UNC CHARLOTTE

The WILLIAM STATES LEE COLLEGE *of* ENGINEERING

TECSAR
Transformative Computer Systems and Architecture Research

Real-time AI
Lecture 1:Working with Pre-trained Networks

Hamed Tabkhi

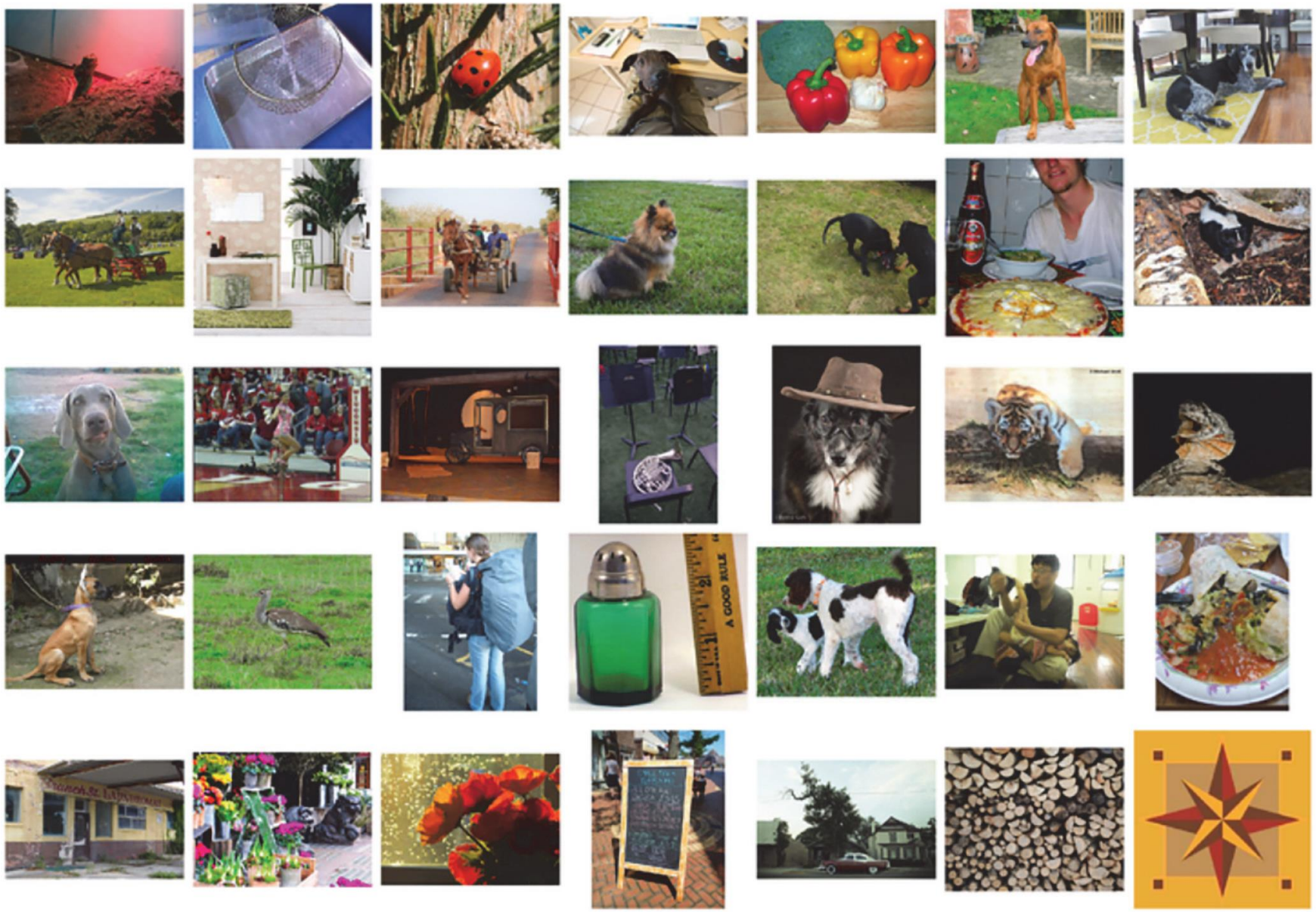
Department of Electrical and Computer Engineering,

University of North Carolina Charlotte (UNCC)

htabkhiv@uncc.edu

Summary

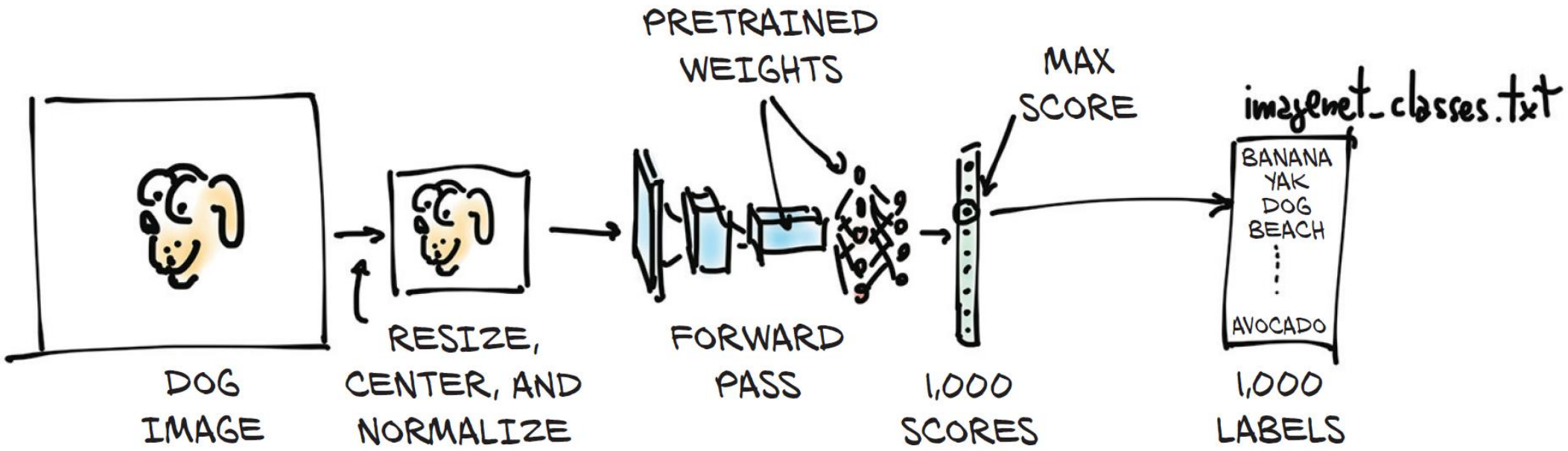
Deep learning models automatically learn to associate inputs and desired outputs from examples.
Libraries like PyTorch allow you to build and train neural network models efficiently.
PyTorch minimizes cognitive overhead while focusing on flexibility and speed. It also defaults to immediate execution for operations.
TorchScript allows us to precompile models and invoke them not only from Python but also from C++ programs and on mobile devices.
Since the release of PyTorch in early 2017, the deep learning tooling ecosystem has consolidated significantly.
PyTorch provides a number of utility libraries to facilitate deep learning projects.



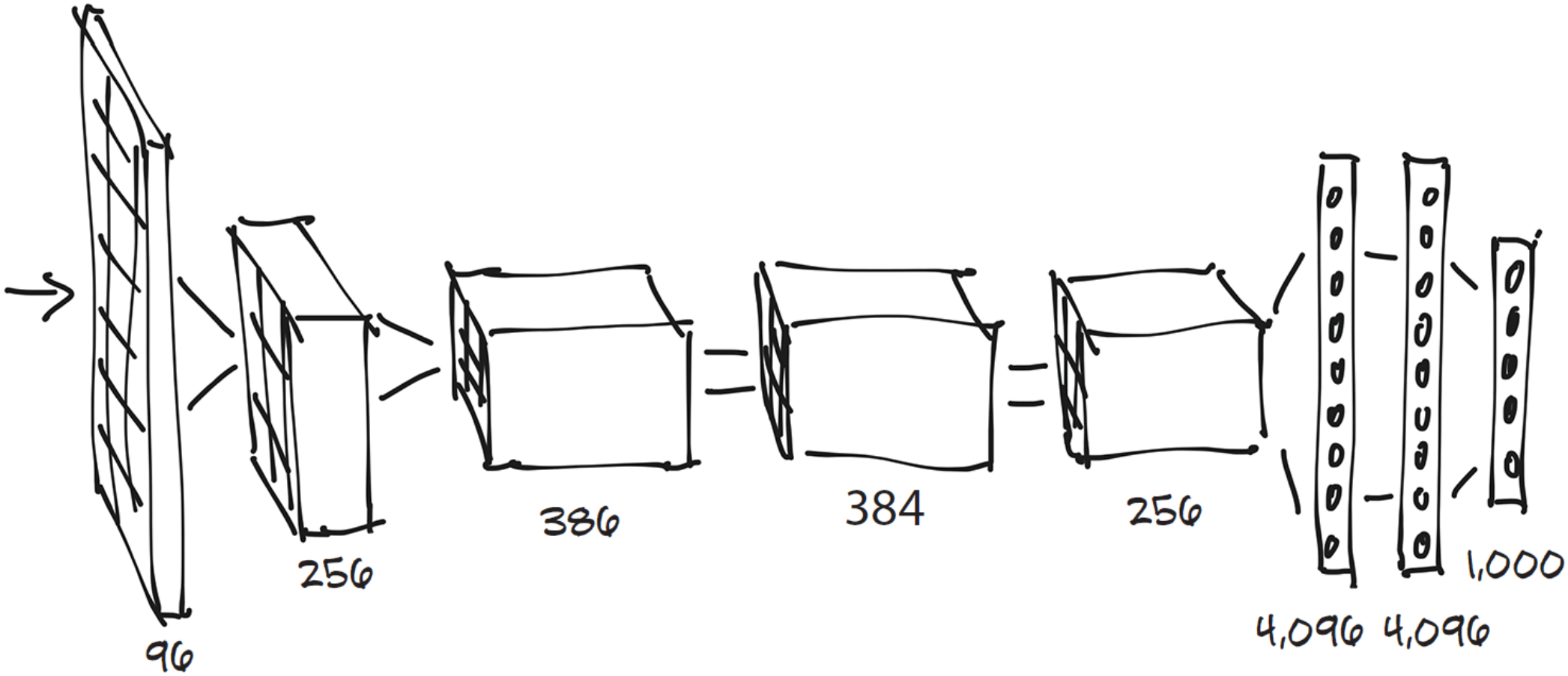
The WILLIAM STATES LEE COLLEGE *of* ENGINEERING
UNC CHARLOTTE

Image Classification

A pretrained network that recognizes the subject of an image

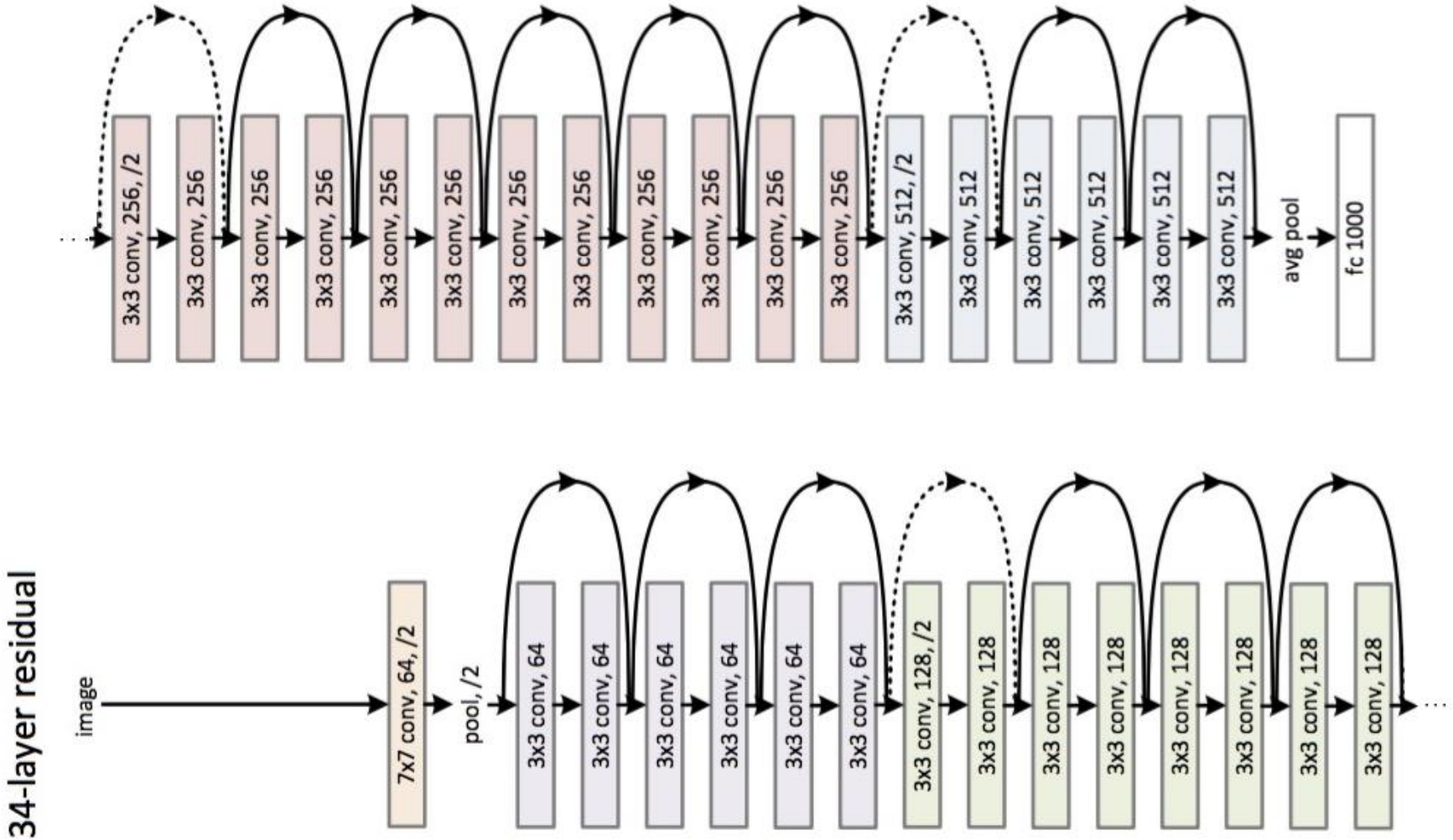


AlexNet: First Image Classfier



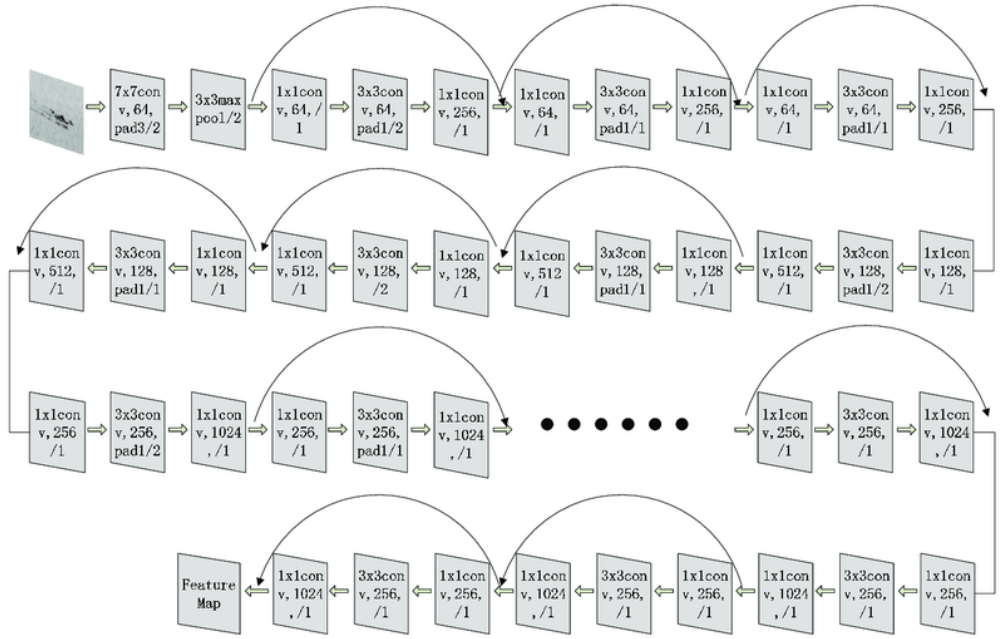
AlexNet Classification Example

ResNet


34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128
3x3 conv, 128, /2
3x3 conv, 256
3x3 conv, 256, /2
3x3 conv, 512
3x3 conv, 512, /2
avg pool
fc 1000

ResNet 101

- A 101-layer convolutional neural network.
- Residual networks pulled a trick that made it possible to train a network composing out of 101 layers.
- resnet101 sports 44.5 million parameters—that's a lot of parameters to optimize automatically!
- It beat several benchmarks.



Time to Try Some Codes

Personal Computer:
Install Pytorch with respect to your target OS. Instruction here:
https://pytorch.org/get-started/locally/

Install Jupyter Notebook or Jupyter Lab
https://jupyter.org/install

Google Co-lab:
Everything has already installed. Instruction to work with google Co-lab provided here:
https://course19.fast.ai/start_colab.html

Creating ResNet 101 Instance

Let's create an instance of the network now.

Download all source codes from
https://github.com/deep-learning-with-pytorch/dlwpt-code

Code related to ResNet 101 are in:
https://github.com/deep-learning-with-pytorch/dlwpt-code/blob/master/p1ch2/2_pre_trained_networks.ipynb

```
# In[4]:
resnet = models.resnet101(pretrained=True)
```

We'll pass an argument that will instruct the function to download the weights of `resnet101` trained on the ImageNet dataset, with 1.2 million images and 1,000 categories:

Creating ResNet 101 Instance

```
# In[5]: resnet

# Out[5]:
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)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
                       ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
...
    )
  )
  (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0)
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)
```

`modules` one per line: they are individual operations, the building blocks of a neural network. They are also called *layers* in other deep learning frameworks.

Bottleneck Module: group of layer modules

Sequential cascade of filters and nonlinear functions, ending with a layer (`fc`) producing scores for each of the 1,000 output classes (`out_features`). See full ResNet 101 code here: https://github.com/deep-learning-with-pytorch/dlwpt-code/blob/master/p1ch2/2_pre_trained_networks.ipynb

The WILLIAM STATES LEE COLLEGE of ENGINEERING
UNC CHARLOTTE

Creating ResNet 101 Instance

we have to preprocess the input images so they are the right size and so that their values (colors) sit roughly in the same numerical range.

```
# In[6]:
from torchvision import transforms
preprocess =
transforms.Compose([
      transforms.Resize(256),
      transforms.CenterCrop(224),
      transforms.ToTensor(),
      transforms.Normalize(
         mean=[0.485, 0.456, 0.406],
         std=[0.229, 0.224, 0.225]
      )])
```

The `torchvision` module provides `transforms`, which allow us to quickly define pipelines of basic preprocessing functions:

We scale the input image to 256 × 256, crop the image to 224 × 224 around the center,

It transform image to a tensor (a PyTorch multidimensional array: A 3D array with color, height, and width)

It also normalizes its RGB (red, green, blue) components so that they have defined means and standard deviations

Note: These need to match what was presented to the network during training

Creating ResNet 101 Instance

Grab a picture of our favorite dog (say, bobby.jpg from the GitHub repo).
We can start by loading an image from the local filesystem using Pillow (https://pillow.readthedocs.io/en/stable)
An image-manipulation module for Python:

```
# In[7]:
from PIL import Image
img = Image.open("../data/p1ch2/bobby.jpg")

# In[8]:
img
```

The `resnet` variable can be called like a function, taking as input one or more images and producing an equal number of scores for each of the 1,000 ImageNet classes.

Creating ResNet 101 Instance

Next, we can pass the image through our preprocessing pipeline:

```
# In[9]:
img_t = preprocess(img)
```

Then we can reshape, crop, and normalize the input tensor in a way that the network expects.
We'll understand more of this in the next two chapters; hold tight for now:

```
# In[10]:
import torch
batch_t = torch.unsqueeze(img_t, 0)
```

The process of running a trained model on new data is called ***inference*** in deep learning circles. In order to do inference, we need to put the network in `eval` mode:

```
# In[11]:
resnet.eval()
```

Creating ResNet 101 Instance

Now, we run the network

```
# In[12]:
out = resnet(batch_t)
out
# Out[12]:
tensor([[ -3.4803, -1.6618, -2.4515, -3.2662, -3.2466, -1.3611,
-2.0465, -2.5112, -1.3043, -2.8900, -1.6862, -1.3055,
...
2.8674, -3.7442, 1.5085, -3.2500, -2.4894, -0.3354,
0.1286, -1.1355, 3.3969, 4.4584]])
```

A staggering set of operations involving 44.5 million parameters has just happened, producing a vector of 1,000 scores, one per ImageNet class.

Creating ResNet 101 Instance

We now need to find out the label of the class that received the highest score.

To see the list of predicted labels, we load a text file listing the labels in the same order they were presented to the network during training:

```
# In[13]:
with open('../data/p1ch2/imagenet_classes.txt') as f:
labels = [line.strip() for line in f.readlines()]
```

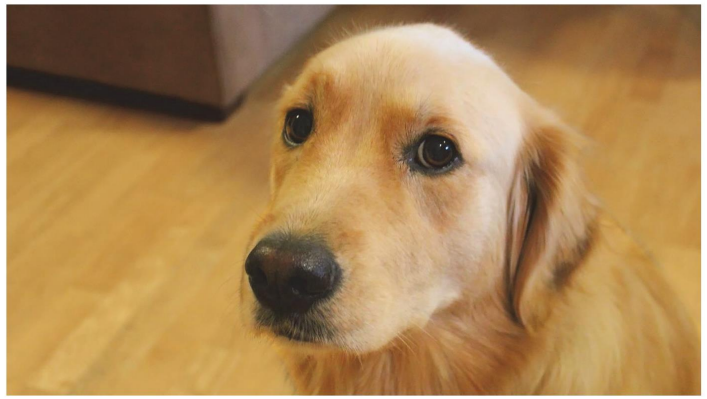
We can do that using the `max` function in PyTorch, which outputs the maximum value in a tensor as well as the indices where that maximum value occurred:

```
# In[14]:
_, index = torch.max(out, 1)
```

Creating ResNet 101 Instance

We also use `torch.nn.functional.softmax` (http://mng.bz/BYnq) to normalize our outputs to the range [0, 1], and divide by the sum.
That gives us something roughly akin to the confidence that the model has in its prediction.

```
percentage =
torch.nn.functional.softmax(out, dim=1)[0]
* 100
labels[index[0]],
percentage[index[0]].item()
# Out[15]:
('golden retriever', 96.29334259033203)
```



In this case, the model is 96% certain that it knows what it's looking at is a golden retriever.

Creating ResNet 101 Instance

Since the model produced scores, we can also find out what the second best, third best, and so on were. To do this, we can use the `sort` function, which sorts the values in ascending or descending order and also provides the indices of the sorted values in the original array:

```
# In[16]:
_, indices = torch.sort(out, descending=True)
[(labels[idx], percentage[idx].item()) for idx in indices[0][:5]]
# Out[16]:
[('golden retriever', 96.29334259033203),
('Labrador retriever', 2.80812406539917),
('cocker spaniel, English cocker spaniel, cocker', 0.2826742827892303 5),
('redbone', 0.2086310237646103),
('tennis ball', 0.1162156909704208 4)]
```

We see that the first four are dogs!