In [1]:

```
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 [2]:

```
# 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])
    ]),
    'val': 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 = 'hymenoptera_data'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                             shuffle=True, num_workers=4)
              for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

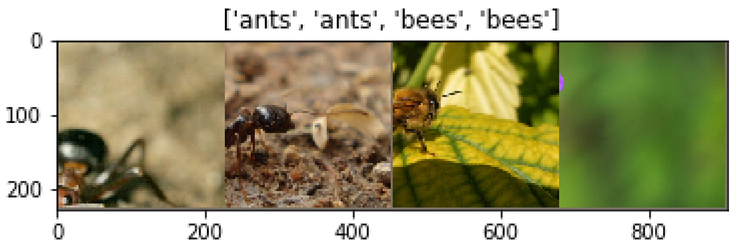
In [3]:

```
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 [4]:

```
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, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            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)
                    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 == 'val' 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
```

In [5]:

```
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['val']):
            inputs = inputs.to(device)
            labels = labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)

            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: {}'.format(class_names[preds[j]]))
                imshow(inputs.cpu().data[j])

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

In [11]:

```
model_ft = models.resnet18(pretrained=False)
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, 2)

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)
```

In [12]:

```
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler,
                       num_epochs=25)
```

```
Epoch 0/24
----------
train Loss: 0.6961 Acc: 0.5775
val Loss: 0.9642 Acc: 0.6078

Epoch 1/24
----------
train Loss: 0.8590 Acc: 0.4718
val Loss: 1.3384 Acc: 0.4575

Epoch 2/24
----------
train Loss: 0.8101 Acc: 0.5915
val Loss: 0.9814 Acc: 0.5556

Epoch 3/24
----------
train Loss: 0.8271 Acc: 0.5211
val Loss: 0.8839 Acc: 0.5359

Epoch 4/24
----------
train Loss: 0.7880 Acc: 0.5775
val Loss: 0.7381 Acc: 0.5556

Epoch 5/24
----------
train Loss: 0.6780 Acc: 0.6338
val Loss: 0.7956 Acc: 0.7059

Epoch 6/24
----------
train Loss: 0.7670 Acc: 0.6268
val Loss: 1.4656 Acc: 0.4837

Epoch 7/24
----------
train Loss: 0.7693 Acc: 0.6408
val Loss: 0.6422 Acc: 0.6863

Epoch 8/24
----------
train Loss: 0.6242 Acc: 0.6408
val Loss: 0.6096 Acc: 0.6536

Epoch 9/24
----------
train Loss: 0.6475 Acc: 0.6338
val Loss: 0.7052 Acc: 0.6471

Epoch 10/24
----------
train Loss: 0.6437 Acc: 0.6338
val Loss: 0.6250 Acc: 0.6471

Epoch 11/24
----------
train Loss: 0.5679 Acc: 0.7183
val Loss: 0.6450 Acc: 0.6732

Epoch 12/24
```

```
----------
train Loss: 0.5846 Acc: 0.6831
val Loss: 0.6418 Acc: 0.6667

Epoch 13/24
----------
train Loss: 0.5985 Acc: 0.6408
val Loss: 0.6527 Acc: 0.6405

Epoch 14/24
----------
train Loss: 0.5768 Acc: 0.7042
val Loss: 0.6304 Acc: 0.6667

Epoch 15/24
----------
train Loss: 0.5921 Acc: 0.6620
val Loss: 0.6235 Acc: 0.6405

Epoch 16/24
----------
train Loss: 0.5195 Acc: 0.7676
val Loss: 0.6219 Acc: 0.6732

Epoch 17/24
----------
train Loss: 0.5319 Acc: 0.7394
val Loss: 0.6333 Acc: 0.6732

Epoch 18/24
----------
train Loss: 0.5614 Acc: 0.6972
val Loss: 0.6465 Acc: 0.6863

Epoch 19/24
----------
train Loss: 0.5807 Acc: 0.6972
val Loss: 0.6211 Acc: 0.6601

Epoch 20/24
----------
train Loss: 0.5924 Acc: 0.6620
val Loss: 0.6099 Acc: 0.6667

Epoch 21/24
----------
train Loss: 0.5754 Acc: 0.6761
val Loss: 0.6295 Acc: 0.6797

Epoch 22/24
----------
train Loss: 0.5881 Acc: 0.7042
val Loss: 0.6428 Acc: 0.6601

Epoch 23/24
----------
train Loss: 0.5793 Acc: 0.6831
val Loss: 0.6342 Acc: 0.6471

Epoch 24/24
----------
```

```
train Loss: 0.5804 Acc: 0.7113
val Loss: 0.6430 Acc: 0.6601

Training complete in 0m 48s
Best val Acc: 0.705882
```

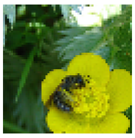
In [8]:

```
visualize_model(model_ft)
```

predicted: bees

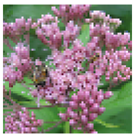
predicted: ants

predicted: bees



predicted: ants

predicted: bees



predicted: bees

In [9]:

```
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that only parameters of final layer are being optimized as
# opposed to before.
optimizer_conv = optim.SGD(model_conv.fc.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_conv, step_size=7, gamma=0.1)
```

In [10]:

```
model_conv = train_model(model_conv, criterion, optimizer_conv,
                         exp_lr_scheduler, num_epochs=25)
```

```
Epoch 0/24
----------
train Loss: 1.0234 Acc: 0.5986
val Loss: 0.4431 Acc: 0.7712

Epoch 1/24
----------
train Loss: 0.7238 Acc: 0.6690
val Loss: 0.4281 Acc: 0.8039

Epoch 2/24
----------
train Loss: 0.6133 Acc: 0.7465
val Loss: 0.2407 Acc: 0.9150

Epoch 3/24
----------
train Loss: 0.4530 Acc: 0.8099
val Loss: 0.3934 Acc: 0.8497

Epoch 4/24
----------
train Loss: 0.3517 Acc: 0.8451
val Loss: 0.2393 Acc: 0.9281

Epoch 5/24
----------
train Loss: 0.5462 Acc: 0.7535
val Loss: 0.2099 Acc: 0.9346

Epoch 6/24
----------
train Loss: 0.4382 Acc: 0.7817
val Loss: 0.2176 Acc: 0.9346

Epoch 7/24
----------
train Loss: 0.3708 Acc: 0.8592
val Loss: 0.1908 Acc: 0.9281

Epoch 8/24
----------
train Loss: 0.3224 Acc: 0.8662
val Loss: 0.1833 Acc: 0.9346

Epoch 9/24
----------
train Loss: 0.3905 Acc: 0.8239
val Loss: 0.1866 Acc: 0.9346

Epoch 10/24
----------
train Loss: 0.4563 Acc: 0.7887
val Loss: 0.2013 Acc: 0.9150

Epoch 11/24
----------
train Loss: 0.3729 Acc: 0.8592
val Loss: 0.1897 Acc: 0.9281

Epoch 12/24
```

```
----------
train Loss: 0.4144 Acc: 0.8028
val Loss: 0.2134 Acc: 0.9346

Epoch 13/24
----------
train Loss: 0.3654 Acc: 0.8169
val Loss: 0.1856 Acc: 0.9216

Epoch 14/24
----------
train Loss: 0.3644 Acc: 0.8310
val Loss: 0.1895 Acc: 0.9346

Epoch 15/24
----------
train Loss: 0.2761 Acc: 0.8732
val Loss: 0.1815 Acc: 0.9477

Epoch 16/24
----------
train Loss: 0.3038 Acc: 0.8873
val Loss: 0.1990 Acc: 0.9346

Epoch 17/24
----------
train Loss: 0.3736 Acc: 0.8169
val Loss: 0.1909 Acc: 0.9346

Epoch 18/24
----------
train Loss: 0.2432 Acc: 0.9155
val Loss: 0.1891 Acc: 0.9281

Epoch 19/24
----------
train Loss: 0.2362 Acc: 0.9014
val Loss: 0.1840 Acc: 0.9346

Epoch 20/24
----------
train Loss: 0.3469 Acc: 0.8451
val Loss: 0.1934 Acc: 0.9281

Epoch 21/24
----------
train Loss: 0.3184 Acc: 0.8662
val Loss: 0.1853 Acc: 0.9346

Epoch 22/24
----------
train Loss: 0.3869 Acc: 0.7958
val Loss: 0.1799 Acc: 0.9346

Epoch 23/24
----------
train Loss: 0.2730 Acc: 0.8662
val Loss: 0.1852 Acc: 0.9281

Epoch 24/24
----------
```

```
train Loss: 0.3820 Acc: 0.8099
val Loss: 0.1809 Acc: 0.9412

Training complete in 0m 38s
Best val Acc: 0.947712
```

In [12]:

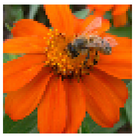
```
visualize_model(model_conv)

plt.ioff()
plt.show()
```

predicted: ants

predicted: bees

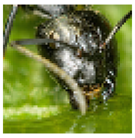
predicted: bees



predicted: ants

predicted: ants

predicted: ants



In [ ]:

```

```