CSC413 Neural Networks and Deep Learning

Lecture 4: Convolutional Neural Networks

January 30/ February 1, 2024

Table of Contents

Lecture Plan

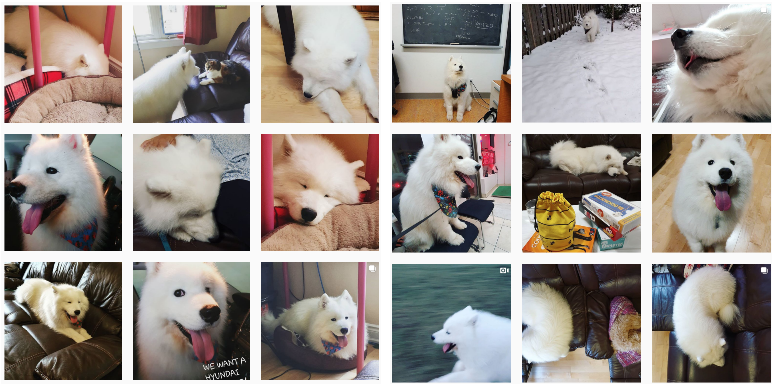
- Challenges of Computer Vision Problems
- Convolutional Neural Networks
- Training of a ConvNet
- Datasets and Architectures for Computer Vision Problems

Section 1

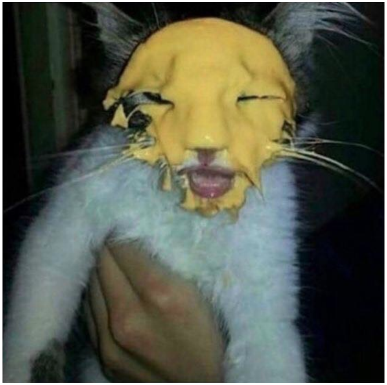
Computer Vision Problem

Computer vision is hard

- Object change in pose, size, viewpoint, background, illumination
- Some objects are hidden behind others: *occlusion*



Computer vision is really hard



How can you “hard code” an algorithm that still recognizes that this is a cat?

Convolutional Layer: A New Layer

So far in the course, we have seen two types of layers:

- Fully connected layer
- Embedding layer (i.e. lookup tables)

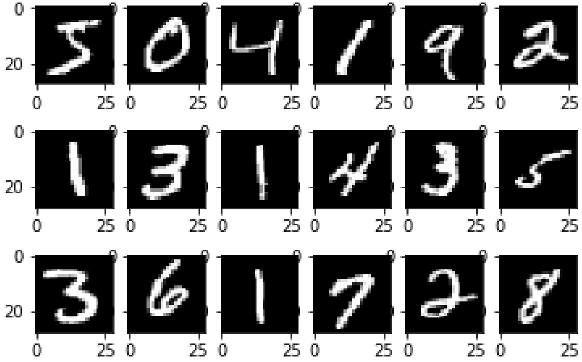
Different layers could be stacked together to build powerful models.

Let's add another layer type: **convolution layer**

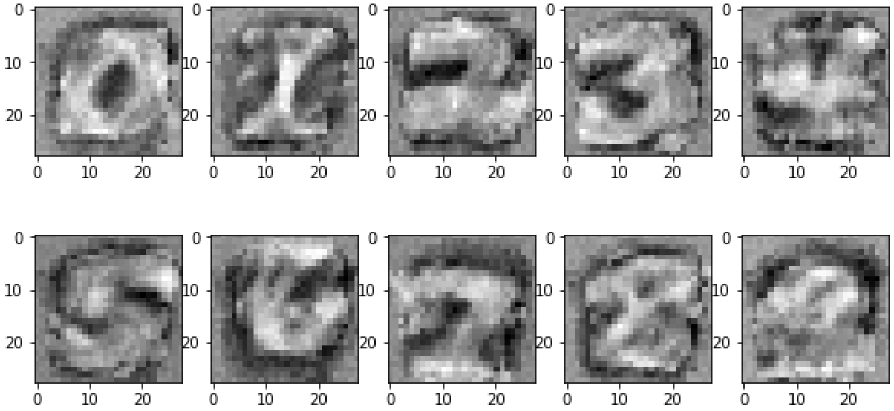
Convolutional layers have **inductive biases** suitable for computer vision tasks.

Working with Small Images

Consider MNIST images, which are 28×28 black and white images.

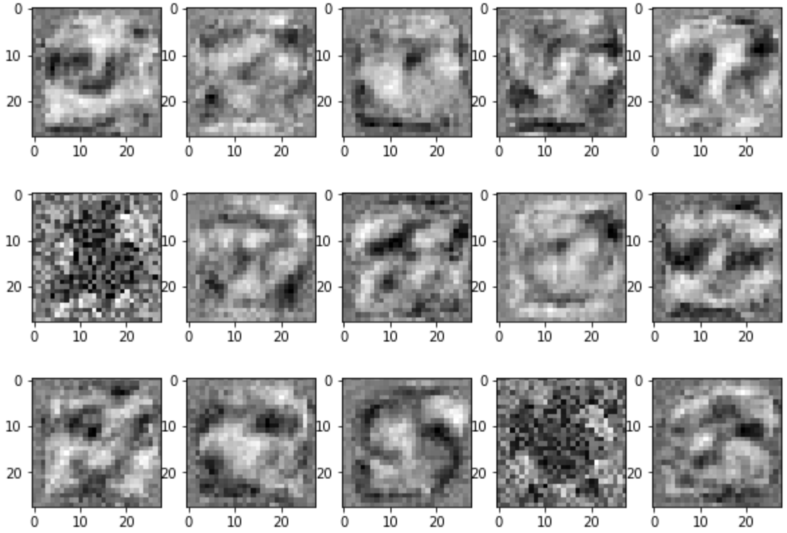


Logistic Regression Weights



Q: How many parameters do we have here?

MLP Weights (first layer)



Working with Large Images

- Suppose you have an image that is 200 pixels x 200 pixels
- There are 500 units in the first hidden layer

Q: How many **parameters** will there be in the first layer?

A: $200 \times 200 \times 500 + 500 =$ over 20 million!

Q: Why might using a fully connected layer be problematic?

- computing predictions (forward pass) will take a long time
- large number of weights requires a lot of training data to avoid overfitting
- small shift in image can result in large change in prediction

Section 2

Convolutional Layers

Limitation of Fully Connected Layers

Suppose we want to train a network that takes a 200 $\times$ 200 RGB image as input.

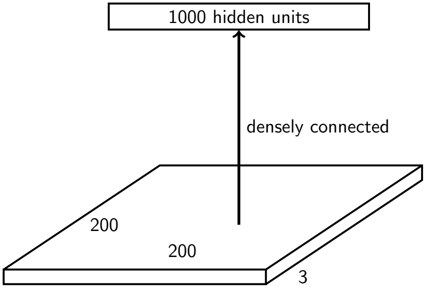


Figure 1: Fully (or densely) Connect Layer

What is the problem with having this as the first layer?

- Too many parameters! Input size = 200 $\times$ 200 $\times$ 3 = 120K. Parameters = 120K $\times$ 1000 = 120 million.

Limitation of Fully Connected Layers

In the fully connected layer, each feature (hidden unit) looks at the **entire image**. Since the image is a BIG object, we end up with lots of parameters.

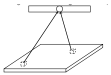


Figure 2: Each neuron connects to all neurons from previous layer

But, do we really expect to learn a useful feature at the first layer which depends on pixels that are spatially far away?

The far away pixels will probably belong to different objects (or object sub-parts). Little correlation.

We want the incoming weights to focus on **local** patterns of the input image.

Limitation of Fully Connected Layers

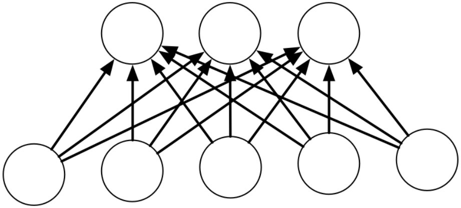
The same sorts of features that are useful in analyzing one part of the image will probably be useful for analyzing other parts as well.

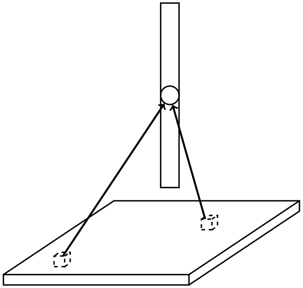
- Examples: edges, corners, contours, object parts

We want a neural net architecture that lets us learn a set of feature detectors **shared** at all image locations.

From Fully Connected to Convolutional Layer

Let's look at the Fully Connected layer closely:

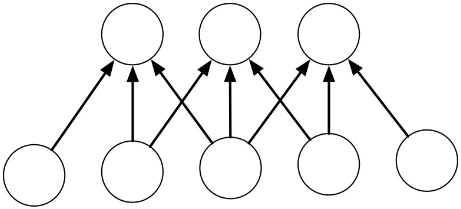
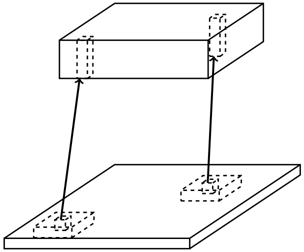




Each hidden unit looks at the entire image.

Locally Connected Layer

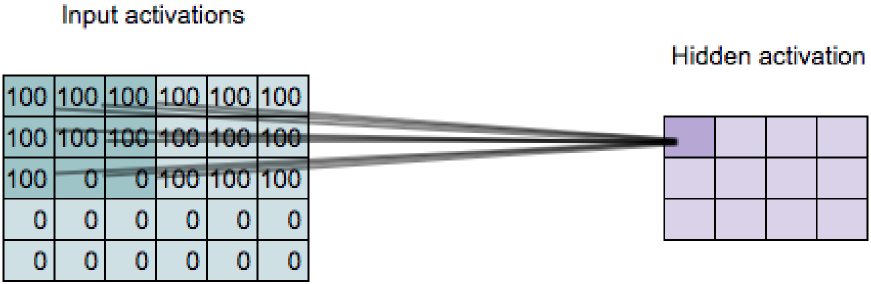
Let us limit the number of neurons from previous layer each unit is connected to. This adds **locality**.

Each column of hidden units looks at a small region of the image.

Locally Connected Layer

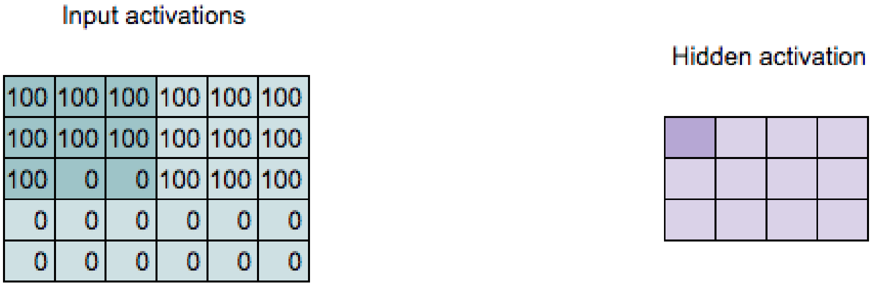
Let us see how this actually work. Suppose each neuron has a 3×3 connection field.



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

Q: How many parameters do need to describe the connections of this neuron?

Locally Connected Layer

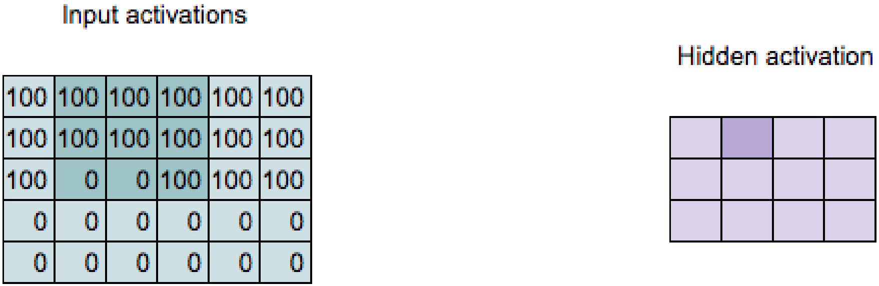


100	100	100	100	100	100
100	100	100	100	100	100
100	0	0	100	100	100
0	0	0	0	0	0
0	0	0	0	0	0

Each neuron has a separate set of weights.

(Remove lines for readability)

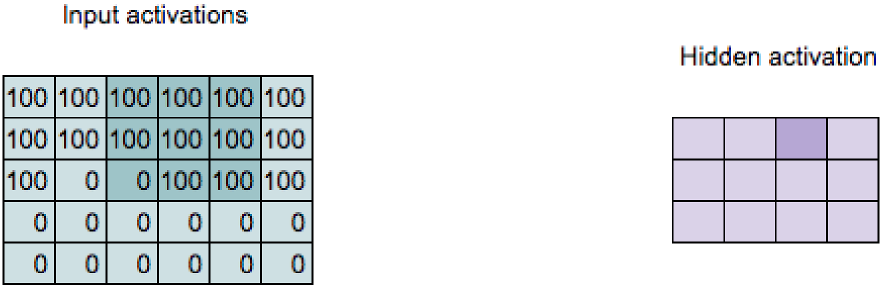
Locally Connected Layer



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

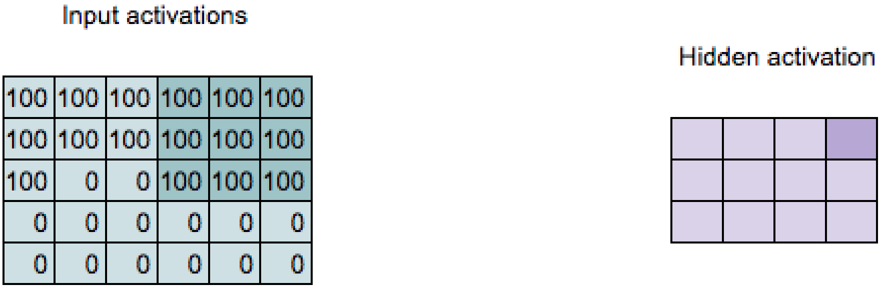
Hidden unit geometry has a 2D geometry consistent with the input

Locally Connected Layer



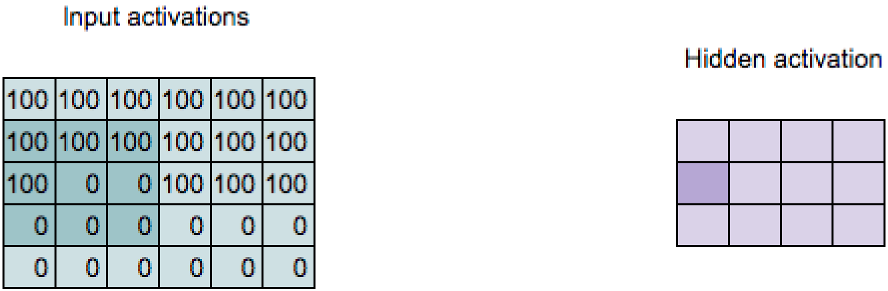
100	100	100	100	100	100
100	100	100	100	100	100
100	0	0	100	100	100
0	0	0	0	0	0
0	0	0	0	0	0

Locally Connected Layer

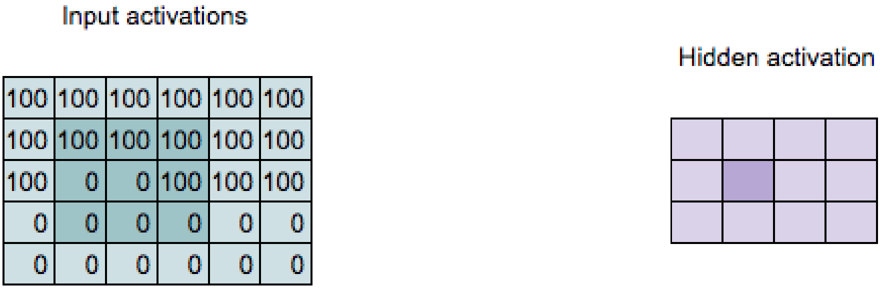


100	100	100	100	100	100
100	100	100	100	100	100
100	0	0	100	100	100
0	0	0	0	0	0
0	0	0	0	0	0

Locally Connected Layer

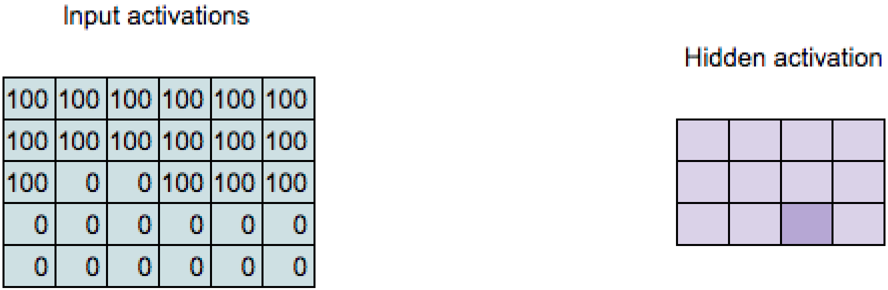


Locally Connected Layer



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

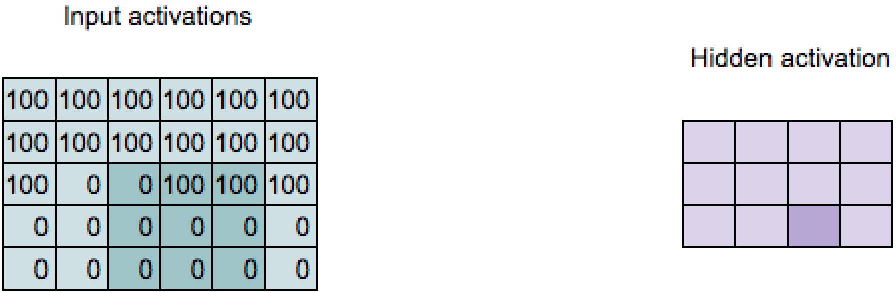
Locally Connected Layer



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

Q: Which region of the input is this hidden unit connected to?

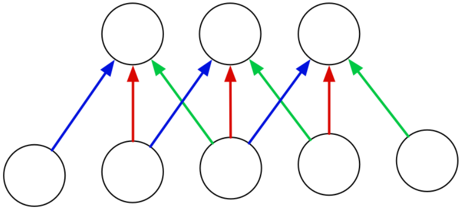
Locally Connected Layer

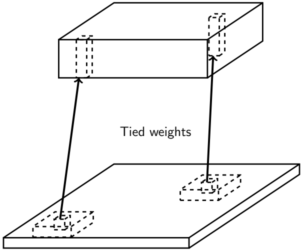


100	100	100	100	100	100
100	100	100	100	100	100
100	0	0	100	100	100
0	0	0	0	0	0
0	0	0	0	0	0

Convolution Layers

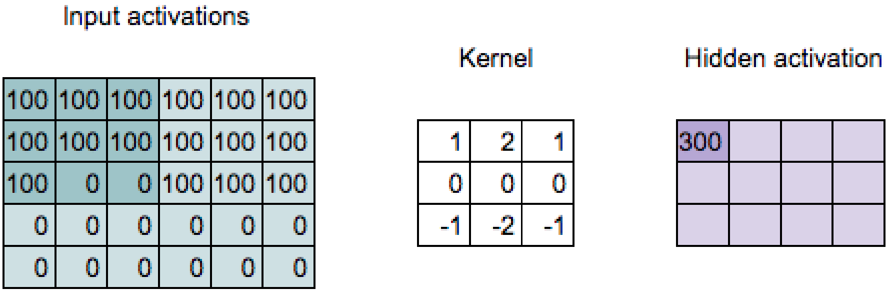
Similar to locally connected layer, but we share the weights between neurons.





Each column of hidden units looks at a small region of the image, and the weights are shared between all image locations.

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

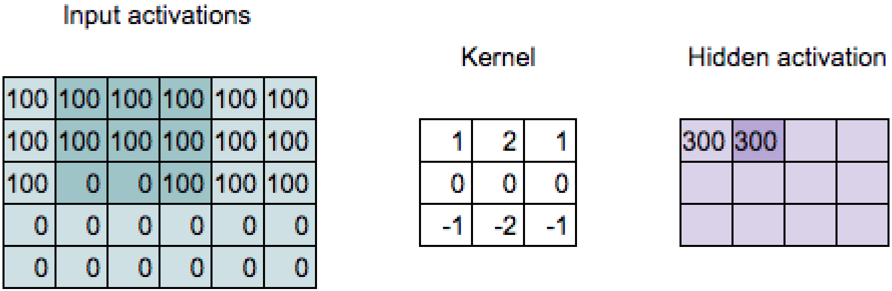
Hidden activation

300			

$$\begin{aligned}300 = &100 \times 1 + 100 \times 2 + 100 \times 1 + \\ &100 \times 0 + 100 \times 0 + 100 \times 0 + \\ &100 \times (-1) + 0 \times (-2) + 0 \times (-1)\end{aligned}$$

- The **kernel** or **filter** (middle) contains the trainable weights
- In our example, the **kernel size** is 3×3
- The "**convolved features**" is another term for the output hidden activation

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

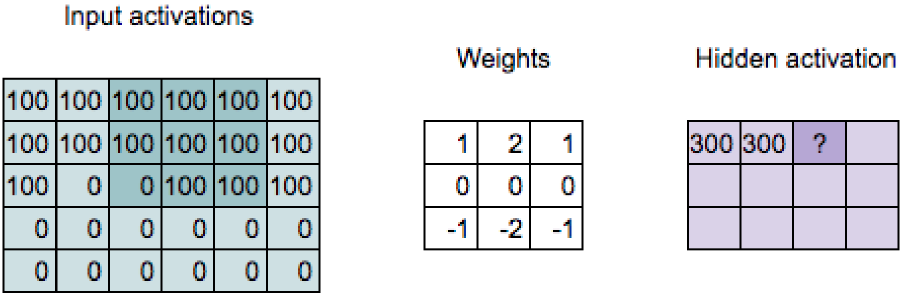
1	2	1
0	0	0
-1	-2	-1

Hidden activation

300	300		

$$\begin{aligned} 300 = & 100 \times 1 + 100 \times 2 + 100 \times 1 + \\ & 100 \times 0 + 100 \times 0 + 100 \times 0 + \\ & 0 \times (-1) + 0 \times (-2) + 100 \times (-1) \end{aligned}$$

Convolution Computation (Your Turn!)



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

Weights

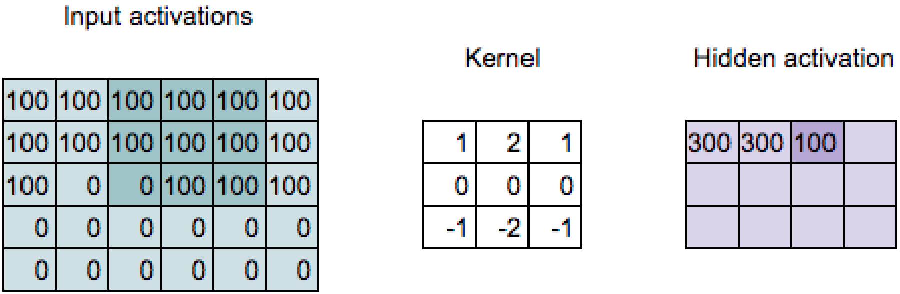
1	2	1
0	0	0
-1	-2	-1

Hidden activation

300	300	?	

Q: What is the value of the highlighted hidden activation?

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

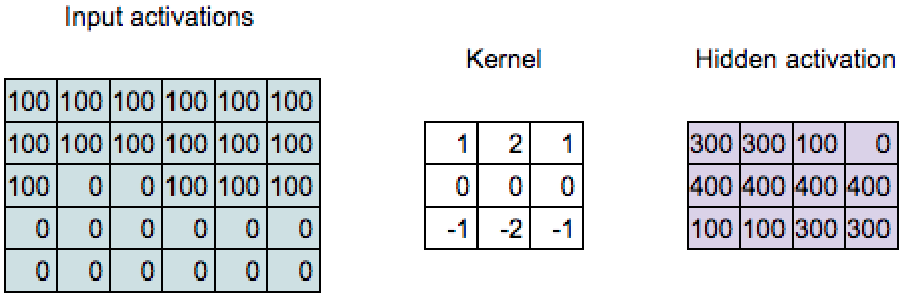
1	2	1
0	0	0
-1	-2	-1

Hidden activation

300	300	100	

$$\begin{aligned}100 = &100 \times 1 + 100 \times 2 + 100 \times 1 + \\ &100 \times 0 + 100 \times 0 + 100 \times 0 + \\ &0 \times (-1) + 100 \times (-2) + 100 \times (-1)\end{aligned}$$

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	0
400	400	400	400
100	100	300	300

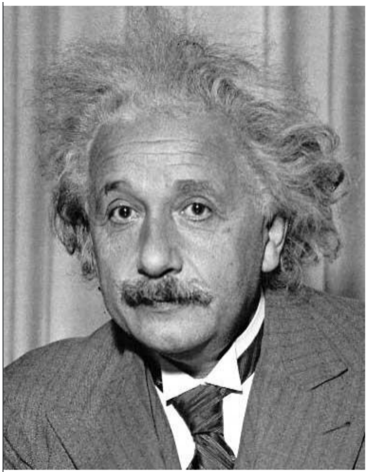
Convolution as a Feature Detection Filter

Convolution acts like a **filter** that glides over the image and detects certain features.

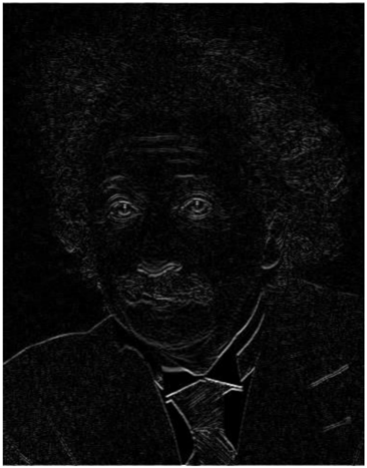
- "Feature": a pattern in a part of the image, like an edge or shape
- "Detection": output (activation) is high if the feature is present

Let us see how this looks like!

Sobel Filter - Weights to Detect Horizontal Edges

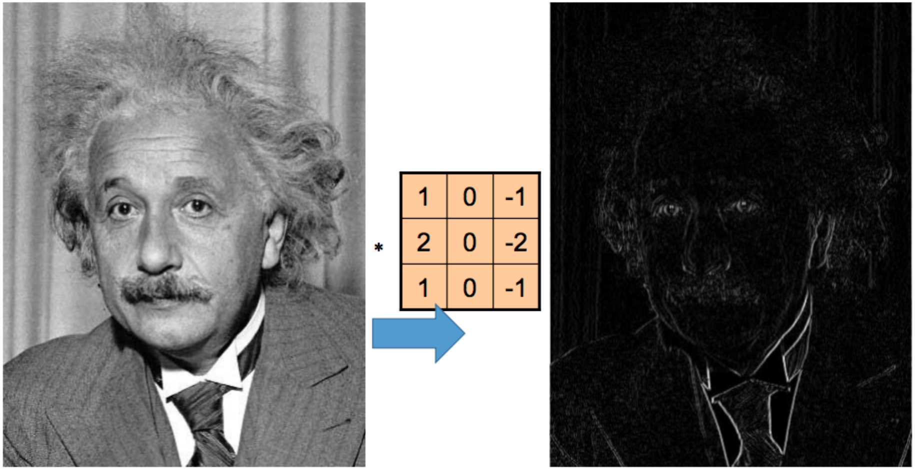


$$* \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$



Horizontal Edge
(absolute value)[8]

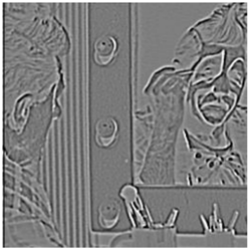
Sobel Filter - Weights to Detect Vertical Edges



Vertical Edge (absolute value)[7]

Weights to Detect Blobs

$$
* \begin{pmatrix}
0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \\
0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\
3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\
2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\
2 & 5 & 0 & -23 & -40 & -23 & 0 & 5 & 2 \\
2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\
3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\
0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\
0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0
\end{pmatrix}
$$



Q: What is the *kernel size* of this convolution?

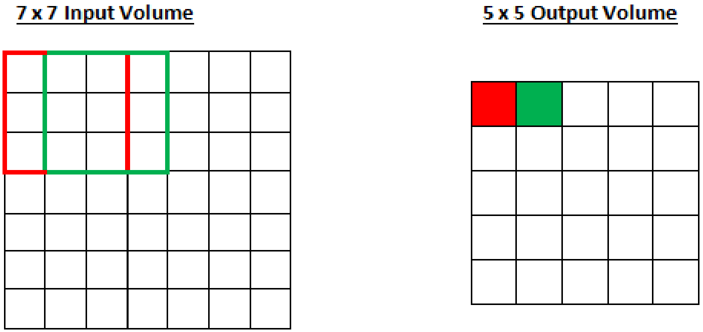
How to design kernels/filters?

- The kernels (filters) in previous examples were all designed manually.
- They could find apparently important features of an image, such as edges or blobs.
  - But why do we think those are important features for an image?
  - More importantly, are they the only important features that can be used by our AI agent to solve a computer vision task?
- What if we **learn** them instead?
  - We can in fact treat the values of a kernel/filter as *parameters* and learn them.

Exercise in Parameter Counting

Before moving on, let us do a quick exercise.

- Greyscale input image: 7×7
- Convolution **kernel**: 3×3
- Q: How many hidden units are in the output of this convolution?



Convolution Operator

Why do we call this thing **convolution** after all?

Convolution is a mathematical operator (similar to addition or multiplication) that is defined between two **signals** (or vectors or functions – depending on the context).

Consider two 1-dimensional signals (or arrays) a and b. Their convolution is defined as a new signal (or array) whose t-th element is

$$(a * b)_t = \sum_{\tau} a_\tau b_{t-\tau}.$$

Some basic properties of the convolution operator:

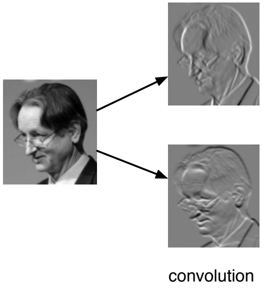
- Commutativity: $a * b = b * a$
- Linearity: $a * (\lambda_1 b + \lambda_2 c) = \lambda_1 a * b + \lambda_2 a * c$

It has many useful properties and interpretations that make it a useful mathematical concept for signal processing, control theory, probability theory, machine learning, etc.

Convolutional Networks

Let's turn to **convolutional networks**. These have two kinds of layers: **detection layers** (or **convolution layers**), and **pooling layers**.

The convolution layer has a set of filters. Its output is a set of **feature maps**, each one obtained by convolving the image with a filter.



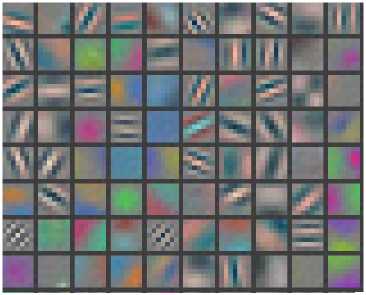


Image (right) Credit: Zeiler and Fergus, 2013, Visualizing and understanding convolutional networks

Convolutional Networks

It is common to apply a linear rectification nonlinearity: $y_i = \max(z_i, 0)$

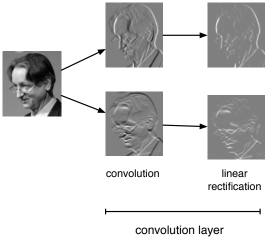


Figure 3: Network Rectify

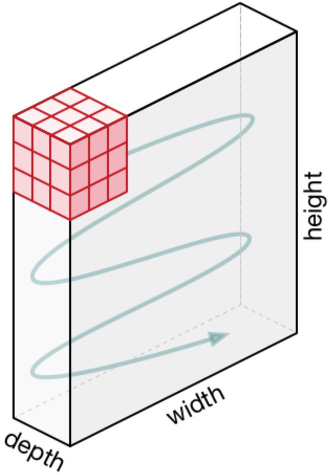
Why might we do this?

- Convolution is a linear operation. Therefore, we need a nonlinearity, otherwise 2 convolution layers would be no more powerful than 1.
- Two edges in opposite directions shouldn't cancel.

Other Considerations in using Convolution Layers

- What if we have a coloured image? How should we define a convolutional layer?
- What if we want to compute *multiple* features, instead of just one?

Convolution Layer with Coloured Image (RGB)



The kernel becomes a 3-dimensional tensor!

In this example, the kernel has size **3** $\times 3 \times 3$

Convolution Layer with Coloured Image (RGB)

Colour input image: $\mathbf{3} \times 7 \times 7$

Convolution kernel: $\mathbf{3} \times 3 \times 3$

Questions:

- How many units are in the output of this convolution?
- How many trainable weights are there?

Terminology

Input image: $3 \times 32 \times 32$

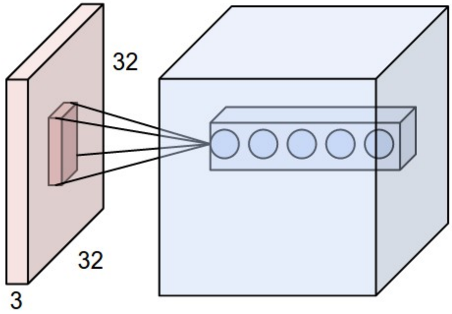
Convolution kernel: $\mathbf{3} \times 3 \times 3$

- The number 3 is the number of **input channels** or **input feature maps**

Detecting Multiple Features

Q: What if we want to detect many features of the input? (i.e. **both** horizontal edges and vertical edges, and maybe even other features?)

A: Have many convolutional filters!



Many Convolutional Filters

Input image: $3 \times 7 \times 7$

Convolution kernel: $3 \times 3 \times 3\times$ **5**

Questions:

- How many units are in the output of this convolution?
- How many trainable weights are there?

More Terminology

Input image of size $3 \times 32 \times 32$

Convolution kernel of $\mathbf{3} \times 3 \times 3 \times \mathbf{5}$

- The number 3 is the number of **input channels** or **input feature maps**
- The number 5 is the number of **output channels** or **output feature maps**

Example

Input features: $5 \times 32 \times 32$

Convolution kernel: $5 \times 3 \times 3 \times 10$

Questions:

- How many input channels are there?
- How many output channels are there?
- How many units are in the higher layer?
- How many trainable weights are there?

Computing Convolution – A Closer Look

Method 1: translate-and-scale

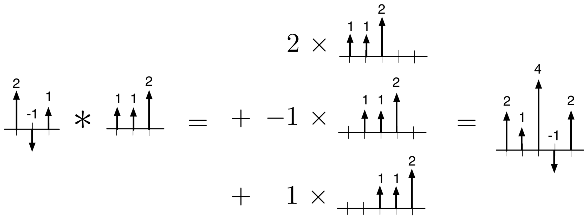


Figure 4: Conv Impulse

Computing Convolution – A Closer Look

Method 2: flip-and-filter

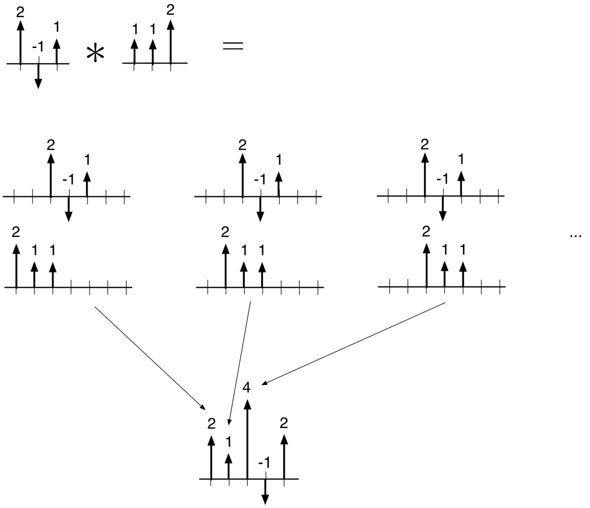


Figure 5: Conv Filter

Computing Convolution – A Closer Look

Convolution can also be viewed as matrix multiplication:

$$(2, -1, 1) * (1, 1, 2) = \begin{pmatrix} 1 & & \\ 1 & 1 & \\ 2 & 1 & 1 \\ & 2 & 1 \\ & & 2 \end{pmatrix} \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}$$

- Naive method to compute the convolution of two N-dimensional array has the computational cost of $O(N^2)$.
- We can use techniques based on Fast Fourier Transform (FFT) to compute it in $O(N \log N)$.
- Because of parallelism of modern GPUs, the matrix multiplication approach is a common and beneficial way to compute the convolution.

2D Convolution

2D convolution is defined analogously to 1D convolution.

If A and B are two 2-D arrays (or signals), then:

$$(A * B)_{ij} = \sum_s \sum_t A_{st} B_{i-s,j-t}.$$

Computing 2D Convolution – A Closer Look

Method 1: Translate-and-Scale

$$1 \times \begin{array}{|c|c|c|c|}\hline 1 & 3 & 1 & \\ \hline 0 & -1 & 1 & \\ \hline 2 & 2 & -1 & \\ \hline & & & \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|}\hline 1 & 3 & 1 \\ \hline 0 & -1 & 1 \\ \hline 2 & 2 & -1 \\ \hline \end{array} * \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 0 & -1 \\ \hline \end{array} = + \; 2 \times \begin{array}{|c|c|c|c|}\hline & 1 & 3 & 1 \\ \hline & 0 & -1 & 1 \\ \hline & 2 & 2 & -1 \\ \hline & & & \\ \hline \end{array} = \begin{array}{|c|c|c|c|}\hline 1 & 5 & 7 & 2 \\ \hline 0 & -2 & -4 & 1 \\ \hline 2 & 6 & 4 & -3 \\ \hline 0 & -2 & -2 & 1 \\ \hline \end{array}$$

$$+ -1 \times \begin{array}{|c|c|c|c|}\hline & & & \\ \hline & 1 & 3 & 1 \\ \hline & 0 & -1 & 1 \\ \hline & 2 & 2 & -1 \\ \hline \end{array}$$

Computing 2D Convolution – A Closer Look

Method 2: Flip-and-Filter

$$\begin{array}{|c|c|c|}\hline 1 & 3 & 1 \\ \hline 0 & -1 & 1 \\ \hline 2 & 2 & -1 \\ \hline \end{array} \ast \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 0 & -1 \\ \hline \end{array}$$

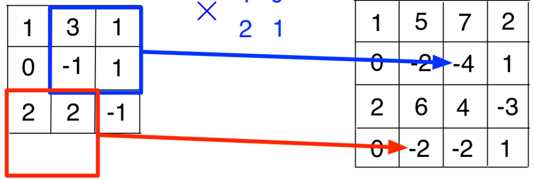


Figure 7: 2D Convolution Filter

Section 3

Pooling Layers

Consolidating Information

In a neural network with fully-connected layers, we reduced the number of units in each hidden layer

Q: Why?

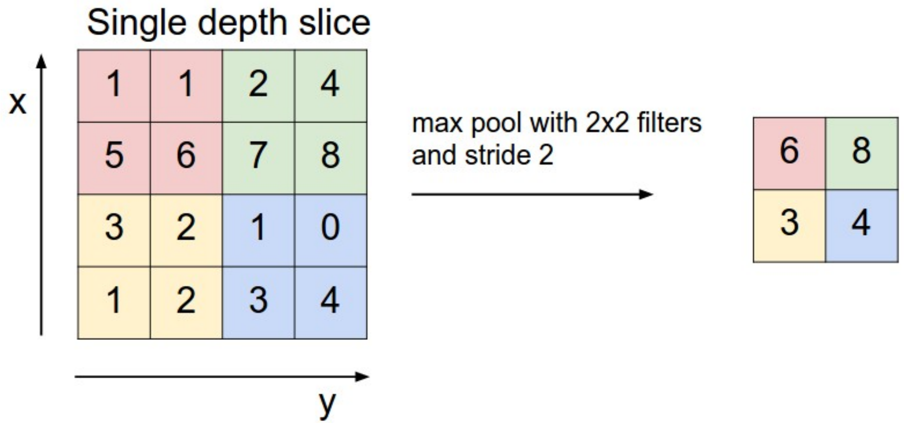
- To be able to consolidate information, and remove out information not useful for the current task

Q: How can we consolidate information in a neural network with convolutional layers?

- max pooling, average pooling, strided convolutions

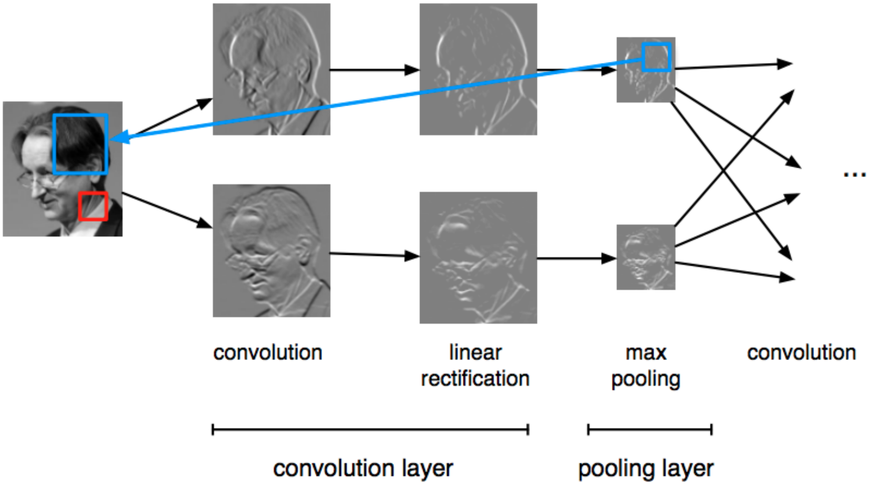
Max-Pooling

Idea: take the **maximum value** in each 2×2 grid.



Max-Pooling Example

We can add a max-pooling layer *after* each convolutional layer

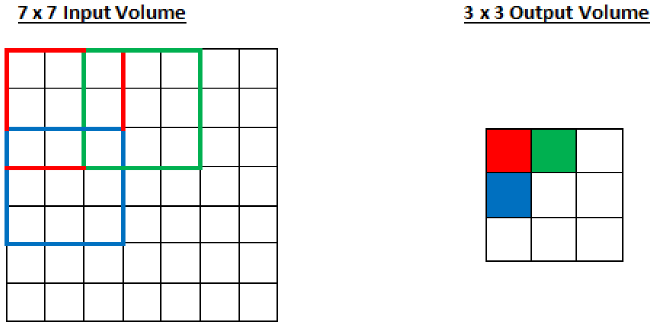


Average Pooling

- Average pooling (compute the average activation of a region)

Strided Convolution

Instead of pooling, we can use **strided** convolutions too:



Shift the kernel by **2** (stride=2) when computing the next output feature.

There are many variations. Take a look:

- https://arxiv.org/pdf/1603.07285.pdf
- https://github.com/vdumoulin/conv_arithmetic

Equivariance and Invariance

A network's responses should be robust to translations of the input. But this can mean two different things.

- Convolution layers are **equivariant**: if you translate the inputs, the outputs are translated by the same amount.
- We'd like the network's predictions to be **invariant**: if you translate the inputs, the prediction should not change.
- Pooling layers provide invariance to small translations.

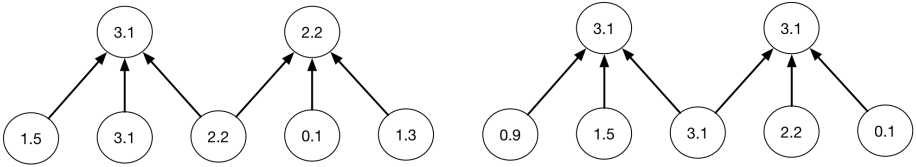
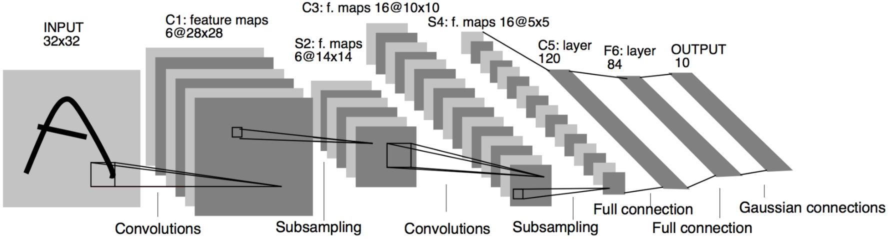


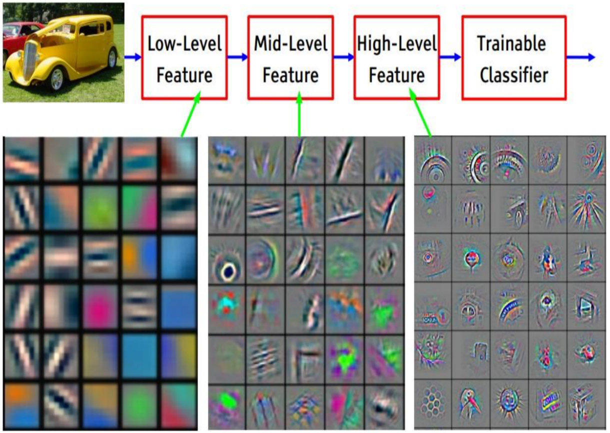
Figure 8: Pooling Invariance

Early Convolutional Architecture: LeNet Architecture



- Input: 32x32 pixel, greyscale image
- First convolution has 6 output features (5x5 convolution?)
- First subsampling is probably a max-pooling operation
- Second convolution has 16 output features (5x5 convolution?)
- ...
- Some number of fully-connected layers at the end

What features do CNN's detect?



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

Section 4

Training a Convolutional Network

Training a Convolutional Network

Q: How do we train a convolutional net?

A: With backpropagation, of course!

Recall what we need to do: Backprop is a message passing procedure, where each layer knows how to pass messages backwards through the computation graph. Let's determine the updates for convolution layers.

- We assume we are given the loss derivatives $\bar{y}_{i,t}$ with respect to the output units.
- We need to compute the cost derivatives with respect to the input units and with respect to the weights.

The only novel aspect is how we perform backpropagation with tied weights.

Let us take a close look . . . !

Convolution Layers

Each layer consists of several **feature maps**, or **channels** each of which is an array.

- If the input layer represents a grayscale image, it consists of one channel. If it represents a color image, it consists of three channels.

Each unit is connected to each unit within its receptive field in the previous layer. This includes *all* of the previous layer's feature maps.

Convolution Layers

For simplicity, focus on 1-D signals (e.g. audio waveforms). Suppose the convolution layer's input has J feature maps and its output has I feature maps. Let t index the locations. Suppose the convolution kernels have radius R, i.e. dimension $K = 2R + 1$.

Each unit in a convolution layer receives inputs from all the units in its receptive field in the previous layer:

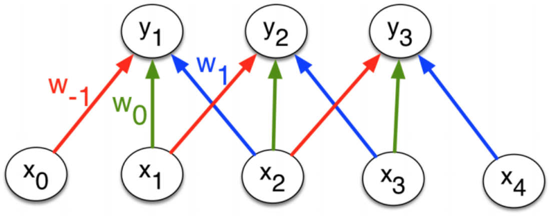
$$y_{i,t} = \sum_{j=1}^{J} \sum_{\tau=-R}^{R} w_{i,j,\tau} x_{j,t+\tau}.$$

In terms of convolution,

$$\mathbf{y}_i = \sum_j \mathbf{x}_j * \text{flip}(\mathbf{w}_{i,j}).$$

Backpropagation for Convolutional Layer w.r.t. Weights

Consider the computation graph for the weights:



$$\begin{tikzcd}
w_{-1} \arrow[r] \arrow[dr] \arrow[ddr] & y_1 \\
w_0 \arrow[ur] \arrow[r] \arrow[dr] & y_2 \\
w_1 \arrow[uur] \arrow[ur] \arrow[r] & y_3
\end{tikzcd}$$

Backpropagation for Convolutional Layer w.r.t. Weights

Each of the weights affects all the output units for the corresponding input and output feature maps.

$$y_{i,t} = \sum_{j=1}^{J} \sum_{\tau=-R}^{R} w_{i,j,\tau} x_{j,t+\tau}.$$

We compute the partial derivatives, which requires summing over all spatial locations:

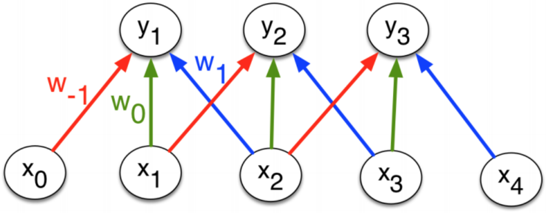
$$\bar{w}_{i,j,\tau} = \sum_{t} \bar{y}_{i,t} \frac{\partial y_{i,t}}{\partial w_{i,j,\tau}} = \sum_{t} \bar{y}_{i,t} x_{j,t+\tau}.$$

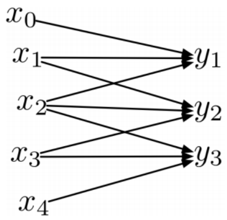
To decipher this, note that

- $y_{i,t}$ is the output at unit t of channel i.
- $x_{j,t+\tau}$ is the input $t+\tau$ of channel j.

Backpropagation for Convolutional Layer w.r.t. Input

Consider the computation graph for the inputs:





Each input unit influences all the output units that have it within their receptive fields. Using the multivariate Chain Rule, we need to sum together the derivative terms for all these edges.

Backpropagation for Convolutional Layer w.r.t. Input

Recall the formula for the convolution layer:

$$y_{i,t} = \sum_{j=1}^{J} \sum_{\tau=-R}^{R} w_{i,j,\tau} x_{j,t+\tau}.$$

We compute the derivatives, which requires summing over all the outputs units which have the input unit in their receptive field:

$$\bar{x}_{j,t} = \sum_{\tau} \bar{y}_{i,t-\tau} \, \frac{\partial y_{i,t-\tau}}{\partial x_{j,t}} = \sum_{\tau} \bar{y}_{i,t-\tau} \, w_{i,j,\tau}.$$

Written in terms of convolution,

$$\bar{\mathbf{x}}_j = \bar{\mathbf{y}}_j * \mathbf{w}_{i,j}.$$

Q: Why should we care about the derivative w.r.t. the input after all?

Section 5

Of Datasets and Architectures

Object Recognition

- Object recognition is the task of identifying which object category is present in an image.
- It is challenging because objects can differ widely in position, size, shape, appearance, etc., and we have to deal with occlusions, lighting changes, etc.
- Why we care about it
  - Direct applications to image search
  - Closely related to **object detection**, the task of locating all instances of an object in an image
    - Example: a self-driving car detecting pedestrians or stop signs
- Convolutional Networks have been behind many of the objective recognizers in the past 10-15 years.
  - More recently, Transformers are changing this!
- Let us look at some of the datasets and architectures behind various object recognition tasks.

Datasets

- 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.
- Some questions to consider:
  - Which categories to include?
  - Where should the images come from?
  - How many images to collect?
  - How to normalize (preprocess) the images?

Image Classification

- Conv nets are just one of many possible approaches to image classification. But they have been behind many successes in the past 10-15 years.
- 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.

MNIST Dataset

- 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
    - The assumption is that the digit recognizer would be part of a larger pipeline that segments and normalizes images.
- 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.

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.

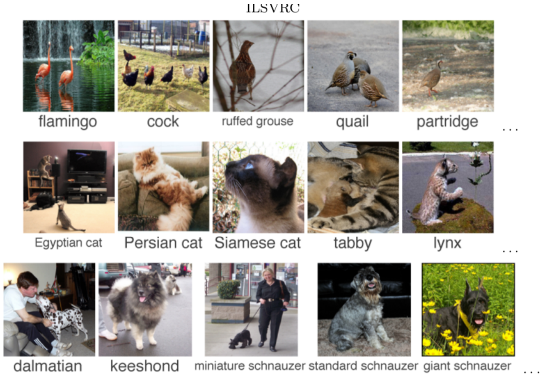


Figure 9: ImageNet

ImageNet

- Used for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual benchmark competition for object recognition algorithms.
- **Design decisions:**
  - **Categories:** Taken from a lexical database called WordNet
    - WordNet consists of “synsets”, or sets of synonymous words
    - They tried to use as many of these as possible; almost 22,000 as of 2010
    - Of these, they chose the 1000 most common for the ILSVRC
    - The categories are really specific, e.g., hundreds of kinds of dogs
  - **Size:** 1.2 million full-sized images for the ILSVRC
  - **Source:** Results from image search engines, hand-labeled by Mechanical Turkers
    - Labeling such specific categories was challenging; annotators had to be given the WordNet hierarchy, Wikipedia, etc.
  - **Normalization:** none, although the contestants are free to do preprocessing

ImageNet

Images and object categories vary on a lot of dimensions

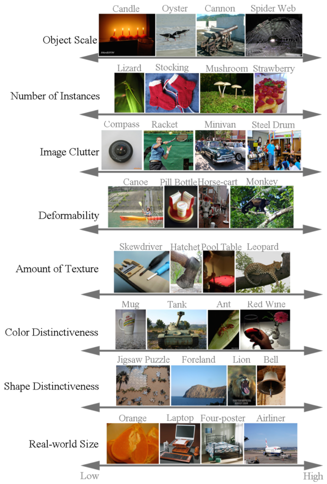


Figure 10: ImageNet

ImageNet

- MNIST: 60MB
- ImageNet: 50GB

LeNet

- Here's the LeNet architecture, which was applied to handwritten digit recognition on MNIST in 1998.

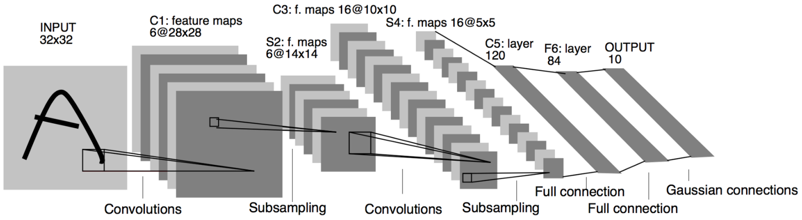


Figure 11: LeNet Architecture

Size of a Convolutional Network

- Ways to measure the size of a network:
    - Number of units: This is important because the activations need to be stored in memory during training (i.e.∼backprop).
    - Number of weights: This is important because the weights need to be stored in memory, and because the number of parameters determines the amount of overfitting.
    - Number of connections: This is important because there are approximately 3 add-multiply operations per connection (1 for the forward pass, 2 for the backward pass).
- We saw that a fully connected layer with M input units and N output units has MN connections and MN weights.
- The story for Convolutional Networks is more complicated.

Size of a Convolutional Network

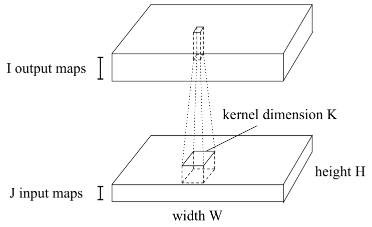


Figure 12: Convolution Layer Size

- **Fully Connected Layer**
  - **# Output Units:** WHI
  - **# Weights:** W^2H^2IJ
  - **# Connections:** W^2H^2IJ
- **Convolution Layer**
  - **# Output Units:** WHI
  - **# Weights:** K^2IJ
  - **# Connections:** WHK^2IJ

Sizes of Layers in LeNet:

Layer	Type	# Units	# Connections	# Weights
C1	Convolution	4704	117,600	150
S2	Pooling	1176	4704	0
C3	Convolution	1600	240,000	2400
S4	Pooling	400	1600	0
F5	Fully Connected	120	48,000	48,000
F6	Fully Connected	84	10,080	10,080
Output	Fully Connected	10	840	840

Conclusions?

- Rule of thumb
  - Most units and connections are in the convolution layers.
  - Most weights are in the fully connected layers.
- Larger layers face resource limitations (computation time, memory).
- Convolutional networks have grown significantly larger since 1998.

Size Comparison:

	LeNet (1989)	LeNet (1998)	AlexNet (2012)
Classification Task	Digits	Digits	Objects
Categories	10	10	1,000
Image Size	16×16	28×28	$256 \times 256 \times 3$
Training Examples	7,291	60,000	1.2 million
Units	1,256	8,084	658,000
Parameters	9,760	60,000	60 million
Connections	65,000	344,000	652 million

AlexNet

- AlexNet, 2012. 8 weight layers. 16.4% top-5 error (i.e. the network gets 5 tries to guess the right category).

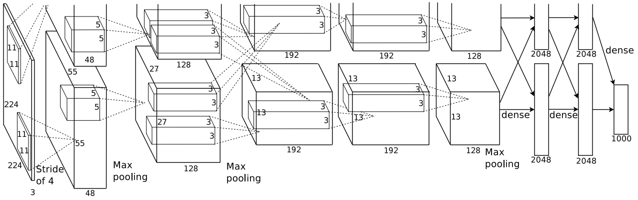


Figure 13: AlexNet Architecture

Image credit: Krizhevsky et al., 2012

- They used lots of tricks we've covered in this course (ReLU units, weight decay, data augmentation, SGD with momentum, dropout).
- AlexNet's stunning performance on the ILSVRC is what set off the deep learning boom of 2010s.

GoogLeNet

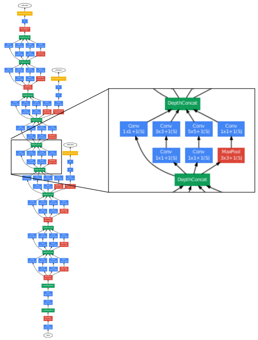


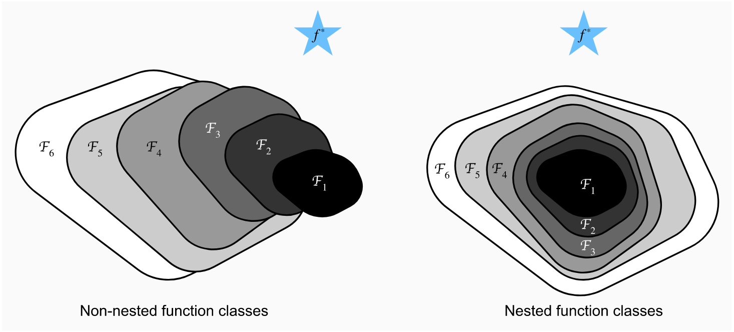
Figure 14: GoogLeNet

- GoogLeNet, 2014. 22 weight layers.
- Fully convolutional (no fully connected layers).
- Convolutions are broken down into a bunch of smaller convolutions.
- 6.6% test error on ImageNet.

GoogLeNet

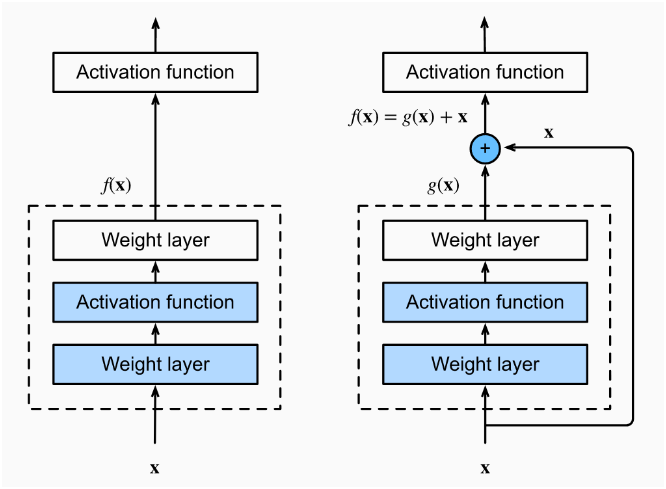
- They were really aggressive about cutting the number of parameters.
    - Motivation: train the network on a large cluster, run it on a cell phone
        - Memory at test time is the big constraint.
        - Having lots of units is OK, since the activations only need to be stored at training time (for backpropagation).
        - Parameters need to be stored both at training and test time, so these are the memory bottleneck.
    - How they did it
        - No fully connected layers (remember, these have most of the weights)
        - Break down convolutions into multiple smaller convolutions (since this requires fewer parameters total)
    - GoogLeNet has "only" 2 million parameters, compared with 60 million for AlexNet
    - This turned out to improve generalization as well. (Overfitting can still be a problem, even with over a million images!)

Residual Networks (ResNet) Architecture



- Suppose we add another layer. How can we ensure that the new set of represented functions contains the old set, before the layer was added?
- Why do we need this? We'd like to get larger (nested) sets of functions as we add more layers and not just different (non-nested) sets.

ResNet Blocks



- Side effect of adding identity $f(x) = x + g(x)$: better gradient propagation
- See https://d2l.ai/chapter_convolutional-modern/resnet.html

Residual Networks (ResNet) Architecture

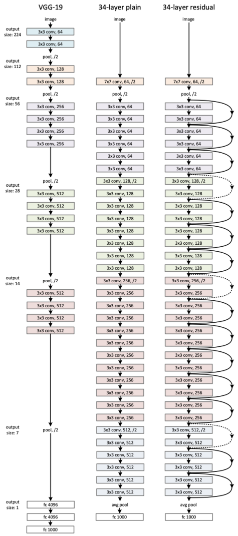


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

DenseNet Blocks

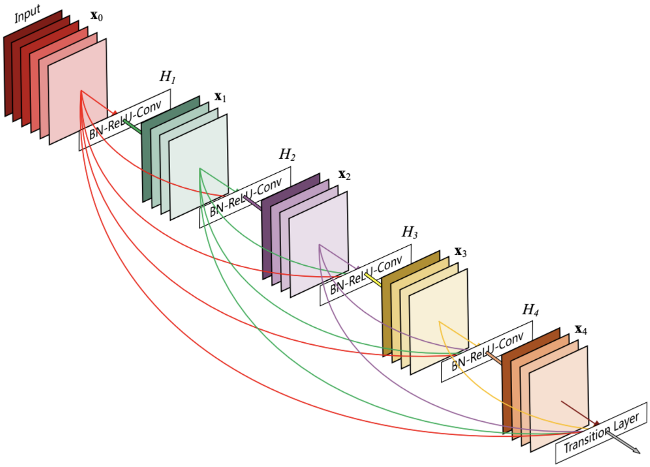


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

DenseNet Architecture

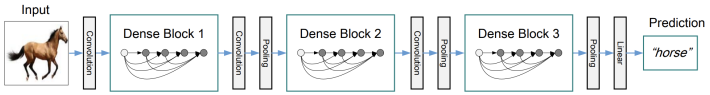


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

See https://d2l.ai/chapter_convolutional-modern/densenet.html

Classification

- ImageNet results over the years. Note that errors are top-5 errors (the network gets to make 5 guesses).

Year	**Model**	**Top-5 error**
2010	Hand-designed descriptors + SVM	28.2%
2011	Compressed Fisher Vectors + SVM	25.8%
2012	AlexNet	16.4%
2013	a variant of AlexNet	11.7%
2014	GoogLeNet	6.6%
2015	deep residual nets	4.5%

- Human-performance is around 5.1%.
- They stopped running the object recognition competition because the performance is already so good.

Beyond Classification

- The classification nets map the entire input image to a pre-defined class categories.
- But there are more than just class labels in an image.
  - where is the foreground object? how many? what is in the background?

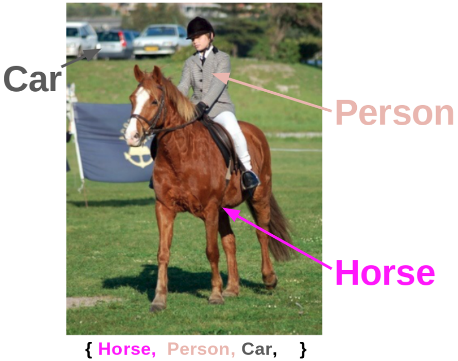


Figure 15: Segmentation

(PASCAL VOC 2012)

Semantic Segmentation

- Semantic segmentation, a natural extension of classification, focuses on making dense classification of class labels for **every pixel**.
- It is an important step towards complete scene understanding in computer vision.
  - Semantic segmentation is a stepping stone for many of the high-level vision tasks, such as object detection, Visual Question Answering (VQA).
- A naive approach is to adapt the existing object classification conv nets for each pixel. This works surprisingly well.

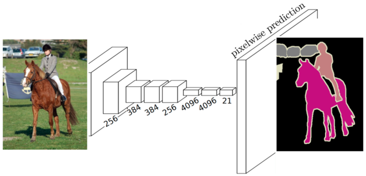


Figure 16: Fully Convolutional Networks

(Fully Convolutional Networks, 2015)

Semantic Segmentation

- After the success of CNN classifiers, segmentation models quickly moved away from hand-crafted features and pipelines but instead use CNN as the main structure.
- Pre-trained ImageNet classification network serves as a building block for all the state-of-the-art CNN-based segmentation models.

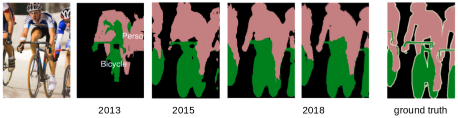


Figure 17: Segmentation Progress

(from left to right: Li, et. al., (CSI), CVPR, 2013; Long, et. al., (FCN), CVPR 2015; Chen et. al., (DeepLab), PAMI 2018)

Supervised Pre-training and Transfer Learning

- In practice, we will rarely train an image classifier from scratch.
  - It is unlikely we will have millions of cleanly labeled images for our specific datasets.
- If the dataset is a computer vision task, it is common to fine-tune a pre-trained conv net on ImageNet or OpenImage.
- Just like semantic segmentation tasks, we will fix most of the weights in the pre-trained network. Only the weights in the last layer will be randomly initialized and learnt on the current dataset/task.

Supervised Pre-training and Transfer Learning

- When to fine-tune?
  - How many training examples we have in the new dataset/task?
    - Fewer new examples: more weights from the pre-trained networks are fixed.
  - How similar is the new dataset to our pre-training dataset? Microscopy images v.s. natural images:
    - More fine-tuning is needed for dissimilar datasets.
  - Learning rate for the fine-tuning stage is often much lower than the learning rate used for training from scratch.