50.039 Theory and Practice of Deep Learning | Coding Homework 4 - Data Loading and Augmentation with CNNs

Joel Huang, 1002530

Problem 1

The `ImageNetDataset` custom class is written in `dataloader.py`. It provides the following functionality:

1. Load paths to images, and load labels from a .csv file
2. Return the length of the dataset
3. Provides access through `__getitem__`. This returns a transformed `dict` with the image and label.
4. Support for grayscale images, which are repeated along the channel axis to get a 3-channel image.

In [1]:
```
import dataloader
%load_ext autoreload
%autoreload 2

import torch
from torchvision import transforms
import torchvision.models as models
from torch.utils.data import DataLoader
```

With normalize

In [2]:

```
transform = transforms.Compose([transforms.CenterCrop(224),
                                transforms.ToTensor(),
                                transforms.Normalize(mean=[0.485, 0.456, 0.
406],
                                                     std=[0.229, 0.224, 0.2
25])])
dataset = dataloader.ImageNetDataset('data/imagespart', 'data.csv',
                                     crop_size=224,
                                     transform=transform)
dataset_loader = DataLoader(dataset,
                            batch_size=4,
                            shuffle=True,
                            num_workers=4)

device = torch.device("cuda")
model = models.resnet18(pretrained=True).to(device)
model.eval()

correct = 0
with torch.no_grad():
    for _, batch in enumerate(dataset_loader):
        images, labels = batch['image'], batch['label']
        images, labels = images.to(device), labels.to(device)
        output = model(images)
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(labels.view_as(pred)).sum().item()

print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(dataset
_loader.dataset),
                                                       100. * correct / len
(dataset_loader.dataset)))
```

```
Test set: Accuracy: 1615/2500 (65%)
```

Without normalize

In [2]:
```
transform = transforms.Compose([transforms.CenterCrop(224),
                                transforms.ToTensor()])

dataset = dataloader.ImageNetDataset('data/imagespart', 'data.csv',
                                     crop_size=224,
                                     transform=transform)
dataset_loader = DataLoader(dataset,
                            batch_size=4,
                            shuffle=True,
                            num_workers=4)

device = torch.device("cuda")
model = models.resnet18(pretrained=True).to(device)
model.eval()

correct = 0
with torch.no_grad():
    for _, batch in enumerate(dataset_loader):
        images, labels = batch['image'], batch['label']
        images, labels = images.to(device), labels.to(device)
        output = model(images)
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(labels.view_as(pred)).sum().item()

print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(dataset
_loader.dataset),
                                                       100. * correct / len
(dataset_loader.dataset)))
```

```
Test set: Accuracy: 987/2500 (39%)
```

Problem 2

1. According to the PyTorch docs, `FiveCrop()` returns a 5-tuple of tensors. So we need to write lambda functions for the next transforms in the pipeline to deal with this shape.
2. We can consider each of the 5 crops as separate images. So now the crops (5) and batch size (4) flatten to (20,).

In [40]:

```
# Following the method at
# https://pytorch.org/docs/stable/torchvision/transforms.html#torchvision.t
ransforms.FiveCrop

transform = transforms.Compose([transforms.FiveCrop(224),
                lambda crops: torch.stack([transforms.ToTensor()(crop) for
crop in crops]),
                lambda norms: torch.stack([transforms.Normalize(mean=[0.485
, 0.456, 0.406],
                                                                std=[0.229,
0.224, 0.225])(norm) for norm in norms])])

dataset = dataloader.ImageNetDataset('data/imagespart', 'data.csv',
                                     crop_size=280,
                                     transform=transform)
dataset_loader = DataLoader(dataset,
                            batch_size=4,
                            shuffle=True,
                            num_workers=4)

device = torch.device("cuda")
model = models.resnet18(pretrained=True).to(device)
model.eval()

correct = 0
with torch.no_grad():
    for _, batch in enumerate(dataset_loader):
        batched_fives = batch['image'].to(device)
        labels = batch['label'].to(device)

        batch_size, num_crops, c, h, w = batched_fives.size()

        # flatten over batch and five crops
        stacked_fives = batched_fives.view(-1, c, h, w)

        result = model(stacked_fives)
        result_avg = result.view(batch_size, num_crops, -1).mean(1) # avg o
ver crops
        pred = result_avg.argmax(dim=1, keepdim=True)
        correct += pred.eq(labels.view_as(pred)).sum().item()

print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(dataset
_loader.dataset),
                                                      100. * correct / len
(dataset_loader.dataset)))
```

```
Test set: Accuracy: 1718/2500 (69%)
```

Mirroring as an augmentation method

Mirroring is a bad augmentation when the viewpoint of the object matters greatly. For example, OCR tasks, where individual characters need to be detected and classified. A 'p' might become a 'q' when mirrored!!!

Problem 3

We need to compose the transforms including fivecrop.

1. According to the PyTorch docs, `FiveCrop()` returns a 5-tuple of tensors. So we need to write lambda functions for the next transforms in the pipeline to deal with this shape.
2. In this PyTorch ver (1.0.0) ResNet does not have an `AdaptiveAvgPool` layer. We allow ResNet to accept a 330x330 image by replacing the `model.avgpool`.

Variable input size for Resnet18

In [50]:
```
transform = transforms.Compose([transforms.FiveCrop(330),
                lambda crops: torch.stack([transforms.ToTensor()(crop) for
crop in crops]),
                lambda norms: torch.stack([transforms.Normalize(mean=[0.485
, 0.456, 0.406],
                                                                  std=[0.229,
0.224, 0.225])(norm) for norm in norms])])

dataset = dataloader.ImageNetDataset('data/imagespart', 'data.csv',
                                     crop_size=330,
                                     transform=transform)

dataset_loader = DataLoader(dataset,
                            batch_size=4,
                            shuffle=True,
                            num_workers=4)

device = torch.device("cuda")
model = models.resnet18(pretrained=True).to(device)
model.avgpool = torch.nn.AdaptiveAvgPool2d((1,1))
model.eval()

correct = 0
with torch.no_grad():
    for _, batch in enumerate(dataset_loader):
        batched_fives = batch['image']
        labels = batch['label'].to(device)

        batch_size, num_crops, c, h, w = batched_fives.size()

        # flatten over batch and five crops
        stacked_fives = batched_fives.view(-1, c, h, w).to(device)

        result = model(stacked_fives)
        result_avg = result.view(batch_size, num_crops, -1).mean(1) # avg o
ver crops
        pred = result_avg.argmax(dim=1, keepdim=True)
        correct += pred.eq(labels.view_as(pred)).sum().item()

print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(dataset
_loader.dataset),
                                                         100. * correct / len
(dataset_loader.dataset)))
```

```
Test set: Accuracy: 1696/2500 (68%)
```

Variable input size for Densenet121

We need to modify the forward pass here to use `F.adaptive_avg_pool2d`. We subclass `DenseNet` for convenience.

In [61]:
```
import re
import torch.nn.functional as F
import torch.utils.model_zoo as model_zoo

model_urls = {
    'densenet121': 'https://download.pytorch.org/models/densenet121-a639ec9
7.pth',
    'densenet169': 'https://download.pytorch.org/models/densenet169-b2777c0
a.pth',
    'densenet201': 'https://download.pytorch.org/models/densenet201-c110357
1.pth',
    'densenet161': 'https://download.pytorch.org/models/densenet161-8d451a5
0.pth',
}

class DenseNetModified(models.DenseNet):
    def forward(self, x):
        features = self.features(x)
        out = F.relu(features, inplace=True)
        out = F.adaptive_avg_pool2d(out, (1, 1)).view(features.size(0), -1)
        out = self.classifier(out)
        return out

def densenet121(pretrained=False, **kwargs):
    r"""Densenet-121 model from
    `"Densely Connected Convolutional Networks" <https://arxiv.org/pdf/1608
.06993.pdf>`_

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
    """
    model = DenseNetModified(num_init_features=64, growth_rate=32, block_co
nfig=(6, 12, 24, 16),
                     **kwargs)
    if pretrained:
        # '.'s are no longer allowed in module names, but pervious _DenseLa
yer
        # has keys 'norm.1', 'relu.1', 'conv.1', 'norm.2', 'relu.2', 'conv.
2'.
        # They are also in the checkpoints in model_urls. This pattern is u
sed
        # to find such keys.
        pattern = re.compile(
            r'^(.*denselayer\d+\.(?:norm|relu|conv))\.((?:[12])\.(?:weight|
bias|running_mean|running_var))$')
        state_dict = model_zoo.load_url(model_urls['densenet121'])
        for key in list(state_dict.keys()):
            res = pattern.match(key)
            if res:
                new_key = res.group(1) + res.group(2)
                state_dict[new_key] = state_dict[key]
                del state_dict[key]
        model.load_state_dict(state_dict)
    return model
```

In [63]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

transform = transforms.Compose([transforms.FiveCrop(330),
                lambda crops: torch.stack([transforms.ToTensor()(crop) for
crop in crops]),
                lambda norms: torch.stack([transforms.Normalize(mean=[0.485
, 0.456, 0.406],
                                                             std=[0.229,
0.224, 0.225])(norm) for norm in norms])])

dataset = dataloader.ImageNetDataset('data/imagespart', 'data.csv',
                                     crop_size=330,
                                     transform=transform)

dataset_loader = DataLoader(dataset,
                            batch_size=4,
                            shuffle=True,
                            num_workers=4)

device = torch.device("cuda")
model = densenet121(pretrained=True).to(device)
model.eval()

correct = 0
with torch.no_grad():
    for _, batch in enumerate(dataset_loader):
        batched_fives = batch['image']
        labels = batch['label'].to(device)

        batch_size, num_crops, c, h, w = batched_fives.size()

        # flatten over batch and five crops
        stacked_fives = batched_fives.view(-1, c, h, w).to(device)

        result = model(stacked_fives)
        result_avg = result.view(batch_size, num_crops, -1).mean(1) # avg o
ver crops
        pred = result_avg.argmax(dim=1, keepdim=True)
        correct += pred.eq(labels.view_as(pred)).sum().item()

print('\nTest set: Accuracy: {}/{} ({:.0f}%)\n'.format(correct, len(dataset
_loader.dataset),
                                                       100. * correct / len
(dataset_loader.dataset)))
```

```
Test set: Accuracy: 1907/2500 (76%)
```