CSC 411: Lecture 11: Neural Networks II

Richard Zemel, Raquel Urtasun and Sanja Fidler

University of Toronto

Today

- Deep learning for Object Recognition

Neural Nets for Object Recognition

- People are very good at recognizing shapes

Neural Nets for Object Recognition

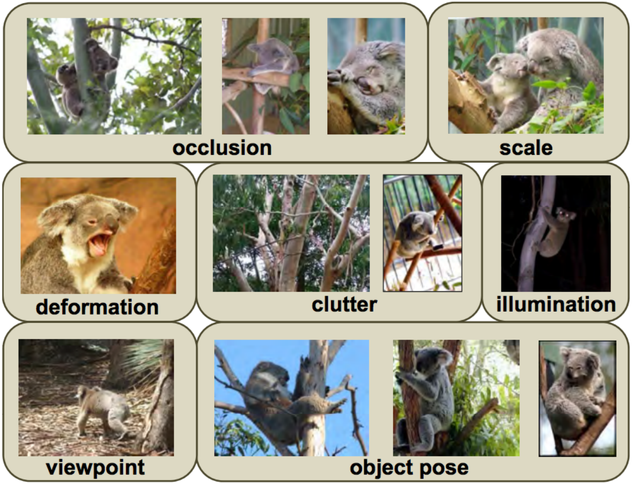
- People are very good at recognizing shapes
  - Intrinsically difficult, computers are bad at it

Neural Nets for Object Recognition

- People are very good at recognizing shapes
  - Intrinsically difficult, computers are bad at it
- Why is it difficult?

Why is it a Problem?

- Difficult scene conditions



[From: Grauman & Leibe]

Why is it a Problem?

- Huge within-class variations. Recognition is mainly about modeling variation.

[Pic from: S. Lazebnik]

Why is it a Problem?

- Tones of classes

[Biederman]

Neural Nets for Object Recognition

- People are very good at recognizing shapes
  - Intrinsically difficult, computers are bad at it
- Some reasons why it is difficult:

Neural Nets for Object Recognition

- People are very good at recognizing shapes
  - Intrinsically difficult, computers are bad at it
- Some reasons why it is difficult:
  - Segmentation: Real scenes are cluttered

Neural Nets for Object Recognition

- People are very good at recognizing shapes
  - Intrinsically difficult, computers are bad at it
- Some reasons why it is difficult:
  - Segmentation: Real scenes are cluttered
  - Invariances: We are very good at ignoring all sorts of variations that do not affect shape

Neural Nets for Object Recognition

- People are very good at recognizing shapes
  - Intrinsically difficult, computers are bad at it
- Some reasons why it is difficult:
  - Segmentation: Real scenes are cluttered
  - Invariances: We are very good at ignoring all sorts of variations that do not affect shape
  - Deformations: Natural shape classes allow variations (faces, letters, chairs)

Neural Nets for Object Recognition

- People are very good at recognizing shapes
  - Intrinsically difficult, computers are bad at it
- Some reasons why it is difficult:
  - Segmentation: Real scenes are cluttered
  - Invariances: We are very good at ignoring all sorts of variations that do not affect shape
  - Deformations: Natural shape classes allow variations (faces, letters, chairs)
  - A huge amount of computation is required

How to Deal with Large Input Spaces

- How can we apply neural nets to images?

How to Deal with Large Input Spaces

- How can we apply neural nets to images?
- Images can have millions of pixels, i.e., $\mathbf{x}$ is very high dimensional

How to Deal with Large Input Spaces

- How can we apply neural nets to images?
- Images can have millions of pixels, i.e., $\mathbf{x}$ is very high dimensional
- How many parameters do I have?

How to Deal with Large Input Spaces

- How can we apply neural nets to images?
- Images can have millions of pixels, i.e., $\mathbf{x}$ is very high dimensional
- How many parameters do I have?
- Prohibitive to have fully-connected layers

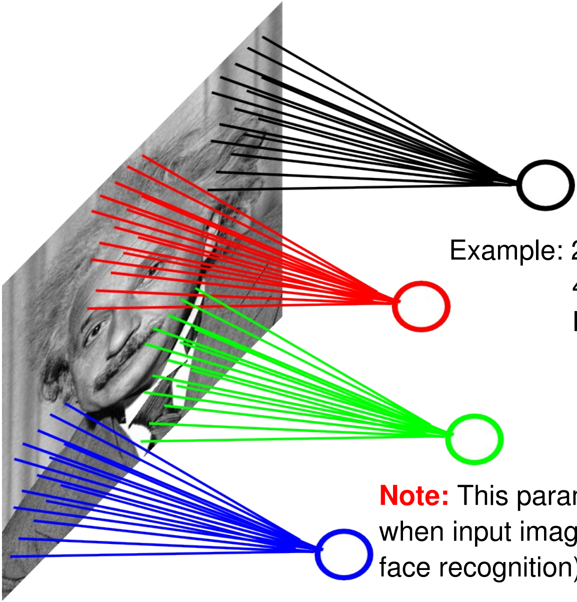
How to Deal with Large Input Spaces

- How can we apply neural nets to images?
- Images can have millions of pixels, i.e., $\mathbf{x}$ is very high dimensional
- How many parameters do I have?
- Prohibitive to have fully-connected layers
- What can we do?

How to Deal with Large Input Spaces

- How can we apply neural nets to images?
- Images can have millions of pixels, i.e., $\mathbf{x}$ is very high dimensional
- How many parameters do I have?
- Prohibitive to have fully-connected layers
- What can we do?
- We can use a locally connected layer

Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

34

Ranzato

When Will this Work?

When Will this Work?

When Will this Work?

When Will this Work?

- This is good when the input is (roughly) registered

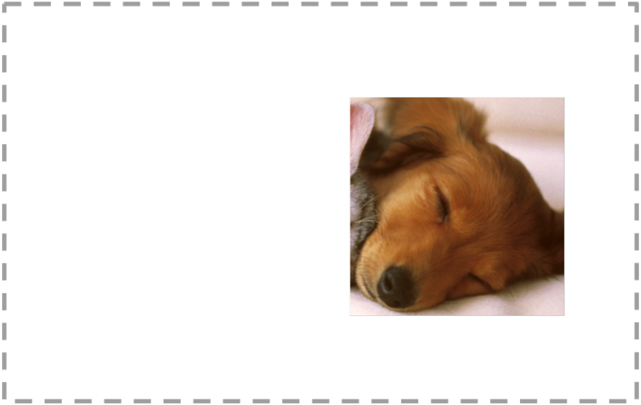
General Images

- The object can be anywhere

[Slide: Y. Zhu]

General Images

- The object can be anywhere



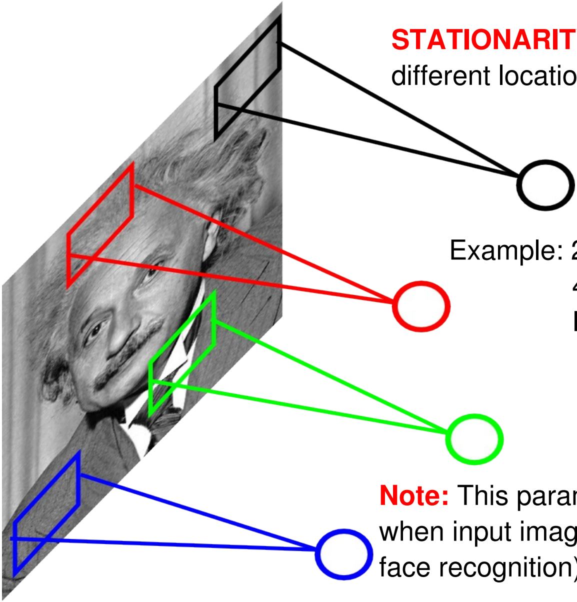
[Slide: Y. Zhu]

General Images

- The object can be anywhere

[Slide: Y. Zhu]

Locally Connected Layer



STATIONARITY? Statistics is similar at different locations

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

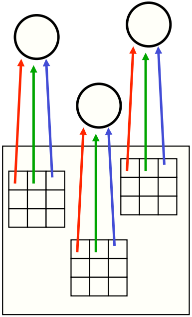
Note: This parameterization is good when input image is registered (e.g., face recognition).

35

Ranzato

The replicated feature approach

The red connections all have the same weight.

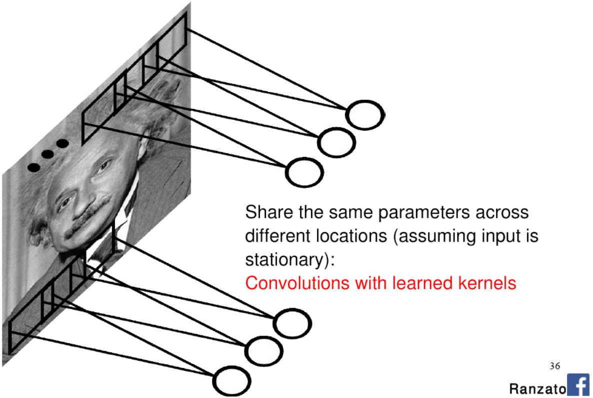


5

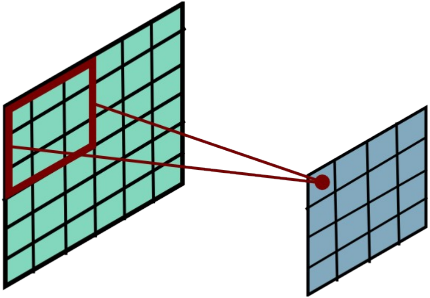
- Adopt approach apparently used in monkey visual systems
- Use many different copies of the same feature detector.
  - Copies have slightly different positions.
  - Could also replicate across scale and orientation.
    - Tricky and expensive
  - Replication reduces number of free parameters to be learned.
- Use several different feature types, each with its own replicated pool of detectors.
  - Allows each patch of image to be represented in several ways.

Convolutional Neural Net

- Idea: statistics are similar at different locations (Lecun 1998)
- Connect each hidden unit to a small input patch and share the weight across space
- This is called a convolution layer and the network is a convolutional network



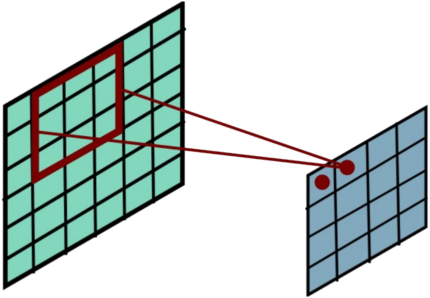
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

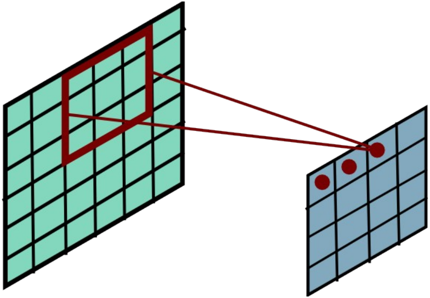
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

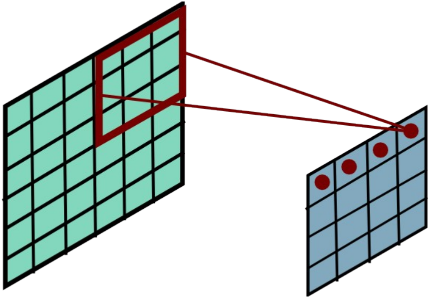
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

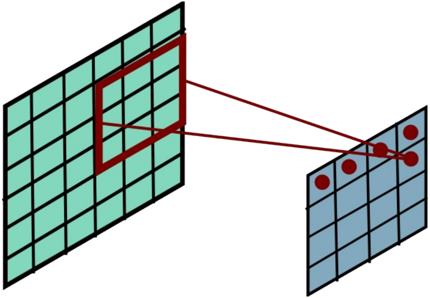
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

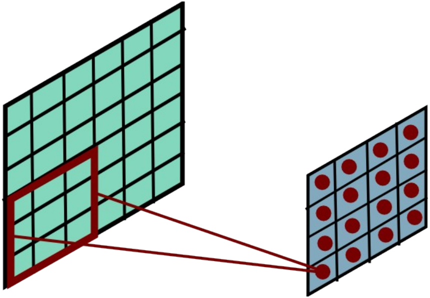
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

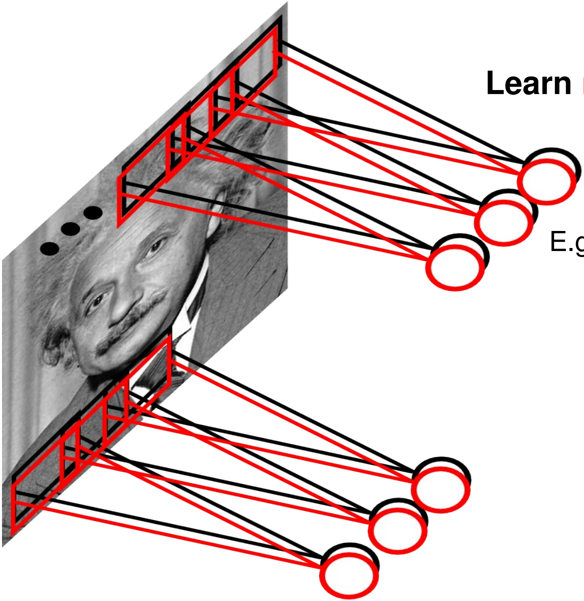
Convolutional Layer



Ranzato

$$h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} * w_{jk}^n)$$

Convolutional Layer



Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

54

Ranzato

Convolutional Layer

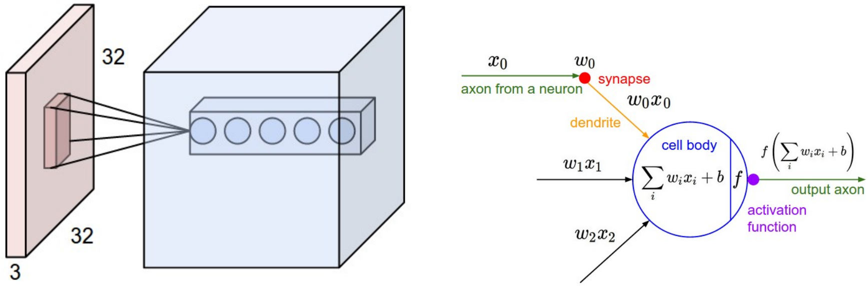


Figure : **Left:** CNN, **right:** Each neuron computes a linear and activation function

Convolutional Layer

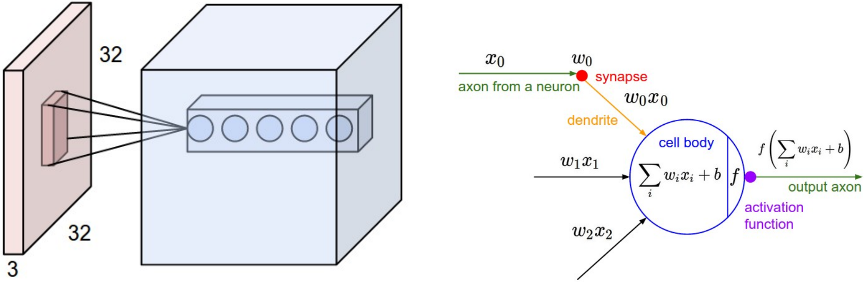


Figure : **Left:** CNN, **right:** Each neuron computes a linear and activation function

<u>Hyperparameters of a convolutional layer:</u>

Convolutional Layer

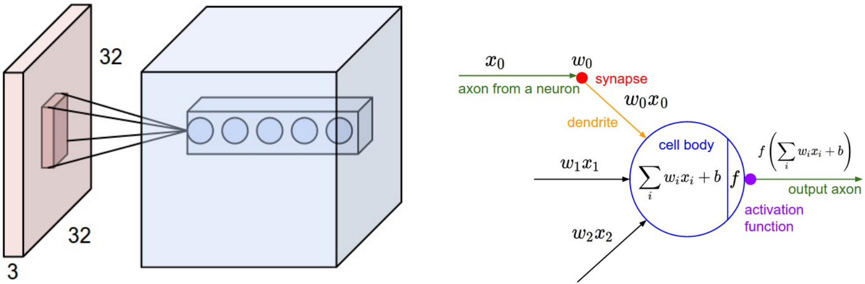


Figure : **Left:** CNN, **right:** Each neuron computes a linear and activation function

Hyperparameters of a convolutional layer:

- The number of filters (controls the **depth** of the output volume)

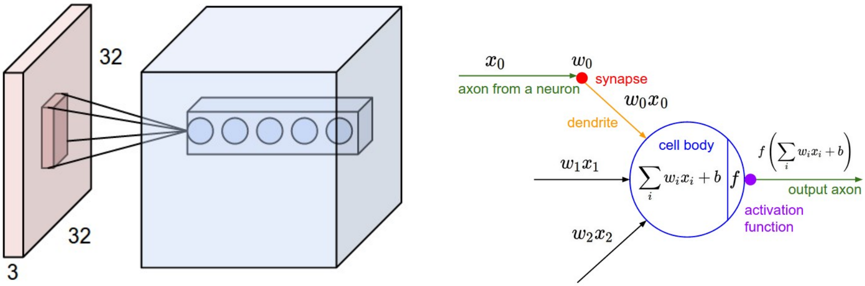


Figure : **Left:** CNN, **right:** Each neuron computes a linear and activation function

Hyperparameters of a convolutional layer:

- The number of filters (controls the **depth** of the output volume)
- The **stride**: how many units apart do we apply a filter spatially (this controls the spatial size of the output volume)

Convolutional Layer

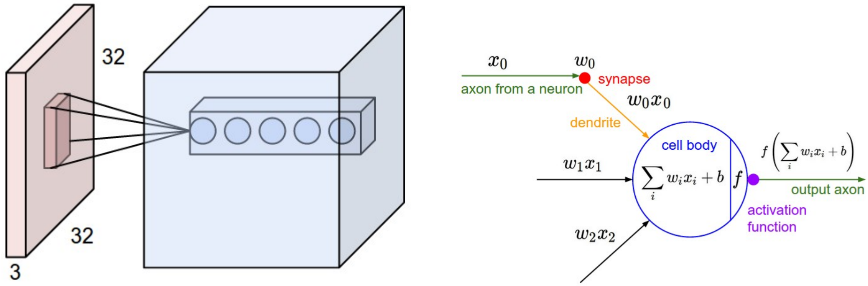
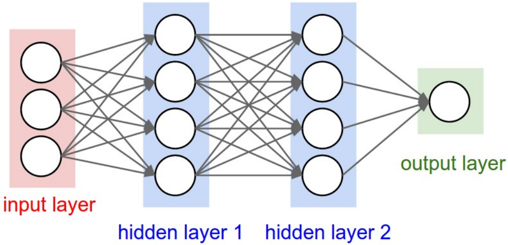


Figure : **Left:** CNN, **right:** Each neuron computes a linear and activation function

Hyperparameters of a convolutional layer:

- The number of filters (controls the **depth** of the output volume)
- The **stride**: how many units apart do we apply a filter spatially (this controls the spatial size of the output volume)
- The size $w \times h$ of the filters

MLP vs ConvNet



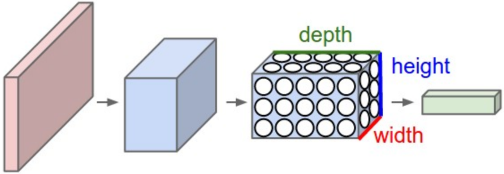
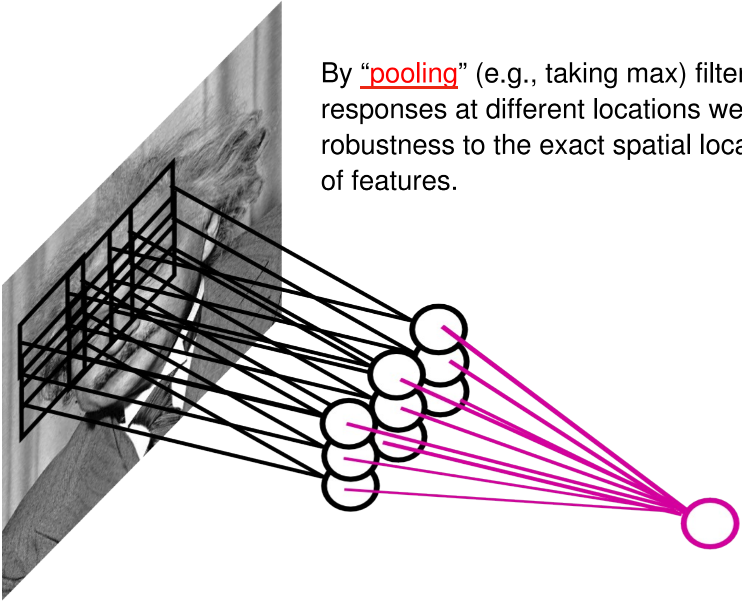


Figure : **Top:** MLP, **bottom:** Convolutional neural network

[http://cs231n.github.io/convolutional-networks/]

Pooling Layer



By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

61

Ranzato

Pooling Options

- Max Pooling: return the maximal argument
- Average Pooling: return the average of the arguments
- Other types of pooling exist.

Pooling

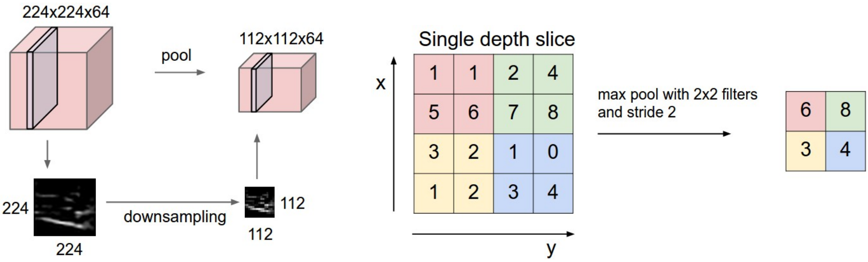


Figure : **Left:** Pooling, **right:** max pooling example

Pooling

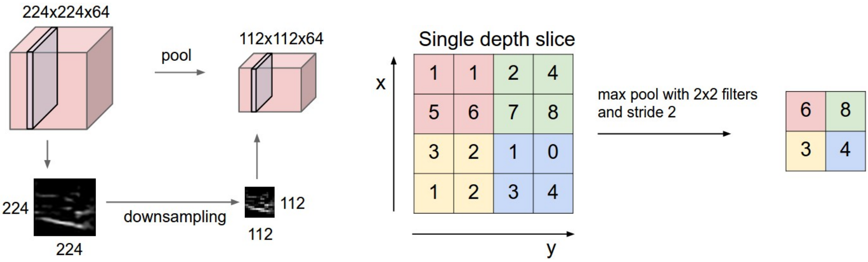


Figure : **Left:** Pooling, **right:** max pooling example

Hyperparameters of a pooling layer:

Pooling

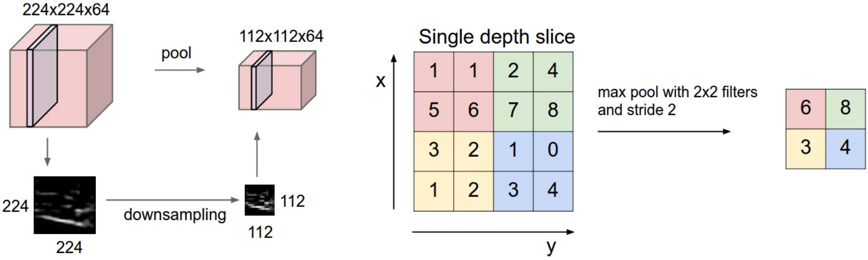


Figure : **Left:** Pooling, **right:** max pooling example

Hyperparameters of a pooling layer:

- The spatial extent F

Pooling

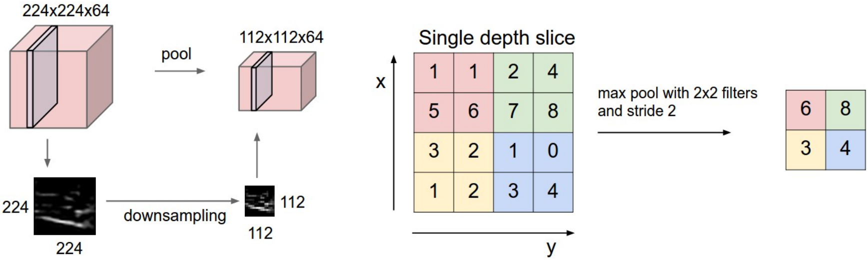


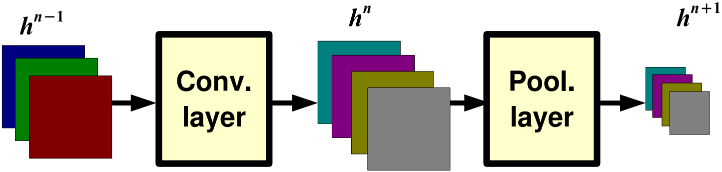
Figure : **Left:** Pooling, **right:** max pooling example

Hyperparameters of a pooling layer:

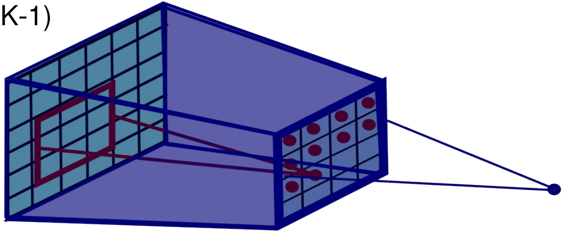
- The spatial extent F
- The stride

[http://cs231n.github.io/convolutional-networks/]

Pooling Layer: Receptive Field Size



If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



67

Ranzato

Backpropagation with Weight Constraints

- It is easy to modify the backpropagation algorithm to incorporate linear constraints between the weights

$$\begin{aligned} \text{To constrain:} \quad w_1 &= w_2 \\ \text{we need:} \quad \Delta w_1 &= \Delta w_2 \end{aligned}$$

Backpropagation with Weight Constraints

- It is easy to modify the backpropagation algorithm to incorporate linear constraints between the weights

$$\begin{aligned} \text{To constrain:} \quad w_1 &= w_2 \\ \text{we need:} \quad \Delta w_1 &= \Delta w_2 \end{aligned}$$

- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.

$$\begin{aligned} \text{compute:} \quad & \frac{\partial E}{\partial w_1} \text{ and } \frac{\partial E}{\partial w_2} \\ \text{use:} \quad & \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \text{ for } w_1 \text{ and } w_2 \end{aligned}$$

Backpropagation with Weight Constraints

- It is easy to modify the backpropagation algorithm to incorporate linear constraints between the weights

$$\begin{aligned} \text{To constrain:} \quad w_1 &= w_2 \\ \text{we need:} \quad \Delta w_1 &= \Delta w_2 \end{aligned}$$

- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.

$$\begin{aligned} \text{compute:} \quad & \frac{\partial E}{\partial w_1} \text{ and } \frac{\partial E}{\partial w_2} \\ \text{use:} \quad & \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \text{ for } w_1 \text{ and } w_2 \end{aligned}$$

- So if the weights started off satisfying the constraints, they will continue to satisfy them.

Backpropagation with Weight Constraints

- It is easy to modify the backpropagation algorithm to incorporate linear constraints between the weights

$$\begin{aligned} \text{To constrain:} \quad w_1 &= w_2 \\ \text{we need:} \quad \Delta w_1 &= \Delta w_2 \end{aligned}$$

- We compute the gradients as usual, and then modify the gradients so that they satisfy the constraints.

$$\begin{aligned} \text{compute:} \quad & \frac{\partial E}{\partial w_1} \text{ and } \frac{\partial E}{\partial w_2} \\ \text{use:} \quad & \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2} \text{ for } w_1 \text{ and } w_2 \end{aligned}$$

- So if the weights started off satisfying the constraints, they will continue to satisfy them.
- This is an intuition behind the backprop. In practice, write down the equations and compute derivatives (it's a nice exercise, do it at home)

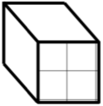
Now let's make this very deep to get a real state-of-the-art object recognition system

Convolutional Neural Networks (CNN)

- Remember from your image processing / computer vision course about filtering?

Input "image"

Filter

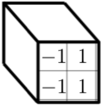


Convolutional Neural Networks (CNN)

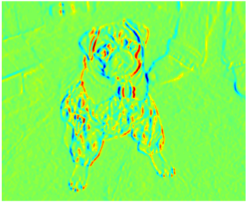
- If our filter is $[-1, 1]$, you get a vertical edge detector

Input “image”

Filter

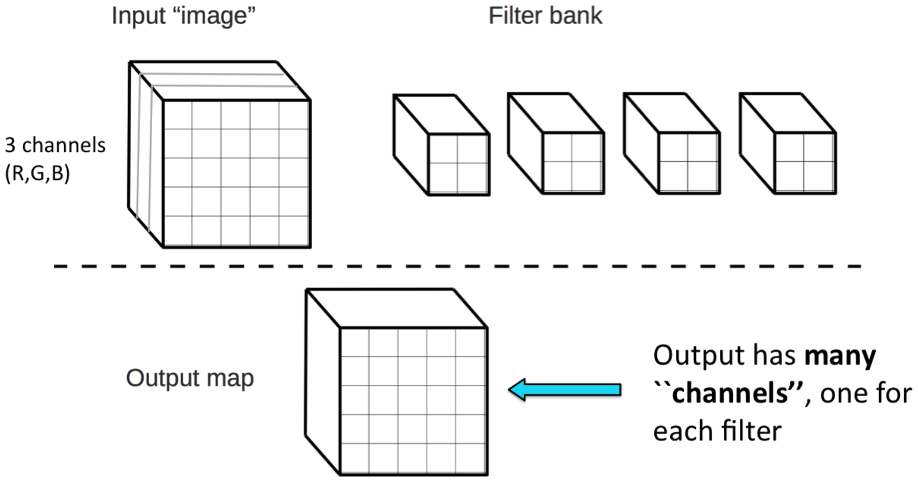


Output map



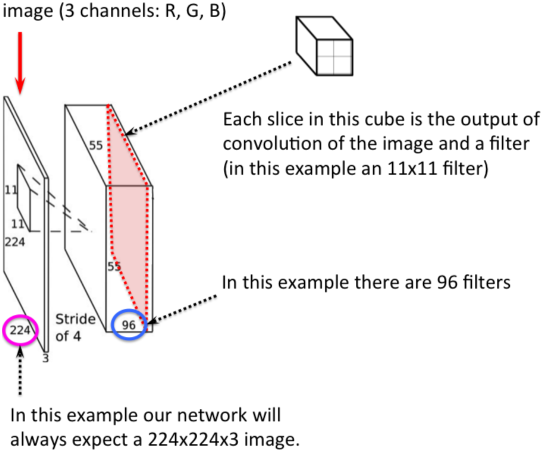
Convolutional Neural Networks (CNN)

- Now imagine we want to have many filters (e.g., vertical, horizontal, corners, one for dots). We will use a filterbank.



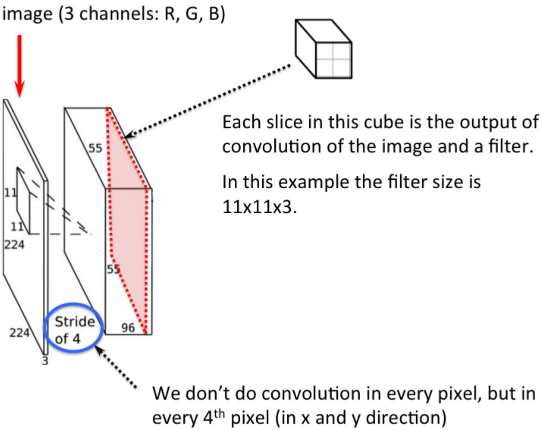
Convolutional Neural Networks (CNN)

- So applying a filterbank to an image yields a cube-like output, a 3D matrix in which each slice is an output of convolution with one filter. We apply an activation function on each hidden unit (typically a ReLU).



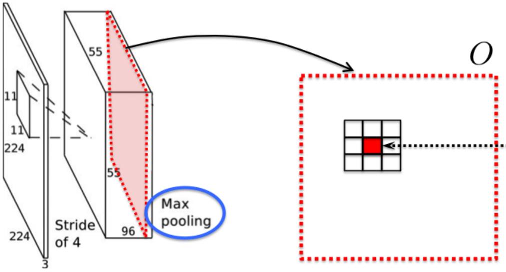
Convolutional Neural Networks (CNN)

- So applying a filterbank to an image yields a cube-like output, a 3D matrix in which each slice is an output of convolution with one filter. We apply an activation function on each hidden unit (typically a ReLU).



Convolutional Neural Networks (CNN)

- Do some additional tricks. A popular one is called max pooling. Any idea why you would do this?

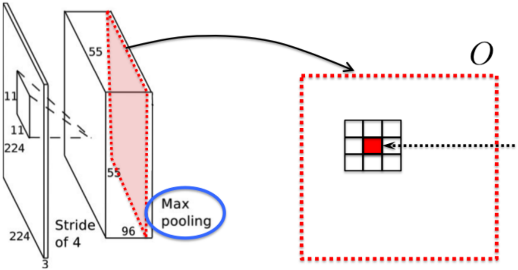


$$O(i,j) = \max_{\substack{k \in \{i-1, i, i+1\} \\ l \in \{j-1, j, j+1\}}} O(k,l)$$

Take each slice in the output cube, and in each pixel compute a max over a small patch around it. This is called **max pooling**.

Convolutional Neural Networks (CNN)

- Do some additional tricks. A popular one is called max pooling. Any idea why you would do this? To get invariance to small shifts in position.

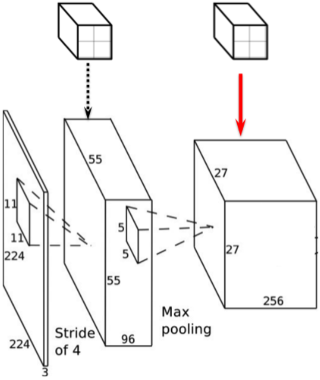


$$O(i,j) = \max_{\substack{k \in \{i-1, i, i+1\} \\ l \in \{j-1, j, j+1\}}} O(k,l)$$

Take each slice in the output cube, and in each pixel compute a max over a small patch around it. This is called **max pooling**.

Convolutional Neural Networks (CNN)

- Now add another "layer" of filters. For each filter again do convolution, but this time with the output cube of the previous layer.



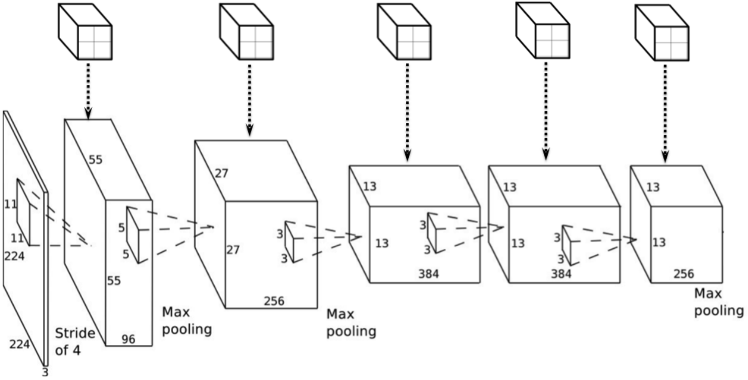
Add one more layer of filters

These filters are convolved with the output of the previous layer. The results of each convolution is again a slice in the cube on the right.

What is the dimension of each of these filters?

Convolutional Neural Networks (CNN)

- Keep adding a few layers. Any idea what's the purpose of more layers? Why can't we just have a full bunch of filters in one layer?

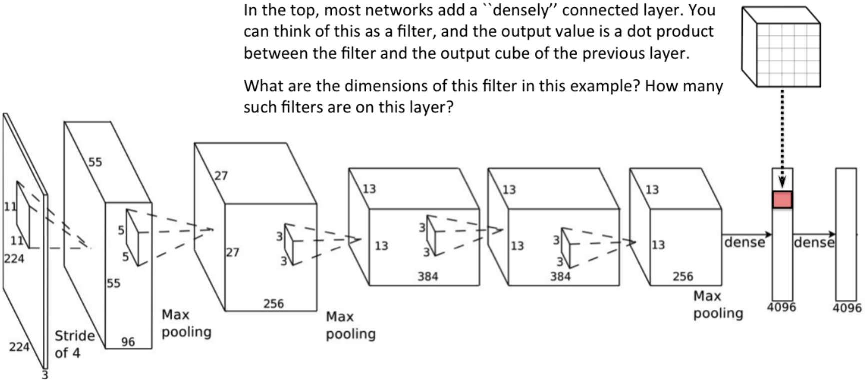


Do it recursively

Have multiple ``layers''

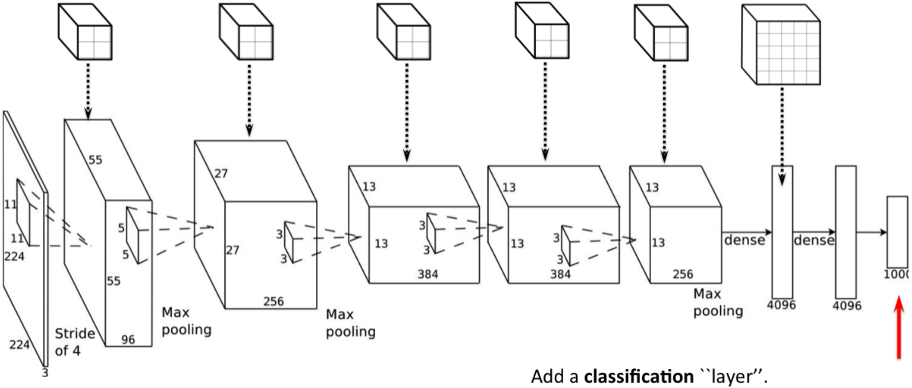
Convolutional Neural Networks (CNN)

- In the end add one or two fully (or densely) connected layers. In this layer, we don't do convolution we just do a dot-product between the "filter" and the output of the previous layer.



Convolutional Neural Networks (CNN)

- Add one final layer: a classification layer. Each dimension of this vector tells us the probability of the input image being of a certain class.

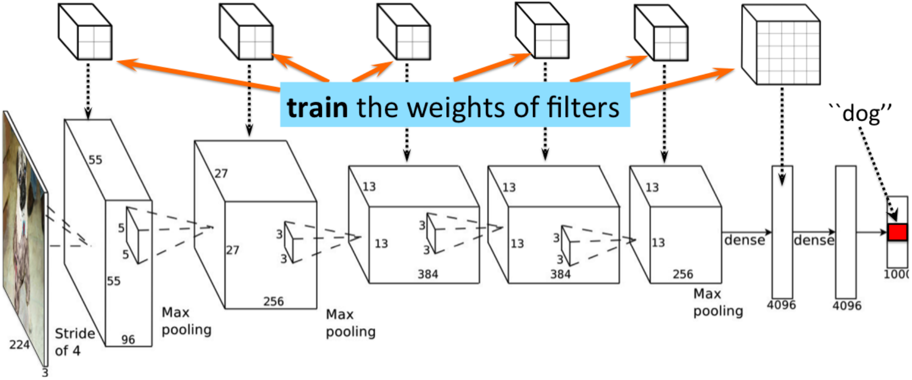


Add a **classification** ``layer''.

For an input image, the value in a particular dimension of this vector tells you the probability of the corresponding object class.

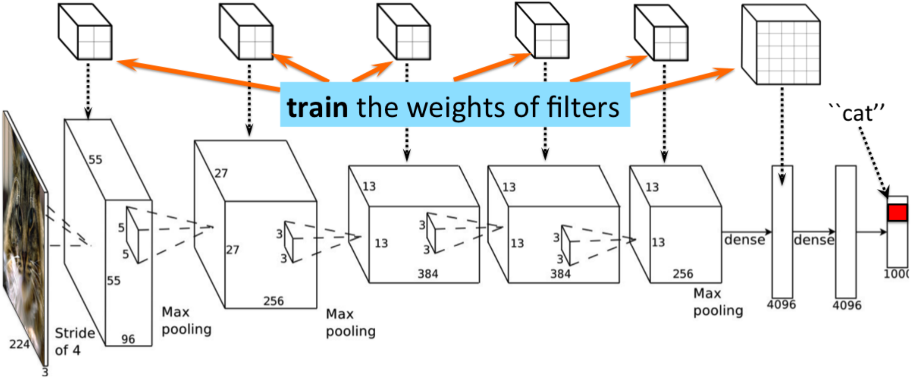
Convolutional Neural Networks (CNN)

- The trick is to not hand-fix the weights, but to train them. Train them such that when the network sees a picture of a dog, the last layer will say "dog".



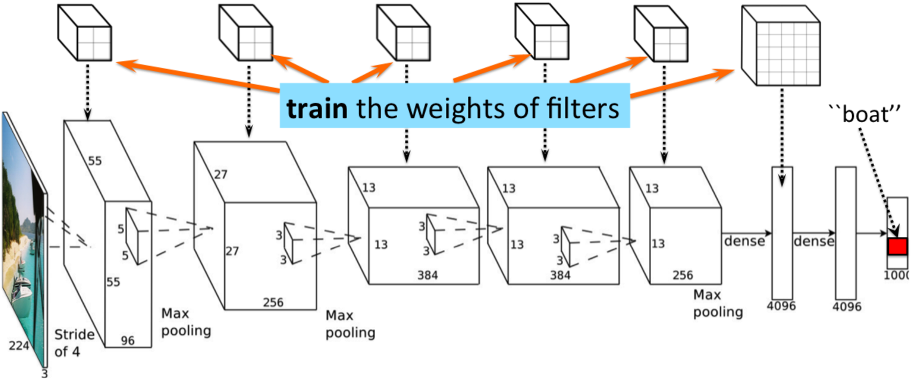
Convolutional Neural Networks (CNN)

- Or when the network sees a picture of a cat, the last layer will say "cat".



Convolutional Neural Networks (CNN)

- Or when the network sees a picture of a boat, the last layer will say "boat"... The more pictures the network sees, the better.



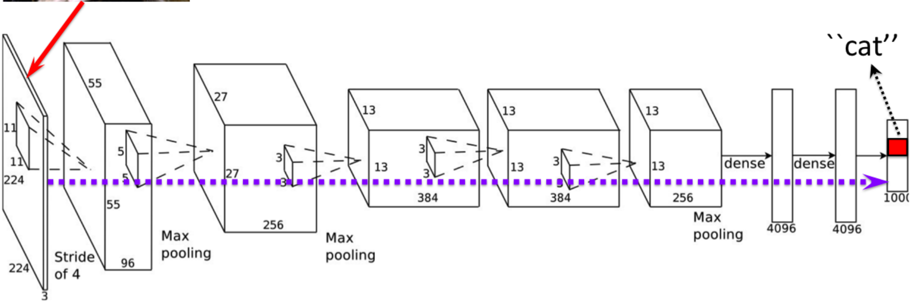
Train on **lots** of examples. Millions. Tens of millions. Wait a week for training to finish.

Share your network (the weights) with others who are not fortunate enough with GPU power.

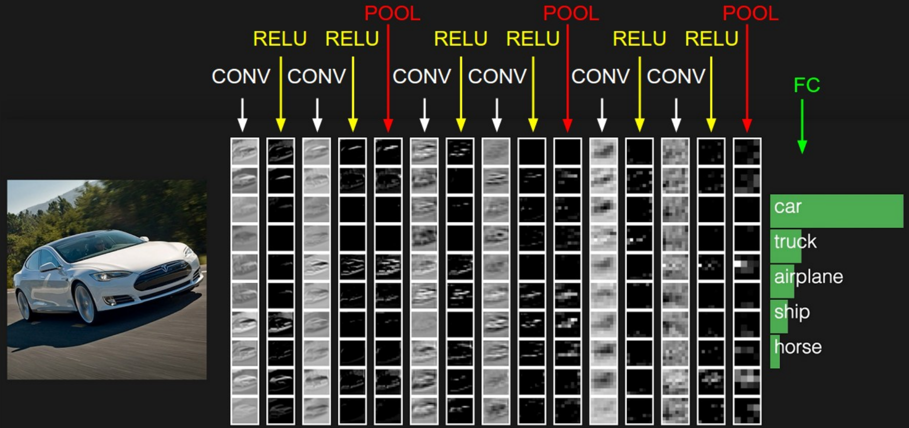
Classification

- Once trained we feed in an image or a crop, run through the network, and read out the class with the highest probability in the last (classif) layer.

What's the class of this object?

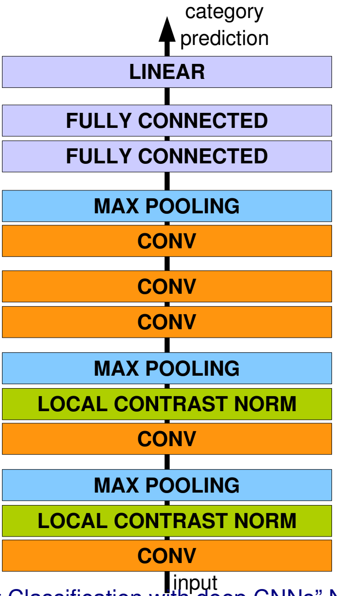


Example



[http://cs231n.github.io/convolutional-networks/]

Architecture for Classification

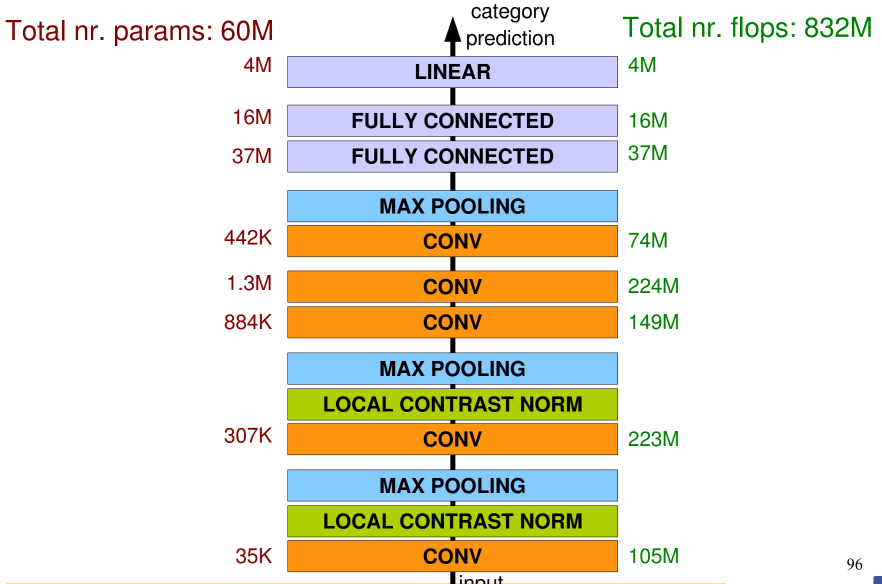


95

Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

Ranzato

Architecture for Classification



Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

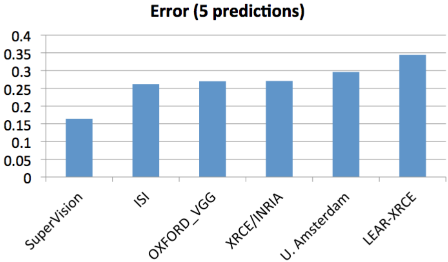
96

Ranzato

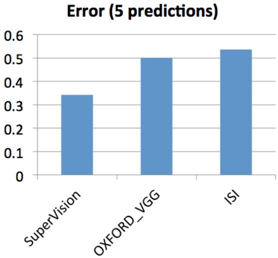
ImageNet

- Imagenet, biggest dataset for object classification: `http://image-net.org/`
- 1000 classes, 1.2M training images, 150K for test

The 2012 Computer Vision Crisis



(Classification)



(Detection)

So Neural Networks are Great

- So networks turn out to be great.
- Everything is deep, even if it's shallow!
- Companies leading the competitions as they have more computational power
- At this point Google, Facebook, Microsoft, Baidu "steal" most neural network professors/students from academia

So Neural Networks are Great

- But to train the networks you need quite a bit of computational power (e.g., GPU farm). So what do you do?

So Neural Networks are Great

- Buy even more.

So Neural Networks are Great

- And train more layers. 16 instead of 7 before. 144 million parameters.

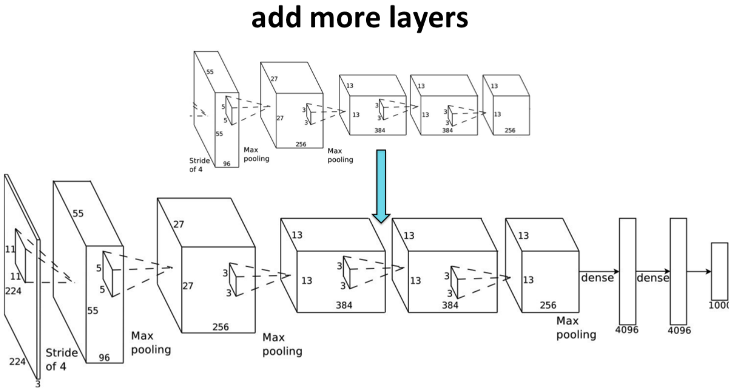
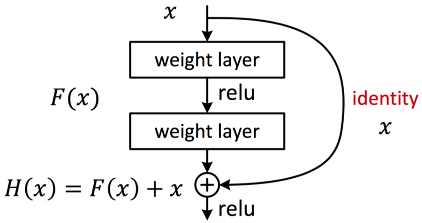
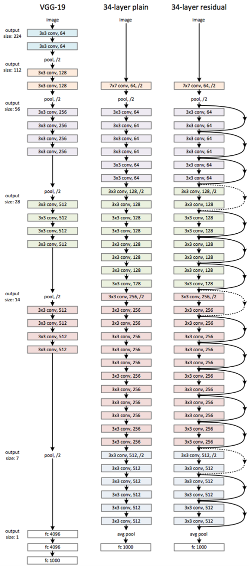


Figure : K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 2014

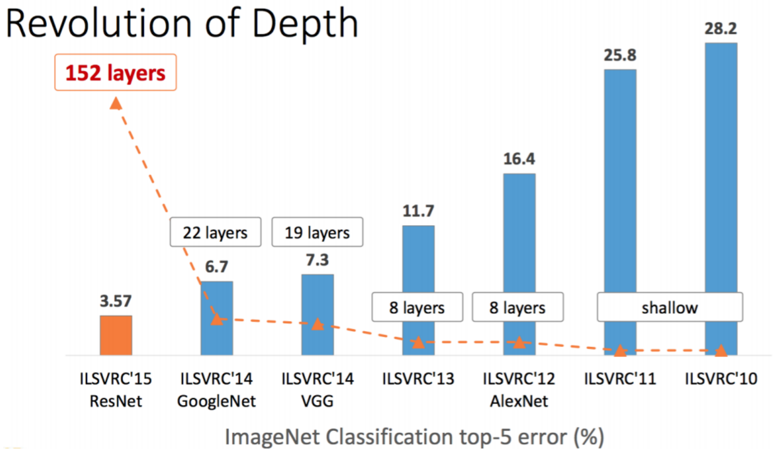
- Networks are now at 150 layers
- They use a skip connections with special form
- In fact, they don't fit on this screen
- Amazing performance!
- A lot of "mistakes" are due to wrong ground-truth





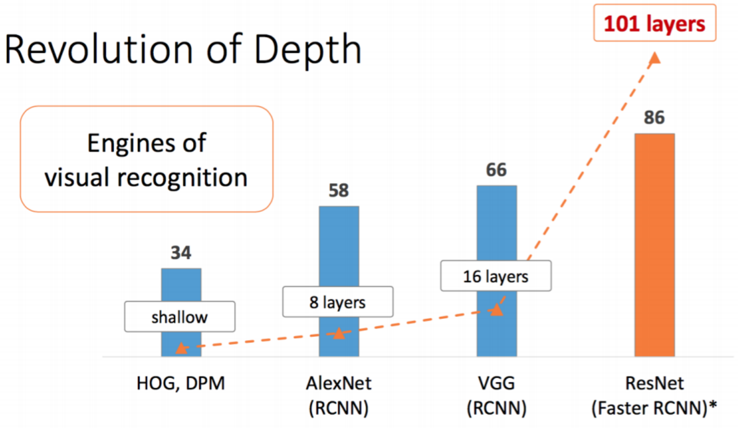
[He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

Results: Object Classification



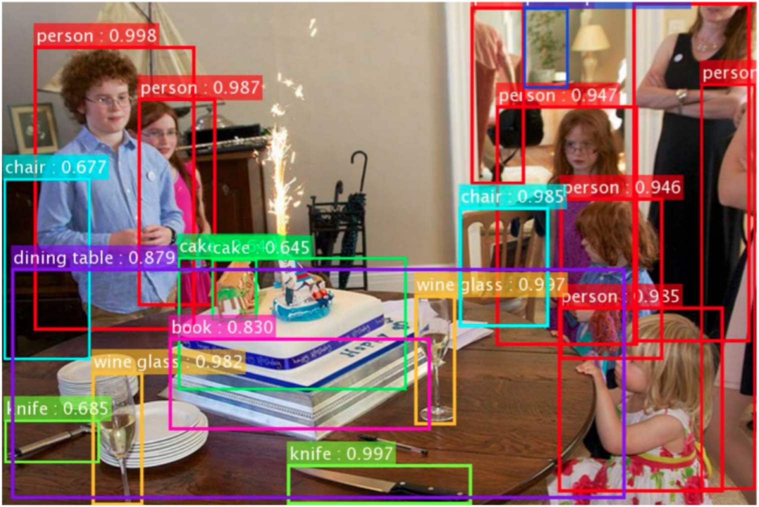
Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

Results: Object Detection



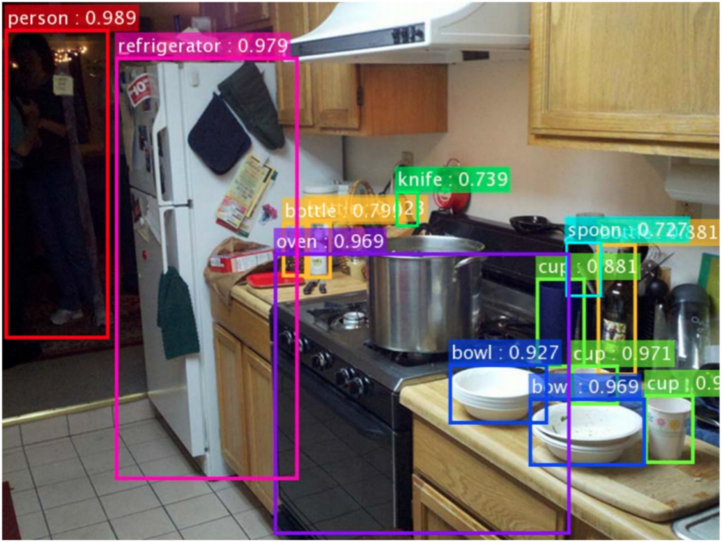
Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

Results: Object Detection



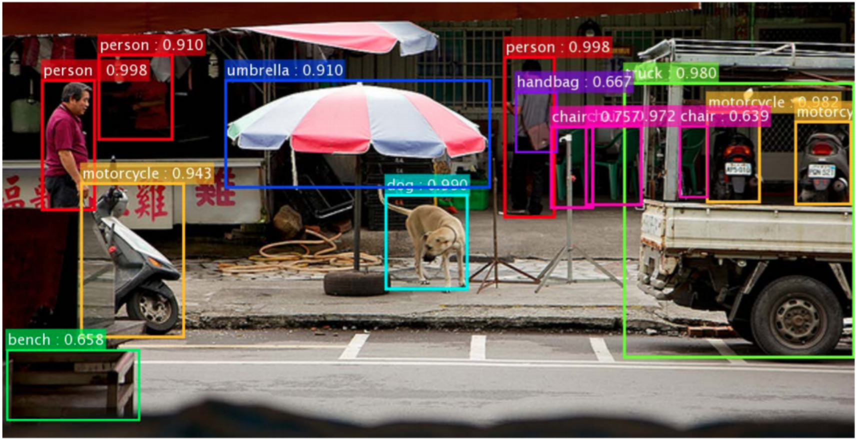
Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

Results: Object Detection



Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition

Results: Object Detection



Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

What do CNNs Learn?

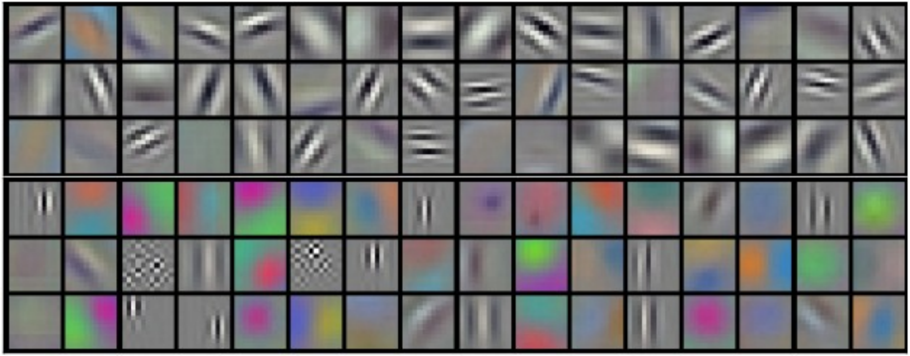


Figure : Filters in the first convolutional layer of Krizhevsky et al

What do CNNs Learn?

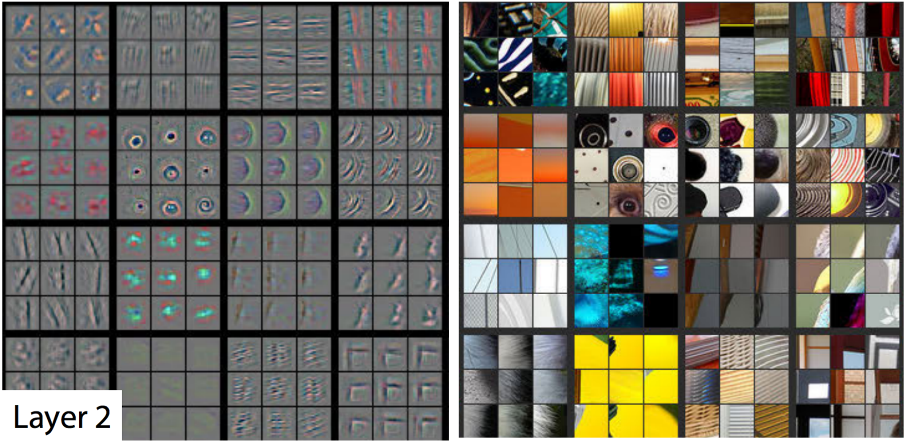


Figure : Filters in the second layer

[http://arxiv.org/pdf/1311.2901v3.pdf]

What do CNNs Learn?

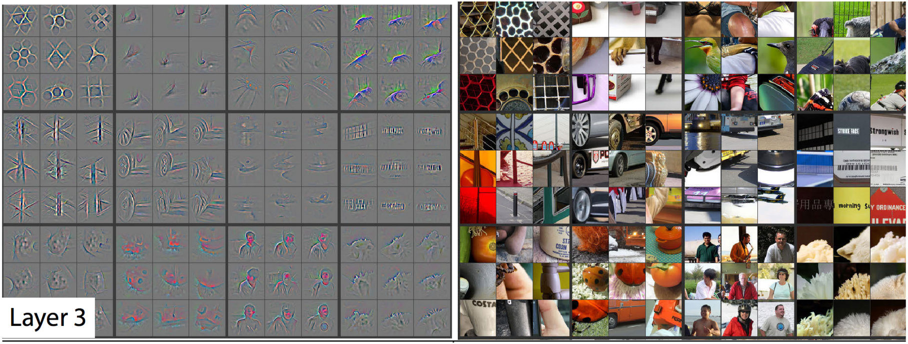
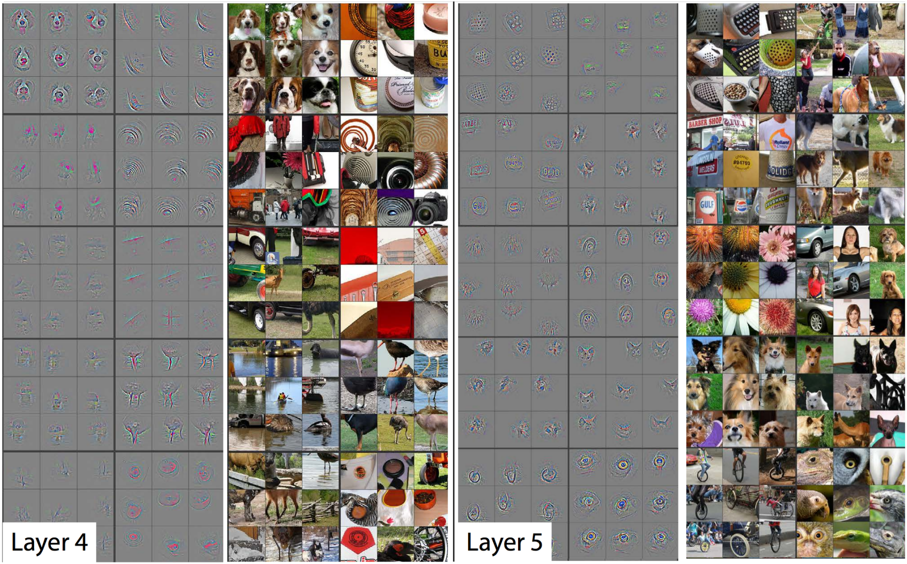


Figure : Filters in the third layer

[http://arxiv.org/pdf/1311.2901v3.pdf]

What do CNNs Learn?



[http://arxiv.org/pdf/1311.2901v3.pdf]

How to Train Good CNNs

- Normalize your data (standard trick: subtract mean, divide by standard deviation)

How to Train Good CNNs

- Normalize your data (standard trick: subtract mean, divide by standard deviation)
- Augment your data (add image flips, rotations, etc)

How to Train Good CNNs

- Normalize your data (standard trick: subtract mean, divide by standard deviation)
- Augment your data (add image flips, rotations, etc)
- Keep training data balanced

How to Train Good CNNs

- Normalize your data (standard trick: subtract mean, divide by standard deviation)
- Augment your data (add image flips, rotations, etc)
- Keep training data balanced
- Shuffle data before batching

How to Train Good CNNs

- Normalize your data (standard trick: subtract mean, divide by standard deviation)
- Augment your data (add image flips, rotations, etc)
- Keep training data balanced
- Shuffle data before batching
- In training: Random initialization of weights with proper variance

How to Train Good CNNs

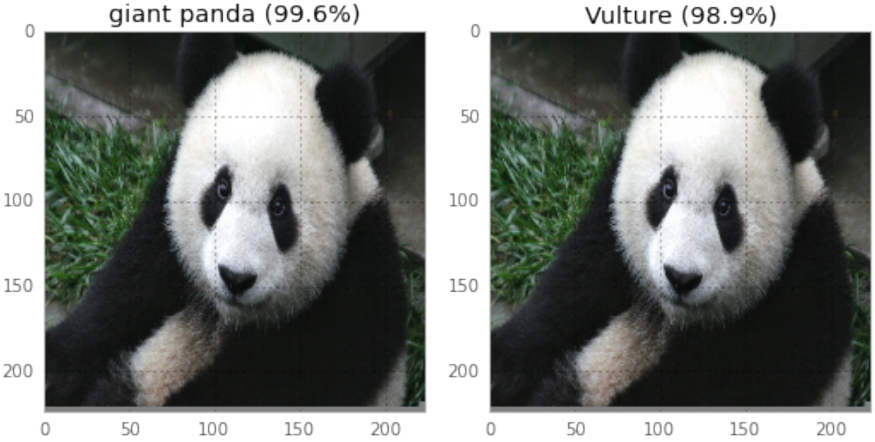
- Normalize your data (standard trick: subtract mean, divide by standard deviation)
- Augment your data (add image flips, rotations, etc)
- Keep training data balanced
- Shuffle data before batching
- In training: Random initialization of weights with proper variance
- Monitor your loss function, and accuracy (performance) on validation

How to Train Good CNNs

- Normalize your data (standard trick: subtract mean, divide by standard deviation)
- Augment your data (add image flips, rotations, etc)
- Keep training data balanced
- Shuffle data before batching
- In training: Random initialization of weights with proper variance
- Monitor your loss function, and accuracy (performance) on validation
- If your labeled image dataset is small: pre-train your CNN on a large dataset (eg Imagenet), and fine-tune on your dataset

[Slide: Y. Zhu, check tutorial slides and code: http://www.cs.utoronto.ca/~fidler/teaching/2015/CSC2523.html]

Tricking a Neural Net



Read about it here (and try it!): https://codewords.recurse.com/issues/five/why-do-neural-networks-think-a-panda-is-a-vulture

Watch: https://www.youtube.com/watch?v=M2IebCN9Ht4

More on NNs

Figure : Generate images: http://arxiv.org/pdf/1511.06434v1.pdf

More on NNs

Generate text: https://vimeo.com/146492001, https://github.com/karpathy/neuraltalk2, https://github.com/ryankiros/visual-semantic-embedding

Figure : Compose music: https://www.youtube.com/watch?v=0VTI1BBLydE

Links

- NNs for computer vision:
  https://github.com/kjw0612/awesome-deep-vision
- Recurrent neural networks: https://github.com/kjw0612/awesome-rnn
- Lots of code, models, tutorials:
  https://github.com/carpedm20/awesome-torch
- More links on our class webpage