```
%matplotlib inline
import numpy as np
import torch
import torchvision
import torchvision.transforms as transforms
from pathlib import Path
import os
import string
import math
from __future__ import print_function, division
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import copy
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
path = Path('/content/drive/My Drive/ml_hw5_q4/data')
```

```
cd /content/drive/My Drive/ml_hw5_q4/data
```

```
/content/drive/My Drive/ml_hw5_q4/data
```

Data setup

```
transforms = transforms.Compose([
transforms.Resize(256),
transforms.CenterCrop(224),
transforms.ToTensor(),
transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])])
```

```
image_datasets = {x: datasets.ImageFolder(os.path.join(path, x), transforms) for x in ['tr
```

```
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4, shuffle=Tru
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")
```

```
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)

    inputs, classes = next(iter(dataloaders['train']))
    out = torchvision.utils.make_grid(inputs)

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

Training the model

```
train_loss = []
val_loss = []
train_acc = []
val_acc = []

def train_model(model, criterion, optimizer, scheduler, num_epochs=20):
    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)
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

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

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
```

```
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                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]

            if phase == 'train':
              train_loss.append(epoch_loss)
              train_acc.append(epoch_acc)
            else:
              val_loss.append(epoch_loss)
              val_acc.append(epoch_acc)

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

            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))

    model.load_state_dict(best_model_wts)
    return model


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 = 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)


model_ft = models.resnet34(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 2)
model_ft = model_ft.to(device)
criterion = nn.CrossEntropyLoss()
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)


model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=20)
```

```
Epoch 6/19
----------
train Loss: 0.2066 Acc: 0.9333
val Loss: 0.0032 Acc: 1.0000

Epoch 7/19
----------
train Loss: 0.0525 Acc: 0.9833
val Loss: 0.0053 Acc: 1.0000

Epoch 8/19
----------
train Loss: 0.0858 Acc: 0.9667
val Loss: 0.0035 Acc: 1.0000

Epoch 9/19
----------
train Loss: 0.0216 Acc: 1.0000
val Loss: 0.0035 Acc: 1.0000

Epoch 10/19
----------
train Loss: 0.1701 Acc: 0.9333
val Loss: 0.0015 Acc: 1.0000

Epoch 11/19
----------
train Loss: 0.0733 Acc: 0.9833
val Loss: 0.0019 Acc: 1.0000

Epoch 12/19
----------
train Loss: 0.1684 Acc: 0.9167
val Loss: 0.0024 Acc: 1.0000

Epoch 13/19
----------
train Loss: 0.1413 Acc: 0.9500
val Loss: 0.0020 Acc: 1.0000

Epoch 14/19
----------
train Loss: 0.1413 Acc: 0.9833
val Loss: 0.0019 Acc: 1.0000

Epoch 15/19
----------
train Loss: 0.1341 Acc: 0.9333
val Loss: 0.0023 Acc: 1.0000

Epoch 16/19
----------
train Loss: 0.0777 Acc: 0.9667
val Loss: 0.0023 Acc: 1.0000

Epoch 17/19
----------
train Loss: 0.0762 Acc: 0.9500
val Loss: 0.0036 Acc: 1.0000

Epoch 18/19
```

```
Epoch 18/19
----------
train Loss: 0.0846 Acc: 0.9667
val Loss: 0.0039 Acc: 1.0000

Epoch 19/19
----------
train Loss: 0.2311 Acc: 0.9167
val Loss: 0.0044 Acc: 1.0000

Training complete in 0m 28s
Best val Acc: 1.000000
```

```
visualize_model(model_ft)
```

predicted: dog

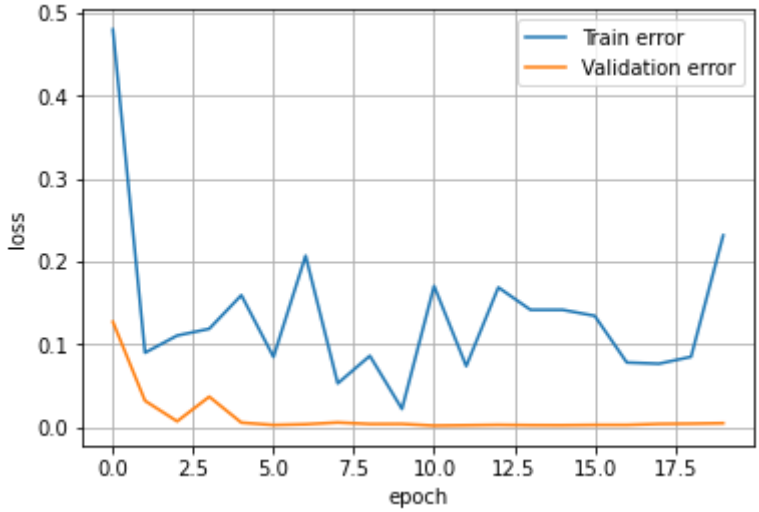
predicted: dog

predicted: dog

- Training Loss at 20th epoch = 0.2311
- Training Accuracy at 20th epoch: 0.9167
- Validation Loss at 20th epoch = 0.0004
- Validation Accuracy at 20th epoch = 1.0000

```
epochs = np.arange(0,20)
plt.plot(epochs, train_loss,label='Train error')
plt.plot(epochs, val_loss,label='Validation error')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.grid(True)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f8ab6fd24e0>
```



```
plt.plot(epochs, train_acc,label='Train accuracy')
plt.plot(epochs, val_acc, label='Validation accuracy')
plt.xlabel('epoch')
plt.ylabel('Accuracy')
plt.grid(True)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f8ab6f5dc88>
```