09_PyTorch_TransferLearning_Manu

January 14, 2020

1 Assignment 9

Neural need huge amount of data to be able to perform well. Huge amount of data means huge computation power... To bypass the fact that we don't own this kind of machine, we will us the transfer leaning. By using a pretrained network (usually on ImageNet) and train it a little bit, we can avoid most of the computationnal power needed to perform our task. We will work on the ResNet network (https://arxiv.org/pdf/1512.03385.pdf) designed in 2014. Then, because we know you all have a degree in medicine, we will try our luck bu doing some! We will then retrain the last layer of the network to be able to recognize leopard and cheetah. The cheetah mini database is in the image folder.

[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
```

[2]:
```
data_dir = 'cheetah_data_mini'
TRAIN = 'train'
TEST = 'val'

# ResNet Takes 224x224 images as input, so we resize all of them
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])
    ]),
}

image_datasets = {
    x: datasets.ImageFolder(
        os.path.join(data_dir, x),
        transform=data_transforms[x]
    )
    for x in [TRAIN, TEST]
}

dataloaders = {
    x: torch.utils.data.DataLoader(
        image_datasets[x], batch_size=4,
        shuffle=True, num_workers=4
    )
    for x in [TRAIN, TEST]
}

dataset_sizes = {x: len(image_datasets[x]) for x in [TRAIN, TEST]}

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

for x in [TRAIN, TEST]:
    print("Loaded {} images under {}".format(dataset_sizes[x], x))

print("Classes: ")
class_names = image_datasets[TRAIN].classes
print(image_datasets[TRAIN].classes)
```

```
Loaded 450 images under train
Loaded 162 images under val
Classes:
['cheetah', 'leopard', 'unknown']
```

[3]:
```
def imshow(inp, title=None):
    inp = inp.numpy().transpose((1, 2, 0))
    # plt.figure(figsize=(10, 10))
    plt.axis('off')
    plt.imshow(inp)
```

```
    if title is not None:
        plt.title(title)
    plt.pause(0.001)

def show_databatch(inputs, classes):
    out = torchvision.utils.make_grid(inputs)
    print(out.shape)
    print(out[:,:,15].shape)
    imshow(out, title=[class_names[x] for x in classes])
"""
# Get a batch of training data
inputs, classes = next(iter(dataloaders[TRAIN]))
show_databatch(inputs, classes)
"""
```

[3]:
```
'\n# Get a batch of training data\ninputs, classes =
next(iter(dataloaders[TRAIN]))\nshow_databatch(inputs, classes)\n'
```

1.1 Pretrained network

Now load a network pre-trained on Imagenet and classify the validation data. You can import a pretrained model directly from pytorch with models.resnet18(pretrained=True). The labels are already used in ImageNet so try to recognize the database directly using the output of the pretrained network on the validation database.

[4]:
```
# NET
model_vanilla = torchvision.models.resnet18(pretrained=True)
```

[5]:
```
for param in model_vanilla.parameters():
    param.requires_grad = False
```

[6]:
```
# cheetah = 293; leopard = 288;
```

[7]:
```
# TEST THE NET ON OUR DATASET
def switch_class_idx(predicted):
    for x in range(predicted.shape[0]):
        if predicted[x] == 293:
            predicted[x] = 0
        elif predicted[x] == 288:
            predicted[x] = 1
        else:
            predicted[x] = 2
    return predicted

def val_pretrained(model):
    correct = 0
```

```
    total = 0
    with torch.no_grad():
        for data in dataloaders[TEST]:
            images, labels = data
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            #print(predicted)
            predicted = switch_class_idx(predicted)
            #print(labels, predicted)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print('Accuracy: %d %%' % (
        100 * correct / total))

val_pretrained(model_vanilla)
```

```
Accuracy: 33 %
```

1.2 Transfer learning

The pre-trained network can now be further trained with our data. Replace the last layer in the network with a fully connected Layer with 3 outputs for our classes cheetah, leopard and unknown. Then train the last layer of the network.

[13]:
```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

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

        model.train()

        running_loss = 0.0
        running_corrects = 0

        # Iterate over data.
        for inputs, labels in dataloaders[TRAIN]:

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward
            # track history if only in train
            with torch.set_grad_enabled(True):
                outputs = model(inputs)
```

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

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            scheduler.step()

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

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

        print()

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

    return model
```

[21]:
```
# FEATURE EXTRACTING
model_extract = torchvision.models.resnet18(pretrained=True)
for param in model_extract.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_extract.fc.in_features
model_extract.fc = nn.Linear(num_ftrs, 3)

criterion = nn.CrossEntropyLoss()
optimizer_ext = optim.SGD(model_extract.fc.parameters(), lr=.0001, momentum=0.9)

# Decay LR by a factor
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ext, step_size=5, gamma=0.1)
```

[22]:
```
model_extract = train_model(model_extract, criterion, optimizer_ext,
                            exp_lr_scheduler, num_epochs=25)
```

```
Epoch 1/25
----------
TRAIN Loss: 1.2268 Acc: 0.2889
```

```
Epoch 2/25
----------
TRAIN Loss: 1.2153 Acc: 0.3289

Epoch 3/25
----------
TRAIN Loss: 1.2009 Acc: 0.3156

Epoch 4/25
----------
TRAIN Loss: 1.2084 Acc: 0.2933

Epoch 5/25
----------
TRAIN Loss: 1.2089 Acc: 0.3067

Epoch 6/25
----------
TRAIN Loss: 1.1987 Acc: 0.3067

Epoch 7/25
----------
TRAIN Loss: 1.2023 Acc: 0.3244

Epoch 8/25
----------
TRAIN Loss: 1.1819 Acc: 0.3378

Epoch 9/25
----------
TRAIN Loss: 1.1965 Acc: 0.3333

Epoch 10/25
----------
TRAIN Loss: 1.1957 Acc: 0.3289

Epoch 11/25
----------
TRAIN Loss: 1.1842 Acc: 0.3178

Epoch 12/25
----------
TRAIN Loss: 1.2038 Acc: 0.3044

Epoch 13/25
----------
TRAIN Loss: 1.2124 Acc: 0.3089
```

```
Epoch 14/25
----------
TRAIN Loss: 1.2141 Acc: 0.3044

Epoch 15/25
----------
TRAIN Loss: 1.2043 Acc: 0.3000

Epoch 16/25
----------
TRAIN Loss: 1.1878 Acc: 0.3111

Epoch 17/25
----------
TRAIN Loss: 1.1792 Acc: 0.3111

Epoch 18/25
----------
TRAIN Loss: 1.2050 Acc: 0.3267

Epoch 19/25
----------
TRAIN Loss: 1.2055 Acc: 0.2911

Epoch 20/25
----------
TRAIN Loss: 1.2051 Acc: 0.3200

Epoch 21/25
----------
TRAIN Loss: 1.2105 Acc: 0.2933

Epoch 22/25
----------
TRAIN Loss: 1.1825 Acc: 0.3378

Epoch 23/25
----------
TRAIN Loss: 1.1880 Acc: 0.3089

Epoch 24/25
----------
TRAIN Loss: 1.2029 Acc: 0.3000

Epoch 25/25
----------
TRAIN Loss: 1.2045 Acc: 0.3111
```

```
Training complete in 27m 45s
```

[23]:
```python
def val_extract(model):
    correct = 0
    total = 0
    with torch.no_grad():
        for data in dataloaders[TEST]:
            images, labels = data
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            #print(labels, predicted)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print('Accuracy: %d %%' % (
        100 * correct / total))

val_extract(model_extract)
```

```
Accuracy: 29 %
```

[ ]: