Lecture 23: Computer Vision and Deep Learning Basics
Logistics

- Programming assignment 2 released. Due 12/04.
- Guest lecture by Adrian Sampson next Wednesday on graphics programming models.

Real-time 3D graphics systems are everywhere, but the tools that software engineers use to build them are from the dark ages. OpenGL was invented in a time when GPUs were fixed-function pipelines, and it has grown haphazardly into a flexible system that resembles an inconvenient, confusing programming language. Essentially, it attempts to be both a convenient programming model and a portable hardware abstraction—and it does poorly at both. Modern APIs like Vulkan and Metal address the problem by acting *only* as low-level system abstractions, relegating the programming model to higher levels in the system stack. This shift creates an opportunity for research on designing programming models for real-time graphics that make it more expressive, faster, and more correct. In this talk, I will complain endlessly about OpenGL and then talk about two projects in my lab that aim to alleviate some of the misery it induces.
Where Are We?

Physical Scene (Objects, lights) → Optics (Gather light) → Image Sensing (Optical to electrical signal transformation) → Image Signal Processing (Signal reconstruction) → Computer Vision (Semantics understanding)

Modeling (Scene, optics) → Computer Graphics (Visibility + Shading) → Display (Generating lights) → Human Vision System (Eye, visual cortex) → Video/Image De/Compression → Cloud/Storage
From Retina to Visual Cortex

Light Spectrum

Eye

Visual Cortex

https://www.eyeworld.org/research-finds-human-visual-cortex-continues-development-mid-30s
One lower-level neuron influences many higher-level neurons.

Level 3: Neural Cells
Level 2: Neural Cells
Level 1: Neural Cells
Level 0: Photoreceptors

The "receptive field" of neuron N.
Single Neuron

Computational neuron

\[ w_0 x_0 \]

\[ w_1 x_1 \]

\[ w_2 x_2 \]

\[ f \left( \sum w_i x_i + b \right) \]

Biological neuron

impulses carried toward cell body

branches of axon

impulses carried away from cell body

axon terminals
A linear combination of inputs then go through a non-linear activation function.

\[ f\left( \sum_i w_i x_i + b \right) \]

**Activation functions**

- **Sigmoid**
  \[ \sigma(x) = \frac{1}{1+e^{-x}} \]

- **tanh**
  \[ \tanh(x) \]

- **ReLU**
  \[ \text{max}(0, x) \]
Single Neuron Processing an Image

Ground-truth: $P(\text{cat}) = 1$

Loss function, e.g., $||P(\text{cat}) - P'(\text{cat})||$. Find $W$ and $b$ that minimizes the loss function.

Adjust $W$ and $b$

Flatten image to a pixel vector (will see a better approach later).

Loss function, e.g., $||P(\text{cat}) - P'(\text{cat})||$. Find $W$ and $b$ that minimizes the loss function.
From a Single Neuron to a Neural Network

Ground-truth: \( P(\text{cat}) = 1 \)

\[ P'(\text{cat}) = 0.8 \]

Loss function, e.g., \( \|P(\text{cat}) - P'(\text{cat})\| \). Find \( W \) and \( b \) that minimizes the loss function.

Adjust \( W \) and \( b \)
Neural Network

Training Dataset

Cat probability
Dog probability
The Learning Objective

- Suppose we have a fixed training set.
- Define the cost function w.r.t. a single example, and suppose we have a (one-half) squared-error cost function.

\[
J(W, b; x, y) = \frac{1}{2} \| h_{W,b}(x) - y \|
\]

Loss of one input
The Learning Objective

- **Loss of one input**
  \[ J(W, b) = \frac{1}{2} \| h_{W,b}(x) - y \| \]

- **Loss of all inputs in the training set**
  \[ J(W, b) = \left[ \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \| h_{W,b}x^{(i)} - y^{(i)} \| \right] \]

- **Loss of all inputs in the training set with regularization**
  \[ J(W, b) = \left[ \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \| h_{W,b}x^{(i)} - y^{(i)} \| \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_i} \sum_{j=1}^{s_{i+1}} \left( W_{ji}^{(l)} \right)^2 \]

- **Goal:** find \( W \) and \( b \) that minimizes \( J(W, b) \).
- **How?** Randomly initialize \( W \) and \( b \) (e.g., sample a Gaussian distribution) and apply **Gradient Descent**.

The “weight decaying” regularization that prevents the weights becoming too complex, i.e., overfitting.
Exploiting Receptive Field

One lower-level neuron influences many higher-level neurons.

Level 3: Neural Cells
Level 2: Neural Cells
Level 1: Neural Cells
Level 0: Photoreceptors

Level 3: Neural Cells
Level 2: Neural Cells
Level 1: Neural Cells
Level 0: Photoreceptors

N

The “receptive field” of neuron N.
Exploiting Receptive Field

Retinal ganglion cells respond to edges

Center-surround receptive fields: emphasize edges.
Convolutional Neural Network

LeNet, By Yann LeCun 1989
A Convolution Layer

32x32x3 image

32 height
32 width
3 depth

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
Filter in A Convolution Layer

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Filter in A Convolution Layer

Convolution Layer

32x32x3 image

5x5x3 filter

Filters always extend the full depth of the input volume

**Convolve** the filter with the image i.e. “slide over the image spatially, computing dot products”

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
Filter in A Convolution Layer

32x32x3 image

5x5x3 filter $w$

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. $5 \times 5 \times 3 = 75$-dimensional dot product + bias)

$$w^T x + b$$

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
Activation Map

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
Multiple Filters

Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

consider a second, green filter

activation maps

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
Multiple Filters

For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
Cascading Convolutional Layers

**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions.
Example

**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

- **Input:** 3x32x32
- **Layer 1:** Convolutional (CONV), ReLU, e.g., 6 filters, size 5x5x3
- **Layer 2:** Convolutional (CONV), ReLU, e.g., 10 filters, size 5x5x6
- **Layer 3:** Convolutional (CONV), ReLU
- **Output:** 10x24x24

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
What Does Each Layer Do?

[From recent Yann LeCun slides]

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

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
Pooling

- makes the representations smaller and more manageable
- operates over each activation map independently:

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
Max Pooling

Invariant to small translations.
Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
Case Study: LeNet-5

Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]
Case Study: AlexNet
Case Study: VGGNet

Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2

<table>
<thead>
<tr>
<th>ConvNet Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
</tr>
<tr>
<td>11 weight layers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input (224 × 224 RGB image)</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv3-64</td>
</tr>
<tr>
<td>LRN</td>
</tr>
<tr>
<td>maxpool</td>
</tr>
<tr>
<td>maxpool</td>
</tr>
<tr>
<td>maxpool</td>
</tr>
<tr>
<td>maxpool</td>
</tr>
<tr>
<td>maxpool</td>
</tr>
<tr>
<td>maxpool</td>
</tr>
<tr>
<td>maxpool</td>
</tr>
</tbody>
</table>
Case Study: GoogLeNet

ILSVRC 2014 winner (6.7% top 5 error)

Fun features:
- Only 5 million params!
  (Removes FC layers completely)

Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)
Case Study: ResNet

Revolution of Depth

- AlexNet, 8 layers (ILSVRC 2012)
- VGG, 19 layers (ILSVRC 2014)
- ResNet, 152 layers (ILSVRC 2015)

ILSVRC 2015 winner (3.6% top 5 error)

2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)
ImageNet Challenges

Revolution of Depth

152 layers

ILSVRC'15
ResNet
3.57

ILSVRC'14
GoogleNet
6.7

ILSVRC'14
VGG
7.3

ILSVRC'13
8 layers
11.7

ILSVRC'12
AlexNet
16.4

ILSVRC'11
shallow
25.8

ILSVRC'10
28.2

ImageNet Classification top-5 error (%)


(slide from Kaiming He’s recent presentation)
Applications

Classification

Retrieval

[Krizhevsky 2012]

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
Applications

Detection

Segmentation

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

[Farabet et al., 2012]

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
Applications

Point-based Rendering

positions $p_1$, $p_2$, ..., $p_n$
descriptors $d_1$, $d_2$, ..., $d_n$
camera

Rasterizer + z-buffer

Rendering network

Result

Ground truth

Loss

Neural Point-Based Graphics, ECCV 2019  https://saic-violet.github.io/npbg/
Applications

[Toshev, Szegedy 2014]

[Mnih 2013]
Applications

Image Captioning

[A person riding a motorcycle on a dirt road.]

[Two dogs play in the grass.]

[A skateboarder does a trick on a ramp.]

[A dog is jumping to catch a frisbee.]

[A group of young people playing a game of frisbee.]

[Two hockey players are fighting over the puck.]

[A little girl in a pink hat is blowing bubbles.]

[A refrigerator filled with lots of food and drinks.]

[A herd of elephants walking across a dry grass field.]

[A close up of a cat laying on a couch.]

[A red motorcycle parked on the side of the road.]

[A yellow school bus parked in a parking lot.]

[Fei-Fei Li & Andrej Karpathy & Justin Johnson]

[Vinyals et al., 2015]
Applications

reddit.com/r/deepdream

Credit: Fei-Fei Li & Andrej Karpathy & Justin Johnson
Vision + Graphics

https://www.tensorflow.org/graphics/overview
Another Perspective: Universal Approximation Theorem

- Very loosely: an arbitrarily deep or wide neural network can be constructed to approximate any function.
- This theorem only states the existence not how to construct.
- Intuitively, the more complex a function is, the more difficult it is to construct (train) a neural network to approximate it.
- Carefully pick when to use a neural network. It’s better to have an analytical solution derived from physics, and then approximate only the portion that’s hard to manually derive.
Another Perspective: Universal Approximation Theorem

Another Perspective: Universal Approximation Theorem

Another Perspective: Universal Approximation Theorem

Deep Learning at the Speed of Light & Zero Energy?

- **Integrated Photonics**
  - Leverage the propagation of light in waveguides — think of them as fibers
  - Compatible with CMOS design
  - Small form factors, a.k.a, nanophotonics
  - Many many start-ups on this

- **Free-space Photonics**
  - Leverage the propagation of light in free-space through custom-design lens
  - Similar to computation cameras
  - Can directly interrogate natural scene as the input
Integrated vs. Free-Space Photonics

Free-space photonics

Integrated photonics

https://medium.com/lightmatter/matrix-processing-with-nanophotonics-998e294dabc1
Point Spread Function

- Describes how a point spreads its energy (light intensity) around itself in 2D space.

[Diagram of Focus and Spherical Aberration]

Point Spread Function

- Describes how a point spreads its energy (light intensity) around itself in 2D space.
- \( \text{PSF}(x, y) = 0.2 \) means that a point \((a, b)\) in the object space sends 20% of its energy to point \(Q=(a+x, b+y)\) on the image plane.
- Usually normalized, i.e., the integral of PSF is 1.

https://en.wikipedia.org/wiki/Point_spread_function
Point Spread Function

› Shift invariant: PSF of a point is independent of the position of the point on the object plane.

› Linear: Light intensity of different points on the object plane can be directly superimposed on the image plane.

https://en.wikipedia.org/wiki/Point_spread_function
Point Spread Function

- Assuming a shift-invariant and linear imaging system

\[ P' = 0.1A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.1G + 0.1H \]

As if convolves PSF at P in the input!
Point Spread Function

- Assuming a shift-invariant and linear imaging system

\[ P' = 0.1A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.1G + 0.1H \]

As if convolves PSF at P in the input!
Point Spread Function

- Assuming a shift-invariant and linear imaging system

\[ P' = 0.1A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.1G + 0.1H \]

As if convolves PSF at P in the input!
Point Spread Function

- Assuming a shift-invariant and linear imaging system

\[
P' = 0.1A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.1G + 0.1H
\]

As if convolves PSF at P in the input!
Point Spread Function

Assuming a shift-invariant and linear imaging system

\[ P' = 0.1A + 0.1B + 0.1C + 0.1D + 0.2E + 0.1F + 0.1G + 0.1H \]

As if convolves PSF at P in the input!
Point Spread Function

- Assuming a shift-invariant and linear imaging system

\[ P' = 0.1A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.1G + 0.1H \]

As if convolves PSF at P in the input!
Point Spread Function

- Assuming a shift-invariant and linear imaging system

\[ P' = 0.1A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.1G + 0.1H \]

As if convolves PSF at P in the input!
Point Spread Function

- Assuming a shift-invariant and linear imaging system

\[ P' = 0.1A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.1G + 0.1H \]

As if convolves PSF at P in the input!
Point Spread Function

- Assuming a shift-invariant and linear imaging system

P' = 0.1A + 0.1B + 0.1C +
  0.1D + 0.2P + 0.1E +
  0.1F + 0.1G + 0.1H

As if convolves PSF at P in the input!
Point Spread Function

- Only true if PSF is “radial symmetric”

\[
P' = 0.2A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.05G + 0.05H
\]
Point Spread Function

- Only true if PSF is “radial symmetric”

\[
P' = 0.2A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.05G + 0.05H
\]
Point Spread Function

- Only true if PSF is “radial symmetric”

\[
P' = 0.2A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.05G + 0.05H
\]
Point Spread Function

- Only true if PSF is “radial symmetric”

\[
P' = 0.2A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.05G + 0.05H
\]
**Point Spread Function**

- Only true if PSF is “radial symmetric”

\[ P' = 0.2A + 0.1B + 0.1C + \\
0.1D + 0.2P + 0.1E + \\
0.1F + 0.05G + 0.05H \]

<table>
<thead>
<tr>
<th></th>
<th>P'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>0.1</td>
<td>0.05</td>
</tr>
<tr>
<td>0.1</td>
<td>0.05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>P'</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
</tr>
</tbody>
</table>

PSF

0.05 0.05 0.1
0.1 0.2 0.1
0.1 0.1 0.2
Point Spread Function

- Only true if PSF is “radial symmetric”

\[
P' = 0.2A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.05G + 0.05H
\]

<table>
<thead>
<tr>
<th>PSF</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05 0.05 0.1</td>
<td>A B C</td>
<td>P'</td>
</tr>
<tr>
<td>0.1 0.2 0.1</td>
<td>D P E</td>
<td></td>
</tr>
<tr>
<td>0.1 0.1 0.2</td>
<td>F G H</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>0.1</td>
<td>0.05</td>
<td>0.05</td>
</tr>
</tbody>
</table>
Point Spread Function

- Only true if PSF is “radial symmetric”

\[
P' = 0.2A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.05G + 0.05H
\]
Point Spread Function

- Only true if PSF is “radial symmetric”

$$P' = 0.2A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.05G + 0.05H$$

<table>
<thead>
<tr>
<th></th>
<th>0.2</th>
<th>0.1</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.05</td>
<td>0.05</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0.05</th>
<th>0.05</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
<td></td>
</tr>
</tbody>
</table>

PSF

Input

Output

$$P'$$
Point Spread Function

› Only true if PSF is “radial symmetric”

\[
P' = 0.2A + 0.1B + 0.1C + 0.1D + 0.2P + 0.1E + 0.1F + 0.05G + 0.05H
\]
Point Spread Function

- Typical imaging systems taking natural light as input (i.e., non-coherent light), e.g., camera lens, are linear and shift invariant
- Their PSFs are radial symmetric
- So, can think of a lens as applying a convolution to the input, where the convolution kernel is the PSF inherent to the lens
Deep Computational Camera

Optical Domain

Light

Sensor (O2E)

Electrical Domain

L_1  L_2  L_3  \ldots  L_N
Deep Computational Camera

Optical Domain

Light → L1 → Sensor (O2E) → Electrical Domain

L2 → L3 → ... → LN