Chapter 2

Pretrained Networks

This chapter covers:

- Running pretrained image-recognition models
- Introduction to GANs and CycleGAN
- Captioning models used to produce text descriptions of images
- Sharing models through Torch Hub

Jupyter Codes Link (https://github.com/deep-learning-with-pytorch/dlwpt-code/tree/master/p1ch2)

- Might be wise to git clone the entire repository for the entire study

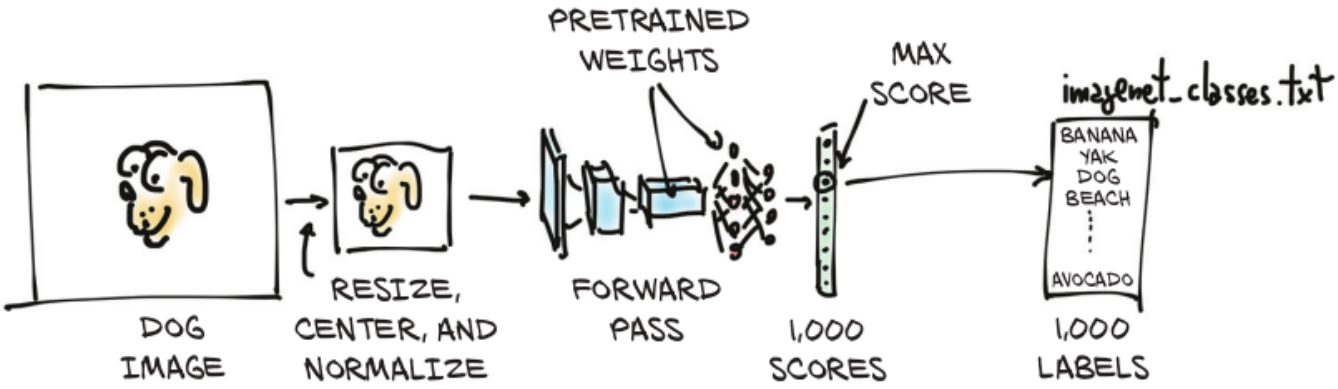
Pretrained models:

- already been trained on open, large-scale datasets by the best researchers in the field
- Provides a quick way to jump-start a deep learning project
  - draws on expertise from researchers who designed the model
  - - the computation time that went into training the weights

Recognition

ImageNet Dataset (http://imagenet.stanford.edu)

- A very large dataset of over 14 million images
- Maintained by Stanford University
- All images labeled with a hierarchy of nouns that come from the WordNet dataset (http://wordnet.princeton.edu)
  - A large lexical(related to words) database of the English language
- Used for ImageNet Large Scale Visual Recognition Challenge (ILSCVRC)
  - Image classification, Object localization/detection, Scene classification, Scene parsing...
  - 1.2mil images labelled with one of 1000 nouns (i.e. 1000 classes, or 1000 labels)



- Input image first processed into an instance of `torch.Tensor`
  - 3 colour channels, two spatial dimensions of a specific size

Torchvision Project (https://github.com/pytorch/vision)

- contains few of the best performing neural network architecture for vision
  - AlexNet, ResNet, Inception...
- Easy access to datasets like ImageNet
- Other utilities for getting up to speed with cv applications in PyTorch

In [5]:

```
from torchvision import models
print(dir(models)[:5])
print(dir(models)[-5:])
```

```
['AlexNet', 'DenseNet', 'GoogLeNet', 'GoogLeNetOutputs', 'Inception3']
['vgg19', 'vgg19_bn', 'video', 'wide_resnet101_2', 'wide_resnet50_2']
```

(just printed first and last 5, too long)

- **Uppercase** names: popular architecture for computer vision
- **lowercase** names: functions that instantiate models with predefined number of layers and units
  - can optionally download and load pretrained weights into them

For now, turn our attention to **Alexnet**

- Won 2012 ILSVRC by a large margin
  - top-5 test error rate (correct label in top 5 predictions) 15.4%
    - runner up was at 26.2%
    - modern architectures get as low as 3%
- considered a small network in today's standards

In [8]:

```
"""
object that can run the Alexnet Architecture
can run forward pass with output = alexnet(input)
This network is uninitialized: i.e. all weights are untrained
current output would be garbage
"""
alexnet = models.AlexNet()

"""
resnet101 function, instantiate 101-layer CNN
Pass an argument to download weights of resnet101 trained on ImageNet
"""
resnet = models.resnet101(pretrained=True)

"""
What does our model look like?
shows modules, one per line
"""
resnet
```

Out[8]:

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=
(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track
_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=
1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, t
rack_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), pad
ding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, t
rack_running_stats=True)
```

In [17]:

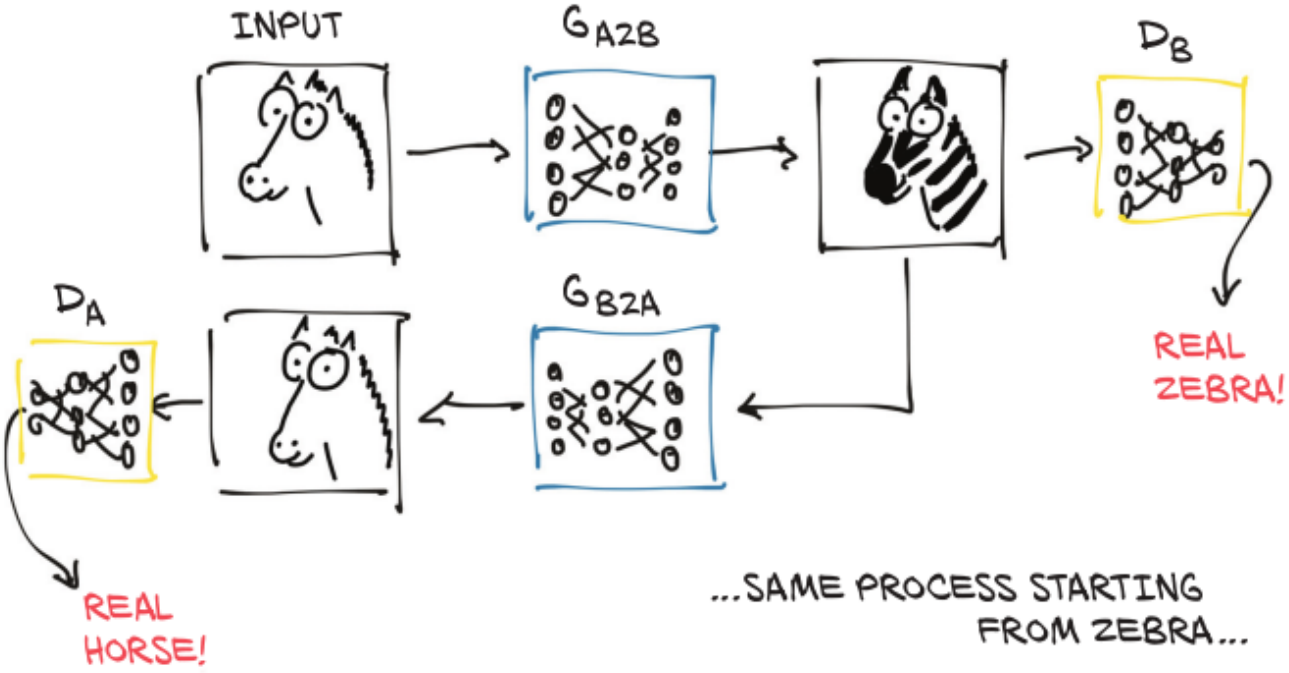
```
"""
We have to preprocess the input images
so that they are the right size and their values(colors) sit roughly in the
same numerical range (i.e. need normalization)
"""
from torchvision import transforms
preprocess = transforms.Compose([
                transforms.Resize(256),
                transforms.CenterCrop(224),
                transforms.ToTensor(), # Converts image to tensor
                transforms.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225]
            )])

"""
Can use the Pillow module to load an image from the local file system
"""
from PIL import Image
import torch
#img = Image.open(PATH_TO_IMGFILE)
#img_t = preprocess(img)
#batch_t = torch.unsqueeze(img_t, 0)

img = Image.open("/home/swkim/Desktop/29.jpg")
img_t = preprocess(img)
print(img_t.shape)
batch_t = torch.unsqueeze(img_t, 0)
print(batch_t.shape)
```

```
torch.Size([3, 224, 224])
torch.Size([1, 3, 224, 224])
```

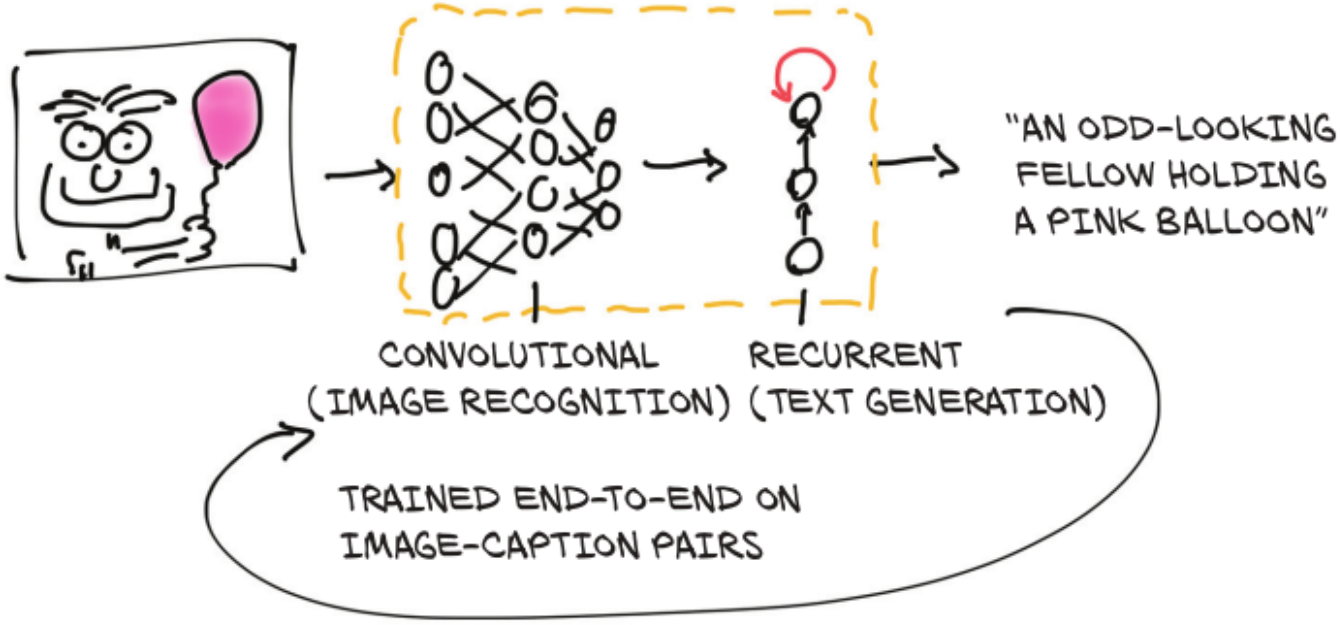
In [23]:

```
"""
Now ready to run our model
First set the model to eval mode
: otherwise, some pretrained models like BATCHNORM and DROPOUT will not produce mea
"""
resnet.eval()
out = resnet(batch_t)
out

# 이후 inference 과정이 있긴 한데 여기서는 생략
# value, index = torch.max(out, 1)
# percentage = torch.nn.functional.softmax(out, dim=1)[0] * 100
# labels[index[0]], percentage[index[0]].item()

"""
For top-5 values:
"""
# _, indices = torch.sort(out, descending=True)
# [labels[index[idx]], percentage[idx].item()] for idx in indices[0][:5]]

"""
This chapter is just for the purpose of introducing pretrained models
skipping GAN and ImageCaptioning, refer to book if you want to know more
참고로 그림들이 귀여워서 읽을만함
"""
pass
```

CycleGAN



Concept of a Captioning Model



Torch Hub

Previously, there was no way to ensure that users would have a uniform interface to get pretrained models

TorchVision was a good example of a clean interface

- But some (such as CycleGAN or NeuralTalk2) chose different designs

Torch Hub:

- mechanism through which authors can publish a model on GitHub
  - with or without pretrained weights
- exposes the model through an interface that PyTorch understands
- makes pretrained model loading (from third party) as easy as loading a TorchVision model

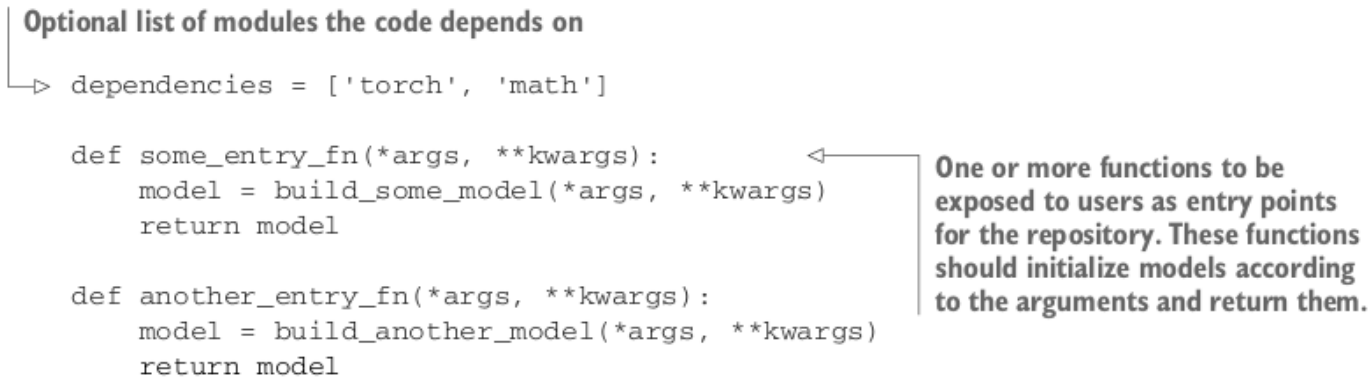
All it takes for an author to publish a model through the Torch Hub mechanism is to place a file named `hubconf.py` in the root directory of the Github repository.

Structure of `hubconf.py`

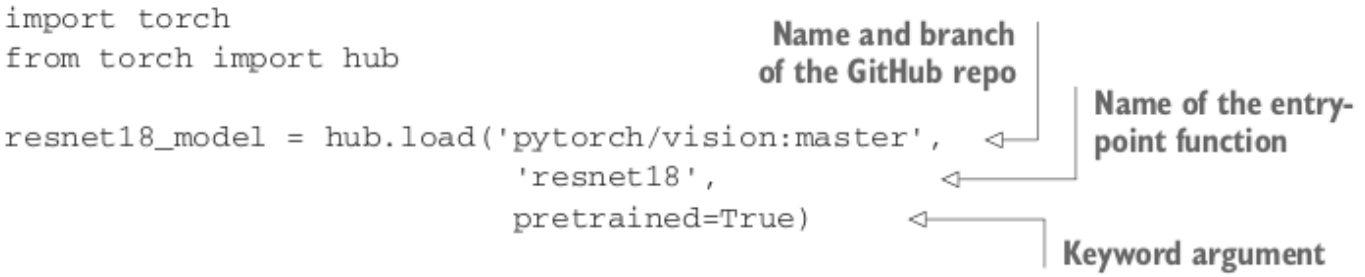
Optional list of modules the code depends on

```
dependencies = ['torch', 'math']

def some_entry_fn(*args, **kwargs):
    model = build_some_model(*args, **kwargs)
    return model

def another_entry_fn(*args, **kwargs):
    model = build_another_model(*args, **kwargs)
    return model
```

One or more functions to be exposed to users as entry points for the repository. These functions should initialize models according to the arguments and return them.



Usage of PyTorch Hub in code:



In the above example, all entry points returns models

- Not necessary!

Can have entry points for anything else, such as:

- Trnasofrming inputs
- Turning output probabilities into a text label
- Includes the model along with pre- and post processing steps

Torch Hub is **quite new** at the time of writing

- Can access models published this way by Googling `github.com hubconf.py`