CSC321 Neural Networks and Machine Learning

Lecture 7

February 26, 2020

Agenda

- Midterms are done, yay!
- More Convolutional Neural Networks
  - Review: Conv2d, MaxPool2d
  - Object Detection, AlexNet, VGG, Sizes of networks
  - Transfer Learning
  - More Ideas: BatchNorm2d, Global Pooling

Convolutional Neural Networks

Convolution Computation

Input activations

100	100	100	100	100	100
100	100	100	100	100	100
100	0	0	100	100	100
0	0	0	0	0	0
0	0	0	0	0	0

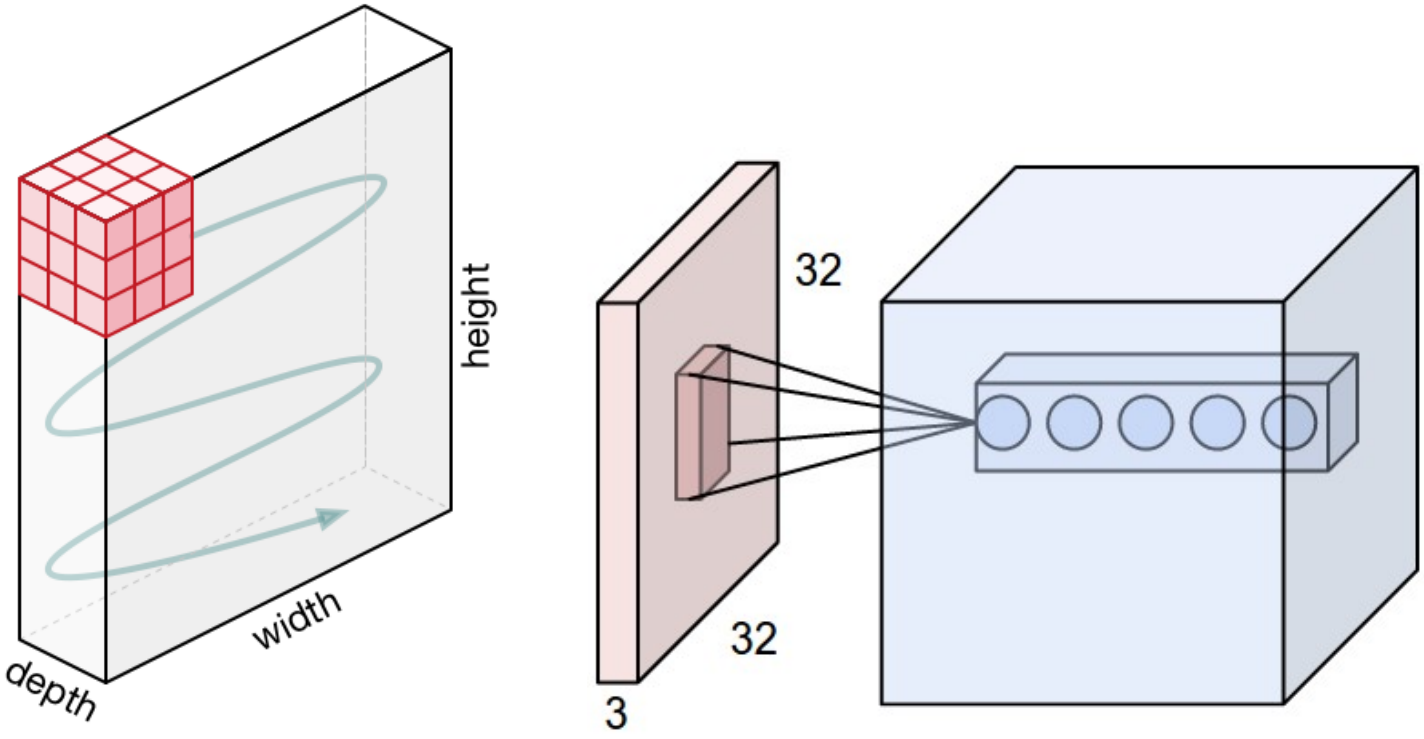
Kernel

1	2	1
0	0	0
-1	-2	-1

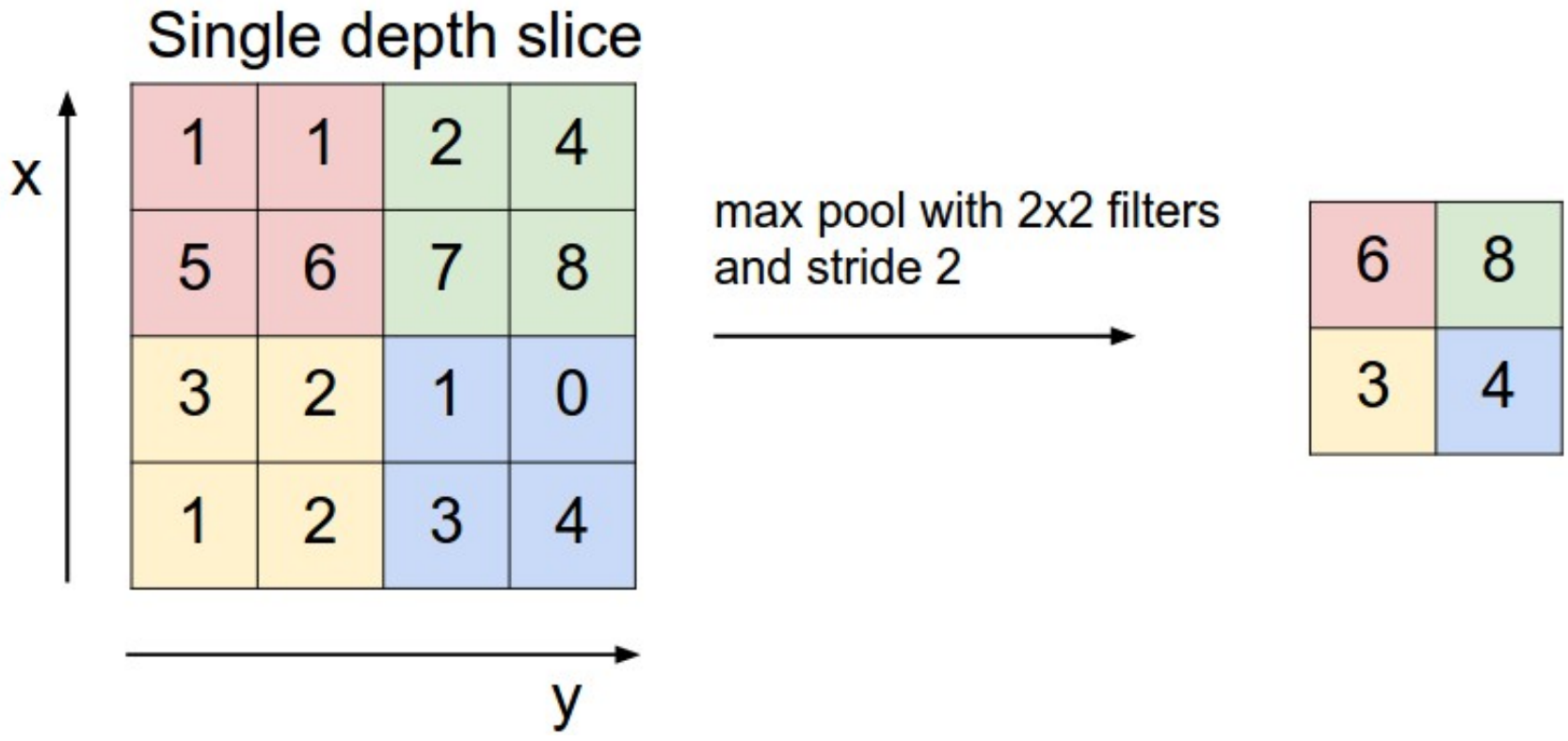
Hidden activation

300	300	100	

Multiple Input and Output Channels

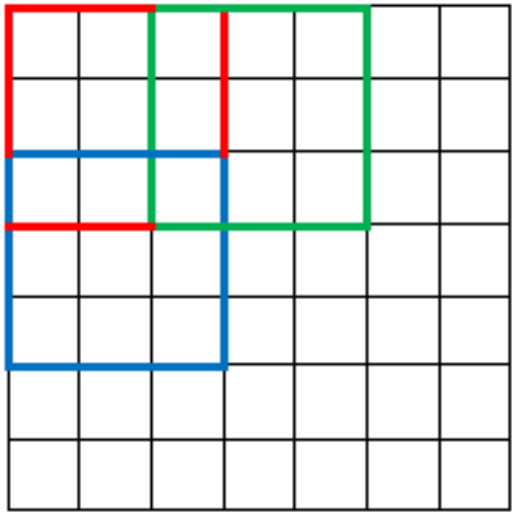


Max-Pooling

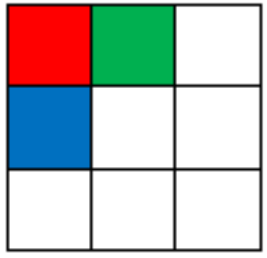

Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

Strided Convolution

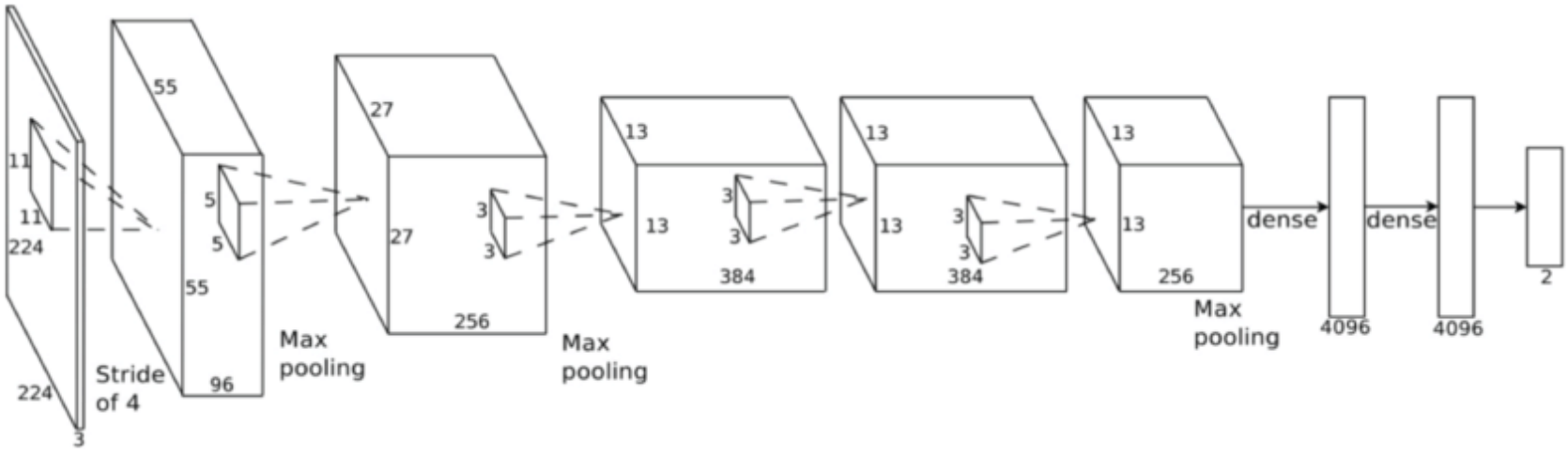
7 x 7 Input Volume



3 x 3 Output Volume

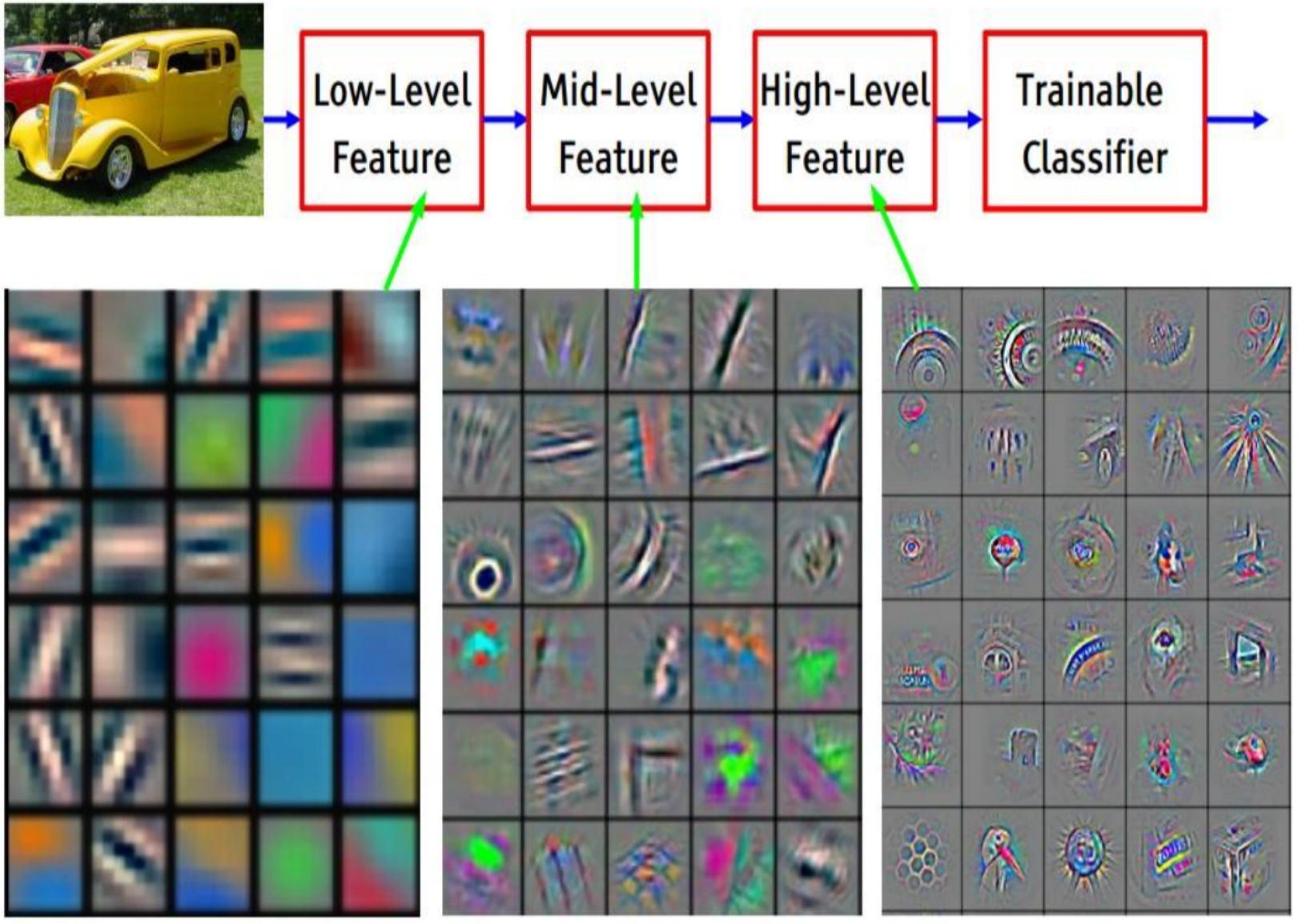


Example CNN: AlexNet




```
import torchvision.models
alexNet = torchvision.models.alexnet(pretrained=False)
```

Convolutional Features



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Receptive Fields

Because of downsampling (pooling and use of strides), higher-layer filters "cover" a larger region of the input than equal-sized filters in the lower layers.

Object Recognition

Object Recognition Task

- Object recognition is the task of identifying which object category is present in an image.
  - Deal with change in position, size, shape, occlusion, lighting, ...
- Direct application to image search
- Closely related to object detection, the task of locating all instances of an object in an image

For the past 6 years, all of the best object recognizers have been various kinds of conv nets.

Object Recognition Dataset

- In order to train and evaluate a machine learning system, we need to collect a dataset.
- The design of the dataset can have major implications

Biggest image classification "advances" of the last two decades:

- Datasets have gotten much larger (because of digital cameras and the Internet)
- Computers got much faster
- Graphics processing units (GPUs) turned out to be really good at training big neural nets; they're generally about 30 times faster than CPUs.

As a result, we could fit bigger and bigger neural nets.

Image Classification Data: MNIST

MNIST dataset of handwritten digits

- Categories: 10 digit classes
- Source: Scans of handwritten zip codes from envelopes
- Size: 60,000 training images and 10,000 test images, grayscale, of size 28×28
- Normalization: centered within in the image, scaled to a consistent size

In 1998, Yann LeCun and colleagues built a conv net called LeNet which was able to classify digits with 98.9% test accuracy.

It was good enough to be used in a system for automatically reading numbers on checks.

Data Size: $\sim$60MB

Image Classification Data: ImageNet

ImageNet is the modern object recognition benchmark dataset. It was introduced in 2009, and has led to amazing progress in object recognition since then

- Categories: 1000 categories (e.g. hundreds of kinds of dogs)
- Size: 1.2 million full-sized images
- Source: Results from image search engines, hand-labeled by Mechanical Turkers
- Normalization: none, although the contestants are free to do preprocessing

Data Size: ~50 GB

ImageNet Images

Size of a network

Ways to measure size of a network:

- **Number of units**: important because the activations need to be stored in memory during training (i.e. backprop)
  - More units imply higher memory requirement
- **Number of connections**: important because there are approximately 3 add-multiply operations per connection (1 for the forward pass, 2 for the backward pass).
  - More units imply higher CPU/GPU requirement
- **Number of weights**: important because the weights need to be stored in memory, and because the number of parameters determines the amount of overfitting.
  - More units imply higher training data requirement

Size of a fully-conneted layer

If we have M input units and N output units, then

- Number of units $= N$
- Number of connections $= MN$
- Number of weights $= MN$

Size of a convolutional layer

If we have M input channels, N output channels, kernel size $K \times K$, input shape $H \times W$, and padding $\frac{K-1}{2}$

- Number of units $= WHN$
- Number of connections $= WHK^2MN$
- Number of weights $= K^2MN$

Homework 4

In homework 4, we'll be looking at the size of AlexNet

When doing homework 4, think about:

- Where are most of the units?
- Where are most of the connections?
- Where are most of the weights?

Why AlexNet?

Reason 1: AlexNet's performance on the ImageNet dataset is what set off the deep learning boom of the last 6 years!

Reason 2: Transfer Learning!

Transfer Learning

Transfer Learning is the idea of using weights/features trained on one task, and using it on another task.

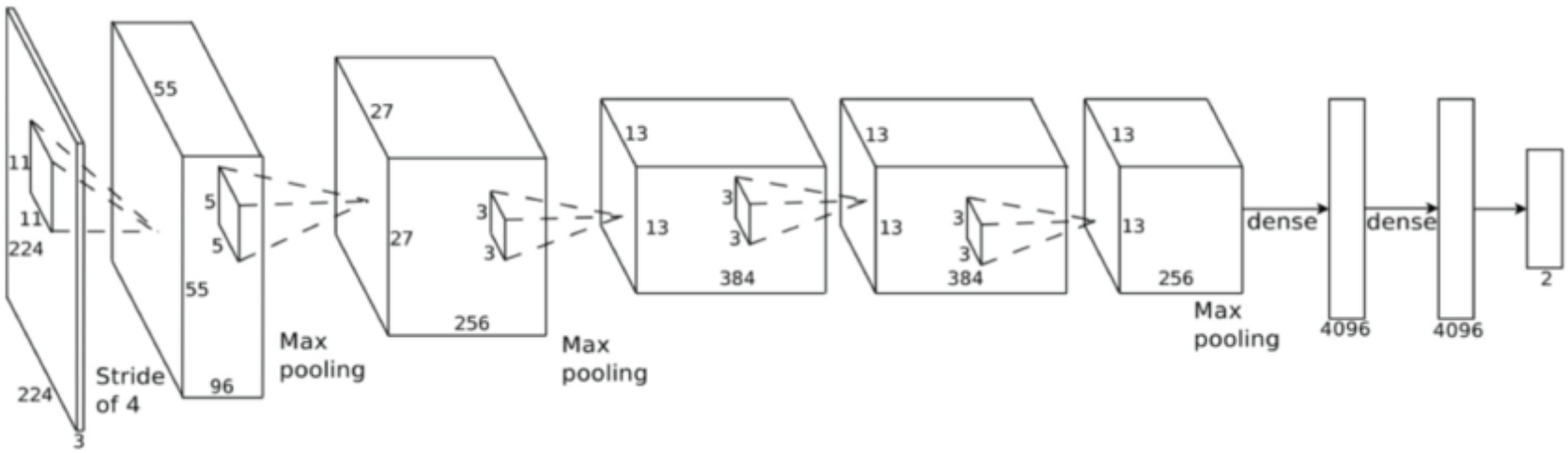
We already saw the idea of transfer learning in project 2:

- Train a model to predict the next work given the previous three
- Use the weights to determine word similarities

Transfer learning with CNN

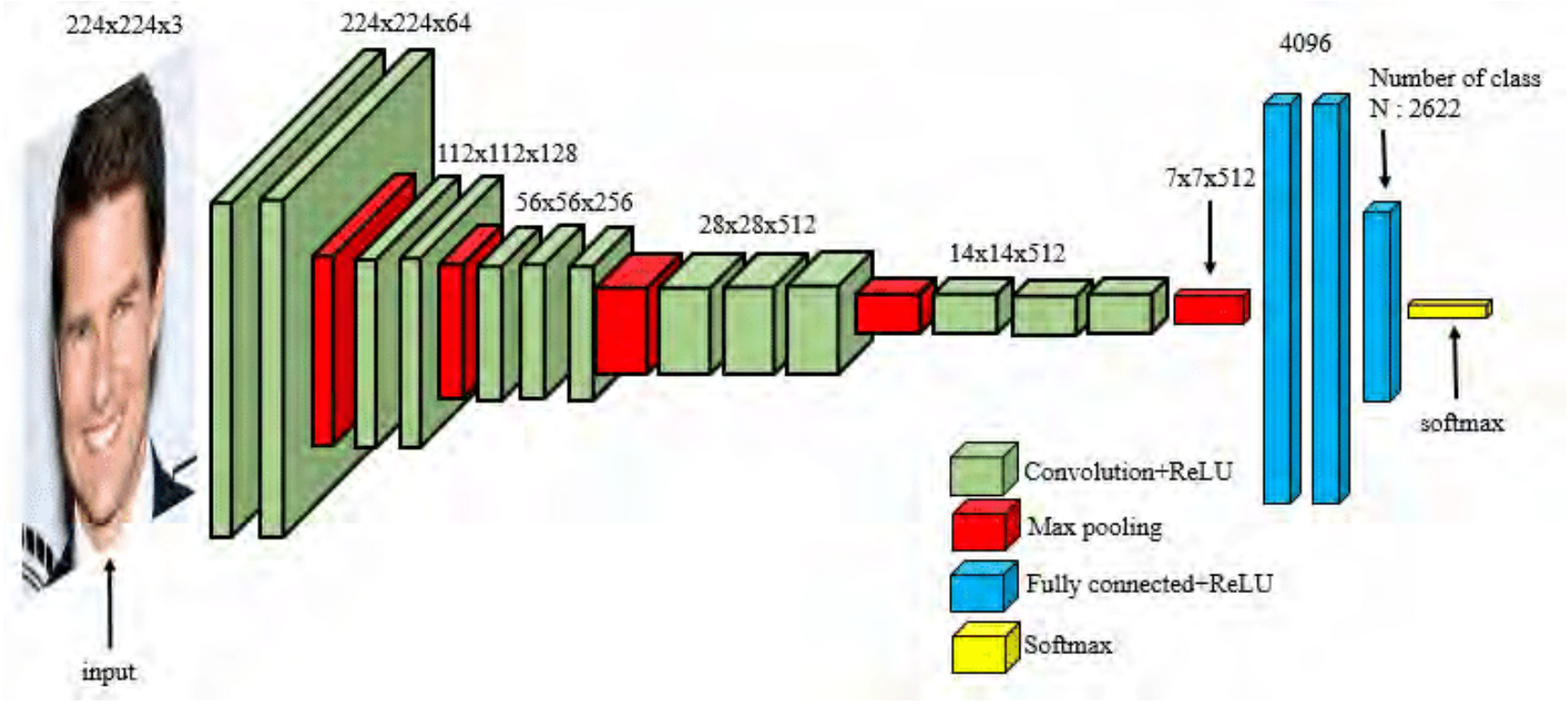
Practioners rarely train a CNN "from scratch". Instead we could:

1. Take a pre-trained CNN model (e.g. AlexNet), and use its features network to compute **image features**, which we then use to classify our own images
2. Initialize our weights using the weights of a pre-trained CNN model (e.g. AlexNet)



Another CNN: VGG

Also trained on ImageNet, and often used for Transfer Learning:




```
# There are a few VGG versions
vgg16 = torchvision.models.vgg.vgg16(pretrained=False)
vgg19 = torchvision.models.vgg.vgg19(pretrained=False)
```

What we want you to know:

- AlexNet and VGG are trained on the ImageNet data (image classification task)
- AlexNet (2012) came before VGG (2014)
- VGG is the larger network (according to any measurement)

What's interesting about VGG?

- VGG uses very small receptive fields (3×3 instead of 11×11)
- VGG incorporates 1×1 convolutional layers (why?)

Modern CNN Ideas

Batch Normalization

Recent convolutional neural networks also use **batch normalization**.

Unlike in a fully-connected layer, we normalize over an entire feature map (an entire channel).

In PyTorch `nn.BatchNorm2d(num_output_channels)` rather than `nn.BatchNorm1d(num_output_features)`.

Global Average Pooling

- Average (or max pool) over the entire channel
- This avoids fully-connected layers altogether
- The CNN can (theoretically) take arbitrary dimension images as input
  - Q: Why theoretically?

Fully Convolutional Networks

Fully convolutional networks do not use any fully connected layers!

Instead, use global average pooling.

More Fully Convolutional Networks

Example: This is an example of a CNN solving a **pixel-wise** prediction problem, i.e. classifying each pixel. We'll talk more about pixel-wise prediction next week!

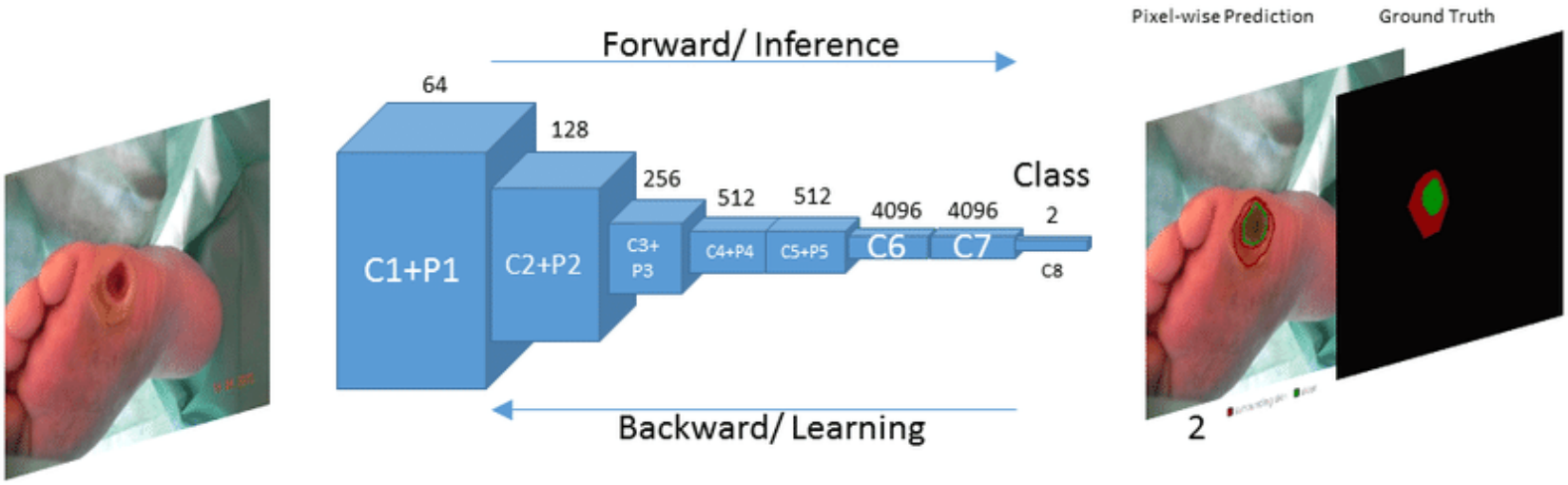


Image from "Fully Convolutional Networks for Diabetic Foot Ulcer Segmentation"