CSC321 Neural Networks and Machine Learning

Lecture 8

March 4, 2020

Agenda

- Midterm + Logistic
- Generative CNNs
- Autoencoder

Remember that homework 4 is due next week, and project 3 is coming up!

Midterm + Logistics

Midterm

- The midterm was a little long, so we'll grade the midterm out of 27 instead of 30
- **The adjustment is not reflect on Markus**
  - Markus still computes your grades ouf of 30 points instead of 27
- This brings the average to a bit above the average on Q2/Q3

We made some changes to the grading after the TAs graded on papers to give more part marks. Markus annotations supersedes the hand-written grading.

Term Marks

Component	Median	Average
Homeworks	87%	82%
Projects	85%	77%
Midterm	60%	60%

Are we overfitting? (i.e. are you trying to memorize the slides without really understanding deeply?)

Remark Requests

Please read and follow the instructions on Quercus or Markus.

Deadline: March 9th, 9pm

Survey Feedback

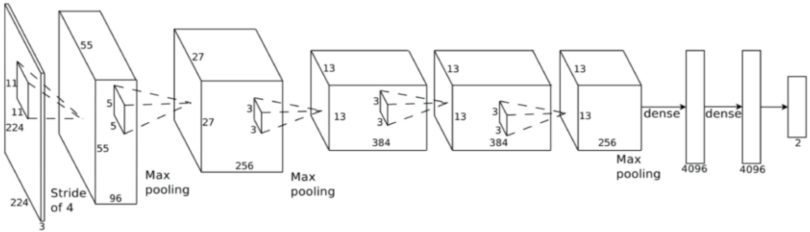
Main takeaway: The tutorials aren't really working for you, even though we're taking a lot of time to develop the materials.

So let's try something else:

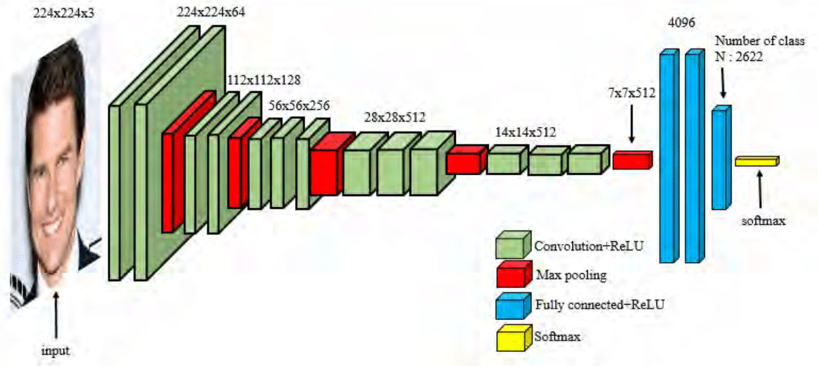
- Hands-on Labs?
- Exam prep questions?
- Bonus quizzes?

CNN Architecture Review

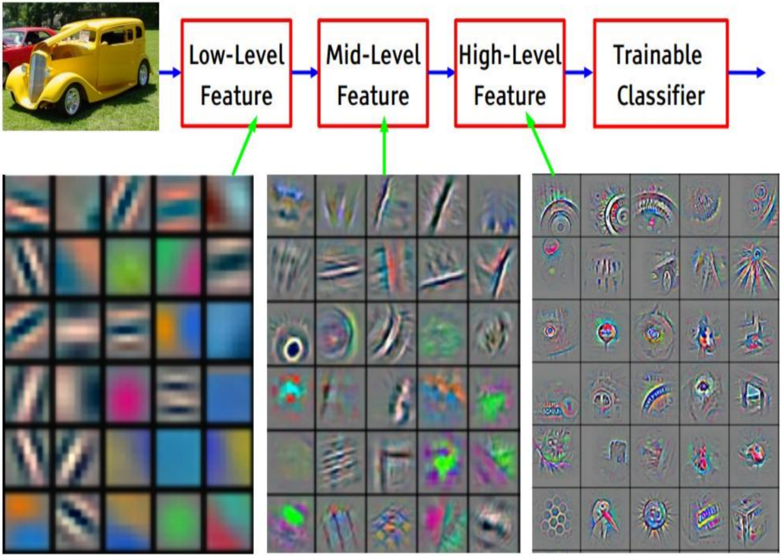
Review: AlexNet



Review: VGG


224x224x3
224x224x64
112x112x128
56x56x256
28x28x512
14x14x512
7x7x512
4096
Number of class
N : 2622
softmax
input
Convolution+ReLU
Max pooling
Fully connected+ReLU
Softmax

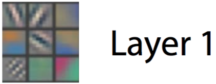
Review: Convolutional Features

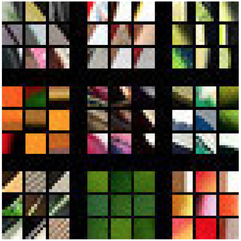


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

What features do convolutional networks detect?

The visualization shows the patterns in the input space (pixels) that cause the highest activation in a unit in the first conv layer.

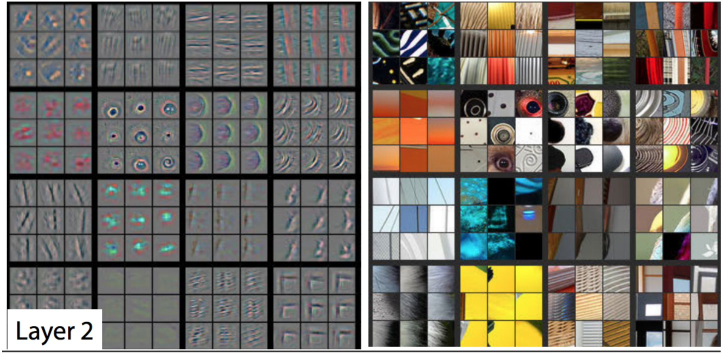




Zeiler & Fergus (2013) Visualizing and Understanding Convolutional Networks
https://arxiv.org/pdf/1311.2901.pdf

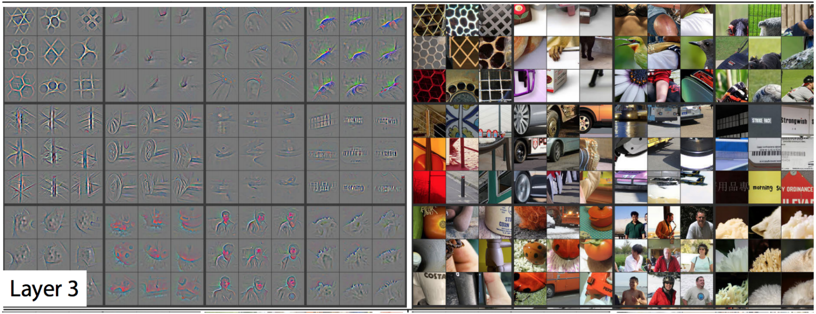
What features do convolutional networks detect?

. . . second conv layer:

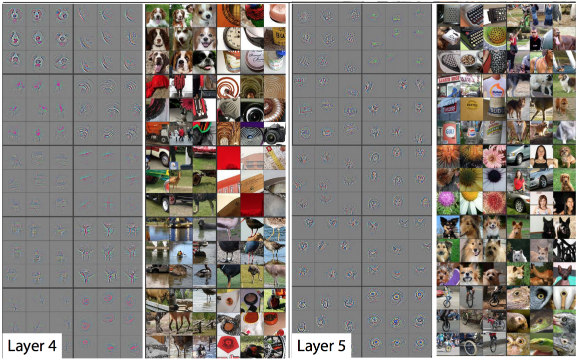


What features do convolutional networks detect?

. . . third conv layer:



What features do convolutional networks detect?



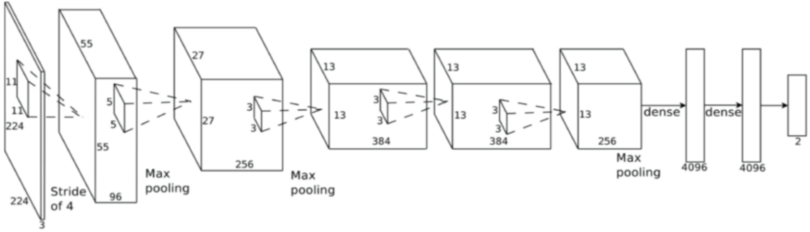
Observations:

- Higher layers look at a larger region of the image (why?)
- Higher layers detect "higher-level" features

Review: Transfer learning

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)



Review: Fully Convolutional Networks

Fully convolutional networks do not use any fully connected layers!

Instead, use global average pooling.

Example: Pixel-wise prediction

Example: This is an example of a CNN solving a **pixel-wise** prediction problem, i.e. classifying each pixel.

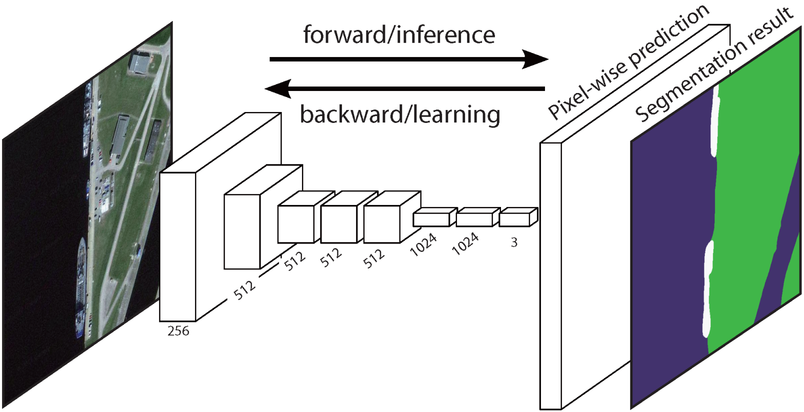


Image from: `https://arxiv.org/pdf/1411.4038.pdf`

Generating predictions

We can solve a problem like pixel-wise prediction by training a neural network that generates an output feature map of size $H \times W \times C$.

- $H \times W$ is the size of the original image
- C is the number of classes
- We have a distribution of classes C at each pixel
- Ground truth targets are a set of $H \times W$ one-hot vectors (one per pixel)

Architecture for pixel-wise prediction

Downsampling

- Reduce the “resolution” of the feature maps (H and W)
- Consolidate information from larger and larger regions of the image to detect higher-level information
- Why? Because the class label of a pixel depends on its surroundings!

Upsampling

Going backwards! Can we increase the “resolution” of the feature maps to match the image?

What we need:

We need to be able to **up-sample** features, i.e. to obtain high-resolution features from low-resolution features

- Opposite of max-pooling OR
- Opposite of a strided convolution

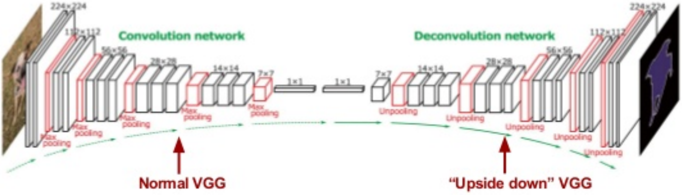
We need an **inverse** convolution – a.k.a a **deconvolution** or **transpose convolution**.

Architectures with Transpose Convolution

More than one upsampling layer

DeconvNet:

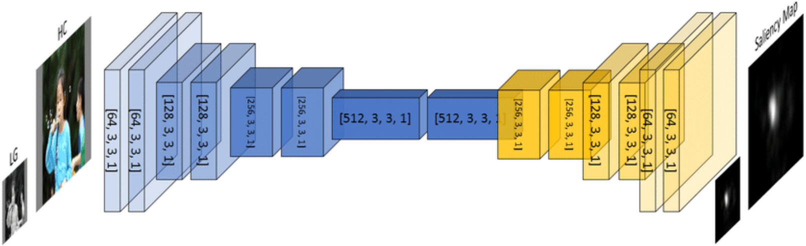
VGG-16 (conv+Relu+MaxPool) + mirrored VGG (Unpooling+’deconv’+Relu)



Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

17

Fully Convolutional Networks



In theory, this architecture can be used to make predictions for arbitrary sized images. (Why?)

Transposes Convolutions in PyTorch

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5)
>>> y = conv(x)
>>> y.shape
```

Transposes Convolutions in PyTorch

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5)
>>> y = conv(x)
>>> y.shape

>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5)
>>> x = convt(y)
>>> x.shape
```

Transposes Convolutions in PyTorch

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5)
>>> y = conv(x)
>>> y.shape
```

```
>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5)
>>> x = convt(y)
>>> x.shape
```

should get the same shape back!

Key Idea

If we have a convolution c and an transpose convolution t with the same kernel size, then applying $t(c(x))$ on a tensor x will yield another tensor with the same shape.

```
>>> convt(conv(x)).shape == x.shape
```

Inverse Convolution + Padding

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  padding=2)
>>> y = conv(x)
>>> y.shape
```

Inverse Convolution + Padding

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  padding=2)
>>> y = conv(x)
>>> y.shape

>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5,
...                            padding=2)
>>> x = convt(y)
>>> x.shape
```

Inverse Convolution + Padding

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  padding=2)
>>> y = conv(x)
>>> y.shape
```

```
>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5,
...                            padding=2)
>>> x = convt(y)
>>> x.shape
```

Also gets the same shape back!

Inverse Convolution + Stride

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  stride=2)
>>> y = conv(x)
>>> y.shape
```

Inverse Convolution + Stride

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  stride=2)
>>> y = conv(x)
>>> y.shape
```

```
>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5,
...                            stride=2)
>>> x = convt(y)
>>> x.shape
```

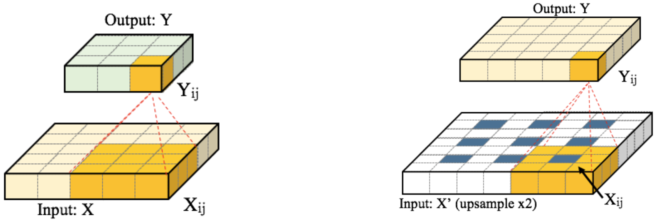
Inverse Convolution + Stride

```
>>> x = torch.randn(2, 8, 64, 64)
>>> conv = nn.Conv2d(in_channels=8,
...                  out_channels=8,
...                  kernel_size=5,
...                  stride=2)
>>> y = conv(x)
>>> y.shape
```

```
>>> convt = nn.ConvTranspose2d(in_channels=8,
...                            out_channels=8,
...                            kernel_size=5,
...                            stride=2)
>>> x = convt(y)
>>> x.shape
```

. . . almost the same shape . . .

Transpose Convolution Layer



(**a**) Convolutional layer: the input size is $W_1 = H_1 = 5$; the receptive field $F = 3$; the convolution is performed with stride $S = 1$ and no padding ($P = 0$). The output Yis of size $W_2 = H_2 = 3$.

(**b**) Transposed convolutional layer: input size $W_1 = H_1 = 3$; transposed convolution with stride $S = 2$; padding with $P = 1$; and a receptive field of $F = 3$. The output Yis of size $W_2 = H_2 = 5$.

Figure 1: https://www.mdpi.com/2072-4292/9/6/522/htm

More at `https://github.com/vdumoulin/conv_arithmetic`

Output Padding

```
nn.ConvTranspose2d(in_channels=8,
                   out_channels=8,
                   kernel_size=5,
                   stride=2,
                   output_padding=1) # +1 to output
                                     # width/height
```

What you need to know

- We won't ask you about transpose convolutional arithmetics (i.e. computing the forward/backward pass of a transpose convolutional layer)
- You should know what the trasnpose convolution setting should be to "invert" a convolution operation (we'll need this for autoencoders later today)
- You should know the difference between the `output_padding` and `padding` setting

Image Autoencoder

Image Autoencoder

An image autoencoder is a neural network used to find a **low-dimensional representation** of some images. This is an **unsupervised learning** task (no labels).

An image autoencoder has two components:

Image Autoencoder

An image autoencoder is a neural network used to find a **low-dimensional representation** of some images. This is an **unsupervised learning** task (no labels).

An image autoencoder has two components:

1. An **encoder** neural network that takes the image as input, and produces a low-dimensional embedding.

Image Autoencoder

An image autoencoder is a neural network used to find a **low-dimensional representation** of some images. This is an **unsupervised learning** task (no labels).

An image autoencoder has two components:

1. An **encoder** neural network that takes the image as input, and produces a low-dimensional embedding.
2. A **decoder** neural network that takes the low-dimensional embedding as input, and reconstructs the image.

Idea: A good, low-dimensional representation should allow us to reconstruct everything about the image.

The components of an autoencoder

Encoder:

- Input = image
- Output = low-dimensional embedding

Decoder:

- Input = low-dimensional embedding
- Output = image

Why autoencoders?

- Dimension reduction:
  - find a low dimensional representation of the image
- Image Generation:
  - generate new images not in the training set
  - (Any guesses on how we can do this?)

Autoencoders are considered a **generative model**.

How to train autoencoders?

- Loss function:
  - How close were the reconstructed image from the original?
  - **Mean Square Error Loss**: look at the mean square error across all the pixels.
- Optimizer:
  - Just like before!
- Training loop:
  - Just like before!

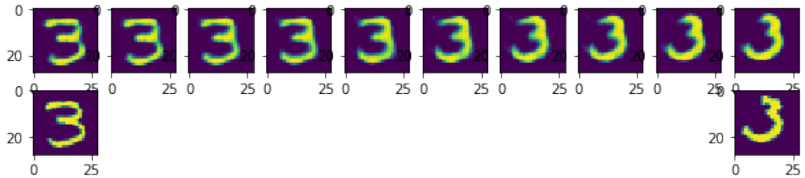
Let's train an autoencoder for MNIST

Structure in the Embedding Space

The dimensionality reduction means that there will be **structure** in the embedding space.

- The **distances** in the embedding space is meaningful (more meaningful than in the input space)
- We can look at **clusters** in the embedding space
- We can generate

Interpolating in the Embedding Space



Generating New Images

Q: Can we pick a random point in the embedding space, and decode it to get an image of a digit?

A: Unfortunately not necessarily. Can we figure out why not?

Autoencoder Overfitting

Overfitting can occur if the size of the embedding space is too large.

If the dimensionality of the embedding space is small, then the neural network needs to map similar images to similar locations.

If the dimensionality of the embedding space is **too large**, then the neural network can simply memorize the images!

Blurriness

Q: Why do autoencoders produce blurry images?

Hint: it has to do with the use of the MSELoss.