In [44]:
```
%matplotlib inline
#Reference from https://pytorch.org/
```

In [45]:
```
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 [46]:
```
# 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.

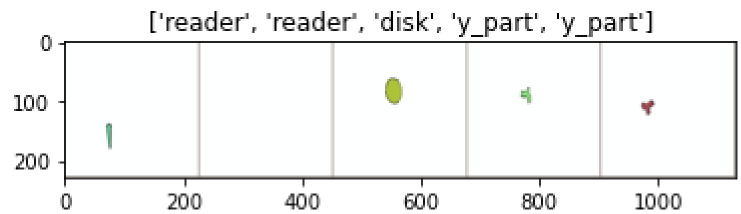
In [47]:
```
def imshow(inp, title=None):
    """Imshow for Tensor."""
```

```
    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 [48]:

```
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+1):
        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 [49]:

```
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)
```

In [50]:

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

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 [51]:

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

```
Epoch 1/100
----------
train Loss: 1.4017 Acc: 0.3731
test Loss: 0.6184 Acc: 0.9000

Epoch 2/100
----------
train Loss: 0.9335 Acc: 0.6448
test Loss: 0.1786 Acc: 1.0000

Epoch 3/100
----------
train Loss: 0.7263 Acc: 0.7343
test Loss: 0.3597 Acc: 0.9000

Epoch 4/100
----------
train Loss: 0.6780 Acc: 0.7373
```

```
test Loss: 0.2399 Acc: 0.9500

Epoch 5/100
----------
train Loss: 0.7448 Acc: 0.7224
test Loss: 0.1258 Acc: 0.9500

Epoch 6/100
----------
train Loss: 0.5884 Acc: 0.7881
test Loss: 0.1478 Acc: 0.9750

Epoch 7/100
----------
train Loss: 0.5671 Acc: 0.7851
test Loss: 0.1594 Acc: 0.9250

Epoch 8/100
----------
train Loss: 0.4945 Acc: 0.8209
test Loss: 0.1164 Acc: 0.9500

Epoch 9/100
----------
train Loss: 0.5618 Acc: 0.8000
test Loss: 0.1472 Acc: 0.9500

Epoch 10/100
----------
train Loss: 0.5479 Acc: 0.7761
test Loss: 0.1134 Acc: 0.9500

Epoch 11/100
----------
train Loss: 0.4440 Acc: 0.8239
test Loss: 0.0949 Acc: 0.9750

Epoch 12/100
----------
train Loss: 0.4722 Acc: 0.8239
test Loss: 0.0931 Acc: 0.9750

Epoch 13/100
----------
train Loss: 0.5079 Acc: 0.8060
test Loss: 0.1562 Acc: 0.9500

Epoch 14/100
----------
train Loss: 0.4357 Acc: 0.8239
test Loss: 0.0846 Acc: 0.9750

Epoch 15/100
----------
train Loss: 0.4635 Acc: 0.8000
test Loss: 0.1140 Acc: 0.9750

Epoch 16/100
----------
train Loss: 0.4659 Acc: 0.8149
test Loss: 0.1201 Acc: 0.9500

Epoch 17/100
----------
train Loss: 0.4606 Acc: 0.8119
```

```
test Loss: 0.1254 Acc: 0.9500

Epoch 18/100
----------
train Loss: 0.4682 Acc: 0.8358
test Loss: 0.0651 Acc: 0.9750

Epoch 19/100
----------
train Loss: 0.4002 Acc: 0.8269
test Loss: 0.1064 Acc: 0.9500

Epoch 20/100
----------
train Loss: 0.4795 Acc: 0.8119
test Loss: 0.0834 Acc: 0.9500

Epoch 21/100
----------
train Loss: 0.4947 Acc: 0.8000
test Loss: 0.0513 Acc: 0.9750

Epoch 22/100
----------
train Loss: 0.5168 Acc: 0.7851
test Loss: 0.1246 Acc: 0.9500

Epoch 23/100
----------
train Loss: 0.4861 Acc: 0.8149
test Loss: 0.0919 Acc: 0.9500

Epoch 24/100
----------
train Loss: 0.4990 Acc: 0.8090
test Loss: 0.0645 Acc: 0.9750

Epoch 25/100
----------
train Loss: 0.4612 Acc: 0.8179
test Loss: 0.1171 Acc: 0.9500

Epoch 26/100
----------
train Loss: 0.4535 Acc: 0.8269
test Loss: 0.0652 Acc: 0.9750

Epoch 27/100
----------
train Loss: 0.4482 Acc: 0.8239
test Loss: 0.0823 Acc: 0.9750

Epoch 28/100
----------
train Loss: 0.4750 Acc: 0.8149
test Loss: 0.0787 Acc: 0.9500

Epoch 29/100
----------
train Loss: 0.5154 Acc: 0.8090
test Loss: 0.0877 Acc: 0.9750

Epoch 30/100
----------
train Loss: 0.4237 Acc: 0.8418
```

```
test Loss: 0.1061 Acc: 0.9500

Epoch 31/100
----------
train Loss: 0.5379 Acc: 0.8030
test Loss: 0.0894 Acc: 0.9750

Epoch 32/100
----------
train Loss: 0.4774 Acc: 0.7940
test Loss: 0.0817 Acc: 0.9500

Epoch 33/100
----------
train Loss: 0.4694 Acc: 0.8388
test Loss: 0.1666 Acc: 0.9500

Epoch 34/100
----------
train Loss: 0.4707 Acc: 0.8060
test Loss: 0.0820 Acc: 0.9750

Epoch 35/100
----------
train Loss: 0.4874 Acc: 0.8030
test Loss: 0.0605 Acc: 0.9750

Epoch 36/100
----------
train Loss: 0.4381 Acc: 0.8299
test Loss: 0.0963 Acc: 0.9750

Epoch 37/100
----------
train Loss: 0.4495 Acc: 0.8179
test Loss: 0.0611 Acc: 0.9750

Epoch 38/100
----------
train Loss: 0.4994 Acc: 0.8060
test Loss: 0.0663 Acc: 0.9750

Epoch 39/100
----------
train Loss: 0.4020 Acc: 0.8388
test Loss: 0.0870 Acc: 0.9500

Epoch 40/100
----------
train Loss: 0.3829 Acc: 0.8776
test Loss: 0.0667 Acc: 0.9750

Epoch 41/100
----------
train Loss: 0.5220 Acc: 0.7851
test Loss: 0.0599 Acc: 0.9750

Epoch 42/100
----------
train Loss: 0.4224 Acc: 0.8657
test Loss: 0.0819 Acc: 0.9500

Epoch 43/100
----------
train Loss: 0.4486 Acc: 0.8358
```

```
test Loss: 0.0881 Acc: 0.9750

Epoch 44/100
----------
train Loss: 0.4235 Acc: 0.8388
test Loss: 0.0684 Acc: 0.9750

Epoch 45/100
----------
train Loss: 0.4774 Acc: 0.8149
test Loss: 0.1151 Acc: 0.9500

Epoch 46/100
----------
train Loss: 0.4644 Acc: 0.8209
test Loss: 0.0812 Acc: 0.9750

Epoch 47/100
----------
train Loss: 0.4678 Acc: 0.8060
test Loss: 0.0717 Acc: 0.9750

Epoch 48/100
----------
train Loss: 0.4548 Acc: 0.8239
test Loss: 0.0958 Acc: 0.9500

Epoch 49/100
----------
train Loss: 0.4416 Acc: 0.8209
test Loss: 0.0657 Acc: 0.9750

Epoch 50/100
----------
train Loss: 0.5141 Acc: 0.8060
test Loss: 0.0987 Acc: 0.9500

Epoch 51/100
----------
train Loss: 0.4723 Acc: 0.8060
test Loss: 0.1170 Acc: 0.9750

Epoch 52/100
----------
train Loss: 0.3933 Acc: 0.8478
test Loss: 0.1033 Acc: 0.9500

Epoch 53/100
----------
train Loss: 0.4597 Acc: 0.8418
test Loss: 0.0489 Acc: 1.0000

Epoch 54/100
----------
train Loss: 0.4260 Acc: 0.8448
test Loss: 0.0563 Acc: 0.9750

Epoch 55/100
----------
train Loss: 0.5065 Acc: 0.8000
test Loss: 0.0831 Acc: 0.9750

Epoch 56/100
----------
train Loss: 0.4218 Acc: 0.8597
```

```
test Loss: 0.0702 Acc: 0.9750

Epoch 57/100
----------
train Loss: 0.5485 Acc: 0.7612
test Loss: 0.1505 Acc: 0.9500

Epoch 58/100
----------
train Loss: 0.3978 Acc: 0.8358
test Loss: 0.0939 Acc: 0.9500

Epoch 59/100
----------
train Loss: 0.4109 Acc: 0.8299
test Loss: 0.0695 Acc: 0.9750

Epoch 60/100
----------
train Loss: 0.4298 Acc: 0.8119
test Loss: 0.0757 Acc: 0.9750

Epoch 61/100
----------
train Loss: 0.3900 Acc: 0.8716
test Loss: 0.1400 Acc: 0.9500

Epoch 62/100
----------
train Loss: 0.4980 Acc: 0.8149
test Loss: 0.1058 Acc: 0.9750

Epoch 63/100
----------
train Loss: 0.4980 Acc: 0.8030
test Loss: 0.0542 Acc: 0.9750

Epoch 64/100
----------
train Loss: 0.4765 Acc: 0.8149
test Loss: 0.1409 Acc: 0.9500

Epoch 65/100
----------
train Loss: 0.5245 Acc: 0.7672
test Loss: 0.0915 Acc: 0.9750

Epoch 66/100
----------
train Loss: 0.5601 Acc: 0.7791
test Loss: 0.0549 Acc: 0.9750

Epoch 67/100
----------
train Loss: 0.5168 Acc: 0.8090
test Loss: 0.0531 Acc: 0.9750

Epoch 68/100
----------
train Loss: 0.4811 Acc: 0.8179
test Loss: 0.0639 Acc: 0.9750

Epoch 69/100
----------
train Loss: 0.5156 Acc: 0.7910
```

```
test Loss: 0.0977 Acc: 0.9750

Epoch 70/100
----------
train Loss: 0.4874 Acc: 0.8149
test Loss: 0.1453 Acc: 0.9500

Epoch 71/100
----------
train Loss: 0.5076 Acc: 0.7881
test Loss: 0.0747 Acc: 0.9750

Epoch 72/100
----------
train Loss: 0.5141 Acc: 0.8000
test Loss: 0.0813 Acc: 0.9500

Epoch 73/100
----------
train Loss: 0.3744 Acc: 0.8806
test Loss: 0.0533 Acc: 0.9750

Epoch 74/100
----------
train Loss: 0.4440 Acc: 0.8179
test Loss: 0.0951 Acc: 0.9500

Epoch 75/100
----------
train Loss: 0.4049 Acc: 0.8537
test Loss: 0.0815 Acc: 0.9500

Epoch 76/100
----------
train Loss: 0.4421 Acc: 0.8179
test Loss: 0.1369 Acc: 0.9500

Epoch 77/100
----------
train Loss: 0.4281 Acc: 0.8507
test Loss: 0.0887 Acc: 0.9500

Epoch 78/100
----------
train Loss: 0.4644 Acc: 0.8209
test Loss: 0.0657 Acc: 0.9750

Epoch 79/100
----------
train Loss: 0.4853 Acc: 0.7970
test Loss: 0.0515 Acc: 0.9750

Epoch 80/100
----------
train Loss: 0.4388 Acc: 0.8149
test Loss: 0.1162 Acc: 0.9500

Epoch 81/100
----------
train Loss: 0.4514 Acc: 0.8269
test Loss: 0.1393 Acc: 0.9500

Epoch 82/100
----------
train Loss: 0.5075 Acc: 0.7761
```

```
test Loss: 0.0873 Acc: 0.9500

Epoch 83/100
----------
train Loss: 0.4416 Acc: 0.8090
test Loss: 0.0798 Acc: 0.9750

Epoch 84/100
----------
train Loss: 0.5856 Acc: 0.7612
test Loss: 0.0809 Acc: 0.9500

Epoch 85/100
----------
train Loss: 0.4325 Acc: 0.8478
test Loss: 0.0994 Acc: 0.9500

Epoch 86/100
----------
train Loss: 0.4381 Acc: 0.8149
test Loss: 0.0671 Acc: 0.9750

Epoch 87/100
----------
train Loss: 0.4478 Acc: 0.8358
test Loss: 0.0757 Acc: 0.9750

Epoch 88/100
----------
train Loss: 0.4520 Acc: 0.8239
test Loss: 0.0743 Acc: 0.9750

Epoch 89/100
----------
train Loss: 0.5353 Acc: 0.7701
test Loss: 0.1447 Acc: 0.9500

Epoch 90/100
----------
train Loss: 0.5090 Acc: 0.7940
test Loss: 0.0821 Acc: 0.9750

Epoch 91/100
----------
train Loss: 0.4820 Acc: 0.8119
test Loss: 0.1073 Acc: 0.9750

Epoch 92/100
----------
train Loss: 0.4932 Acc: 0.8149
test Loss: 0.1141 Acc: 0.9500

Epoch 93/100
----------
train Loss: 0.4638 Acc: 0.8328
test Loss: 0.1358 Acc: 0.9500

Epoch 94/100
----------
train Loss: 0.4901 Acc: 0.8000
test Loss: 0.0451 Acc: 0.9750

Epoch 95/100
----------
train Loss: 0.4935 Acc: 0.8060
```

```
test Loss: 0.1025 Acc: 0.9750

Epoch 96/100
----------
train Loss: 0.4942 Acc: 0.8000
test Loss: 0.0916 Acc: 0.9500

Epoch 97/100
----------
train Loss: 0.4327 Acc: 0.8299
test Loss: 0.1004 Acc: 0.9750

Epoch 98/100
----------
train Loss: 0.4503 Acc: 0.8388
test Loss: 0.1699 Acc: 0.9500

Epoch 99/100
----------
train Loss: 0.4646 Acc: 0.8179
test Loss: 0.0923 Acc: 0.9750

Epoch 100/100
----------
train Loss: 0.4155 Acc: 0.8567
test Loss: 0.0797 Acc: 0.9750

Epoch 101/100
----------
train Loss: 0.4514 Acc: 0.8328
test Loss: 0.0408 Acc: 1.0000

Training complete in 7m 52s
Best val Acc: 1.000000
```

In [52]:

```
visualize_model(model_ft)
```

predicted: hard_disk with 0.977 probabilities

predicted: hard_disk with 0.821 probabilities

predicted: reader with 0.9765 probabilities

predicted: chip with 0.9415 probabilities predicted: chip with 0.96 probabilities

predicted: y_part with 0.5157 probabilities

In [53]:

```
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: 82 %
tensor([[66.,  0.,  0.,  1.,  0.],
        [13., 54.,  0.,  0.,  0.],
        [ 5.,  0., 62.,  0.,  0.],
        [16.,  0.,  0., 51.,  0.],
        [19.,  0.,  0.,  3., 45.]])
tensor([[0.9851, 0.0000, 0.0000, 0.0149, 0.0000],
        [0.1940, 0.8060, 0.0000, 0.0000, 0.0000],
        [0.0746, 0.0000, 0.9254, 0.0000, 0.0000],
        [0.2388, 0.0000, 0.0000, 0.7612, 0.0000],
        [0.2836, 0.0000, 0.0000, 0.0448, 0.6716]])
```

In [54]:

```
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 test images: %d %%' % (total,
    100 * correct / total))

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

```
Accuracy of the network on the 40 test 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 [55]:

```
PATH = './GoogLeNetEP%s.pth' %(ep)
torch.save(model_ft.state_dict(), PATH)
```

In [ ]:

```

```