CSE/STAT 416
Convolutional Neural Networks

Hunter Schafer
Paul G. Allen School of Computer Science & Engineering
University of Washington

May 26, 2021
Logistics

Almost the end of the quarter!

- 🎓 Wed (5/26, today!): Convolutional Neural Networks
- 😊 Thur (5/27): Section on PyTorch
- 🧠 Fri (5/28): LR9 Due
- 🚗 Mon (6/1): Holiday
- 📝 Tue (6/2): HW9 Due (out today, more on this later)
- 🎓 Wed (6/3): Victory Lap and next steps
  - No pre-lecture video
  - Will have a Checkpoint for review from the quarter
- 😊 Thur (6/4): Review
- 😊 Fri (6/5): LR10 Due (quarter reflection)
- 📝 Mon – Wednesday (6/7 - 6/9): Final exam
  - Will send email in next few days with more info and resources
A lot of the buzz about ML recently has come from recent advancements in **deep learning**.

When people talk about “deep learning” they are generally talking about a class of models called **neural networks** that are a loose approximation of how our brains work.
XOR

Notice that we can represent

\[ x[1] \text{ XOR } x[2] = (x[1] \text{ AND } \neg x[2]) \text{ OR } (\neg x[1] \text{ AND } x[2]) \]
NN to the Rescue

Neural Networks implicitly find these low level features for us!

![Diagram showing layers of neural network](Zeiler & Fergus '13)

Each layer learns more and more complex features
Hyperparameter Optimization

How do we choose hyperparameters to train and evaluate?

Grid search:
- Hyperparameters on 2d uniform grid

Random search:
- Hyperparameters randomly chosen

Bayesian Optimization:
- Hyperparameters adaptively chosen
Images are extremely high dimensional

- CIFAR-10 dataset are very small: $3@32\times32$
  - # inputs:
    \[
    3 \cdot 32 \cdot 32 = 3072
    \]

- For moderate sized images: $3@200\times200$
  - # inputs:
    \[
    3 \cdot 200 \cdot 200 = 120,000
    \]

Images are structured, we should leverage this
Idea: Reduce the number of weights that need to be learned by looking at local neighborhoods of image.

Use the idea of a convolution to reduce the number of inputs by combing information about local pixels.
Convolution

Use a kernel that slides across the image, computing the sum of the element-wise product between the kernel and the overlapping part of the image.

<table>
<thead>
<tr>
<th>Image</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 0 1</td>
<td>0 1 2</td>
</tr>
<tr>
<td>1 0 2</td>
<td>2 2 0</td>
</tr>
<tr>
<td>2 2 2</td>
<td>2 1 2</td>
</tr>
<tr>
<td>2 2 2</td>
<td>0 1 2</td>
</tr>
<tr>
<td>2 2 0</td>
<td>0 1 2</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>3 1 2</td>
<td>0 3 4</td>
</tr>
<tr>
<td>2 0 0</td>
<td>0 0 4</td>
</tr>
<tr>
<td>0 0 2</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

\[ \text{sum} \]
Convolution

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)
Convolution

The input image (blue), the kernel (dark blue, numbers lower right) slide over the image to produce a result (green)
You can specify a few more things about a kernel

- Kernel dimensions and values
- Padding size and padding values
- Stride (how far to jump) values

For example, a 3x3 kernel applied to a 5x5 image with 1x1 zero padding and a 2x2 stride.
What is the result of applying a convolution using this kernel on this input image?

Use 1x1 zero padding and a 2x2 stride

<table>
<thead>
<tr>
<th>Image</th>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4</td>
<td>1 1</td>
</tr>
<tr>
<td>5 6 7 8</td>
<td>0 2</td>
</tr>
<tr>
<td>9 10 11 12</td>
<td></td>
</tr>
<tr>
<td>13 14 15 16</td>
<td></td>
</tr>
</tbody>
</table>
What is the result of applying a convolution using this kernel on this input image?

Use 1x1 zero padding and a 2x2 stride
Another core operation that is similar to a convolution is a **pool**.

- Idea is to down sample an image using some operation
- Combine local pixels using some operation (e.g. max, min, average, median, etc.)

Typical to use **max pool** with 2x2 filter and stride 2

- Tends to work better than average pool
Combine convolutions and pools into pre-processing layers on image to learn a smaller, information dense representation.

Example architecture for hand-written digit recognition

- Each convolution section uses many different kernels (increasing depth of channels)
- Pooling layers down sample each channel separately
- Usually ends with fully connected neural network
Why does this help?

- Only need to learn a small number of values (kernel weights) that get applied to the entire image region by region
  - This is called weight-sharing
  - Gives efficiency + shift invariance
- Pooling helps reduce the number of inputs by “blurring” the image without losing too much info.
Brain Break
How does this work if there is more than one input channel?

- Usually, use a 3 dimensional tensor as the kernel to combine information from each input channel.

```
size: 3x3x3
```

How many weights in the kernel? 
3 \times 3 \times 3 = 27
Another way of thinking about this process is each kernel is a neuron that looks at the kernel-size pixels in a neighborhood.

If there are 5 output channels in a conv layer, only need to learn the weights for the 5 neurons.

- These neurons are a bit different since they look at the pixels that overlap with the window at each position.
Consider a plain neural network below, how many weights need to be learned?

Completely ignore intercept terms

\[ 210 + 17 = 227 \]
Consider a plain neural network below, how many weights need to be learned?

Completely ignore intercept terms

15 10 5 2
Consider solving a digit recognition task on 28x28 images. Suppose I wanted to use a hidden layer with 84 neurons

**Without Convolutions:**

Number of weights: \[ 784 \times 84 + 84 \times 10 = 66,696 \]
Consider solving a digit recognition task on 28x28 images. Suppose I wanted to use a hidden layer with 84 neurons with Convolutions (assume \( n_1 = 10 \), \( n_2 = 20 \)).

\[
\text{Total weights learned} = 250 + 5000 + 27,720 = 32,970
\]

With Convolutions:

- **Conv-1**: 5x5 kernel, valid padding, \( 5 \times 5 \times 10 \times 20 = 5000 \) weights for one kernel
- **Conv-2**: 5x5 kernel, valid padding, \( 5 \times 5 \times 10 \times 20 = 5000 \) weights for one kernel
- **Max-Pooling** (2x2) for both layers
- **fc-3** (Fully-Connected Neural Network) with 84 units
- **fc-4** (Fully-Connected Neural Network) with 10 units

- **Inputs flattened**: \( 4 \times 4 \times 20 = 320 \)
- **Weights in fc-3 + fc-4**: \( 320 \times 84 + 84 \times 10 = 27,720 \)
CNNs generally (not always) have architectures that look like the following:

- A series of Convolution + Activation Functions and Pooling layers. It's very common to do a pool after each convolution.
- Then after some number of these operations, flatten the image to work with the final neural network.
The learned kernels are exactly the “features” for computer vision!
They start simple (corners, edges) and get more complex after more layers

Example detectors learned

Example interest points detected

[Zeiler & Fergus '13]
CNNs have had remarkable success in practice

LeNet, 1990s
LeNet made 82 errors on MNIST (popular hand-written digit dataset).
CNN Success

ImageNet 2012 competition:
- 1.2M training images
- 1000 categories

Winner: SuperVision
- 8 layers, 60M parameters [Krizhevsky et al. ‘12]
- Top-5 Error: 17%
CNN Success

Won 2014 ImageNet challenge with 6.66% top-5 error rate

Huge CNN depth has proven helpful in recognition systems... Maybe because images contain hierarchical structure (faces contain eyes contain edges, etc.)
Applications

Image Classification

Input: $\mathbf{x}$
Image pixels

Output: $\mathbf{y}$
Predicted object

Scene Parsing [Farabet et al. ‘13]
Applications


Product Recommendation
Brain Break
Pros

No need to manually engineer features, enable automated learning of features

Impressive performance gains
- Image processing
- Natural Language Processing
- Speech recognition

Making huge impacts in most fields
Cons

Requires a LOT of data

Computationally really expensive
  - Environmentally, extremely expensive (Green AI)

Hard to tune hyper-parameters
  - Choice of architecture (we’ve added even more hyper-parameters)
  - Learning algorithm

Still not very interpretable
A Tale of 2 Tasks

If we don’t have a lot of data for Task 2, what can we do?

**Idea:** Use a model that was trained for one task to help learn another task.

- An old idea, explored for deep learning by Donahue et al. ‘14 & others
What is learned in a neural network?

Initial layers are low-level and very general.
- Usually not sensitive-specific to the task at hand

**Neural net trained for Task 1: cat vs. dog**

- More generic
  - Can be used as feature extractor

- Very specific
to Task 1
  - Should be ignored for other tasks
Transfer Learning

Share the weights for the general part of the network

Neural net trained for Task 1: cat vs. dog

Use simple classifier e.g., logistic regression, SVMs, nearest neighbor, ...

More generic
Can be used as feature extractor

Very specific
to Task 1
Should be ignored for other tasks

Keep weights fixed!

Re-train
Transfer Learning

If done successfully, transfer learning can really help. Can give you

- A higher **start**
- A higher **slope**
- A higher **asymptote**
While NNs have had amazing success, they also have some baffling failures.

“No one adds noise to things in real applications”

**Not true!**

- Hackers will hack
- Sensors (cameras) are noisy!
NN Failures

They even fail with “natural” transformations of images

[Azulay, Weiss preprint]
Objects can be created to trick neural networks!

i made a breakthrough. it turns out juggalo makeup defeats facial recognition successfully. if you want to avoid surveillance, become a juggalo i guess
Datasets, like ImageNet, are generally biased.

One approach is to augment your dataset to add random permutations of data to avoid bias.
Further Reading

Dealing with Variable Length Sequences (e.g. language)
- Recurrent Neural Networks (RNNs)
- Long Short Term Memory Nets (LSTMs)
- [http://colah.github.io/posts/2015-08-Understanding-LSTMs/](http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Reinforcement Learning
- [Google DeepMind AlphaGo Zero](https://www.youtube.com/watch?v=ZByr9E04fzE)

Generative Adversarial Networks
- [How to learn synthetic data](https://distill.pub/2017/gan-tutorial/)

Green AI
Your last assignment involves using a modern neural network library to make predictions using the CIFAR-10 dataset.

We recommend you use Google Colab for this assignment so that you can use their free GPU.

Your first task is to read through the PyTorch tutorial to learn how to use their library.

- Nobody
- Google Colab:

You get a GPU and you get a GPU

Everyone gets a GPU