Images

Classification and Segmentation

Prof. Pei-Jun Lee

Course: Deep Learning based image recognition

Video Signal Processing and Application Lab

Outline

- **Image classification: CFAR-10**
- Image classification: dogs and cats

Image classification: CFAR-10

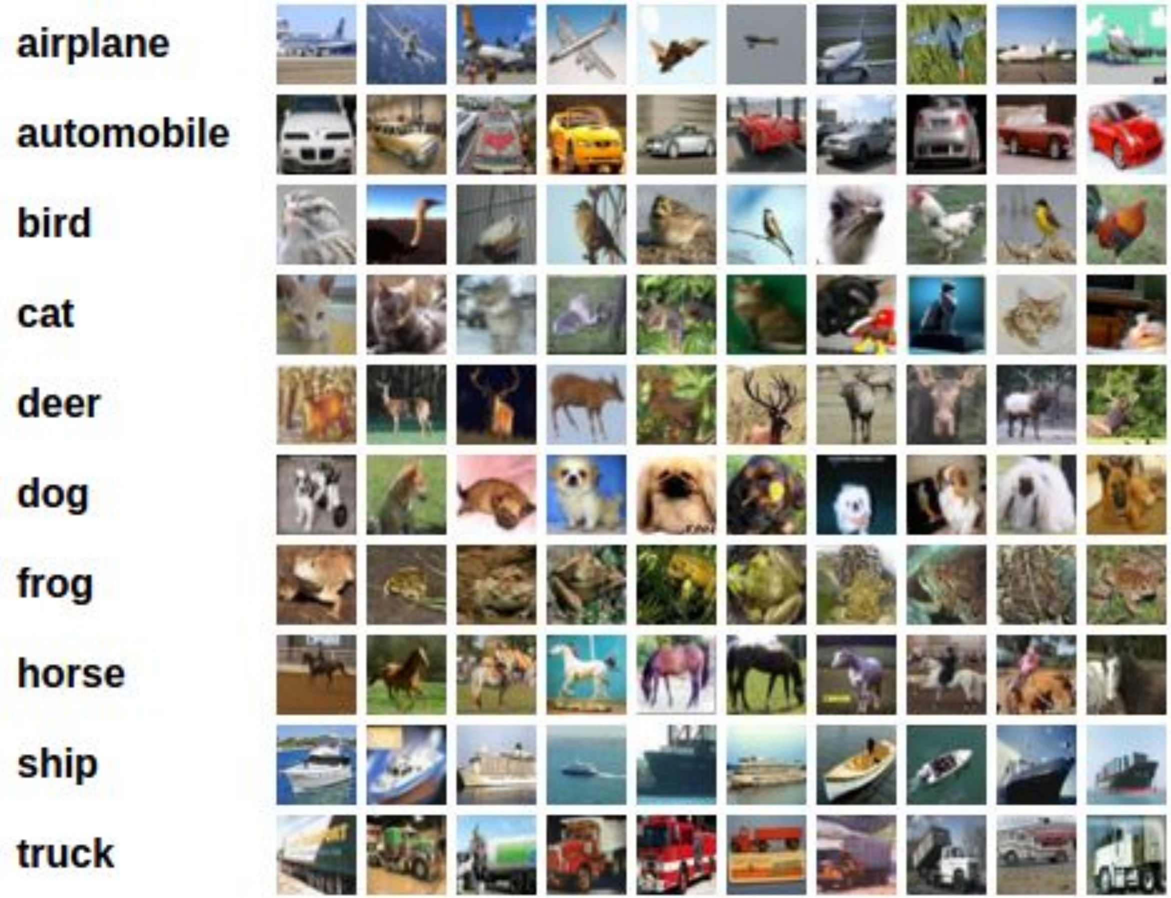
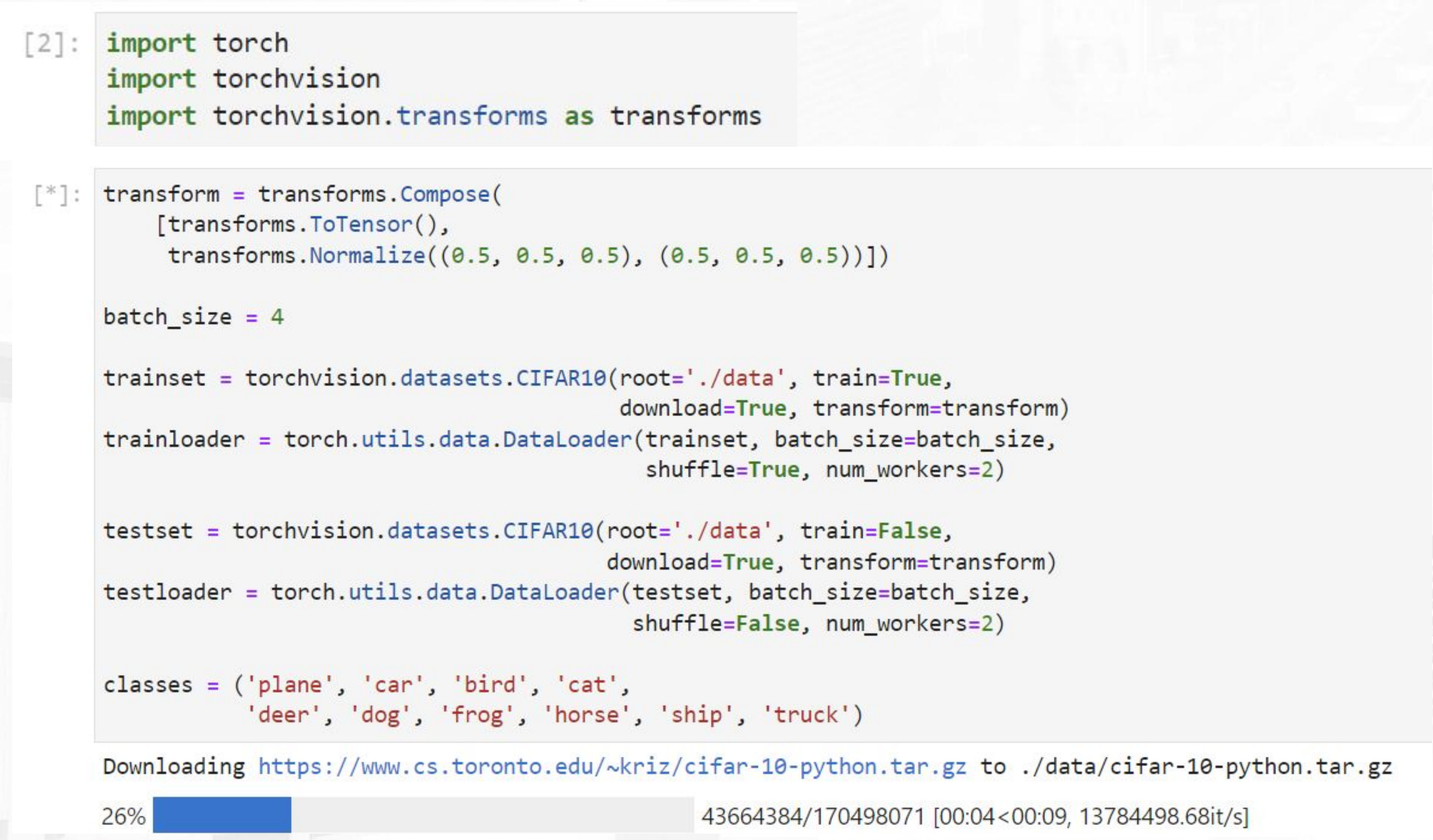


Image classification: CFAR-10

Load and normalize the CIFAR10 training and test datasets using torchvision



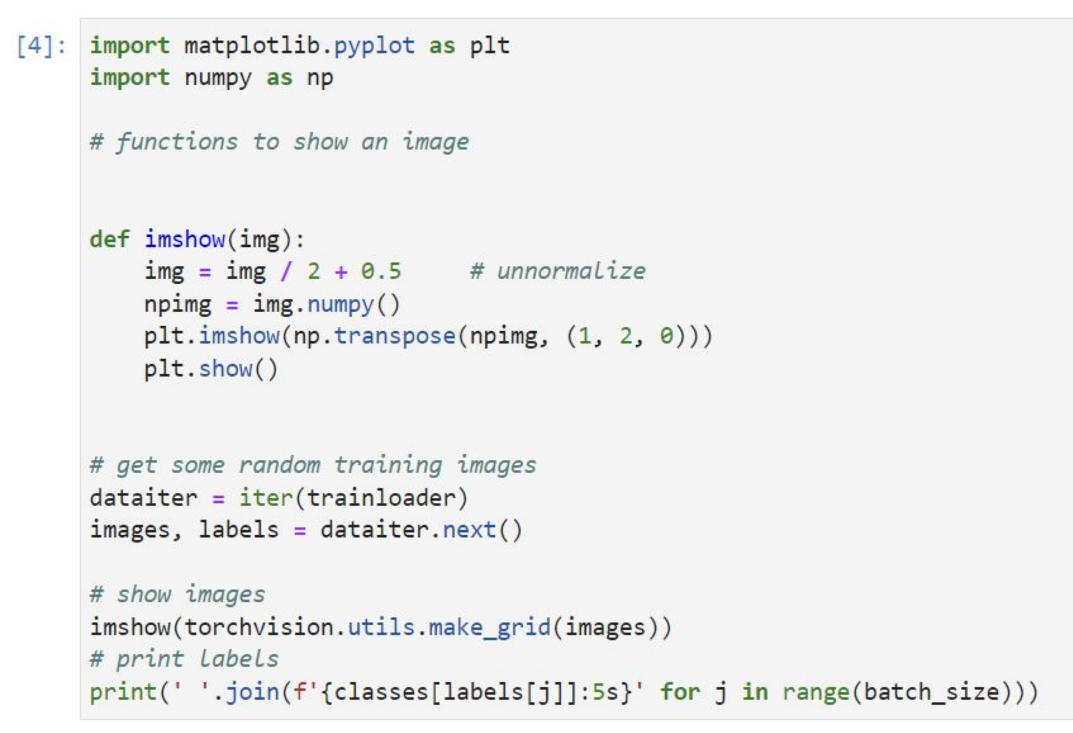
```
[2]: import torch
     import torchvision
     import torchvision.transforms as transforms
```

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

     batch_size = 4

     trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                             download=True, transform=transform)
     trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                               shuffle=True, num_workers=2)

     testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                            download=True, transform=transform)
     testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                              shuffle=False, num_workers=2)

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

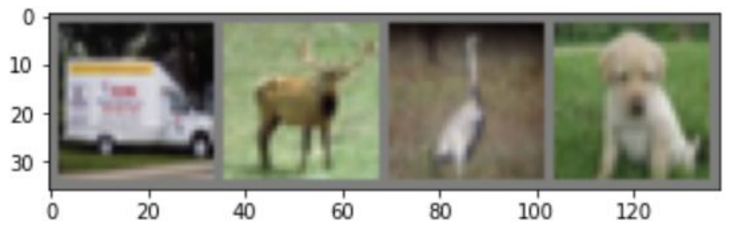
```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz

26%                     43664384/170498071 [00:04<00:09, 13784498.68it/s]
```

Image classification: CFAR-10

Verify CIFAR10 datasets



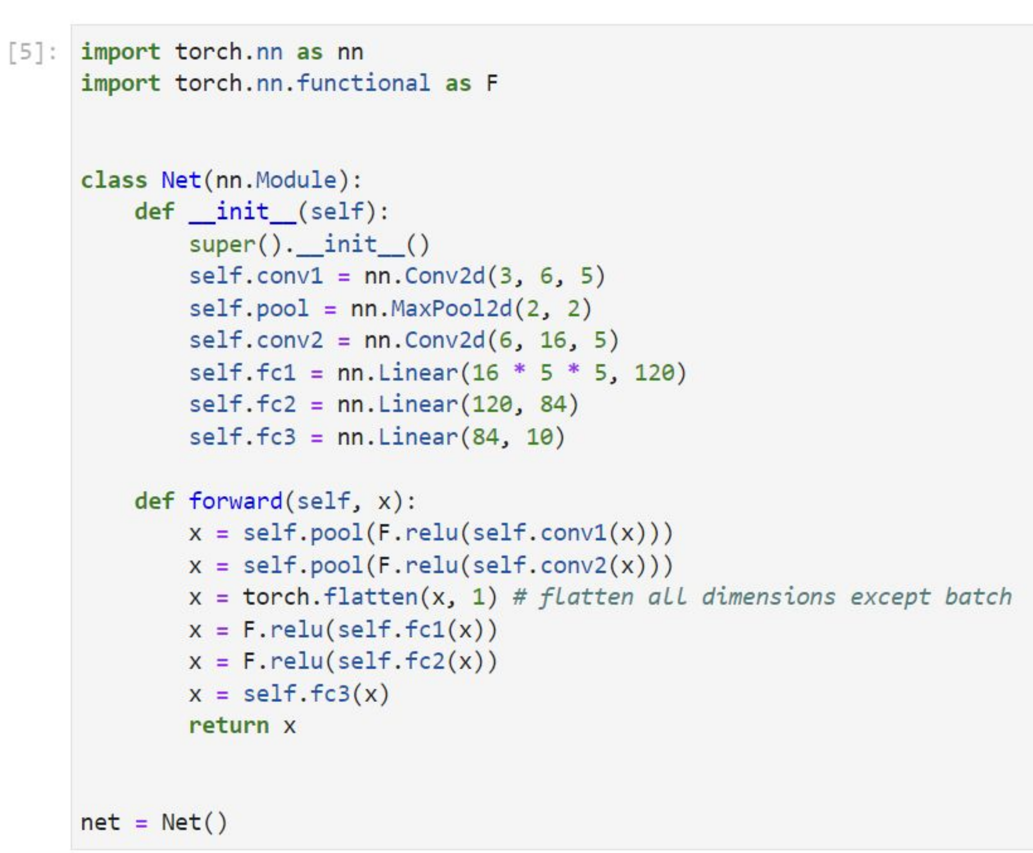
```
[4]: import matplotlib.pyplot as plt
     import numpy as np

     # 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 = dataiter.next()

     # show images
     imshow(torchvision.utils.make_grid(images))
     # print labels
     print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
```



```
truck deer  bird  dog
```

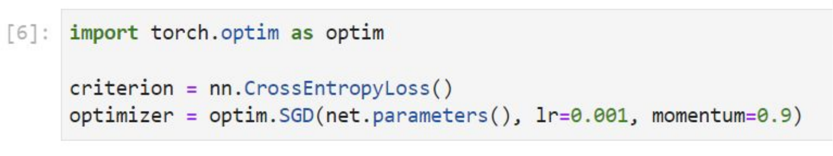
Image classification: CFAR-10

Define a simple Neural Network



```
[5]: import torch.nn as nn
     import torch.nn.functional as F


     class Net(nn.Module):
         def __init__(self):
             super().__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 = torch.flatten(x, 1) # flatten all dimensions except batch
             x = F.relu(self.fc1(x))
             x = F.relu(self.fc2(x))
             x = self.fc3(x)
             return x


     net = Net()
```

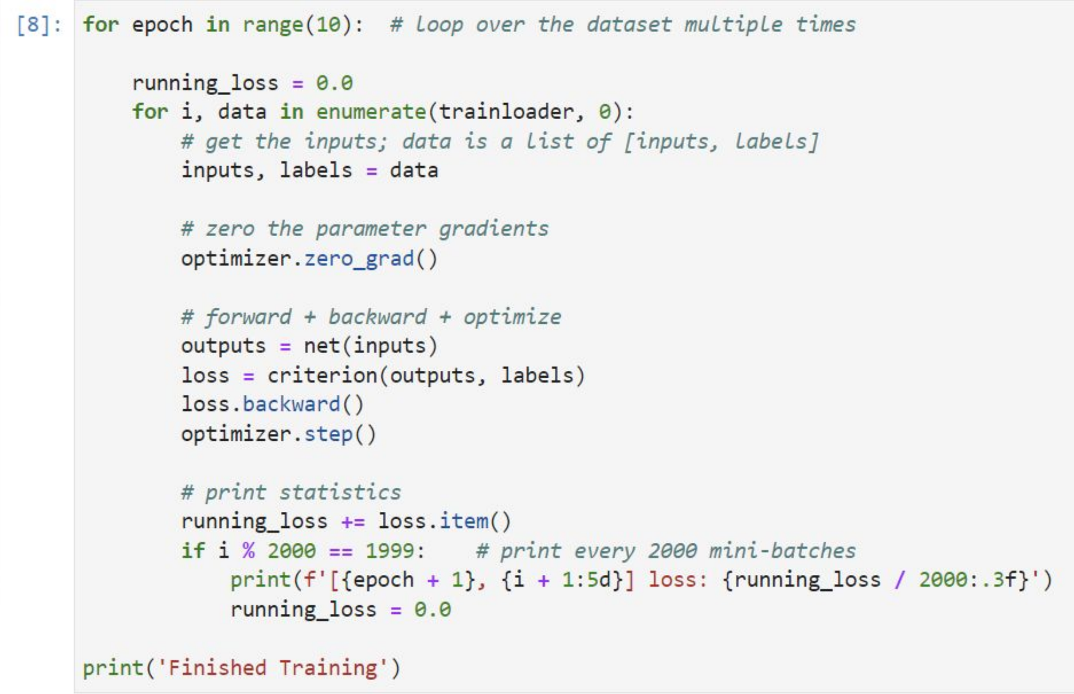
Define Cross-Entropy loss



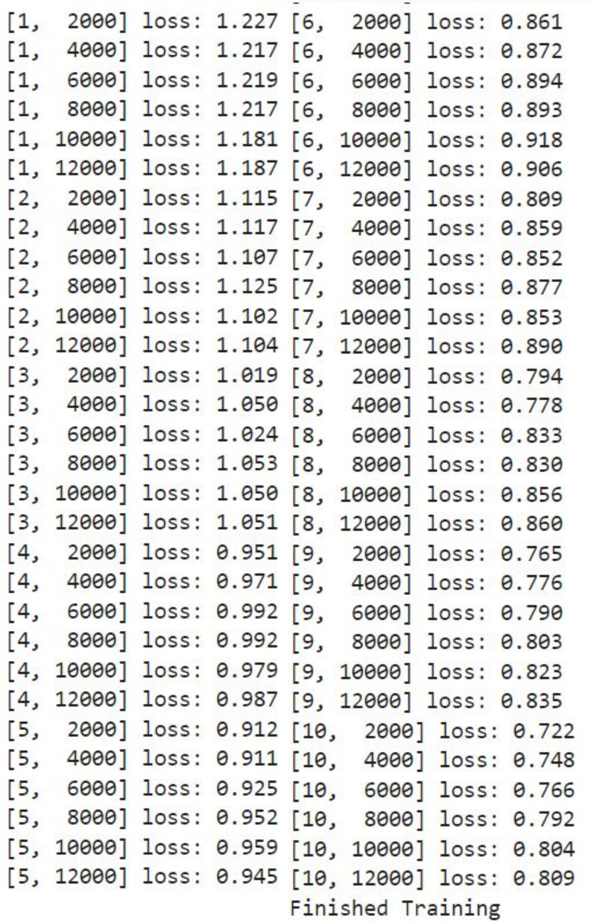
```
[6]: import torch.optim as optim

     criterion = nn.CrossEntropyLoss()
     optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Image classification: CFAR-10

Train simple Neural Network



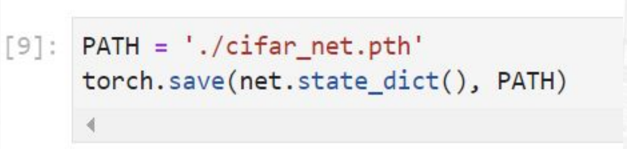
```
[8]: for epoch in range(10):  # loop over the dataset multiple times

         running_loss = 0.0
         for i, data in enumerate(trainloader, 0):
             # get the inputs; data is a list of [inputs, labels]
             inputs, labels = data

             # zero the parameter gradients
             optimizer.zero_grad()

             # forward + backward + optimize
             outputs = net(inputs)
             loss = criterion(outputs, labels)
             loss.backward()
             optimizer.step()

             # print statistics
             running_loss += loss.item()
             if i % 2000 == 1999:    # print every 2000 mini-batches
                 print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
                 running_loss = 0.0

     print('Finished Training')
```



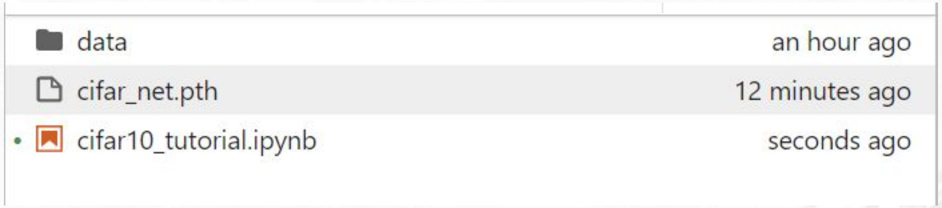
```
[1,  2000] loss: 1.227 [6,  2000] loss: 0.861
[1,  4000] loss: 1.217 [6,  4000] loss: 0.872
[1,  6000] loss: 1.219 [6,  6000] loss: 0.894
[1,  8000] loss: 1.217 [6,  8000] loss: 0.893
[1, 10000] loss: 1.181 [6, 10000] loss: 0.918
[1, 12000] loss: 1.187 [6, 12000] loss: 0.906
[2,  2000] loss: 1.115 [7,  2000] loss: 0.809
[2,  4000] loss: 1.117 [7,  4000] loss: 0.859
[2,  6000] loss: 1.107 [7,  6000] loss: 0.852
[2,  8000] loss: 1.125 [7,  8000] loss: 0.877
[2, 10000] loss: 1.102 [7, 10000] loss: 0.853
[2, 12000] loss: 1.104 [7, 12000] loss: 0.890
[3,  2000] loss: 1.019 [8,  2000] loss: 0.794
[3,  4000] loss: 1.050 [8,  4000] loss: 0.778
[3,  6000] loss: 1.024 [8,  6000] loss: 0.833
[3,  8000] loss: 1.053 [8,  8000] loss: 0.830
[3, 10000] loss: 1.050 [8, 10000] loss: 0.856
[3, 12000] loss: 1.051 [8, 12000] loss: 0.860
[4,  2000] loss: 0.951 [9,  2000] loss: 0.765
[4,  4000] loss: 0.971 [9,  4000] loss: 0.776
[4,  6000] loss: 0.992 [9,  6000] loss: 0.790
[4,  8000] loss: 0.992 [9,  8000] loss: 0.803
[4, 10000] loss: 0.979 [9, 10000] loss: 0.823
[4, 12000] loss: 0.987 [9, 12000] loss: 0.835
[5,  2000] loss: 0.912 [10,  2000] loss: 0.722
[5,  4000] loss: 0.911 [10,  4000] loss: 0.748
[5,  6000] loss: 0.925 [10,  6000] loss: 0.766
[5,  8000] loss: 0.952 [10,  8000] loss: 0.792
[5, 10000] loss: 0.959 [10, 10000] loss: 0.804
[5, 12000] loss: 0.945 [10, 12000] loss: 0.809
                       Finished Training
```

Image classification: CFAR-10

Save trained model

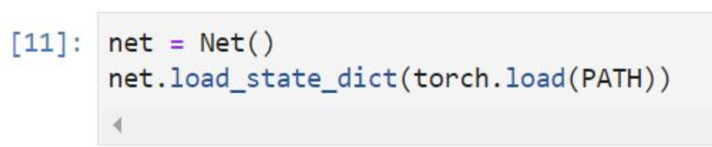


```
[9]: PATH = './cifar_net.pth'
     torch.save(net.state_dict(), PATH)
```

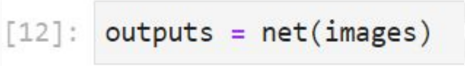


data	an hour ago
cifar_net.pth	12 minutes ago
cifar10_tutorial.ipynb	seconds ago

Load trained model



```
[11]: net = Net()
      net.load_state_dict(torch.load(PATH))
```

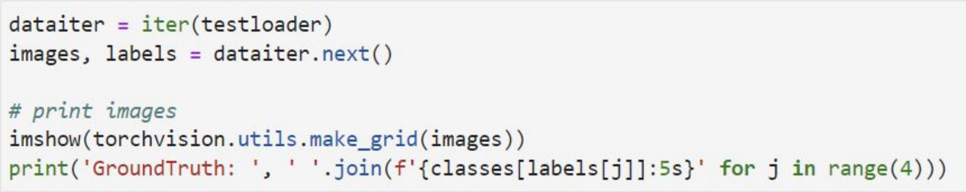


```
[12]: outputs = net(images)
```

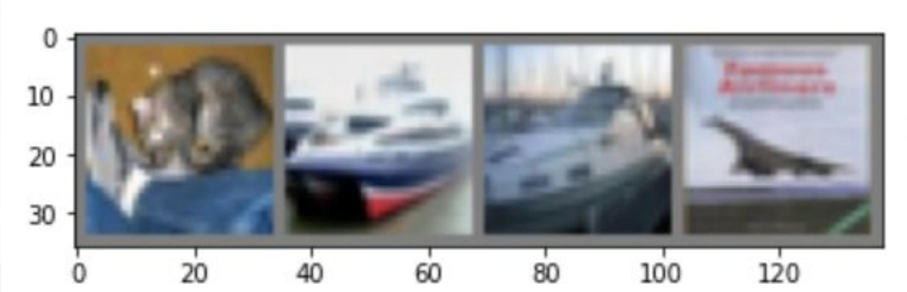
Image classification: CFAR-10

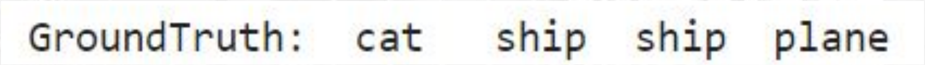
Verify the trained model

Showing the ***Ground truth***



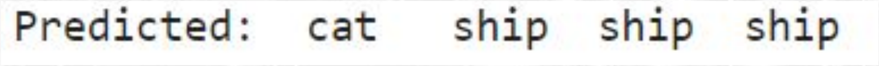
```
dataiter = iter(testloader)
images, labels = dataiter.next()

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join(f'{classes[labels[j]]:5s}' for j in range(4)))
```





```
GroundTruth:  cat   ship  ship  plane
```

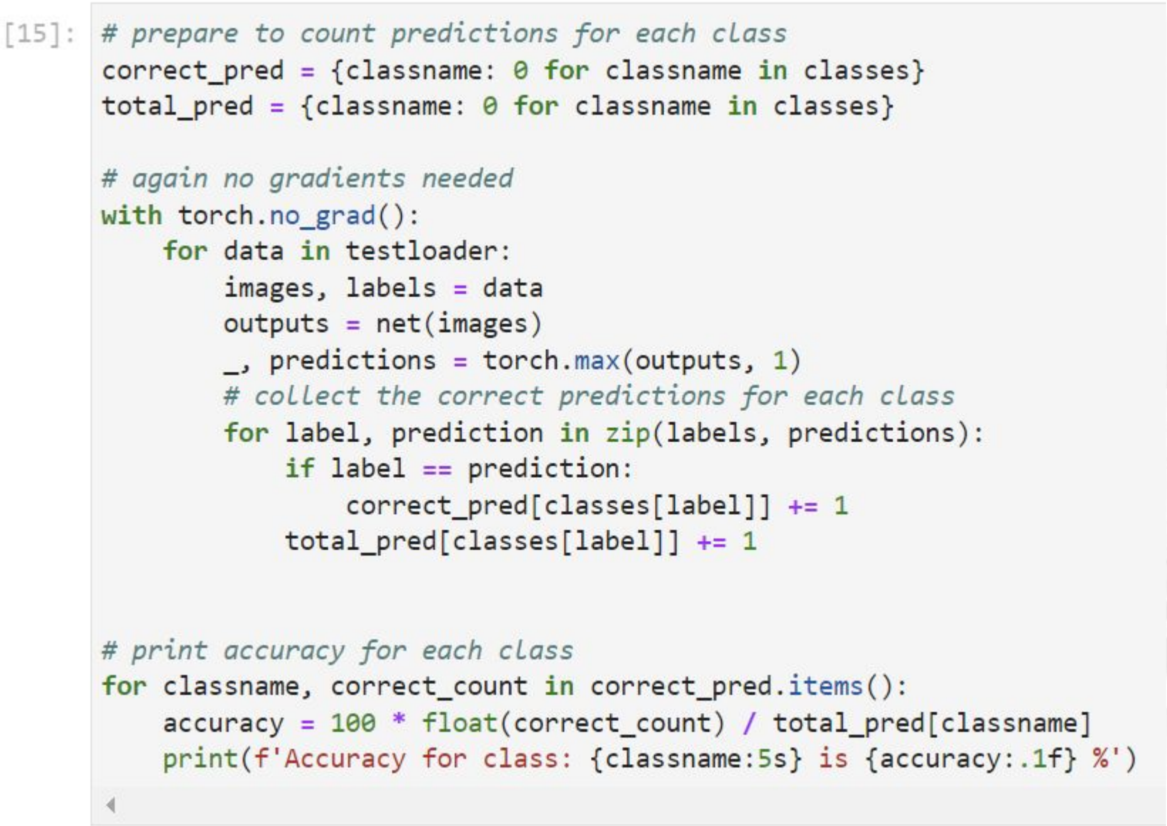
***Predict** classification:*



```
Predicted:  cat   ship  ship  ship
```

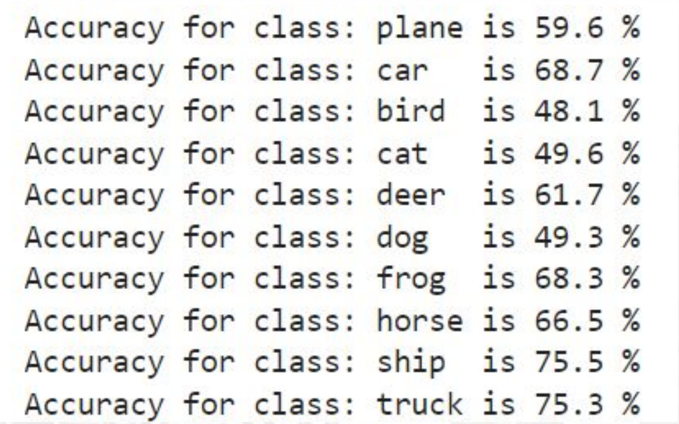
Image classification: CFAR-10

Evaluation Trained data



[15]:

```
# prepare to count predictions for each class
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

# again no gradients needed
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1


# print accuracy for each class
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```



```
Accuracy for class: plane is 59.6 %
Accuracy for class: car   is 68.7 %
Accuracy for class: bird  is 48.1 %
Accuracy for class: cat   is 49.6 %
Accuracy for class: deer  is 61.7 %
Accuracy for class: dog   is 49.3 %
Accuracy for class: frog  is 68.3 %
Accuracy for class: horse is 66.5 %
Accuracy for class: ship  is 75.5 %
Accuracy for class: truck is 75.3 %
```

Question 1:

Above is the training and verify the model by using CPU

Question: Train and verify the model to classify the CIFAR labels by using GPU.

Outline

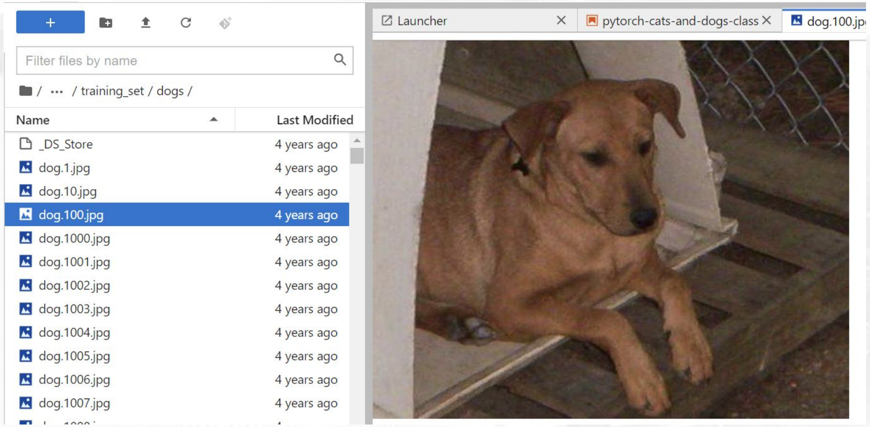
- Image classification: CFAR-10
- **Image classification: dogs and cats**

Binary image classification: dogs and cat

Dataset preparing:

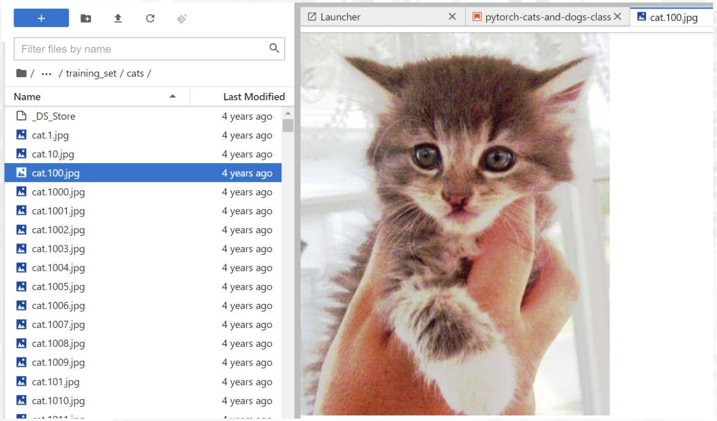
Download the training folder and testing folders (.zip files), and extracts to your PC.

Training data: https://140.118.102.50:5001/sharing/A6T9dFfB5
Testing data: https://140.118.102.50:5001/sharing/cXdQtqjp0



/ ... / training_set / dogs /

Name	Last Modified
_DS_Store	4 years ago
dog.1.jpg	4 years ago
dog.10.jpg	4 years ago
dog.100.jpg	4 years ago
dog.1000.jpg	4 years ago
dog.1001.jpg	4 years ago
dog.1002.jpg	4 years ago
dog.1003.jpg	4 years ago
dog.1004.jpg	4 years ago
dog.1005.jpg	4 years ago
dog.1006.jpg	4 years ago
dog.1007.jpg	4 years ago

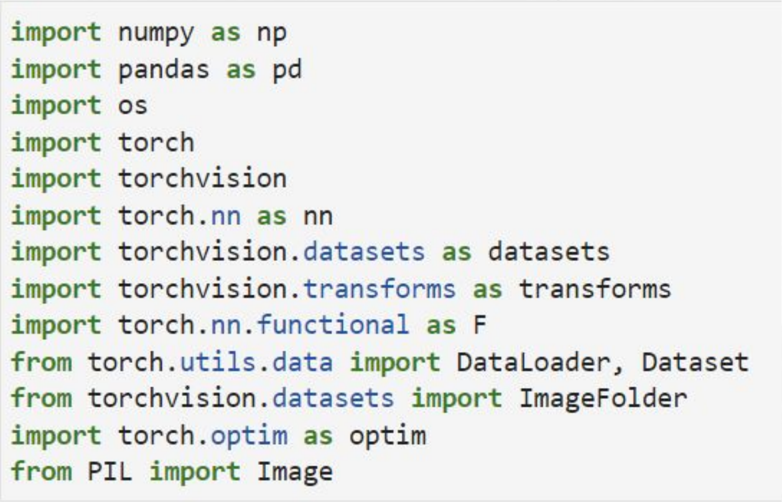


/ ... / training_set / cats /

Name	Last Modified
_DS_Store	4 years ago
cat.1.jpg	4 years ago
cat.10.jpg	4 years ago
cat.100.jpg	4 years ago
cat.1000.jpg	4 years ago
cat.1001.jpg	4 years ago
cat.1002.jpg	4 years ago
cat.1003.jpg	4 years ago
cat.1004.jpg	4 years ago
cat.1005.jpg	4 years ago
cat.1006.jpg	4 years ago
cat.1007.jpg	4 years ago
cat.1008.jpg	4 years ago
cat.1009.jpg	4 years ago
cat.101.jpg	4 years ago
cat.1010.jpg	4 years ago

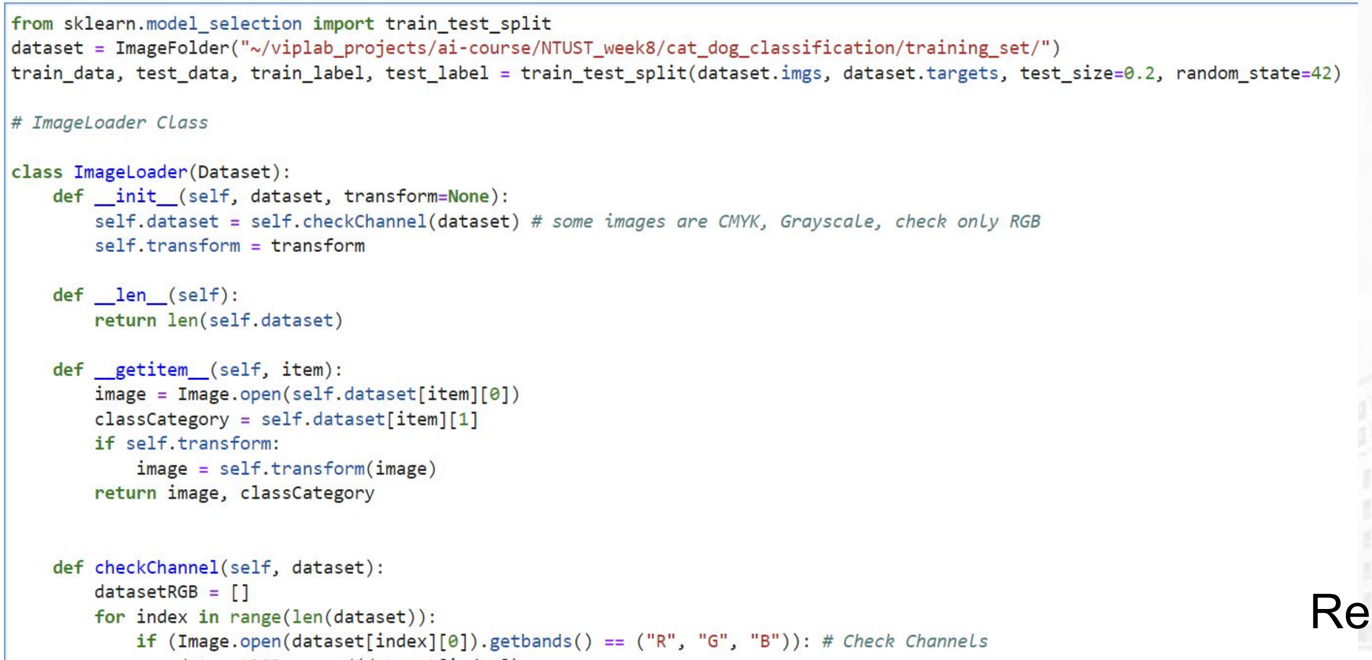
Binary image classification: dogs and cat

Import library



```
import numpy as np
import pandas as pd
import os
import torch
import torchvision
import torch.nn as nn
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset
from torchvision.datasets import ImageFolder
import torch.optim as optim
from PIL import Image
```

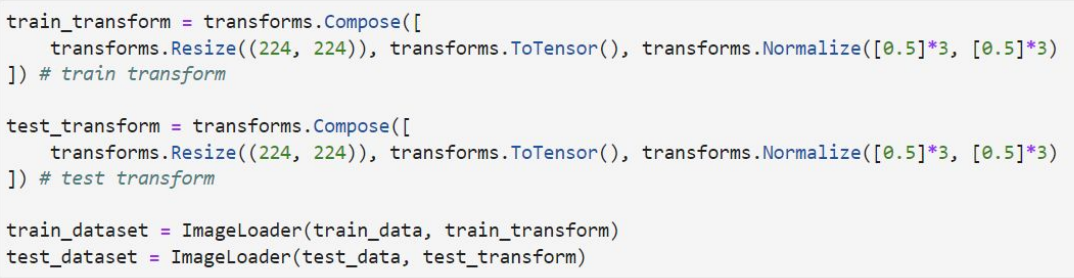
Binary image classification: dogs and cat



```
from sklearn.model_selection import train_test_split
dataset = ImageFolder("~/viplab_projects/ai-course/NTUST_week8/cat_dog_classification/training_set/")
train_data, test_data, train_label, test_label = train_test_split(dataset.imgs, dataset.targets, test_size=0.2, random_state=42)

# ImageLoader Class

class ImageLoader(Dataset):
    def __init__(self, dataset, transform=None):
        self.dataset = self.checkChannel(dataset) # some images are CMYK, Grayscale, check only RGB
        self.transform = transform

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, item):
        image = Image.open(self.dataset[item][0])
        classCategory = self.dataset[item][1]
        if self.transform:
            image = self.transform(image)
        return image, classCategory


    def checkChannel(self, dataset):
        datasetRGB = []
        for index in range(len(dataset)):
            if (Image.open(dataset[index][0]).getbands() == ("R", "G", "B")): # Check Channels
                datasetRGB.append(dataset[index])
        return datasetRGB
```

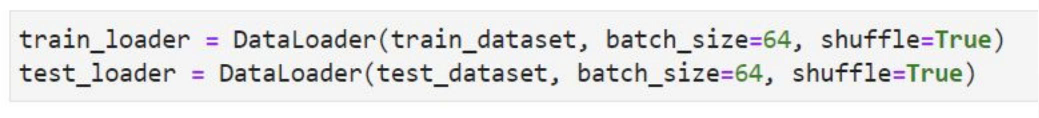
Image loader: from file to data

Resize images to 224x224



```
train_transform = transforms.Compose([
    transforms.Resize((224, 224)), transforms.ToTensor(), transforms.Normalize([0.5]*3, [0.5]*3)
]) # train transform

test_transform = transforms.Compose([
    transforms.Resize((224, 224)), transforms.ToTensor(), transforms.Normalize([0.5]*3, [0.5]*3)
]) # test transform

train_dataset = ImageLoader(train_data, train_transform)
test_dataset = ImageLoader(test_data, test_transform)
```

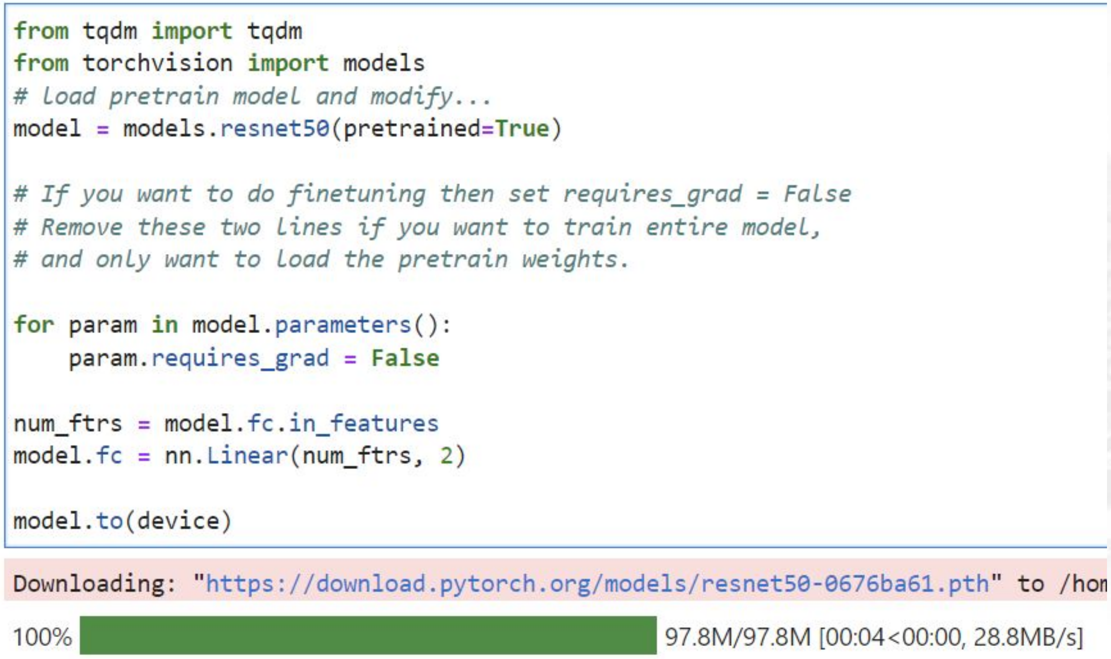
Binary image classification: dogs and cat



```
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=True)
```

Data loader: from data to tensors

Depending on the GPUs memories size, choose your ***batch_size*** number



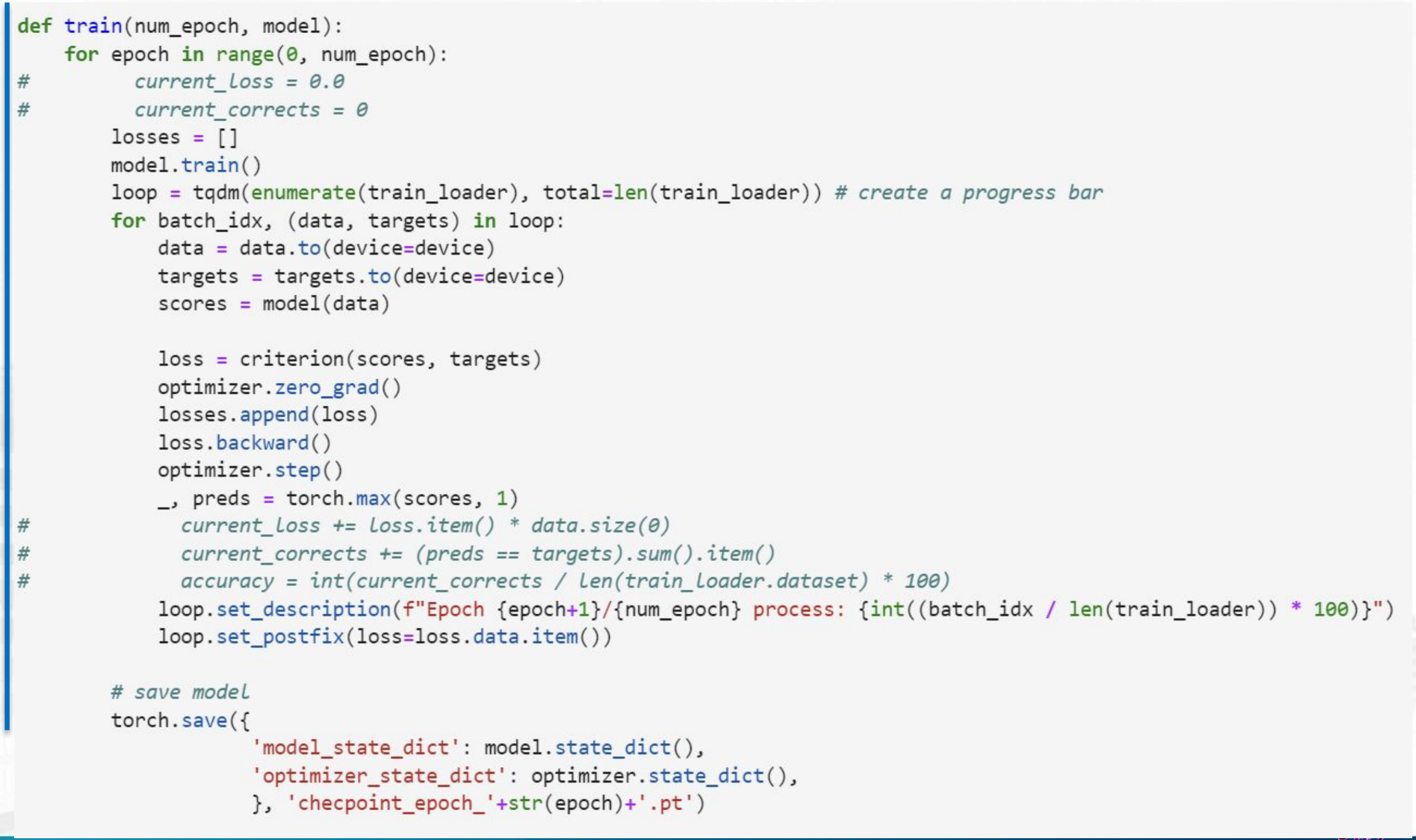
```
from tqdm import tqdm
from torchvision import models
# Load pretrain model and modify...
model = models.resnet50(pretrained=True)

# If you want to do finetuning then set requires_grad = False
# Remove these two lines if you want to train entire model,
# and only want to load the pretrain weights.

for param in model.parameters():
    param.requires_grad = False

num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)

model.to(device)
```

```
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /hom
100% 97.8M/97.8M [00:04<00:00, 28.8MB/s]
```

Using a pre-training model to speed up the training process.

Binary image classification: dogs and cat

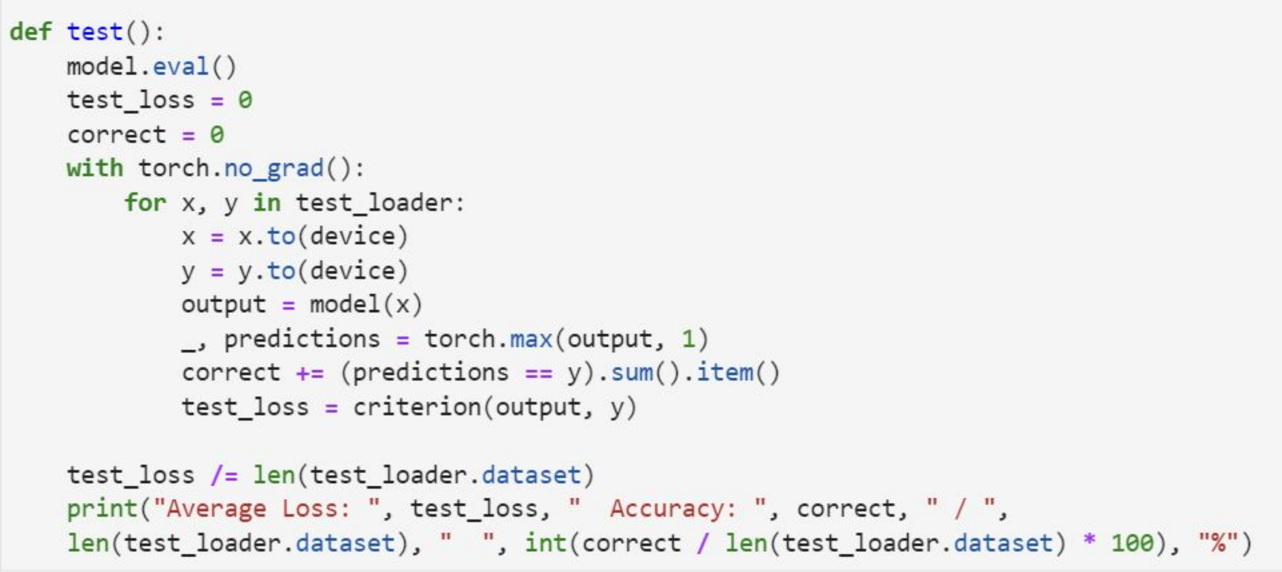
Train function



```
def train(num_epoch, model):
    for epoch in range(0, num_epoch):
#         current_loss = 0.0
#         current_corrects = 0
        losses = []
        model.train()
        loop = tqdm(enumerate(train_loader), total=len(train_loader)) # create a progress bar
        for batch_idx, (data, targets) in loop:
            data = data.to(device=device)
            targets = targets.to(device=device)
            scores = model(data)

            loss = criterion(scores, targets)
            optimizer.zero_grad()
            losses.append(loss)
            loss.backward()
            optimizer.step()
            _, preds = torch.max(scores, 1)
#             current_loss += loss.item() * data.size(0)
#             current_corrects += (preds == targets).sum().item()
#             accuracy = int(current_corrects / len(train_loader.dataset) * 100)
            loop.set_description(f"Epoch {epoch+1}/{num_epoch} process: {int((batch_idx / len(train_loader)) * 100)}")
            loop.set_postfix(loss=loss.data.item())

        # save model
        torch.save({
                    'model_state_dict': model.state_dict(),
                    'optimizer_state_dict': optimizer.state_dict(),
                    }, 'checpoint_epoch_'+str(epoch)+'.pt')
```

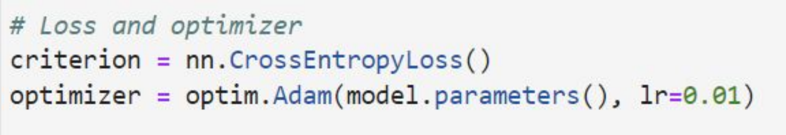
Binary image classification: dogs and cat

Test function



```
def test():
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for x, y in test_loader:
            x = x.to(device)
            y = y.to(device)
            output = model(x)
            _, predictions = torch.max(output, 1)
            correct += (predictions == y).sum().item()
            test_loss = criterion(output, y)

    test_loss /= len(test_loader.dataset)
    print("Average Loss: ", test_loss, "  Accuracy: ", correct, " / ",
    len(test_loader.dataset), "  ", int(correct / len(test_loader.dataset) * 100), "%")
```

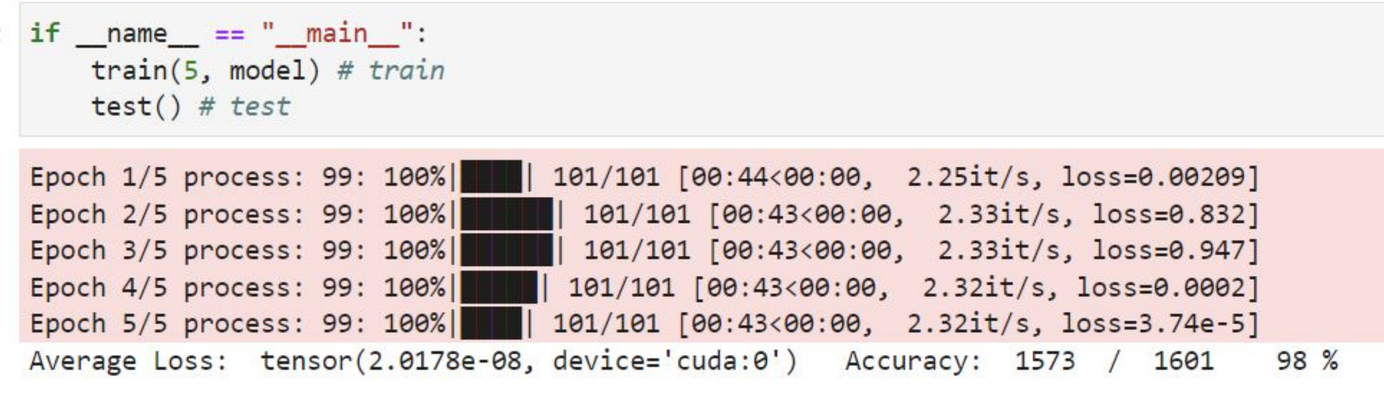
Binary image classification: dogs and cat



```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

Config the loss functions

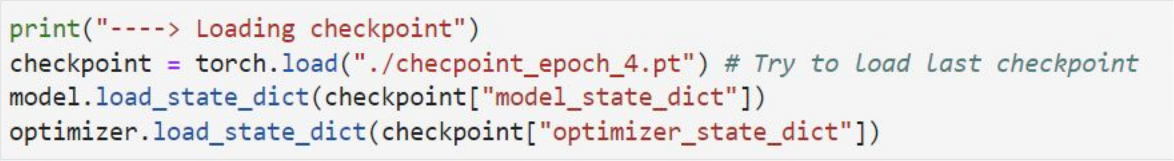
Train and test



```
if __name__ == "__main__":
    train(5, model) # train
    test() # test
```

```
Epoch 1/5 process: 99: 100%|████| 101/101 [00:44<00:00,  2.25it/s, loss=0.00209]
Epoch 2/5 process: 99: 100%|██████| 101/101 [00:43<00:00,  2.33it/s, loss=0.832]
Epoch 3/5 process: 99: 100%|██████| 101/101 [00:43<00:00,  2.33it/s, loss=0.947]
Epoch 4/5 process: 99: 100%|█████| 101/101 [00:43<00:00,  2.32it/s, loss=0.0002]
Epoch 5/5 process: 99: 100%|████| 101/101 [00:43<00:00,  2.32it/s, loss=3.74e-5]
Average Loss:  tensor(2.0178e-08, device='cuda:0')   Accuracy:  1573  /  1601    98 %
```

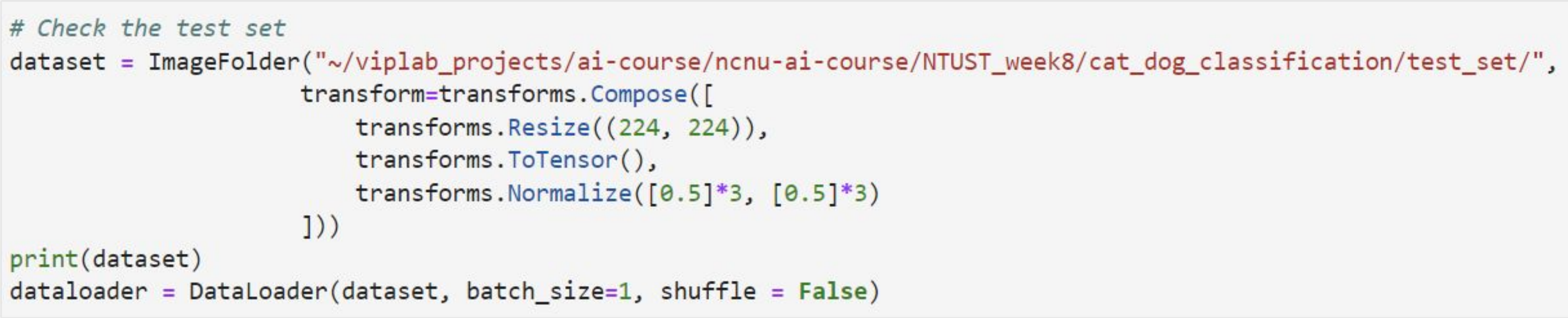
Binary image classification: dogs and cat



```
print("----> Loading checkpoint")
checkpoint = torch.load("./checpoint_epoch_4.pt") # Try to load last checkpoint
model.load_state_dict(checkpoint["model_state_dict"])
optimizer.load_state_dict(checkpoint["optimizer_state_dict"])
```

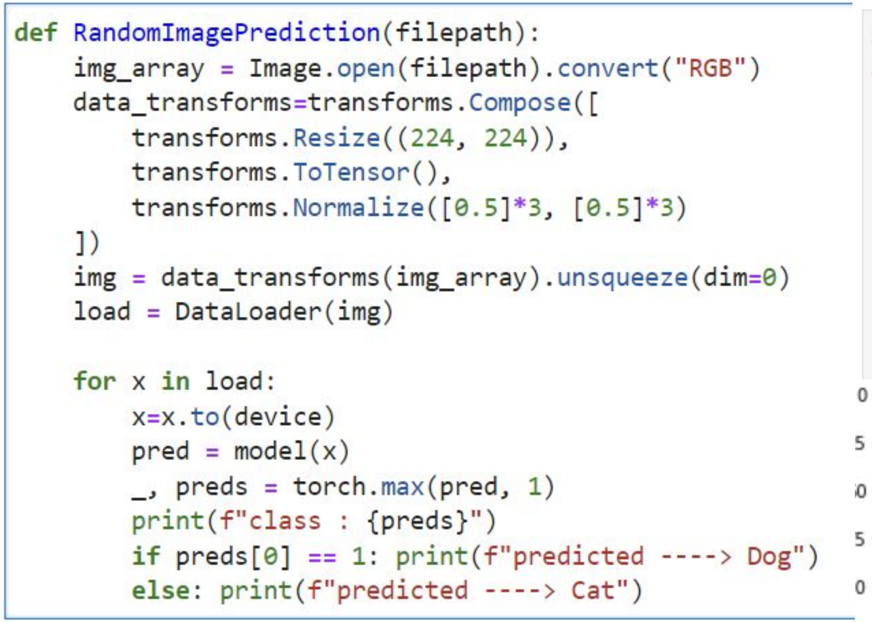
Check the checkpoint and save the trained model

Validate the dataset

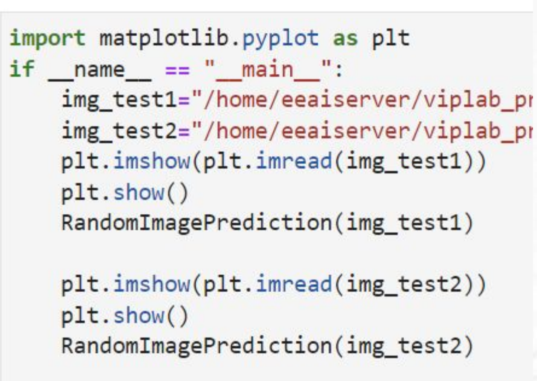


```
# Check the test set
dataset = ImageFolder("~/viplab_projects/ai-course/ncnu-ai-course/NTUST_week8/cat_dog_classification/test_set/",
                     transform=transforms.Compose([
                         transforms.Resize((224, 224)),
                         transforms.ToTensor(),
                         transforms.Normalize([0.5]*3, [0.5]*3)
                     ]))
print(dataset)
dataloader = DataLoader(dataset, batch_size=1, shuffle = False)
```

Binary image classification: dogs and cat

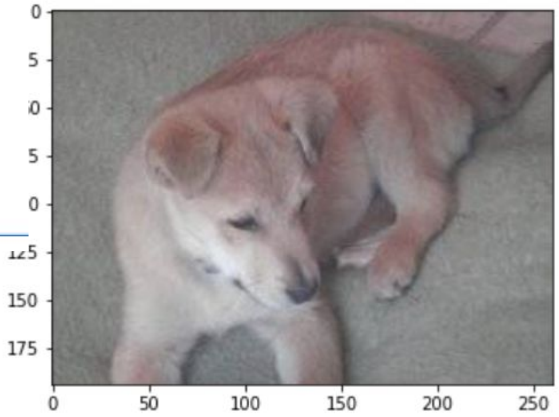


```python
def RandomImagePrediction(filepath):
    img_array = Image.open(filepath).convert("RGB")
    data_transforms=transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.5]*3, [0.5]*3)
    ])
    img = data_transforms(img_array).unsqueeze(dim=0)
    load = DataLoader(img)

    for x in load:
        x=x.to(device)
        pred = model(x)
        _, preds = torch.max(pred, 1)
        print(f"class : {preds}")
        if preds[0] == 1: print(f"predicted ----> Dog")
        else: print(f"predicted ----> Cat")
```

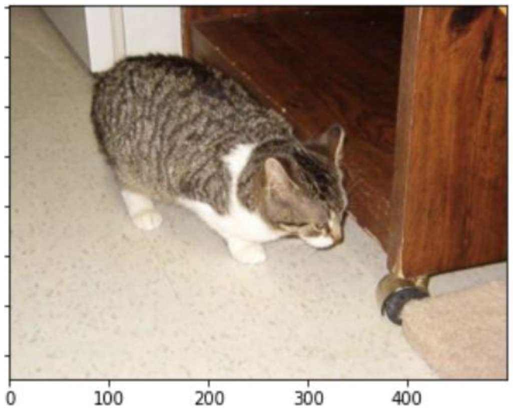


```python
import matplotlib.pyplot as plt
if __name__ == "__main__":
    img_test1="/home/eeaiserver/viplab_pr
    img_test2="/home/eeaiserver/viplab_pr
    plt.imshow(plt.imread(img_test1))
    plt.show()
    RandomImagePrediction(img_test1)

    plt.imshow(plt.imread(img_test2))
    plt.show()
    RandomImagePrediction(img_test2)
```

Testing prediction function



```
class : tensor([1], device='cuda:0')
predicted ----> Dog
```



```
class : tensor([0], device='cuda:0')
predicted ----> Cat
```

img_test1="……./test_set/dogs/dog.4091.jpg"

img_test2="…….//test_set/cats/cat.4641.jpg"

Question 2:

<u>Question:</u> Train and predict the testing images to “dog” or “cat” classified.

Requirements: Setting the device to using GPUs on your PC.