CSci 4270 and 6270
Computational Vision, Spring 2021
Lecture 17: Convolutional Neural Networks
March 29, 2021

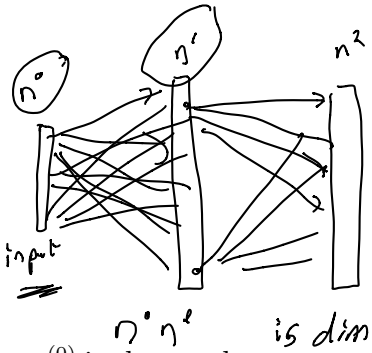
Overview — Beyond the Basics

- Convolutional neural networks
- LeNet architecture
- Loss functions
- Image category recognition and results
- Regularization and training methods
- Summary

In addition to the references provided in the Lecture 15 notes, here is an additional source:

`https://cs231n.github.io/`

Limitations of Our Network So Far



... especially for computer vision ....

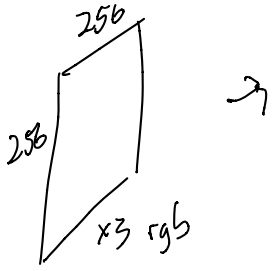
- Lots of parameters:

$$\sum_{l=1}^{L} n^{(l-1)} n^{(l)} + \sum_{l=1}^{L} n^l$$

bias

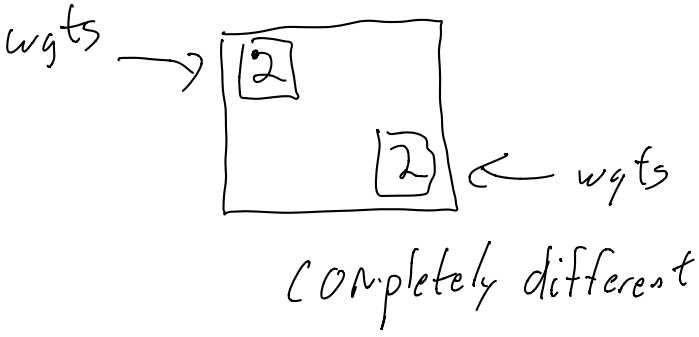
Think about how big this is, starting from the fact that $n^{(0)}$ is the number of pixels in the input image!

- No sharing of information:
  - Must "re-learn" the same feature at each location where it could appear



unravel
65k x 3
~ 200k
at layer 0

20x reduction
$n^1 = 10{,}000$
$2 \times 10^5 \times 10^4$
2×10^9
weights



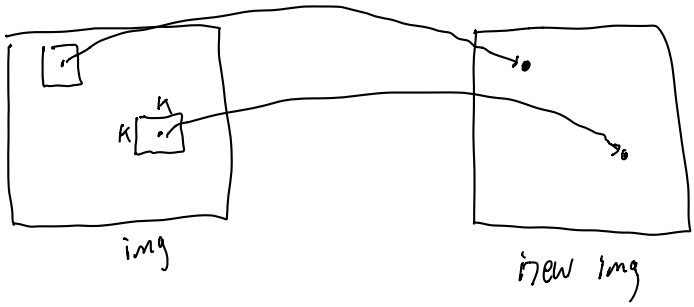
Back to the Notion of a Convolution

- Compare this to "classical" convolution operations in computer vision, including smoothing, edge detection, interest-point extraction and many others, where the same operations are applied throughout the image.
- In a convolution, a $k \times k$ filter — like a smoothing filter or a differentiation filter — is applied to each pixel of an image to produce a new image.
- Mathematically, a convolution is written as:

$$I_{\text{out}}(x, y) = \sum_{u=-k/2}^{k/2} \sum_{v=-k/2}^{k/2} w(u, v) * I_{\text{in}}(x + u, y + v).$$

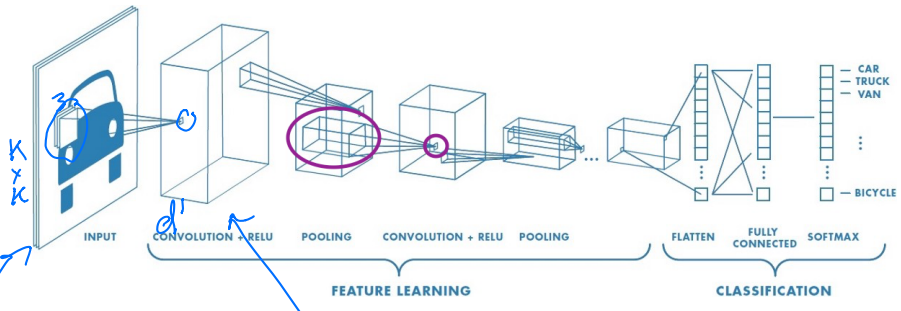
  We will remind ourselves of the form of w for several operations

- Important notes about a *single convolution*:
  1. w is defined over only a small area.
  2. The same w is used to compute the convolution each pixel.



Convolutional Layers of a Neural Network

- We will apply a set of convolution filters to the input image to produce a set of new images. This is a convolution layer.
- Each layer is an image, and each such image stores a vector of the $d^{(l)}$ values that result from applying convolution to the previous layer's image vector.
  - Layer 0, corresponding to the input image, will typically have $d^{(0)} = 3$ for RGB images.
- Suppose each convolution kernel at layer $l \geq 1$ covers $k^{(l)} \times k^{(l)}$ pixels. Then, as illustrated by the ovals in the following diagram, each convolution operation involves all $d^{(l-1)}$ values at the previous layer to produce a single value at layer l.



rgb

d^0 values at each pixel at layer 0

$d^0 = 3$

d^1 values at each pixel at next level

each is the result of a convolution $K \times K \times d^0$

d^1 convolutions $K \times K \times d^0$ each

Think of convolutions you'd like (intuition).

1. smooth
2. derviativ in x
   derivative in y
   2nd dcrivative
3. texture measure

many things we'd like to measure.

d^4 convolution values at each pixel

The Convolution Equations

- Let (x, y, f) be a combination of a pixel location (x, y) and a feature index f.

- Then the input to the neuron with these indices at layer l is

$$z^{(l)}(x, y, f^l) = \sum_{u=-k/2}^{k/2} \sum_{v=-k/2}^{k/2} \sum_{f^{(l-1)}=1}^{d^{(l-1)}} w(u, v, f^{(l-1)}; f^l)\, a^{(l-1)}(x+u, y+v, f^{(l-1)})$$

- And the activation of the neuron (its output) is

$$a^{(l)}(x, y, f) = \text{ReLU}(z^{(l)}(x, y, f))$$

where "ReLU" is the "Rectified Linear Unit" activation function, introduced in the previous lecture notes and to be revisited soon.

- We will go over these equations and relate them to the diagram above in class.

Summary So Far

- Each network layer is an 3d array (tensor):
  - $M^{(l)} \times N^{(l)}$ rows and columns
  - $d^{(l)}$ filter outputs at each pixel
- Each filter at layer l uses all $d^{(l-1)}$ values at the pixels in the $k^{(l)} \times k^{(l)}$ neighborhood surrounding the pixel of interest in layer $l-1$.
  - Example: for RGB input images, all layer 1 filters use all RGB values at their input pixels.
- We structure $d^{(l)}$ different convolution filters at each layer. Each is three-dimensional.
- If each filter covers $k^{(l)} \times k^{(l)}$ pixels then there are

  $$k^{(l)} \cdot k^{(l)} \cdot d^{(l-1)}$$

  parameters (weights) that define each filter and

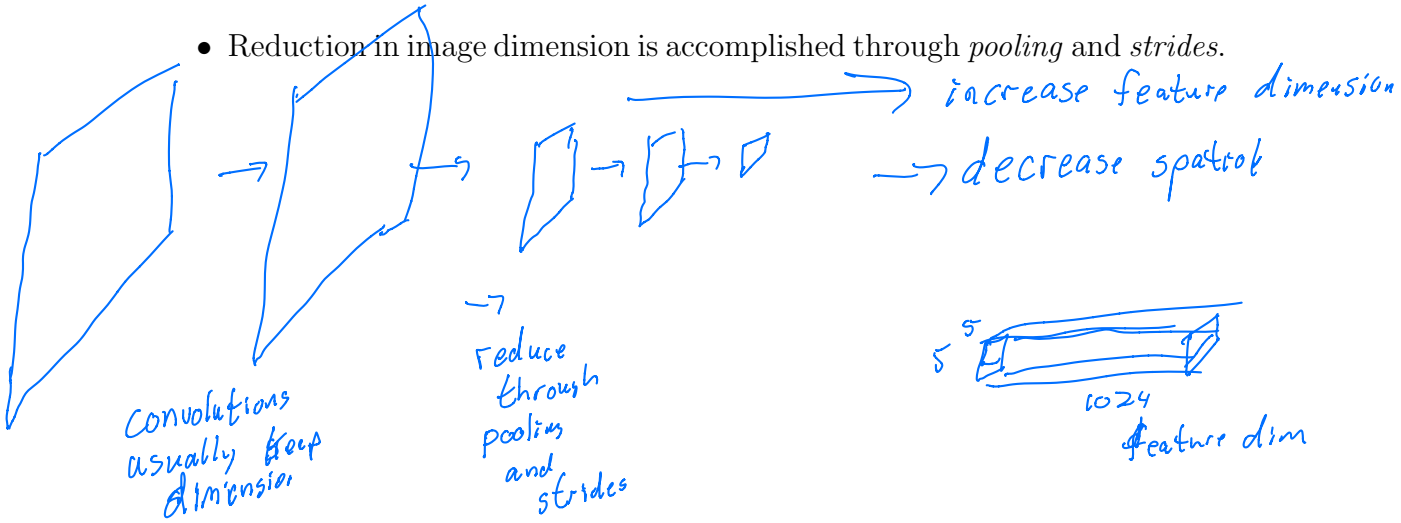
  $$k^{(l)} \cdot k^{(l)} \cdot d^{(l-1)} \cdot d^{(l)}$$

  weights overall in each layer.

  } independent of image size!!
- These parameters are learned, as explained below, through a modified form of backpropagation.

Changes Through the Layers

- As the computation progresses through the layers, we tend to see
  - Increase in the filter dimensions, resulting in more values at each pixel.
  - Smaller images.
- Reduction in image dimension is accomplished through *pooling* and *strides*.

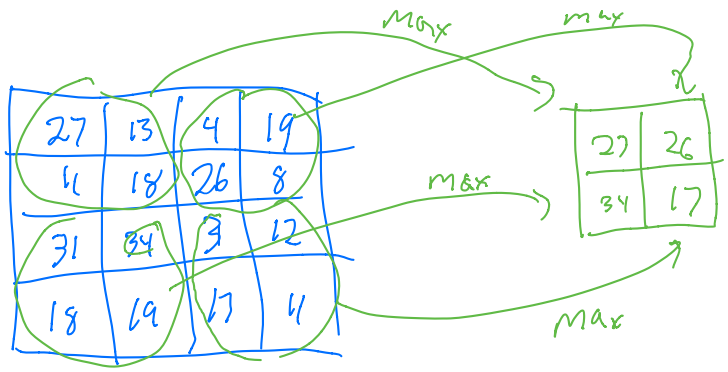


Max Pooling and Striding

- At some layers — called "pooling layers" — instead of applying a linear convolution followed by ReLU activation, apply a max operation over a small neighborhood at each pixel.
  - Usually, the neighborhood of a pooling layer is 2×2 (i.e. $k^l = 2$)
  - Applied to each of the d^{l-1} dimensions at each pixel independently.
- Pooling operations are usually applied at every other pixel in both rows and columns, reducing the overall image dimension by 4.
  - In this case, what we refer to as the "stride length" is 2.
- Purpose of max pooling
  - Adds non-linearity,
  - Reduces sensitivity to position, and
  - Decreases dimensionality.

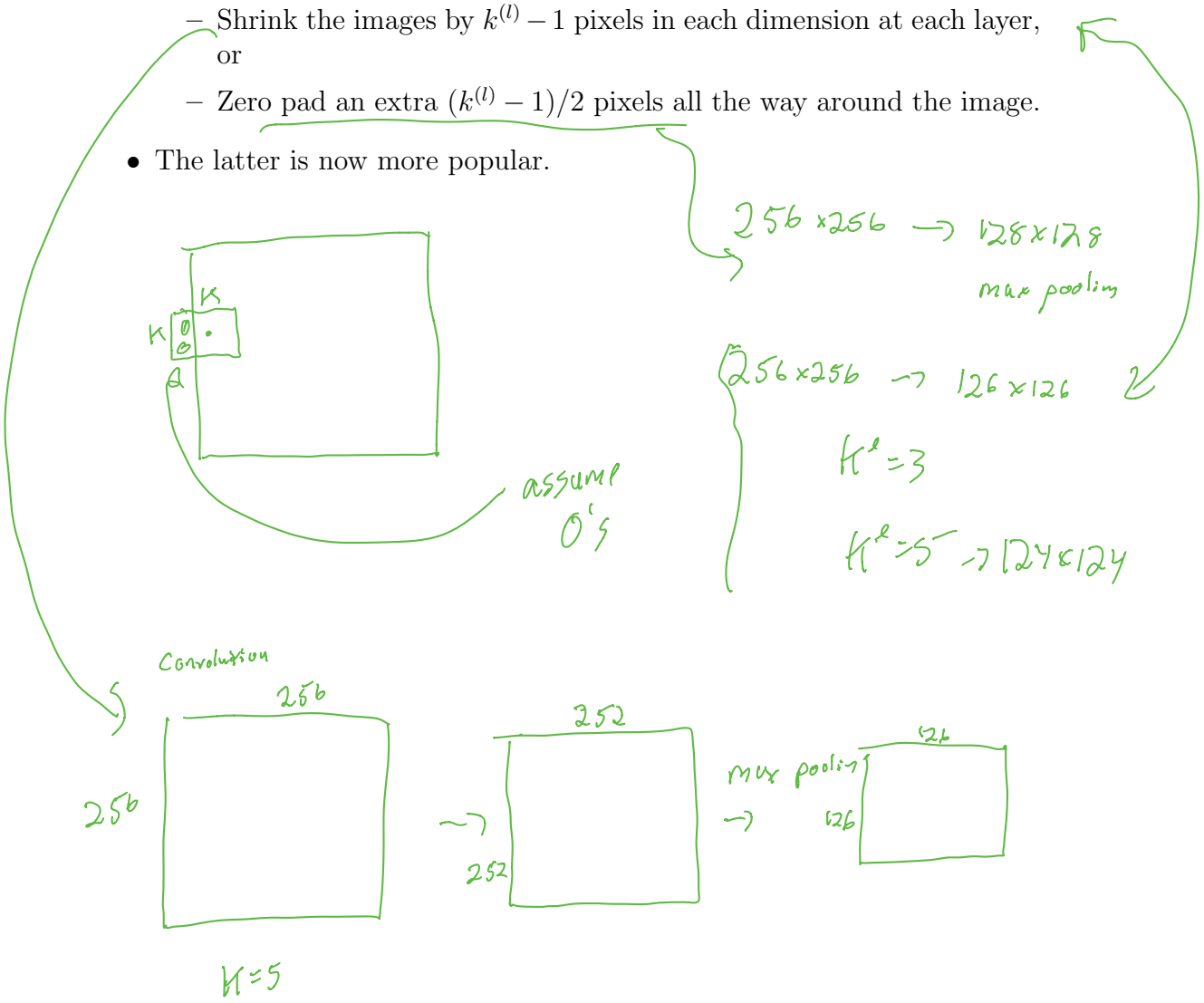
small image and look at one feature dim

2x2 max pooling with stride of 2



Dealing with Shrinkage

- Within $(k^{(l)} - 1)/2$ pixels of the image border, the convolution kernel does not fit within the image.
- Options:
  - Shrink the images by $k^{(l)} - 1$ pixels in each dimension at each layer, or
  - Zero pad an extra $(k^{(l)} - 1)/2$ pixels all the way around the image.
- The latter is now more popular.



Fully Connected Layers and the LeNet Classifier Architecture

- At the top of the network as we've been describing it, with convolution layers and pooling layers, add two or three additional layers where each is "fully-connected" to its previous layer.
  - Mathematically, these are like "hidden" layers in our previous set of notes.
  - They are also known as "dense" layers.
- Resulting architecture (see Figure 1, downloaded from `https://irenelizihui.files.wordpress.com/2016/03/tf43.png`, Nov 16, 2017):
  - Lower levels alternate convolution and pooling layers — although could have consecutive layers of one or the other.
  - Top few layers are series of fully-connected layers
  - Last / output layer has one node for each class we are trying to recognize.

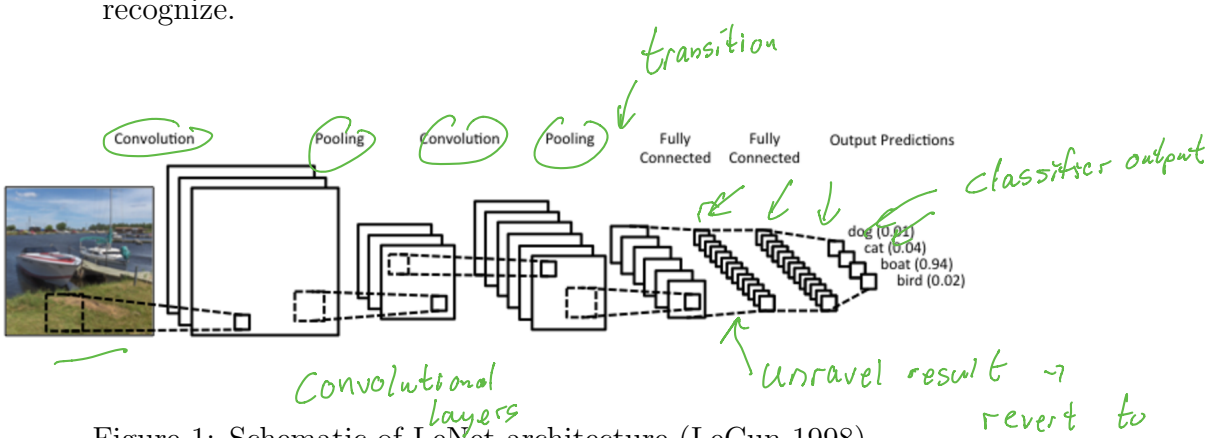
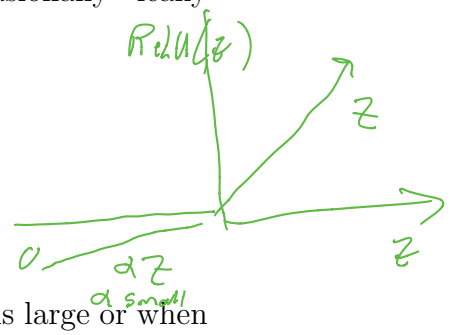


Figure 1: Schematic of LeNet architecture (LeCun 1998).

ReLU Activation Function

- Activation function: use ReLU instead of sigmoid (occasionally "leaky ReLU")
- Reminder:

$$\text{ReLU(z)} = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$



- Replace sigmoid because:
  - Can show that sigmoid learning is slow when error is large or when activations are large.
  - Unbounded activation of ReLU tends to make training more efficient.

Output Layer: Use Soft-Max Activation

- Used in cases where the desired result is a single label / classification decision.
- Converts the input to the final layer of neurons to a probability distribution.
- Recalling that z_i^L is combined input value to the i neuron at the final layer (layer L) , the activation/output of neuron i is

$$a_i^L = \sigma(z_i^L) = \frac{\exp(z_i^L)}{\sum_j \exp(z_j^L)}$$

- By playing around with different values of z you can study the effect of this. For example, given a vector of z values

$$\mathbf{z}^L = [3, 5, 1, 1]$$

the output activation vector is approximately

$$\mathbf{a}^L = [0.115, 0.853, 0.016, 0.016]$$

- You notice that this:
  1. Exaggerates differences between z values
  2. Enforces $0 < a_j^L < 1$ for all output layer activations
  3. Enforces

$$\sum_{j=1}^{n^L} a_j^L = 1$$

Cross-entropy (cost) Loss Function

- Replaces quadratic loss function.
- Notation as before
  - y is the expected result for training input x, and
  - we treat y as a binary vector with a 1 for the desired class and 0 otherwise.
- Cross-entropy loss function is

$$C(x, y) = -\sum_j [y_j \log(a_j^L) + (1 - y_j) \log(1 - a_j^L)]$$

- Think about each in terms of the extreme cases when the network is exactly right, when it is completely uncertain, or it is completely wrong.

$\vec{y} = [0, 0, 1, 0, 0]$ for five digits 0, 1, 2, 3, 4

$\rightsquigarrow C$ becomes $-[\log(1-a_0^L) + \log(1-a_1^L) + \log(a_2^L) + \log(1-a_3^L) + \log(1-a_4^L)]$

$a_i^L = 0$??

$1 - \cdots$ force that part of a toward 0

i better not be 2!! $\to \infty$

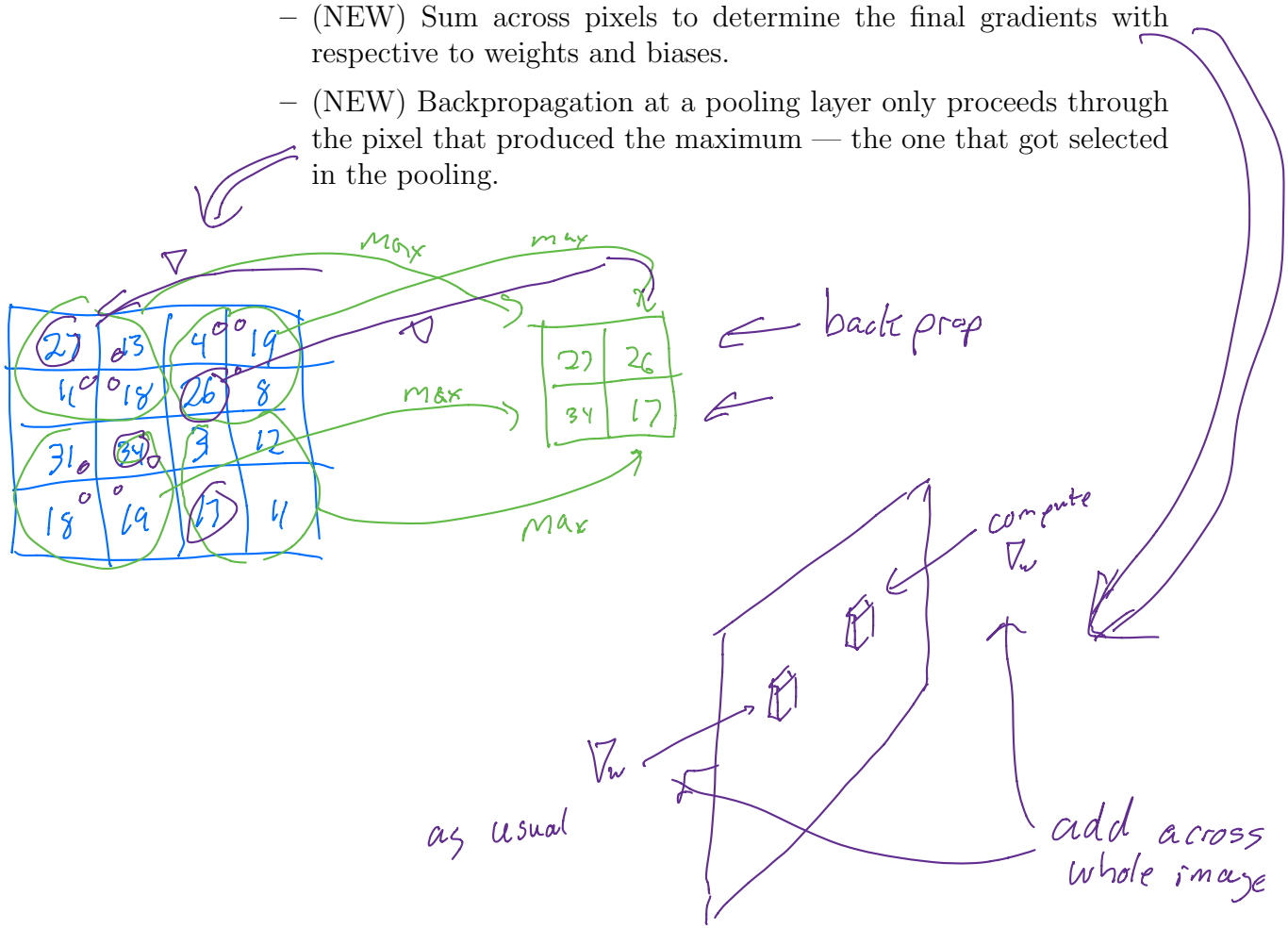
$\log(a_2^L)$ pushes toward 1

$a_4^L = 1 \Rightarrow C \to \infty$

pushes network not to be over confident.

Changes to Backpropagation

- Back propagation through fully-connected layers is as before
- The pooling layers remember which neuron produced the max during forward propagation and backpropagate error only through that neuron.
  - Gradient signal to the other three neurons — the ones that didn't get picked — is 0.
- Back propagation through convolutional layers is only a little more complicated:
  - Compute the "error vectors", $\boldsymbol{\delta}^L$ for the top layer and $\boldsymbol{\delta}^l$ for $l = L-1$ downto 0 as before.
  - Compute the cost gradients $\partial C/\partial \mathbf{b}^l$ and $\partial C/\partial \mathbf{w}^l$ as though the weights were distinct — i.e. different weights at each pixel.
  - (NEW) Sum across pixels to determine the final gradients with respective to weights and biases.
  - (NEW) Backpropagation at a pooling layer only proceeds through the pixel that produced the maximum — the one that got selected in the pooling.



What Do the Learned Filters Look Like?

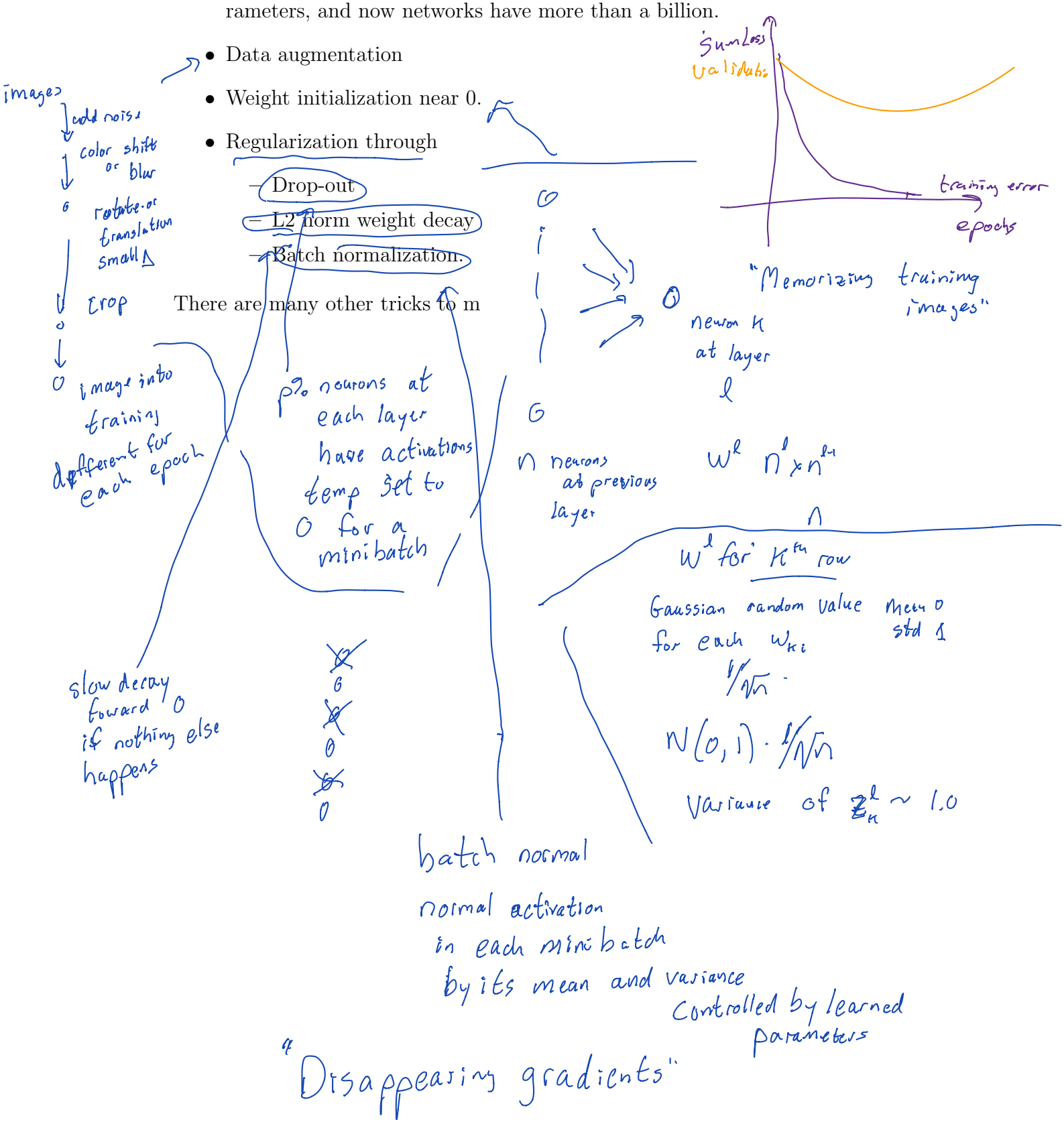
- See Figure 2 from Krizhevsky et al., NIPS 2012. This is the AlexNet architecture.
- 96 $11 \times 11 \times 3$ kernels, split into two sets.

Figure 2: First layer convolution filters from AlexNet. Taken from Krizhevsky 2012.

Additional Details

- Over-fitting is a big concern. AlexNet, for example, had 60 million parameters, and now networks have more than a billion.
- Data augmentation
- Weight initialization near 0.
- Regularization through
  - Drop-out
  - L2 norm weight decay
  - Batch normalization.

There are many other tricks to m



Data Sets and Challenges

This is a preliminary discussion

- Pascal VOC 2007-2012
- Imagenet Large Scale Visual Recognition Challenge (ILSVRC) 2010-2017
- Kaggle takes over ILSVRC in summer 2017.
- Problems:
  - Image classification (category recognition): 1,000 categories
  - Objection localization: 1,000 categories
  - Object detection: 200 categories.
  - Fine-grained classification, e.g. species of dogs

A History of ConvNets and Competitions

- LeNet-5 1998: zip codes, digits, etc.
- Break-through in 2012: AlexNet 10% lower error rate than best previous, non-ConvNet results.
  - See Figure 3.

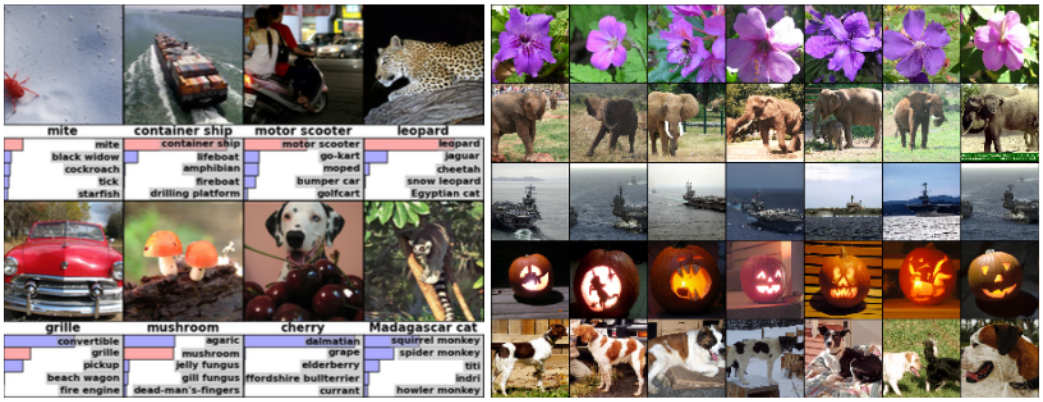


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Figure 3: Alex net example results

- 2013 winner: ZFNet
- 2014 winners: GoogLeNet and VGGNet
- ...
- Currently better than human performance
  - Although humans recognize far more than 1,000 categories

"Modern" Feedforward Architectures

- Narrow and tall
- Towards fully convolutional
- De-emphasize pooling
- No strict hierarchy of layers
- Ensembles

Summary

- Neural networks and convolutional neural networks in particular are the dominant from of solution to image classification problems.
- Millions training images are needed — easily 10K - 100K per class.
- If you can formulate your problem as a classification problem and you can obtain massive amounts of labeled data these types of DCNN's can often solve your problem!
- More sophisicated networks, including recurrent networks, are needed for other problems.