CPSC 340:
Machine Learning and Data Mining

Convolutional Neural Networks

Fall 2022

Admin

- Recording of Wednesday's/today's lecture is online.
- I expect to reach end of "testable content" today.
- Next week, I plan to cover different topics in different sections.
  - AM sections: autoencoders, fully-convolutional networks, "what do we learn?" (and maybe Dalle 2).
  - PM sections: autoencoders, recurrent neural networks, transformers (and maybe ChatGPT).
- Assignment 6 due Wednesday.
  - 1 late day for Friday, 2 for Monday.
- On Friday, I will have office hours during lecture times.
  - To help people prepare for the final if you are panicking.
  - 12-1 in ICICS 104.
  - 4-x in ICICS 246.
- Final exam is Sunday December 11th at 8:30am (SRC A).
  - Two doubled-sided pages for cheat sheet.
  - Exams of previous years will posted soon.
  - Similar format to midterm, but probably more calculation and long answer.
    - Questions related to non-bonus lecture slides, and topics from assignments.
- Project will be due at least one week after the final.
  - Deadline and submission details coming soon.

Last Time: Automatic Differentiation (AD)

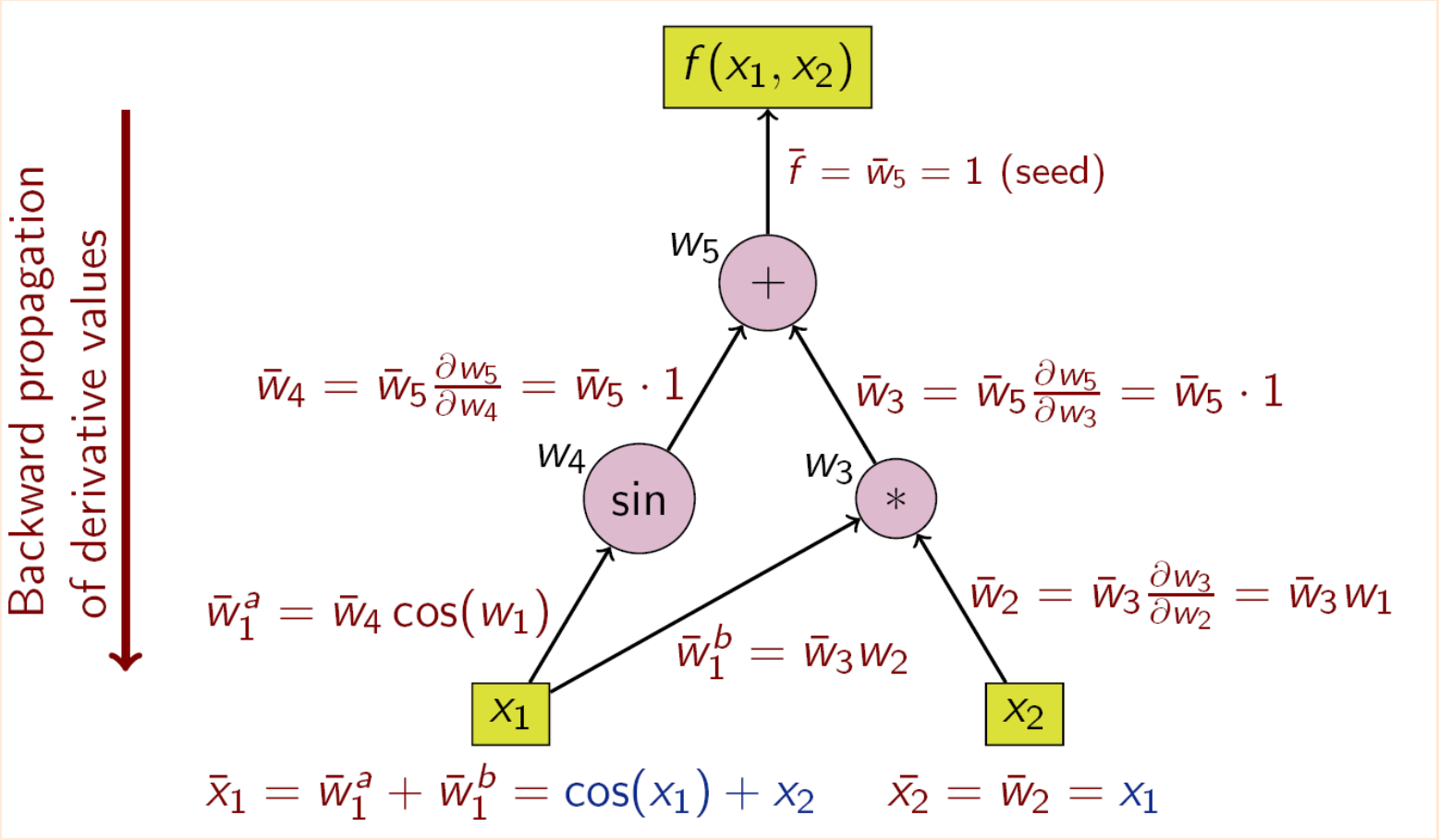
- Automatic differentiation (AD):
  - Takes code that computes a function.
  - Produces code that computes derivatives.
    - No approximation error but no formula (computed algorithmically).
- AD writes functions as a sequence of simple compositions:
  - $f_5(f_4(f_3(f_2(f_1(x)))))$
- AD writes derivatives using chain rule:
  - $f'(x) = f_5'(f_4(f_3(f_2(f_1(x)))))*f_4'(f_3(f_2(f_1(x))))*f_3'(f_2(f_1(x)))*f_2'(f_1(x))f_1'(x)$.
- We showed "forward pass" and "backward pass" for single-variable AD:
  - $\alpha_1 = f_1(x)$, $\alpha_2 = f_2(\alpha_1)$, $\alpha_3 = f_3(\alpha_2)$, $\alpha_4 = f_4(\alpha_3)$, $\alpha_5 = f_5(\alpha_4) = f(x)$.
  - $\beta_5 = 1*f_5'(\alpha_4)$, $\beta_4 = \beta_5*f_4'(\alpha_3)$, $\beta_3 = \beta_4*f_3'(\alpha_2)$, $\beta_2 = \beta_3*f_2'(\alpha_1)$, $\beta_1 = \beta_2*f_1'(x) = f'(x)$.
- Cost of computing f'(x) is same cost as of computing f(x).

Automatic Differentiation – Multiple Parameters

- In ML problems, we often have more than 1 parameter.
  - And we want to compute the gradient for the same cost as the function.
- To generalize AD to this case, we define a computation graph:
  - A directed acyclic graph (DAG).
  - Root nodes are the parameters (and inputs).
  - Intermediate nodes are computed values (α values).
  - Leaf node is the function value.
- Computing the gradient with AD:
  - The forward pass evaluates the function and stores intermediate values.
    - Going from the roots through the intermediate nodes to the leaf.
  - The backward pass applies the f_i' functions to the α values.
    - Accumulating the needed pieces of the chain rule until each root has its partial derivative.

Automatic Differentiation – Multiple Parameters

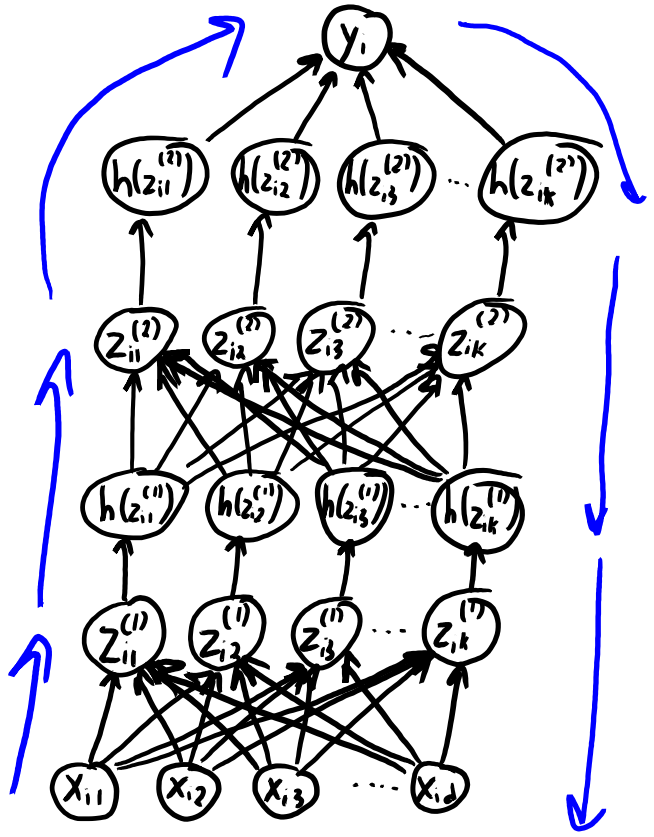
- Wikipedia's example of a computation graph:
  - For computing the gradient of $f(x_1,x_2) = \sin(x_1) + x_1x_2$.
  - Using 'w' for α.
  - Using '$\bar{w}$' for β.



https://en.wikipedia.org/wiki/Automatic_differentiation

Automatic Differentiation – Neural Networks

- Computing gradient for neural networks is a special case of automatic differentiation:
  - Forward AD pass is called forward propagation:
    - Starts from x_i and processes layers to reach $\hat{y}_i$.
      - Storing intermediate calculations.
  - Backward AD pass is called backpropagation:
    - Computes gradient of last layers and works backwards.
      - Using intermediate calculations stored during forward pass.



- Do you need to know how to do this?
  - Exact details are probably not vital (exist many implementations).
  - But understanding basic idea helps you know what can go wrong.
    - Or give hints about what to do when you run out of memory.
  - See discussion here by a neural network expert.

- Backward pass has same cost as forward pass.
  - So cost of computing gradient is same as cost of making predictions.

Automatic Differentiation - Discussion

- AD is amazing – get gradient for the same cost as the function.
  - You can try out lots of stuff, and enjoy thoroughly overfitting validation set!
  - Modern AD codes have lots of features, like built-in derivatives of matrix operations.

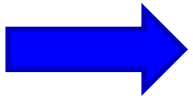
- But reverse-mode AD has some drawbacks:
  - Need to store all intermediate calculations, so requires a lot of storage.
    - For basic deep neural networks, hand-written code would only need to store the activations.
      - Modern codes have some of these space savings built in.
    - For other functions, the storage cost of AD is much higher than handwritten derivative code.
      - "Checkpointing" exists to reduce storage, but increases computational cost.
  - Has the same cost as computing the function, which is a pro and a con.
    - For basic deep neural networks, these have the same cost so this is what we want.
    - For other functions, the gradient can be computed at a lower cost than the function value.
  - May miss opportunities for parallelism, or miss tricks to avoid numerically problems.

- AD only makes sense at points where the function is differentiable.
  - TensorFlow and PyTorch can give incorrect "subderivatives" at non-differentiable ReLU points.
  - AD cannot do things like "take the derivative of a function of a sample from the distribution".

Next Topic: Convolutional Neural Networks

Motivation: X-Ray Abnormality Detection

- Want to build a system that recognizes abnormalities in x-rays:

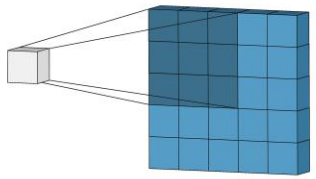


"Abnormality detected"
(binary classification)

- Applications:
  - Fast detection of tuberculosis, pneumonia, lung cancer, and so on.
- Deep learning has led to incredible progress on computer vision tasks.
  - Much of this progress has been driven by convolutional neural networks (CNNs).

Convolutional Neural Network (CNN) Motivation

- Consider training neural networks on 500 pixel by 500 pixel images.
  - So the number of inputs 'd' to first layer is 250,000 inputs.
- If first layer has k=10,000, then 'W' has 2.5 billion parameters.
  - We want to avoid this huge number (due to storage and overfitting).
- CNNs drastically reduce the number of parameters.
  - Main way they do this is using layers that look like convolutions:

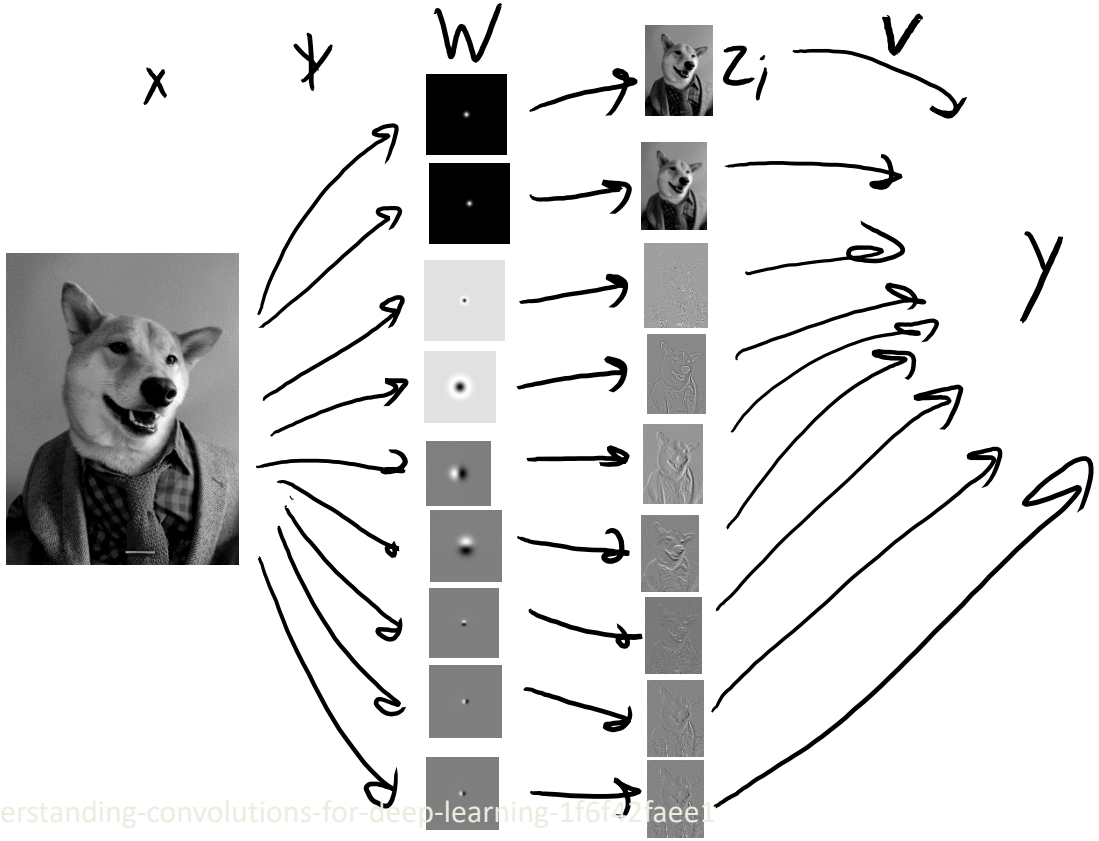


  - Hidden units only depend on a small number of inputs.
  - Using the same parameters on the connections of many activations.

https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Motivation for Convolutional Neural Networks

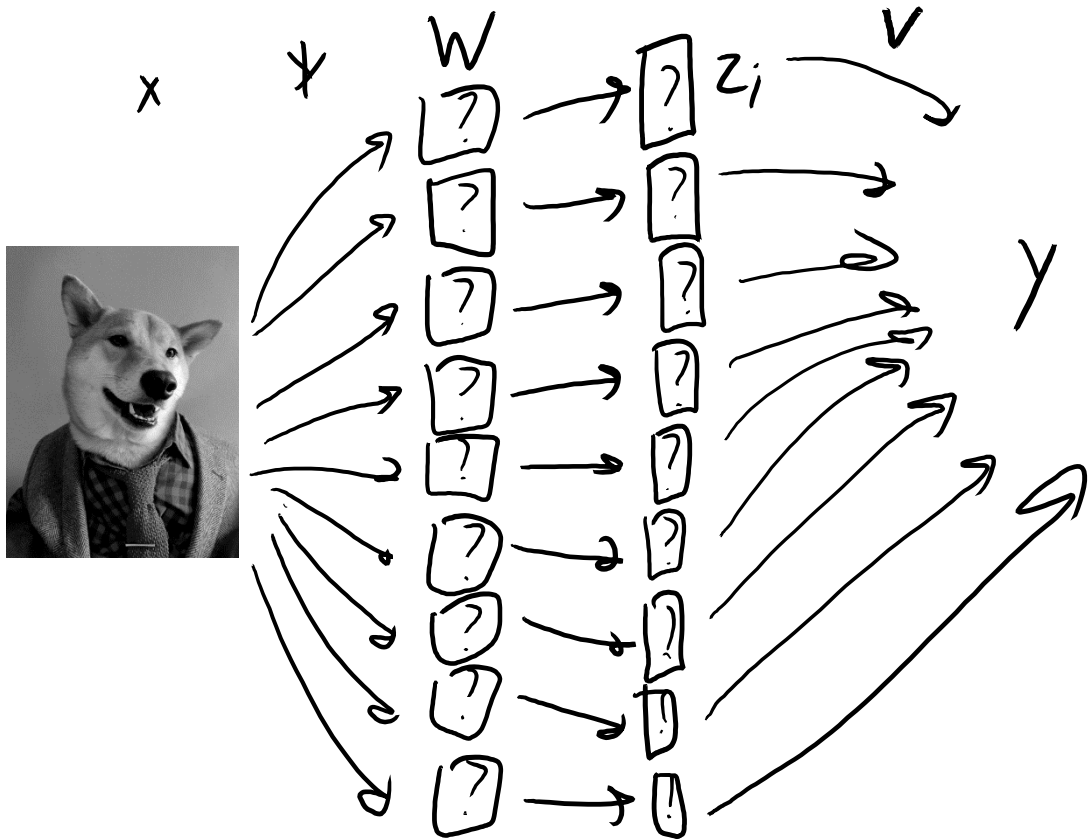
- Classic vision methods uses fixed convolutions as features:
  - Usually have different types/variances/orientations.
  - Can do subsampling or take maxes across locations/orientations/scales.



https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

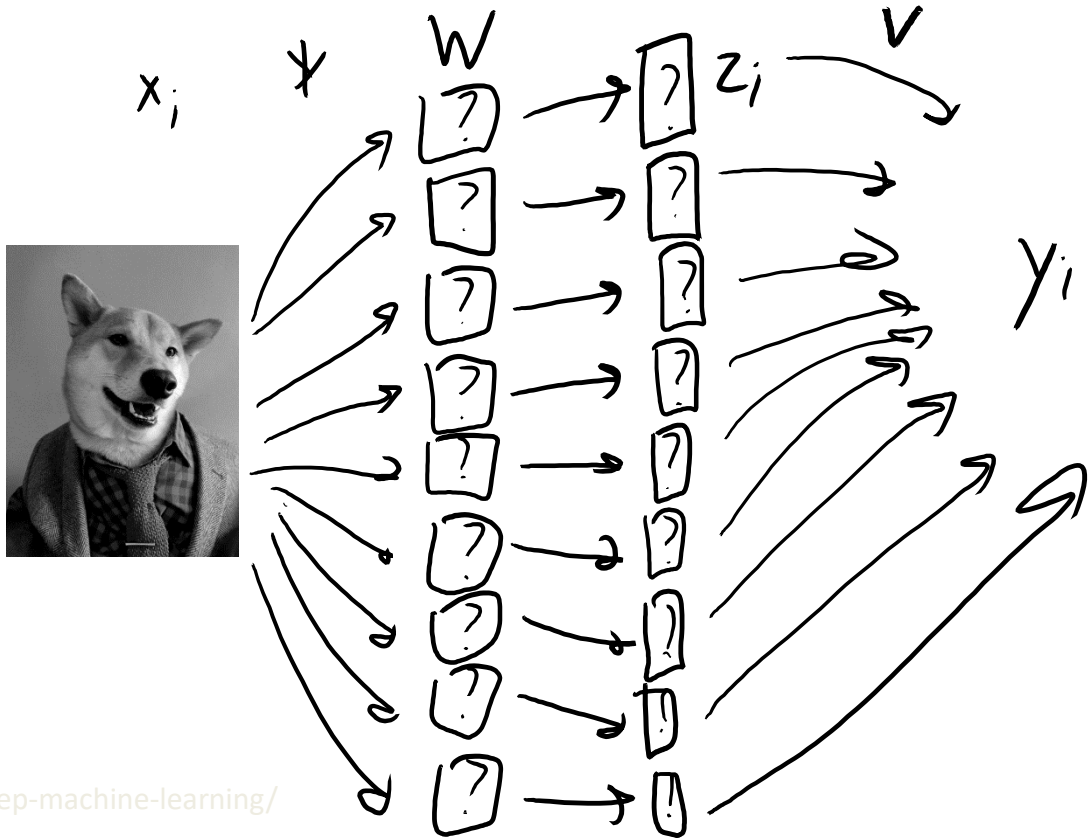
Motivation for Convolutional Neural Networks

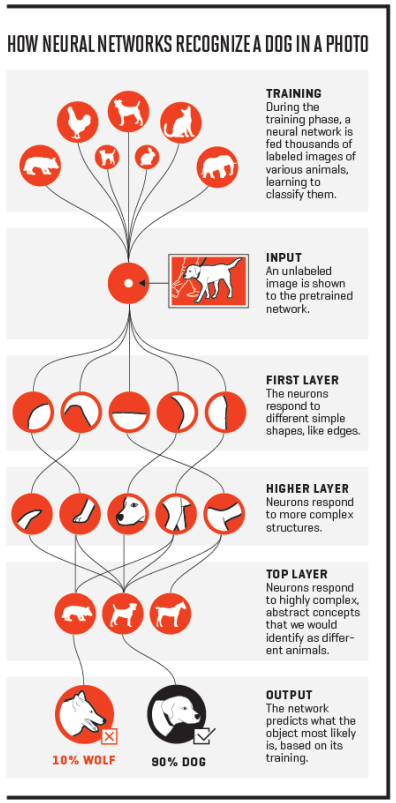
- Convolutional neural networks learn the convolutions:
  - Learning ‘W’ and ‘v’ automatically chooses types/variances/orientations.
  - Don’t pick from fixed convolutions, but learn the elements of the filters.



Motivation for Convolutional Neural Networks

- Convolutional neural networks learn the convolutions:
  - Learning ‘W’ and ‘v’ automatically chooses types/variances/orientations.
  - Can do multiple layers of convolution to get deep hierarchical features.

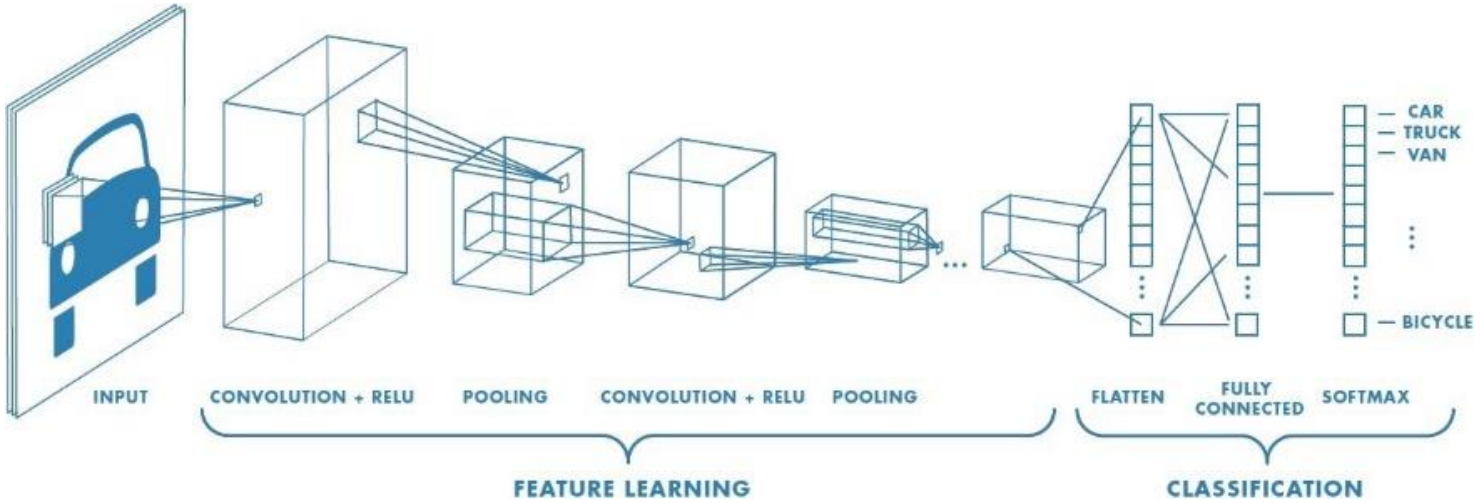




http://fortune.com/ai-artificial-intelligence-deep-machine-learning/

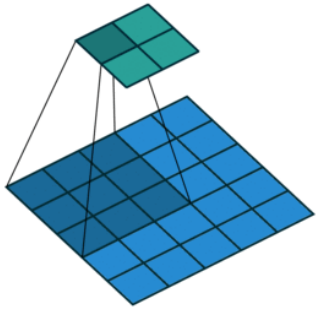
Convolutional Neural Networks

- Classic architecture of a convolutional neural network:



- Convolution layers:
  - Apply convolution with several different filters.
  - Sometimes these have a "stride": skip several pixels between applying filter.
- Pooling layers:
  - Aggregate regions to create smaller images (usually "max pooling").
- Fully-connected layers: usual "multiplication by W^l" in layer.

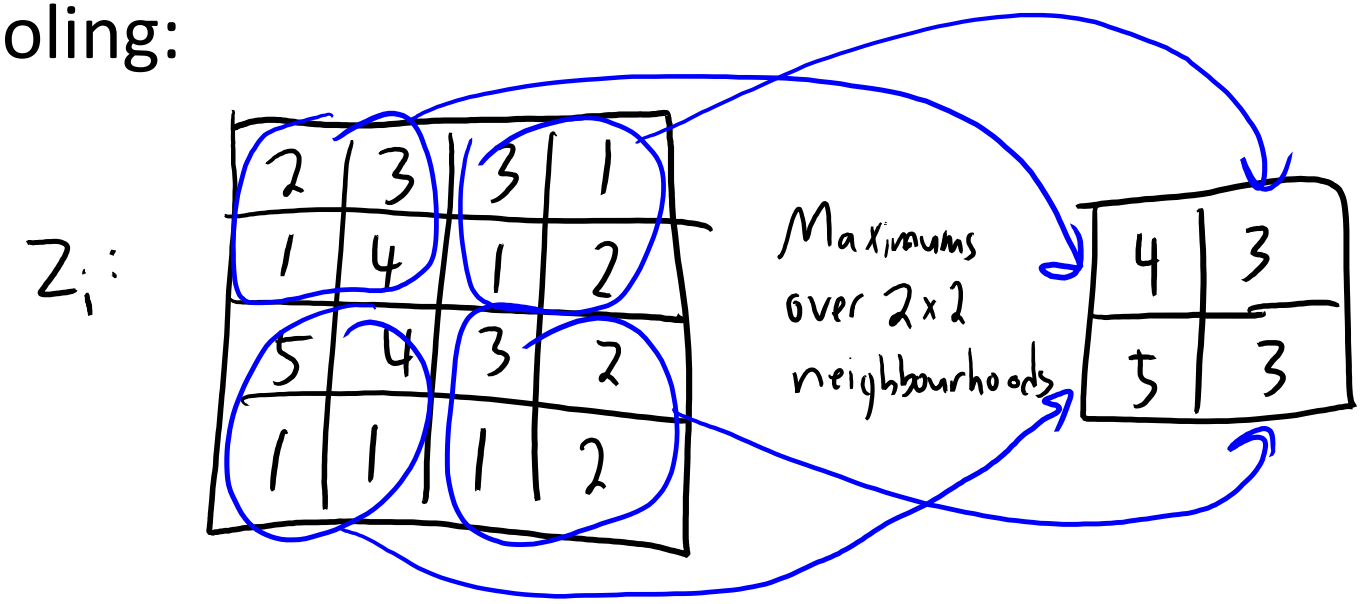
"Stride" of 2



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
https://github.com/vdumoulin/conv_arithmetic

Max Pooling Example

- Max pooling:

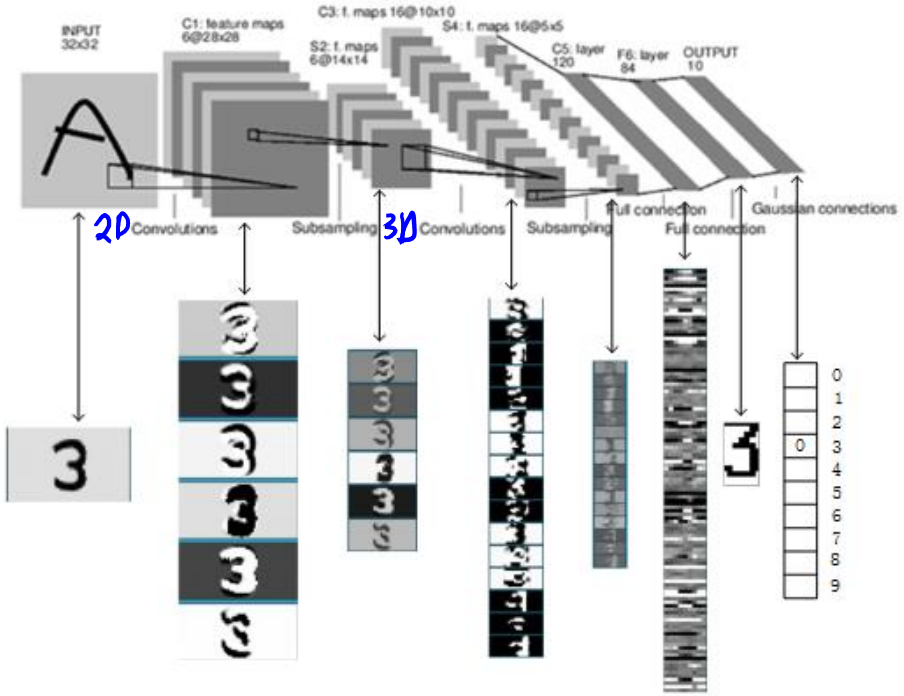


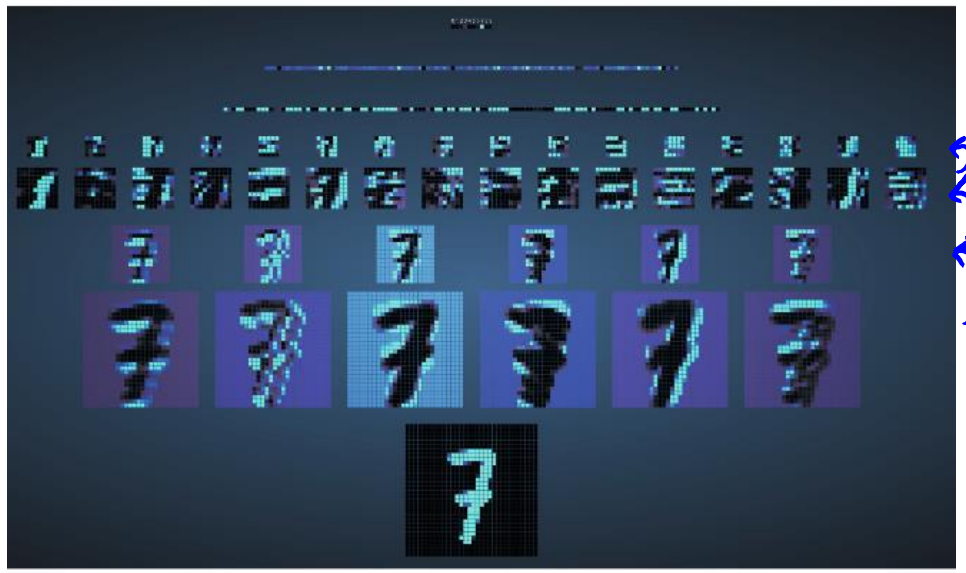
- Decreases size of hidden layer, which speeds up calculations.

- This is continuous and piecewise-linear but non-differentiable.
  - Like ReLU, we can still optimize this type of objective with SGD.

LeNet Convolutional Neural Networks

- Classic convolutional neural network (LeNet):



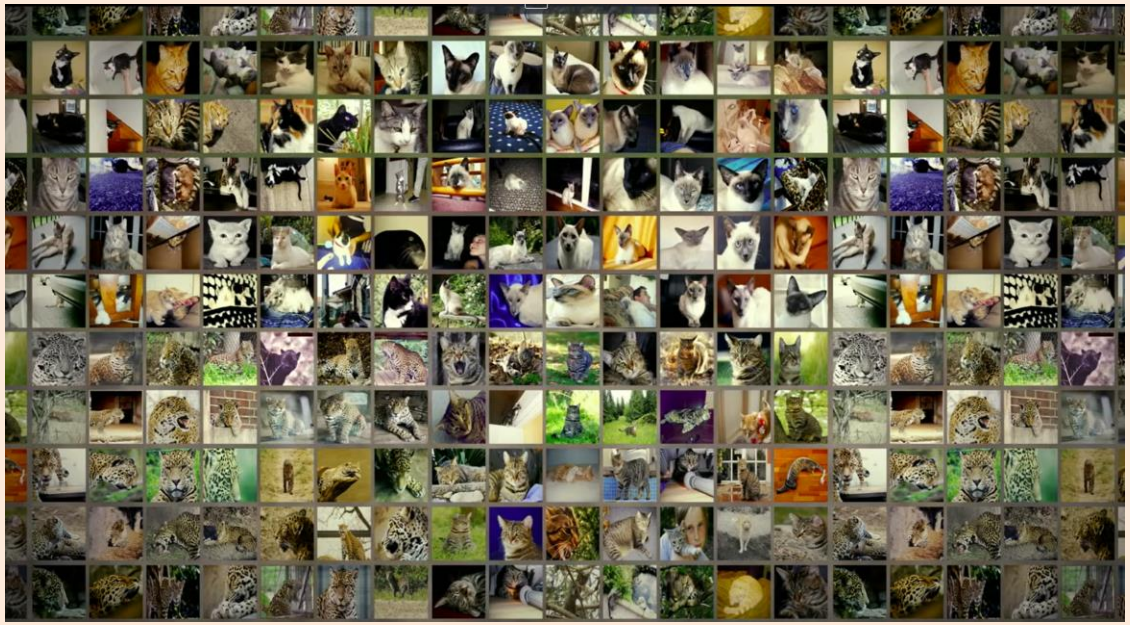


- Visualizing the "activations":
  - An Interactive Node-Link Visualization of Convolutional Neural Networks (adamharley.com)
  - http://cs231n.stanford.edu

http://scs.ryerson.ca/~aharley/vis/harley_vis_isvc15.pdf

ImageNet Competition

- ImageNet: Millions of labeled images, 1000 object classes.
  - Task is to classify images into one of the 1000 class labels.
  - Everyone submits their “best” model, winners announced.
    - Humans error level estimated at ~5%.



Syberian Husky

Canadian Husky

https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
https://www.youtube.com/watch?v=40riCqvRoMs

AlexNet Convolutional Neural Network

- Modern CNN era started with AlexNet (won 2012 competition):
  - 15.4% error vs. 26.2% for closest competitor.
  - 5 convolutional layers.
  - 3 fully-connected layers.
  - SG with momentum.
  - ReLU non-linear functions.
  - Data translation/reflection/ cropping.
  - L2-regularization + Dropout.
  - 5-6 days on two GPUs.

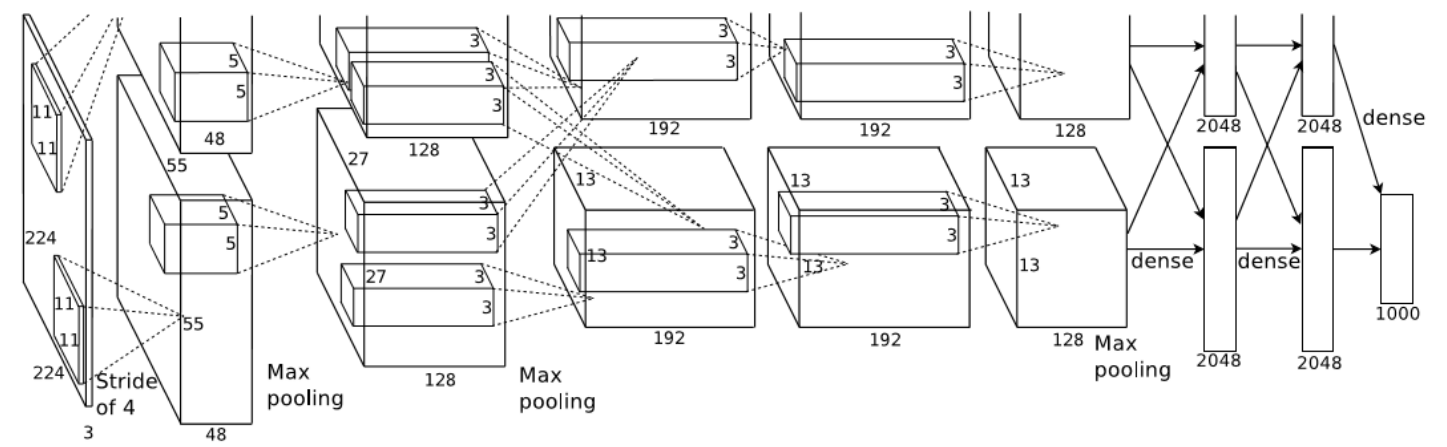
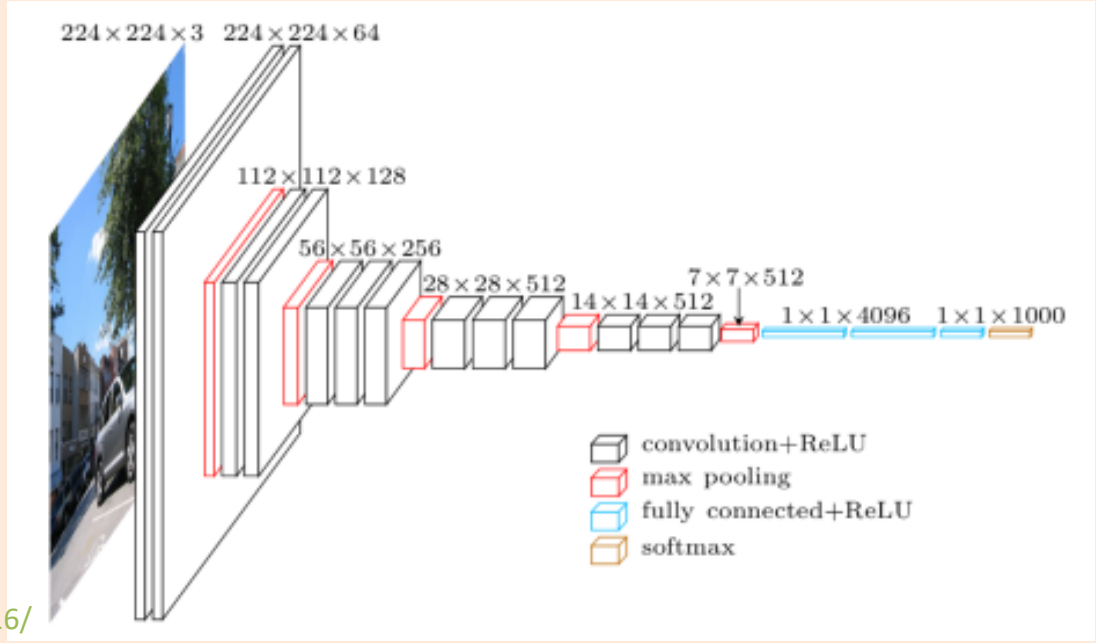


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

ImageNet Insights

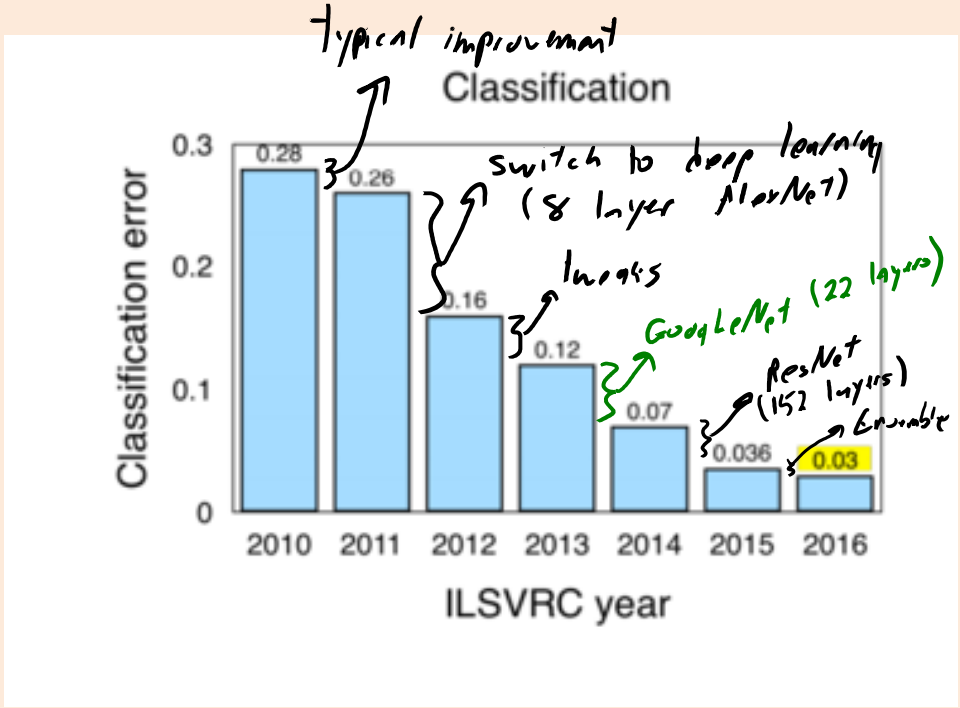
- Filters and stride got smaller over the years.
  - Popular VGG approach uses 3x3 convolution layers with stride of 1.
    - 3x3 followed by 3x3 simulates a 5x5, and another 3x3 simulates a 7x7, and so on.
    - Speeds things up and reduces number of parameters.
    - Also increases number of non-linear ReLU operations.



https://www.cs.toronto.edu/~frossard/post/vgg16/

ImageNet Insights

- Filters and stride got smaller over time.
  - Popular VGG approach uses 3x3 convolution layers with stride of 1.
  - GoogLeNet used multiple filter sizes ("inception layer"), but not as popular.
- Eventual switch to "fully-convolutional" networks.
  - No fully connected layers.
- ResNets introduced in 2015.
  - Won all 5 tasks with 152 layers.
  - Trained for 2-3 weeks on 8 GPUs.
- Ensembles help.
  - 2016 winner: ensemble of previous networks.
- Competition stopped in 2017!



http://www.themtank.org/a-year-in-computer-vision

Discussion of CNNs

- Convolutional layers reduce the number of parameters in several ways:
  - Each hidden unit only depends on small number of inputs from previous layer.
  - We use the same filters across the image.
    - So we do not learn a different weight for each "connection" like in classic neural networks.
  - Pooling layers decrease the image size.

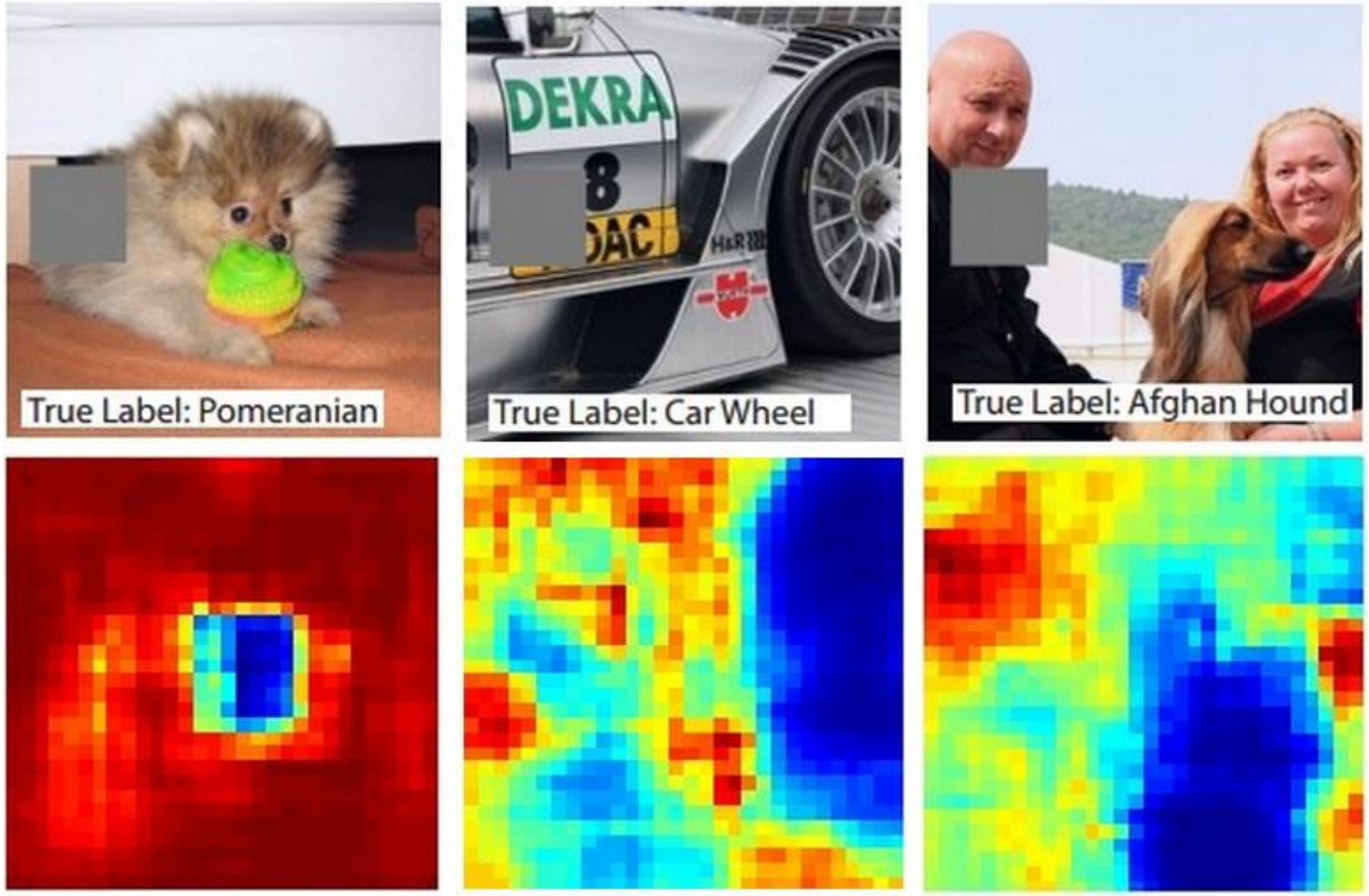
- CNNs give some amount of translation invariance:
  - Because same filters used across the image, they can detect a pattern anywhere in the image.
    - Even in image locations where the pattern has never been seen.

- CNNs are not only for images!
  - Can use CNNs for 1D sequences like sound or language or biological sequences.
  - Can use CNNs for 3D objects like videos or medical image volumes.
  - Can use CNNs for graphs.
- But you do need some notion of "neighbourhood" for convolutions to make sense.

(End of Testable Content)

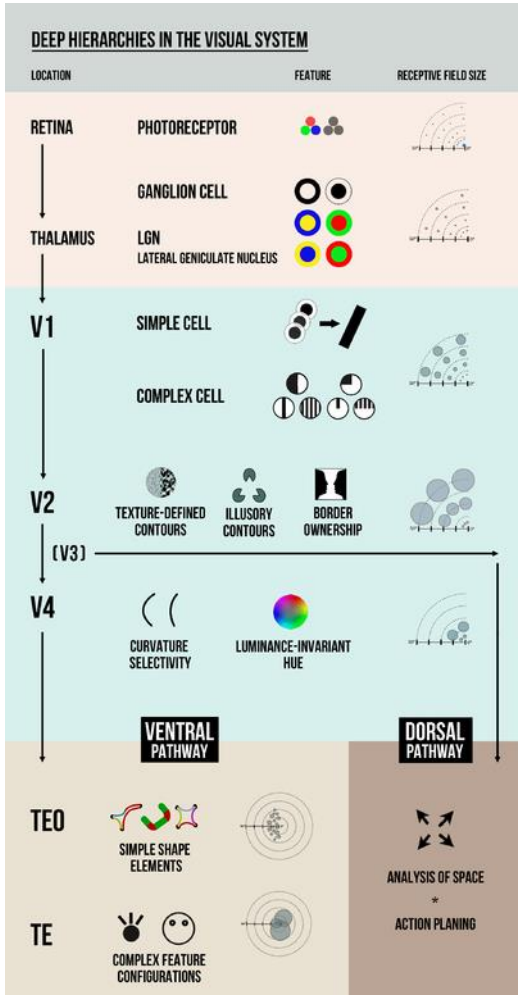
Are CNNs learning something sensible?

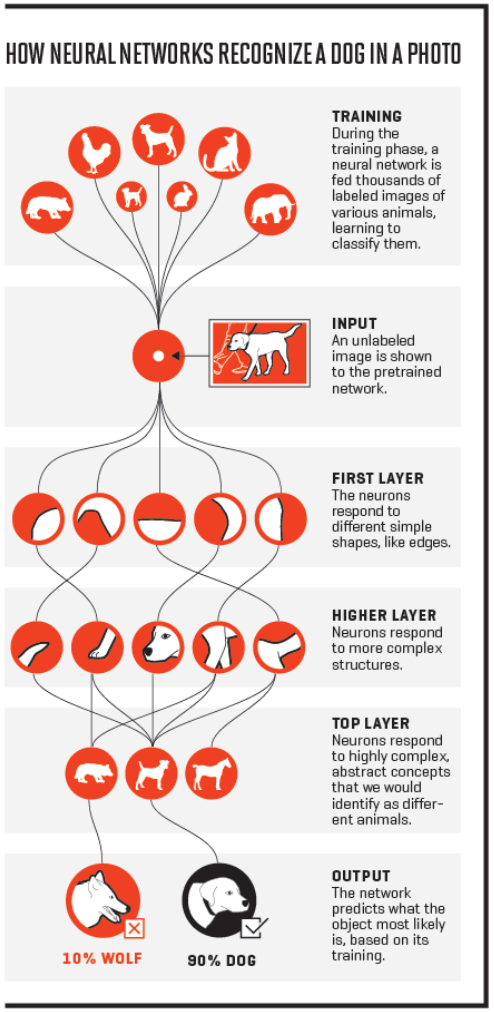
- We can look at how prediction changes if we hide part of image:



Are CNNs learning something sensible?

- Recall that deep learning and CNNs are motivated by ideas about human vision.
  - First layers detect simple features like Gaussians and Gabors filters.
  - Later layers detect more complicated features like corners, repeating patterns.
  - Deeper layers starts to recognize complex parts of objects.
  - Deepest layers recognize full object concepts.
- Is this what trained CNNs actually do?





https://en.wikibooks.org/wiki/Sensory_Systems/Visual_Signal_Processing
http://fortune.com/ai-artificial-intelligence-deep-machine-learning/

Are CNNs learning something sensible?

- Filters learned by first layer of original AlexNet (first CNN winner):

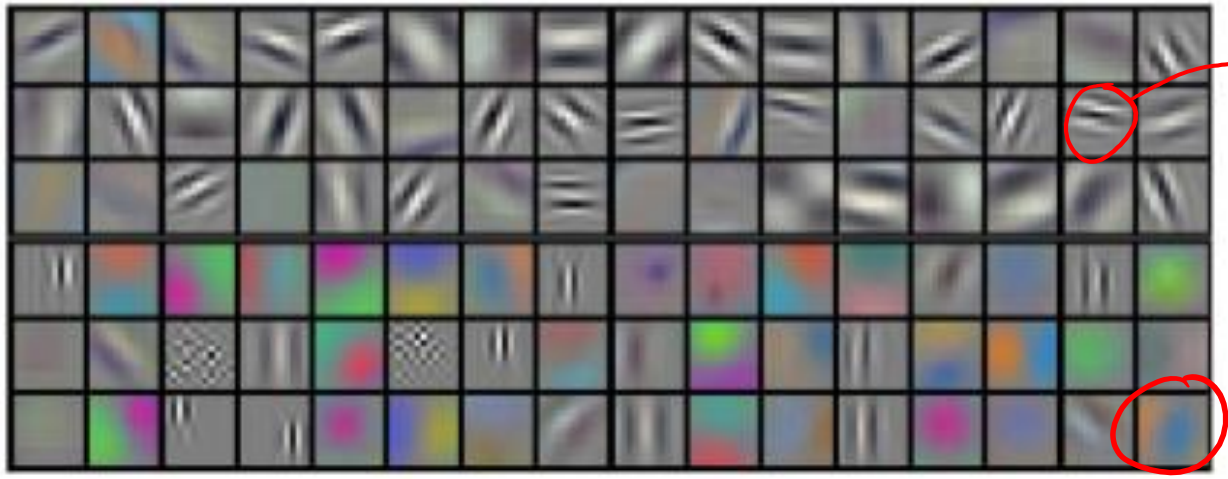


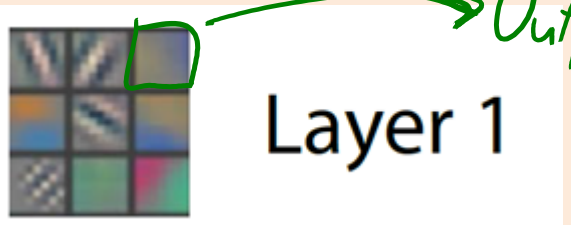
Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The

- Many single-layer models give similar results.

http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Are CNNs learning something sensible?

- It is harder to visualize what is learned in other layers.
  - Approach 1:
    - Search for training data image patches that maximally-activates a filter.
    - Then try to reason about what the filter is doing.
  - Approach 2:
    - Apply deconvolution network to these patches to try to "reverse" the operations.
    - Uses "transposed convolutions" and "unpooling" to visualize "what activated the filter".



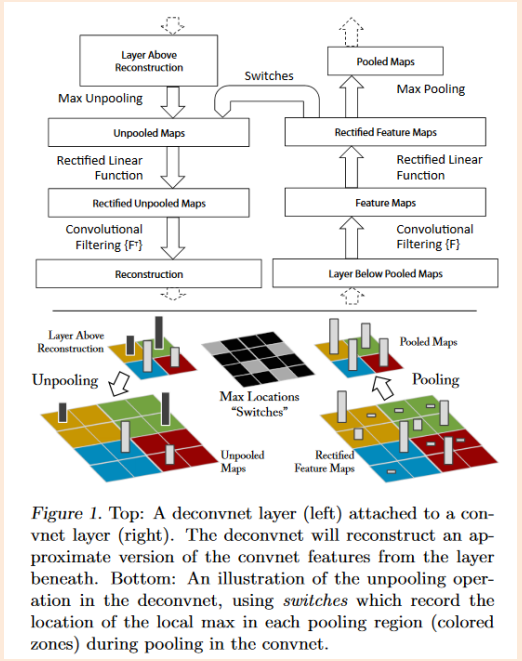
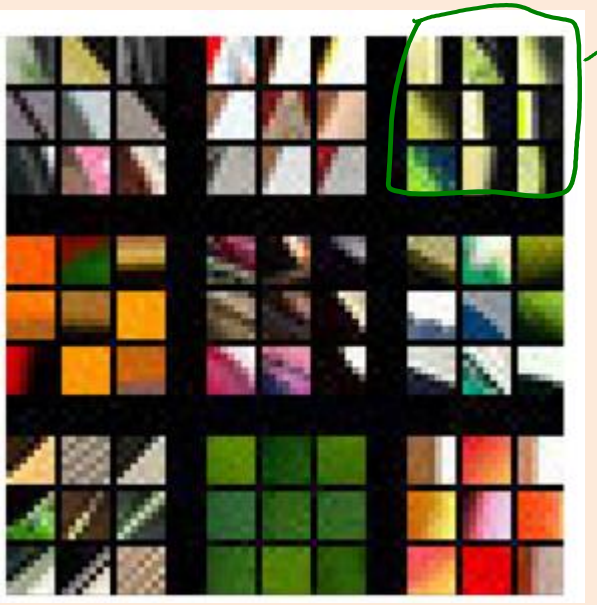
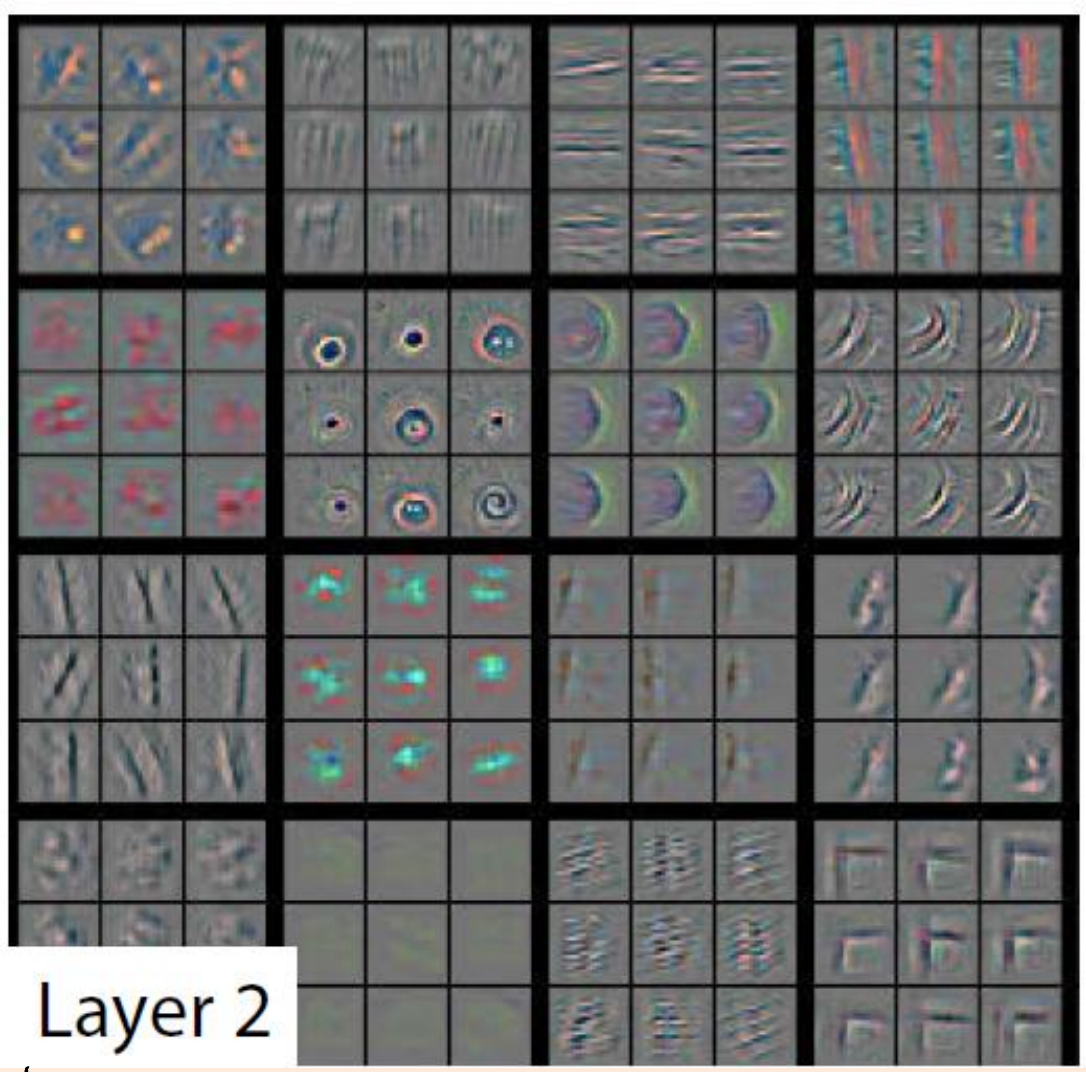


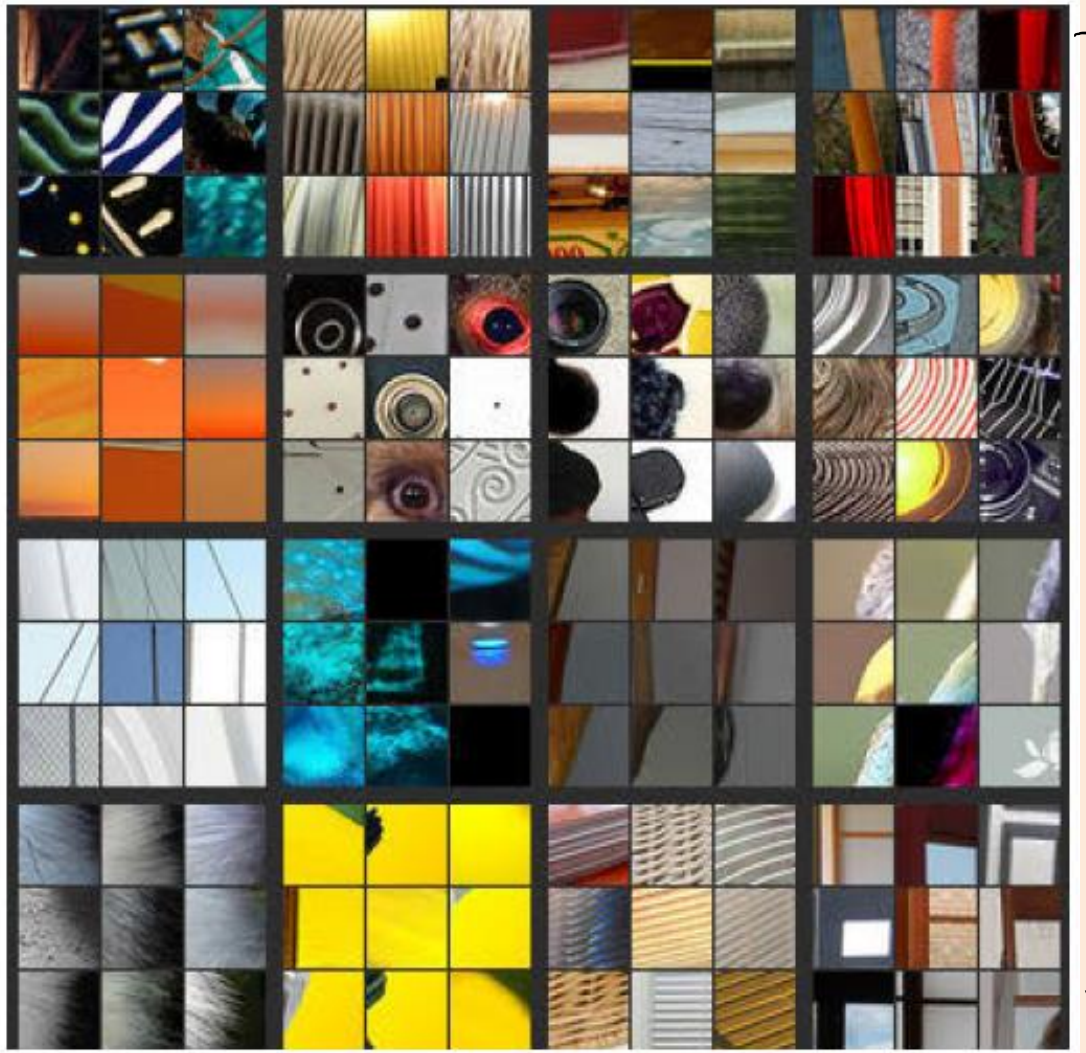
Figure 1. Top: A deconvnet layer (left) attached to a convnet layer (right). The deconvnet will reconstruct an approximate version of the convnet features from the layer beneath. Bottom: An illustration of the unpooling operation in the deconvnet, using *switches* which record the location of the local max in each pooling region (colored zones) during pooling in the convnet.



https://arxiv.org/pdf/1311.2901v3.pdf

Are CNNs learning something sensible?



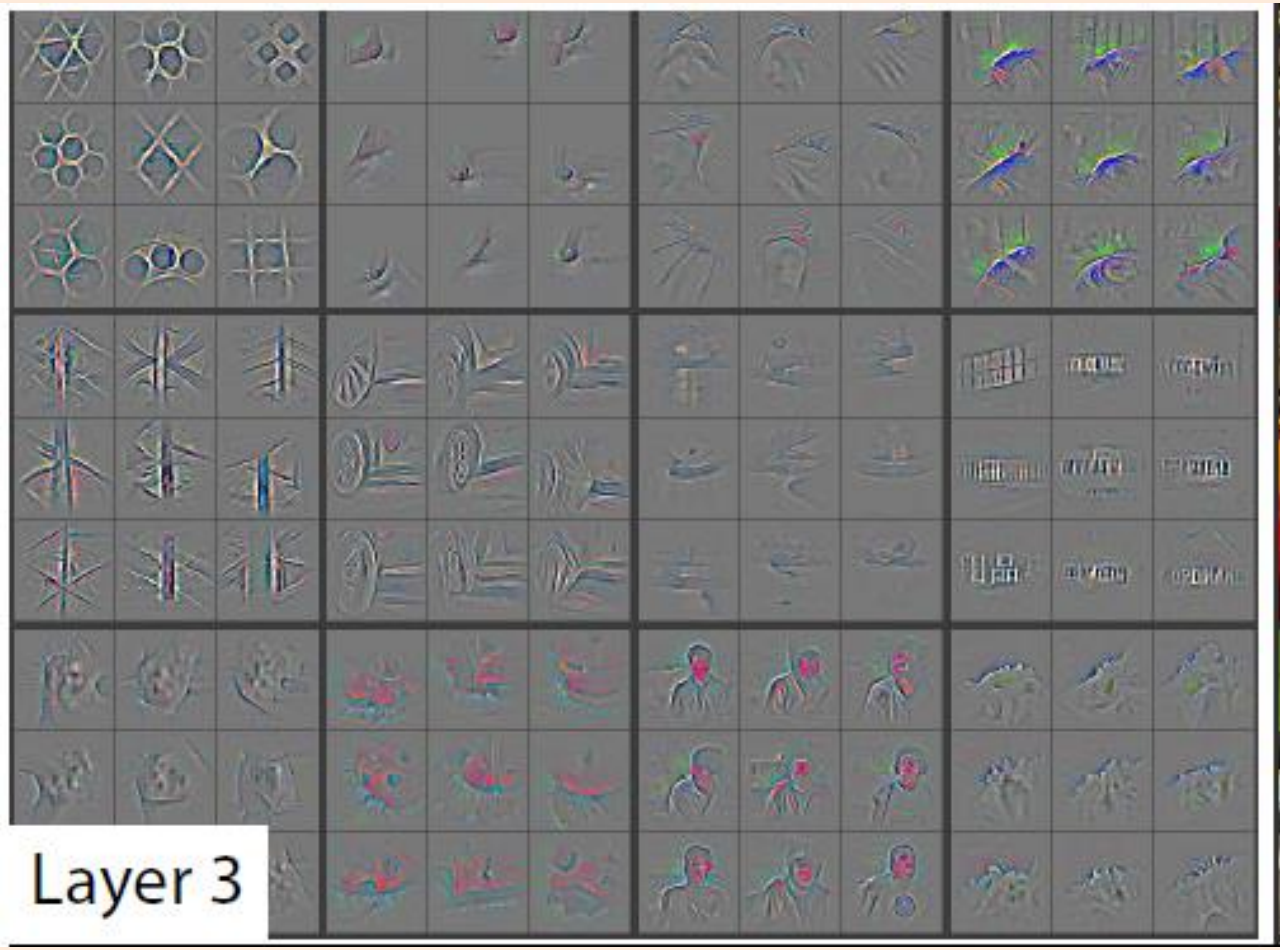


Patch from data giving largest response

Result of deconvolution network

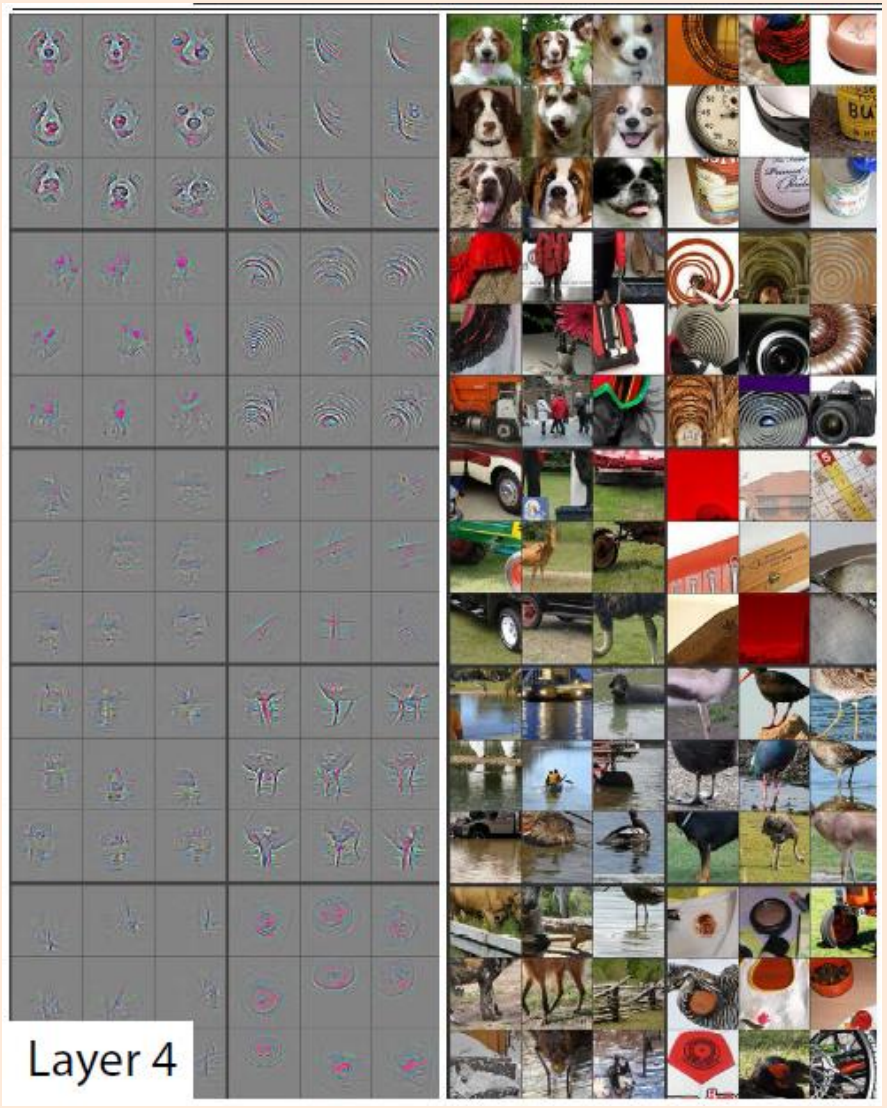
https://arxiv.org/pdf/1311.2901v3.pdf

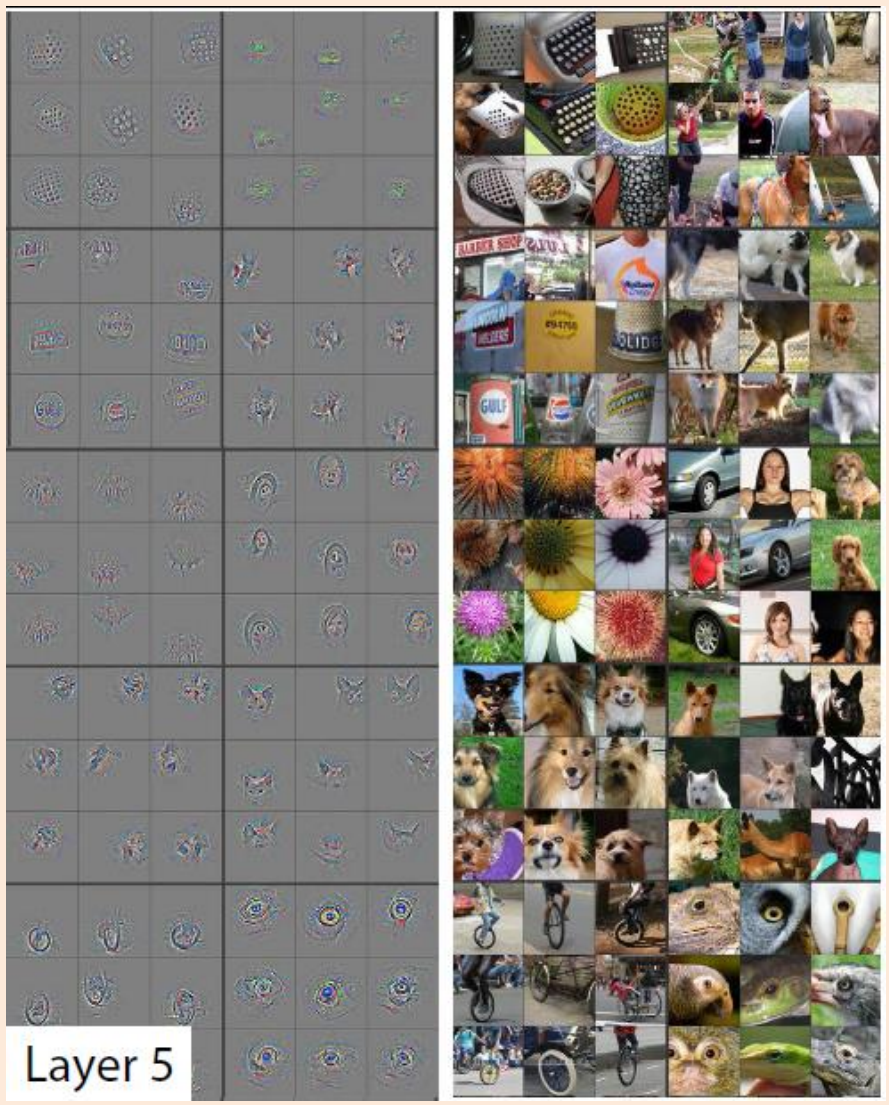
Are CNNs learning something sensible?



https://arxiv.org/pdf/1311.2901v3.pdf

Are CNNs learning something sensible?





https://arxiv.org/pdf/1311.2901v3.pdf

Summary

- Automatic differentiation:
  - Can compute gradient for same cost as objective function.
  - But has some disadvantages compared to human-written code.
  - Backpropagation for computing neural network gradient is a special case.
- Convolutional neural networks:
  - Include layers that apply several (learned) convolutions.
  - Significantly decreases number of parameters.
  - Achieves a degree of translation invariance.
  - Often combined with pooling operations like max pooling.
- CNNs seem to be learning sensible things.
  - Earlier layers seem to represent low-level features.
  - Later layers seem to represent complex object-level features.

- Next time: colourizing black and white images.

Forward-Mode Automatic Differentiation

- We discussed "reverse-mode" automatic differentiation.
  - Given a function, writes code to compute its gradient.
  - Has same cost as original function.
  - But has high memory requirements.
    - Since you need to store all the intermediate calculations.

- There is also "forward-mode" automatic differentiation.
  - Given a function, writes code to compute a directional derivative.
    - Scalar value measuring how much the function changes in one direction.
  - Has same memory requirements as original function.
  - But has high cost if you want the gradient.
    - Need to use it once to get each partial derivative.

Failure of AD on ReLUs

In many settings, our underlying function $f(x)$ is a nonsmooth function, and we resort to subgradient methods. This work considers the question: is there a *Cheap Subgradient Principle*? Specifically, given a program that computes a (locally Lipschitz) function f and given a point x, can we automatically compute an element of the (Clarke) subdifferential $\partial f(x)$ [Clarke, 1975], and can we do this at a cost which is comparable to computing the function $f(x)$ itself? Informally, the set $\partial f(x)$ is the convex hull of limits of gradients at nearby differentiable points. It can be thought of as generalizing the gradient (for smooth functions) and the subgradient (for convex functions).

Let us briefly consider how current approaches handle nonsmooth functions, which are available to the user as functions in some library. Consider the following three equivalent ways to write the identity function, where $x \in \mathbb{R}$,

$$f_1(x) = x, \quad f_2(x) = \text{ReLU}(x) - \text{ReLU}(-x), \quad f_3(x) = 10f_1(x) - 9f_2(x),$$

where $\text{ReLU}(x) = \max\{x, 0\}$, and so $f_1(x) = f_2(x) = f_3(x)$. As these functions are differentiable at 0, the unique derivative is $f_1'(0) = f_2'(0) = f_3'(0) = 1$. However, both TensorFlow [Abadi et al., 2015] and PyTorch [Paszke et al., 2017], claim that $f_1'(0) = 1$, $f_2'(0) = 0$, $f_3'(0) = 10$. This particular answer is due to using a subgradient of 0 at $x = 0$. One may ask if a more judicious choice fixes such issues; unfortunately, it is not difficult to see that no such universal choice exists[1].

[1] By defining $\text{ReLU}'(0) = 1/2$, the reader may note we obtain the correct derivative on f_2, f_3; however, consider $f_4(x) = \text{ReLU}(\text{ReLU}(x)) - \text{ReLU}(-x)$, which also equals $f_1(x)$. Here, we would need $\text{ReLU}'(0) = \frac{\sqrt{5}-1}{2}$ to obtain the correct answer.

https://arxiv.org/pdf/1809.08530.pdf