In [171…

```
%matplotlib inline
#Reference from https://pytorch.org/
```

In [172…

```
from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy

plt.ion()   # interactive mode
```

In [173…

```
# Data augmentation and normalization for training
# Just normalization for validation
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'test': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}

data_dir = '../Dataset/Mask4Classification'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'test']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=5,
                                             shuffle=True, num_workers=4)
              for x in ['train', 'test']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'test']}
class_names = image_datasets['train'].classes
print(class_names)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
['chip', 'disk', 'hard_disk', 'reader', 'y_part']
```

Visualize a few images ^^^^^^^^^^^^^^^^^^^^^^ Let's visualize a few training images so as to understand the data augmentations.

In [174…

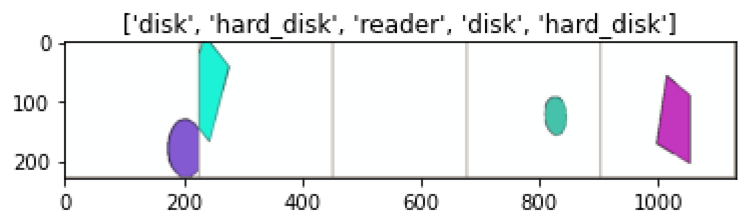
```
def imshow(inp, title=None):
    """Imshow for Tensor."""
```

```
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # pause a bit so that plots are updated


# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

# Make a grid from batch
out = torchvision.utils.make_grid(inputs)

imshow(out, title=[class_names[x] for x in classes])
```



In [175…

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch+1, num_epochs))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'test']:
            if phase == 'train':
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
```

```
                    #print(outputs)
                    #print(labels)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)
            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # deep copy the model
            if phase == 'test' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

Visualizing the model predictions ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Generic function to display predictions for a few images

In [176…

```
def visualize_model(model, num_images=6):
    was_training = model.training
    model.eval()
    images_so_far = 0
    fig = plt.figure()

    with torch.no_grad():
        for i, (inputs, labels) in enumerate(dataloaders['test']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)

            m = nn.Softmax(dim=1)
            proOutput = m(outputs)

            _, preds = torch.max(outputs, 1)
            pro = torch.max(proOutput, dim=1)[0]
```

```
            for j in range(inputs.size()[0]):
                images_so_far += 1
                ax = plt.subplot(num_images//2, 2, images_so_far)
                ax.axis('off')
                ax.set_title('predicted: %s with %s probabilities' %(class_names
                imshow(inputs.cpu().data[j])

                if images_so_far == num_images:
                    model.train(mode=was_training)
                    return

        model.train(mode=was_training)

# visualize_model(model_ft)
```

Finetuning the convnet

Load a pretrained model and reset final fully connected layer.

In [177…

```
model_ft = models.resnet50(pretrained=True) #load resnet50.

num_ftrs = model_ft.fc.in_features

# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to nn.Linear(num_ftrs, len(class_names)).
model_ft.fc = nn.Linear(num_ftrs, 5)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

Train and evaluate ^^^^^^^^^^^^^^^^^^

It should take around 15-25 min on CPU. On GPU though, it takes less than a minute.

In [178…

```
ep = 100
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=ep)
```

```
Epoch 1/100
----------
train Loss: 1.3398 Acc: 0.4418
test Loss: 0.3050 Acc: 0.9250

Epoch 2/100
----------
train Loss: 0.8704 Acc: 0.6567
test Loss: 0.1051 Acc: 0.9500

Epoch 3/100
----------
```

```
train Loss: 0.9275 Acc: 0.6179
test Loss: 0.7251 Acc: 0.7750

Epoch 4/100
----------
train Loss: 0.7055 Acc: 0.7493
test Loss: 0.8281 Acc: 0.7750

Epoch 5/100
----------
train Loss: 0.7274 Acc: 0.7075
test Loss: 0.1893 Acc: 0.9000

Epoch 6/100
----------
train Loss: 0.6436 Acc: 0.7522
test Loss: 0.4338 Acc: 0.8250

Epoch 7/100
----------
train Loss: 0.5781 Acc: 0.7791
test Loss: 0.5228 Acc: 0.8500

Epoch 8/100
----------
train Loss: 0.5945 Acc: 0.7642
test Loss: 0.1324 Acc: 0.9500

Epoch 9/100
----------
train Loss: 0.4772 Acc: 0.8030
test Loss: 0.0663 Acc: 0.9750

Epoch 10/100
----------
train Loss: 0.5096 Acc: 0.7881
test Loss: 0.0282 Acc: 1.0000

Epoch 11/100
----------
train Loss: 0.4780 Acc: 0.8149
test Loss: 0.0227 Acc: 1.0000

Epoch 12/100
----------
train Loss: 0.4373 Acc: 0.8269
test Loss: 0.0260 Acc: 1.0000

Epoch 13/100
----------
train Loss: 0.4532 Acc: 0.8209
test Loss: 0.0276 Acc: 1.0000

Epoch 14/100
----------
train Loss: 0.4536 Acc: 0.8030
test Loss: 0.0158 Acc: 1.0000

Epoch 15/100
----------
train Loss: 0.4896 Acc: 0.8149
test Loss: 0.0186 Acc: 1.0000

Epoch 16/100
----------
```

```
train Loss: 0.4470 Acc: 0.8299
test Loss: 0.0256 Acc: 1.0000

Epoch 17/100
----------
train Loss: 0.5030 Acc: 0.7881
test Loss: 0.0680 Acc: 0.9750

Epoch 18/100
----------
train Loss: 0.4385 Acc: 0.8149
test Loss: 0.0236 Acc: 1.0000

Epoch 19/100
----------
train Loss: 0.5030 Acc: 0.7821
test Loss: 0.0430 Acc: 0.9750

Epoch 20/100
----------
train Loss: 0.4982 Acc: 0.7970
test Loss: 0.1376 Acc: 0.9750

Epoch 21/100
----------
train Loss: 0.5306 Acc: 0.7821
test Loss: 0.0248 Acc: 1.0000

Epoch 22/100
----------
train Loss: 0.4762 Acc: 0.8149
test Loss: 0.0837 Acc: 0.9750

Epoch 23/100
----------
train Loss: 0.4689 Acc: 0.7940
test Loss: 0.0253 Acc: 1.0000

Epoch 24/100
----------
train Loss: 0.4698 Acc: 0.8179
test Loss: 0.0563 Acc: 0.9750

Epoch 25/100
----------
train Loss: 0.3928 Acc: 0.8358
test Loss: 0.0493 Acc: 1.0000

Epoch 26/100
----------
train Loss: 0.4547 Acc: 0.8358
test Loss: 0.0269 Acc: 1.0000

Epoch 27/100
----------
train Loss: 0.4519 Acc: 0.8000
test Loss: 0.0192 Acc: 1.0000

Epoch 28/100
----------
train Loss: 0.4360 Acc: 0.8478
test Loss: 0.0305 Acc: 1.0000

Epoch 29/100
----------
```

```
train Loss: 0.4434 Acc: 0.8179
test Loss: 0.0287 Acc: 1.0000

Epoch 30/100
----------
train Loss: 0.3902 Acc: 0.8567
test Loss: 0.0509 Acc: 0.9750

Epoch 31/100
----------
train Loss: 0.4598 Acc: 0.8060
test Loss: 0.0522 Acc: 0.9750

Epoch 32/100
----------
train Loss: 0.4007 Acc: 0.8537
test Loss: 0.0260 Acc: 1.0000

Epoch 33/100
----------
train Loss: 0.4017 Acc: 0.8418
test Loss: 0.0382 Acc: 1.0000

Epoch 34/100
----------
train Loss: 0.4970 Acc: 0.8090
test Loss: 0.0247 Acc: 1.0000

Epoch 35/100
----------
train Loss: 0.4273 Acc: 0.8119
test Loss: 0.0303 Acc: 1.0000

Epoch 36/100
----------
train Loss: 0.4839 Acc: 0.8060
test Loss: 0.0154 Acc: 1.0000

Epoch 37/100
----------
train Loss: 0.3981 Acc: 0.8478
test Loss: 0.0719 Acc: 0.9750

Epoch 38/100
----------
train Loss: 0.4695 Acc: 0.8000
test Loss: 0.0226 Acc: 1.0000

Epoch 39/100
----------
train Loss: 0.4892 Acc: 0.8030
test Loss: 0.0247 Acc: 1.0000

Epoch 40/100
----------
train Loss: 0.4661 Acc: 0.8119
test Loss: 0.0441 Acc: 0.9750

Epoch 41/100
----------
train Loss: 0.3949 Acc: 0.8328
test Loss: 0.0380 Acc: 1.0000

Epoch 42/100
----------
```

```
train Loss: 0.4225 Acc: 0.8060
test Loss: 0.0134 Acc: 1.0000

Epoch 43/100
----------
train Loss: 0.4517 Acc: 0.8239
test Loss: 0.0288 Acc: 1.0000

Epoch 44/100
----------
train Loss: 0.4345 Acc: 0.8299
test Loss: 0.0143 Acc: 1.0000

Epoch 45/100
----------
train Loss: 0.4663 Acc: 0.8119
test Loss: 0.0236 Acc: 1.0000

Epoch 46/100
----------
train Loss: 0.4845 Acc: 0.8060
test Loss: 0.0224 Acc: 1.0000

Epoch 47/100
----------
train Loss: 0.3657 Acc: 0.8537
test Loss: 0.0201 Acc: 1.0000

Epoch 48/100
----------
train Loss: 0.4216 Acc: 0.8358
test Loss: 0.0415 Acc: 0.9750

Epoch 49/100
----------
train Loss: 0.3901 Acc: 0.8478
test Loss: 0.0401 Acc: 1.0000

Epoch 50/100
----------
train Loss: 0.4666 Acc: 0.8090
test Loss: 0.0164 Acc: 1.0000

Epoch 51/100
----------
train Loss: 0.4101 Acc: 0.8299
test Loss: 0.0423 Acc: 0.9750

Epoch 52/100
----------
train Loss: 0.4424 Acc: 0.8149
test Loss: 0.0347 Acc: 1.0000

Epoch 53/100
----------
train Loss: 0.4257 Acc: 0.8328
test Loss: 0.0453 Acc: 0.9750

Epoch 54/100
----------
train Loss: 0.4468 Acc: 0.8299
test Loss: 0.0416 Acc: 1.0000

Epoch 55/100
----------
```

```
train Loss: 0.4668 Acc: 0.8179
test Loss: 0.0304 Acc: 1.0000

Epoch 56/100
----------
train Loss: 0.4548 Acc: 0.8299
test Loss: 0.0408 Acc: 1.0000

Epoch 57/100
----------
train Loss: 0.4568 Acc: 0.8269
test Loss: 0.0430 Acc: 1.0000

Epoch 58/100
----------
train Loss: 0.4754 Acc: 0.7940
test Loss: 0.0244 Acc: 1.0000

Epoch 59/100
----------
train Loss: 0.3855 Acc: 0.8537
test Loss: 0.0284 Acc: 1.0000

Epoch 60/100
----------
train Loss: 0.5235 Acc: 0.7791
test Loss: 0.0166 Acc: 1.0000

Epoch 61/100
----------
train Loss: 0.4076 Acc: 0.8239
test Loss: 0.0196 Acc: 1.0000

Epoch 62/100
----------
train Loss: 0.4648 Acc: 0.8209
test Loss: 0.0316 Acc: 1.0000

Epoch 63/100
----------
train Loss: 0.4373 Acc: 0.8149
test Loss: 0.0332 Acc: 1.0000

Epoch 64/100
----------
train Loss: 0.4087 Acc: 0.8478
test Loss: 0.0356 Acc: 1.0000

Epoch 65/100
----------
train Loss: 0.4779 Acc: 0.8090
test Loss: 0.0384 Acc: 1.0000

Epoch 66/100
----------
train Loss: 0.4280 Acc: 0.8328
test Loss: 0.0221 Acc: 1.0000

Epoch 67/100
----------
train Loss: 0.4314 Acc: 0.8328
test Loss: 0.0281 Acc: 1.0000

Epoch 68/100
----------
```

```
train Loss: 0.4210 Acc: 0.8358
test Loss: 0.0245 Acc: 1.0000

Epoch 69/100
----------
train Loss: 0.4300 Acc: 0.8090
test Loss: 0.0130 Acc: 1.0000

Epoch 70/100
----------
train Loss: 0.4357 Acc: 0.8358
test Loss: 0.0264 Acc: 1.0000

Epoch 71/100
----------
train Loss: 0.3520 Acc: 0.8657
test Loss: 0.0306 Acc: 1.0000

Epoch 72/100
----------
train Loss: 0.3561 Acc: 0.8537
test Loss: 0.0197 Acc: 1.0000

Epoch 73/100
----------
train Loss: 0.4055 Acc: 0.8328
test Loss: 0.0623 Acc: 1.0000

Epoch 74/100
----------
train Loss: 0.4758 Acc: 0.8090
test Loss: 0.0321 Acc: 1.0000

Epoch 75/100
----------
train Loss: 0.4602 Acc: 0.8090
test Loss: 0.0225 Acc: 1.0000

Epoch 76/100
----------
train Loss: 0.4442 Acc: 0.8090
test Loss: 0.0159 Acc: 1.0000

Epoch 77/100
----------
train Loss: 0.3989 Acc: 0.8507
test Loss: 0.0166 Acc: 1.0000

Epoch 78/100
----------
train Loss: 0.4584 Acc: 0.8030
test Loss: 0.0396 Acc: 1.0000

Epoch 79/100
----------
train Loss: 0.4166 Acc: 0.8328
test Loss: 0.0365 Acc: 1.0000

Epoch 80/100
----------
train Loss: 0.4716 Acc: 0.8299
test Loss: 0.0238 Acc: 1.0000

Epoch 81/100
----------
```

```
train Loss: 0.4579 Acc: 0.8239
test Loss: 0.0207 Acc: 1.0000

Epoch 82/100
----------
train Loss: 0.4470 Acc: 0.8448
test Loss: 0.0237 Acc: 1.0000

Epoch 83/100
----------
train Loss: 0.5435 Acc: 0.7672
test Loss: 0.0222 Acc: 1.0000

Epoch 84/100
----------
train Loss: 0.4626 Acc: 0.8060
test Loss: 0.0166 Acc: 1.0000

Epoch 85/100
----------
train Loss: 0.3945 Acc: 0.8388
test Loss: 0.0386 Acc: 1.0000

Epoch 86/100
----------
train Loss: 0.5728 Acc: 0.7701
test Loss: 0.0152 Acc: 1.0000

Epoch 87/100
----------
train Loss: 0.4085 Acc: 0.8418
test Loss: 0.0129 Acc: 1.0000

Epoch 88/100
----------
train Loss: 0.4067 Acc: 0.8328
test Loss: 0.0405 Acc: 1.0000

Epoch 89/100
----------
train Loss: 0.4470 Acc: 0.8030
test Loss: 0.0196 Acc: 1.0000

Epoch 90/100
----------
train Loss: 0.4238 Acc: 0.8060
test Loss: 0.0238 Acc: 1.0000

Epoch 91/100
----------
train Loss: 0.3918 Acc: 0.8358
test Loss: 0.0412 Acc: 1.0000

Epoch 92/100
----------
train Loss: 0.4309 Acc: 0.8179
test Loss: 0.0296 Acc: 1.0000

Epoch 93/100
----------
train Loss: 0.4647 Acc: 0.8000
test Loss: 0.0267 Acc: 1.0000

Epoch 94/100
----------
```

```
train Loss: 0.4790 Acc: 0.7881
test Loss: 0.0435 Acc: 0.9750

Epoch 95/100
----------
train Loss: 0.4127 Acc: 0.8358
test Loss: 0.0272 Acc: 1.0000

Epoch 96/100
----------
train Loss: 0.4572 Acc: 0.7910
test Loss: 0.0275 Acc: 0.9750

Epoch 97/100
----------
train Loss: 0.4621 Acc: 0.8179
test Loss: 0.0601 Acc: 1.0000

Epoch 98/100
----------
train Loss: 0.5279 Acc: 0.7731
test Loss: 0.0624 Acc: 0.9750

Epoch 99/100
----------
train Loss: 0.4654 Acc: 0.8060
test Loss: 0.0160 Acc: 1.0000

Epoch 100/100
----------
train Loss: 0.5067 Acc: 0.7851
test Loss: 0.0159 Acc: 1.0000

Training complete in 10m 16s
Best val Acc: 1.000000
```

In [179…

```
visualize_model(model_ft)
```

predicted: reader with 0.8462 probabilities

predicted: disk with 0.7022 probabilities

```
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x0000
01E430F5E280>
Traceback (most recent call last):
  File "C:\Users\haoyuliao\Anaconda3\envs\pytorch2\lib\site-packages\torch\utils
\data\dataloader.py", line 1324, in __del__
    self._shutdown_workers()
  File "C:\Users\haoyuliao\Anaconda3\envs\pytorch2\lib\site-packages\torch\utils
\data\dataloader.py", line 1291, in _shutdown_workers
    if self._persistent_workers or self._workers_status[worker_id]:
AttributeError: '_MultiProcessingDataLoaderIter' object has no attribute '_worke
rs_status'
```

predicted: disk with 0.9905 probabilities

predicted: disk with 0.9983 probabilities

predicted: reader with 0.9993 probabilities

predicted: hard_disk with 0.9999 probabilities

In [180…

```
correct = 0
total = 0
nb_classes = 5
confusion_matrix = torch.zeros(nb_classes, nb_classes)
with torch.no_grad():
    for data in dataloaders['train']:
        images, labels = data[0].cuda(), data[1].cuda()
        outputs = model_ft(images)
        predicted = torch.round(outputs)
        _, predicted = torch.max(outputs, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        for t, p in zip(labels.view(-1), predicted.view(-1)):
                confusion_matrix[t.long(), p.long()] += 1


print('Accuracy of the network on the %s train images: %d %%' % (total,
    100 * correct / total))

print(confusion_matrix)
print(confusion_matrix/(total/5)) #Normalizing
```

```
Accuracy of the network on the 335 train images: 80 %
tensor([[61.,  0.,  1.,  2.,  3.],
        [10., 57.,  0.,  0.,  0.],
        [14.,  0., 51.,  0.,  2.],
        [16.,  0.,  0., 50.,  1.],
        [16.,  0.,  0.,  0., 51.]])
tensor([[0.9104, 0.0000, 0.0149, 0.0299, 0.0448],
        [0.1493, 0.8507, 0.0000, 0.0000, 0.0000],
        [0.2090, 0.0000, 0.7612, 0.0000, 0.0299],
        [0.2388, 0.0000, 0.0000, 0.7463, 0.0149],
        [0.2388, 0.0000, 0.0000, 0.0000, 0.7612]])
```

In [181…
```
correct = 0
total = 0
nb_classes = 5
confusion_matrix = torch.zeros(nb_classes, nb_classes)
with torch.no_grad():
    for data in dataloaders['test']:
        images, labels = data[0].cuda(), data[1].cuda()
        outputs = model_ft(images)
        predicted = torch.round(outputs)
        _, predicted = torch.max(outputs, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        for t, p in zip(labels.view(-1), predicted.view(-1)):
                confusion_matrix[t.long(), p.long()] += 1


print('Accuracy of the network on the %s train images: %d %%' % (total,
    100 * correct / total))

print(confusion_matrix)
print(confusion_matrix/(total/5)) #Normalizing
```

```
Accuracy of the network on the 40 train images: 100 %
tensor([[8., 0., 0., 0., 0.],
        [0., 8., 0., 0., 0.],
        [0., 0., 8., 0., 0.],
        [0., 0., 0., 8., 0.],
        [0., 0., 0., 0., 8.]])
tensor([[1., 0., 0., 0., 0.],
        [0., 1., 0., 0., 0.],
        [0., 0., 1., 0., 0.],
        [0., 0., 0., 1., 0.],
        [0., 0., 0., 0., 1.]])
```

In [182…
```
PATH = './Resnet50EP%s.pth' %(ep)
torch.save(model_ft.state_dict(), PATH)
```

In [ ]:
```
```