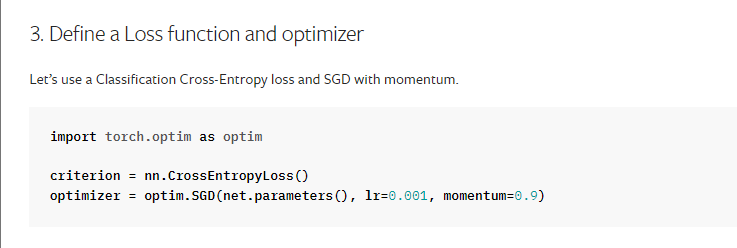
        return x

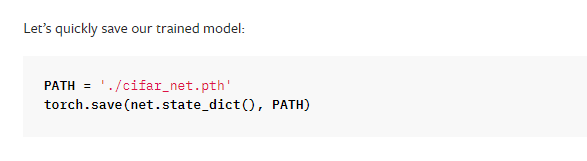
so… save it as imagnet.py in same image folder…

now open classier code and now we define the methods

that are







So the code set for training ….. after update the above code(3,4) to classifier file

import torch

import torch.nn as nn

import torchvision

import torchvision.transforms as transforms

import matplotlib.pyplot as plt

import torch.optim as optim

import numpy as np

from imgnet import Net

transform = transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10('./data', train=True,download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch\_size=4,shuffle=True, num\_workers=0)

testset = torchvision.datasets.CIFAR10('./data', train=False,download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch\_size=4,shuffle=True, num\_workers=0)

classes = ('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# functions to show an image

def imshow(img):

    img = img / 2 + 0.5     # unnormalize

    npimg = img.numpy()

    plt.imshow(np.transpose(npimg, (1, 2, 0)))

    plt.show()

# get some random training images

dataiter = iter(trainloader)

images, labels = next(dataiter)

# show images

imshow(torchvision.utils.make\_grid(images))

print(' '.join('%5s' % classes[labels[j]] for j in range(4)))

#Loss function and optimizer

net = Net()

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

#Train the network

for epoch in range(2):  # loop over the dataset multiple times

    running\_loss = 0.0

    for i, data in enumerate(trainloader, 0):

        # get the inputs; data is a list of [inputs, labels]

        images, labels = data

        # zero the parameter gradients

        optimizer.zero\_grad()

        # forward + backward + optimize

        output = net(images)

        loss = criterion(output, labels)

        loss.backward()

        optimizer.step()

        # print statistics

        running\_loss += loss.item()

        if i % 2000 == 1999:    # print every 2000 mini-batches

            print('[%d, %5d] loss:%.3f' % (epoch +1,i+1, running\_loss/2000))

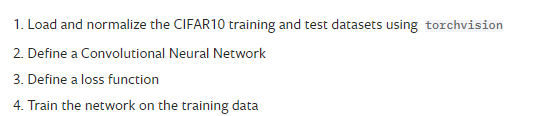
            running\_loss = 0.0

#Saving our trained model

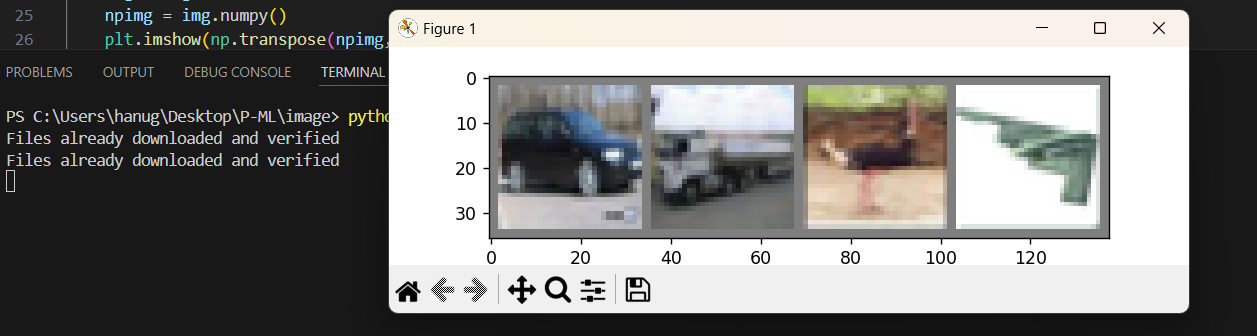
PATH = './cifar\_net.pth'

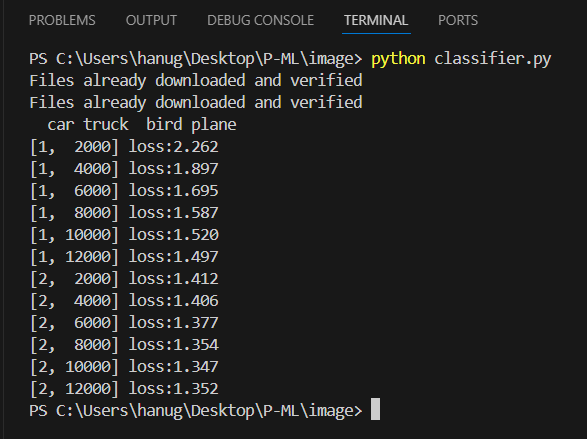
torch.save(net.state\_dict(), PATH)

run the code.. to execute the model on basis of

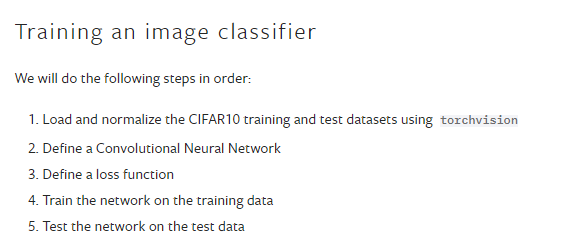


After running the code… we get





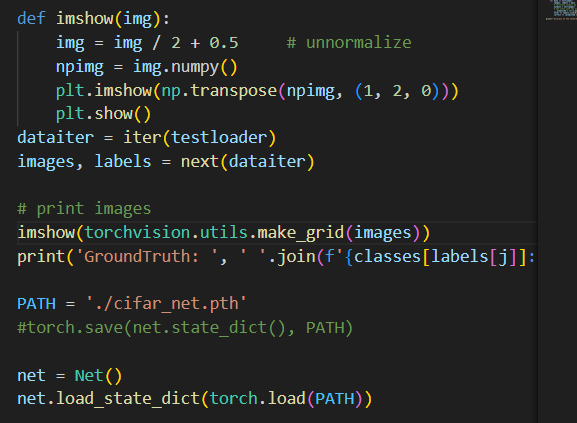
As far we completed up to 4 steps in



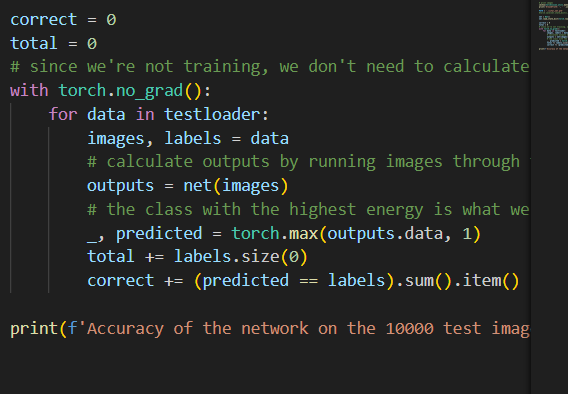
So now final one .. testing… and check the network performance

So to desing testing code we need

1. Load the datasets
2. Then image function



1. Then performance accuracy evaluator method



So the final testing code….

import torch

import torch.nn as nn

import torchvision

import torchvision.transforms as transforms

import matplotlib.pyplot as plt

import torch.optim as optim

import numpy as np

from imgnet import Net

transform = transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10('./data', train=True,download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch\_size=4,shuffle=True, num\_workers=0)

testset = torchvision.datasets.CIFAR10('./data', train=False,download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch\_size=4,shuffle=True, num\_workers=0)

classes = ('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse', 'ship', 'truck')

def imshow(img):

    img = img / 2 + 0.5     # unnormalize

    npimg = img.numpy()

    plt.imshow(np.transpose(npimg, (1, 2, 0)))

    plt.show()

dataiter = iter(testloader)

images, labels = next(dataiter)

# print images

imshow(torchvision.utils.make\_grid(images))

print('GroundTruth: ', ' '.join(f'{classes[labels[j]]:5s}' for j in range(4)))

PATH = './cifar\_net.pth'

net = Net()

net.load\_state\_dict(torch.load(PATH))

correct = 0

total = 0

# since we're not training, we don't need to calculate the gradients for our outputs

with torch.no\_grad():

    for data in testloader:

        images, labels = data

        # calculate outputs by running images through the network

        outputs = net(images)

        # the class with the highest energy is what we choose as prediction

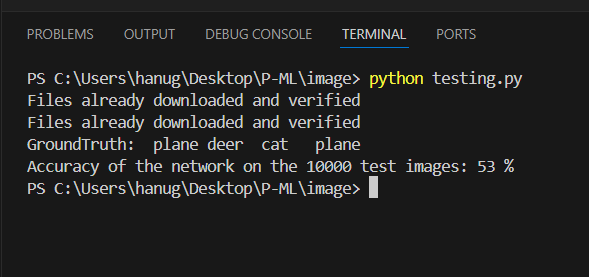
        \_, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)

        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 \* correct // total} %')

Result:



……………………………………………………………………………………………………………………………………………………………

As we seen we created 3 files..

1 classifier.py –

Consists of

\*load and normalize of CIFAR10 Dataset

\*image method to detect and one sample to test

\*loss function and optimizer

2 imagnet.py

Consists of Train model of a network

3 testing.py

Consists of

\*load and normalize of CIFAR10 Dataset

\*image method to detect and one sample to test

\*network performance method

Description :

For 1:

At first we imported the packages and setups the link for the dataset and program code

Then we define a method called imshow to detect the image and identify it

Then we set up the class ie based on cifar10 dataset consists plane,car,bird , cat…..etc

Now set a path to save model default and then

Mention loss function and optimizer

And also from 2 we can import the method net() for creation of NN….

At the time of execution we notice display of test data if we close it then in our terminal we can see the names of the images and loss function… of the data

For 2:

It just a code to write a method call net to create CNN in a separate file

So that we can simply import the method for further classification or testing purpose

For 3:

It is for testing purpose so we only required

\*to load data set

\* image method

Again we invole here network performance so that we import net() from 2

Then we write performance checking info code

Then execute to display the image detection wit accuracy……

So entire process consist of

