

Convolutional Neural Networks

Jay Urbain, PhD

Credits:

- Deep Learning with Python, Francios Challet
- Andrej Karpathy, Chris Olah, Andrew Ng
- nVidia DLI
- HSE Faculty for Advanced Machine Learning: Pavel Shvechikov, Researcher at HSE and Sberbank AI Lab; Anna Kozlova, Team Lead; Evgeny Sokolov, Senior Lecturer; Alexey Artemov, Senior Lecturer and Sergey Yudin, Analyst-developer among multiple other trainers.

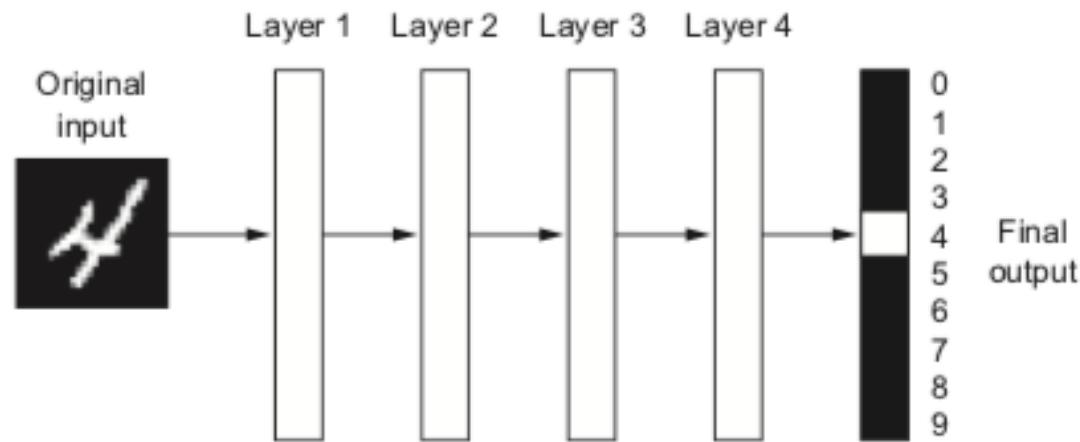
Vehicle Detection

cv_op_2018/1 Deep Learning Introduction.pptx



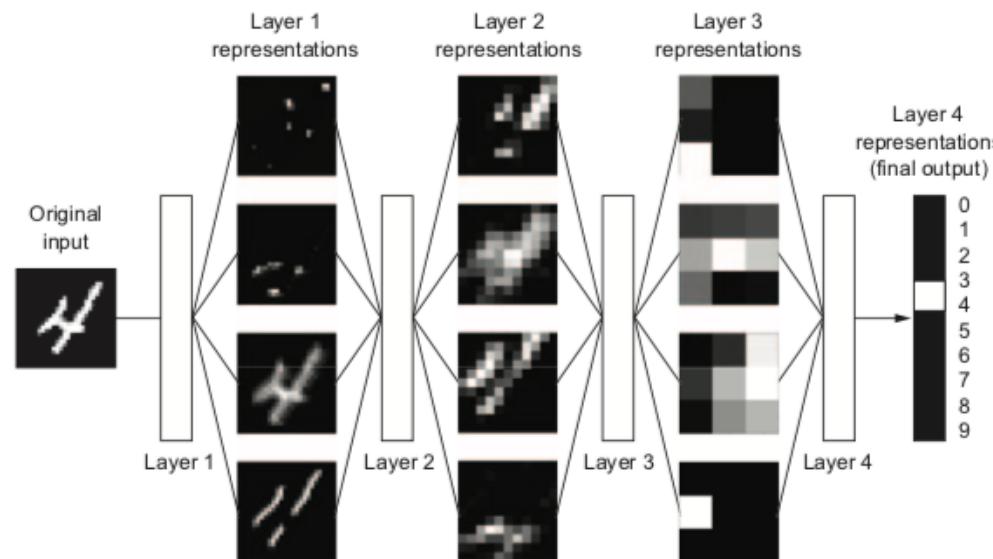
Deep learning as a language

- Provide hints of what to learn by defining structure of the model.
- Think about representations that would transform inputs to outputs.



Deep representations learned by a digit-classification model

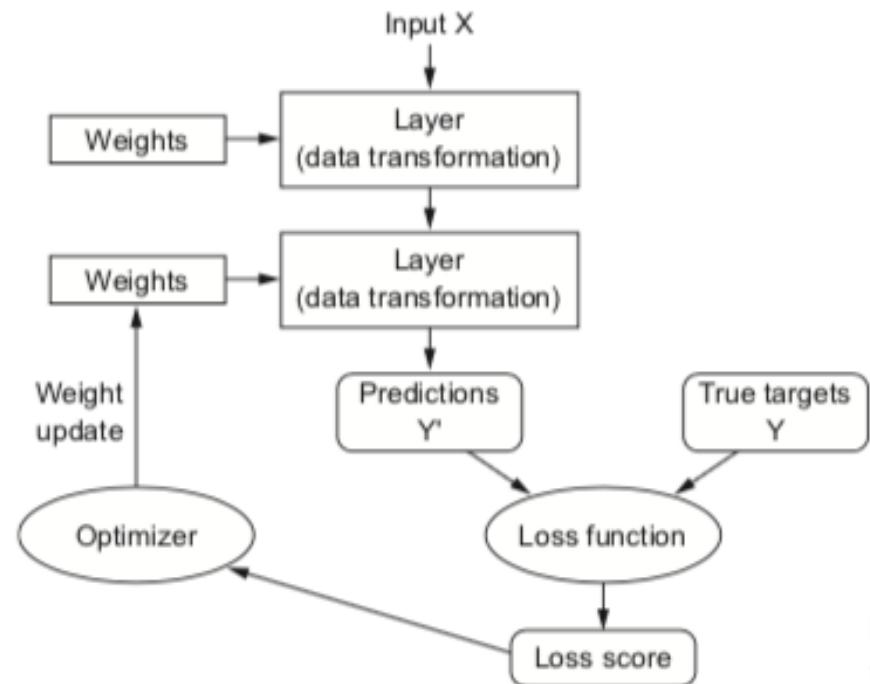
Deep learning is a multistage way to learn data representations



Deep representations learned by a digit-classification model

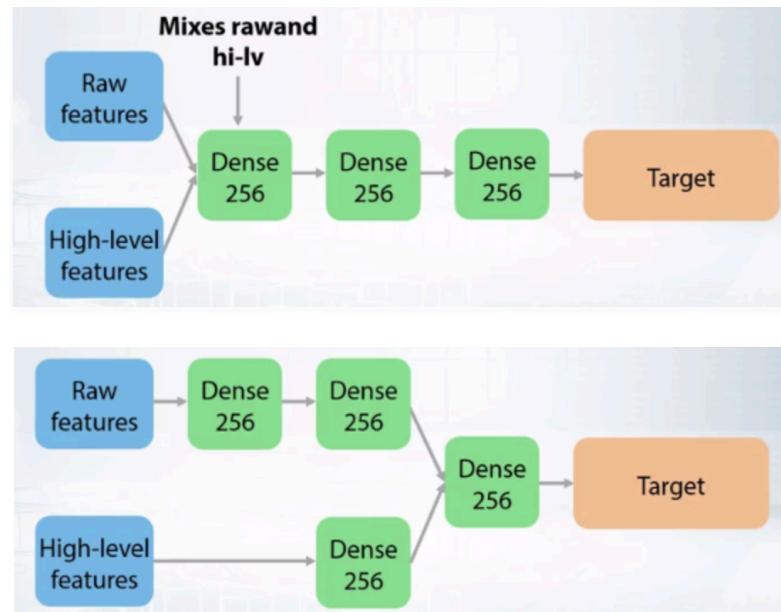
Neural network

- Parameterized by its weights
- A loss function measures the quality of the network's output
- The loss score is used as a feedback signal to adjust the weights



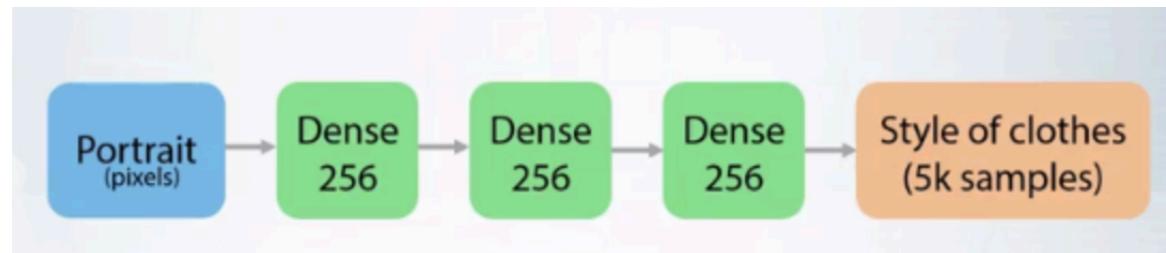
Deep learning as a language

- Provide hints of what to learn by defining structure of the model.
- Think about representations that would transform inputs to outputs.



Deep learning as a language

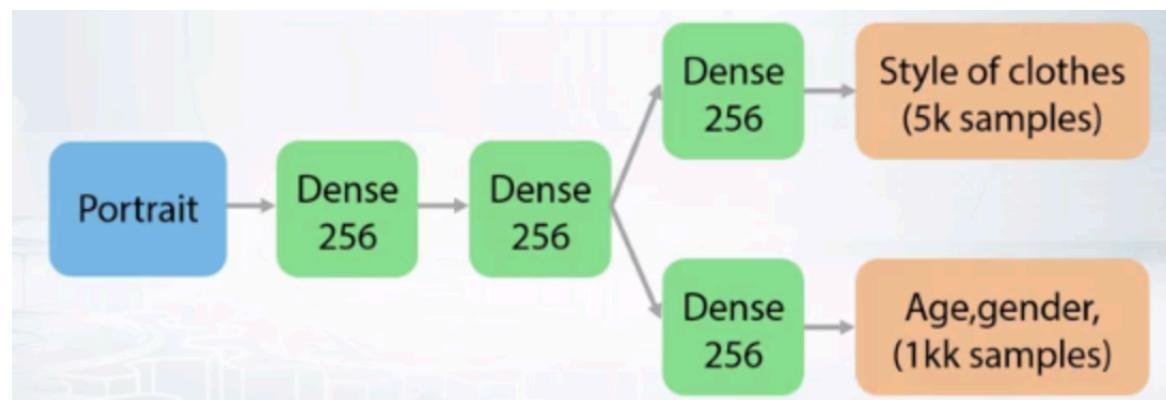
- You have a small dataset



- Consider introducing other problems the network needs to solve

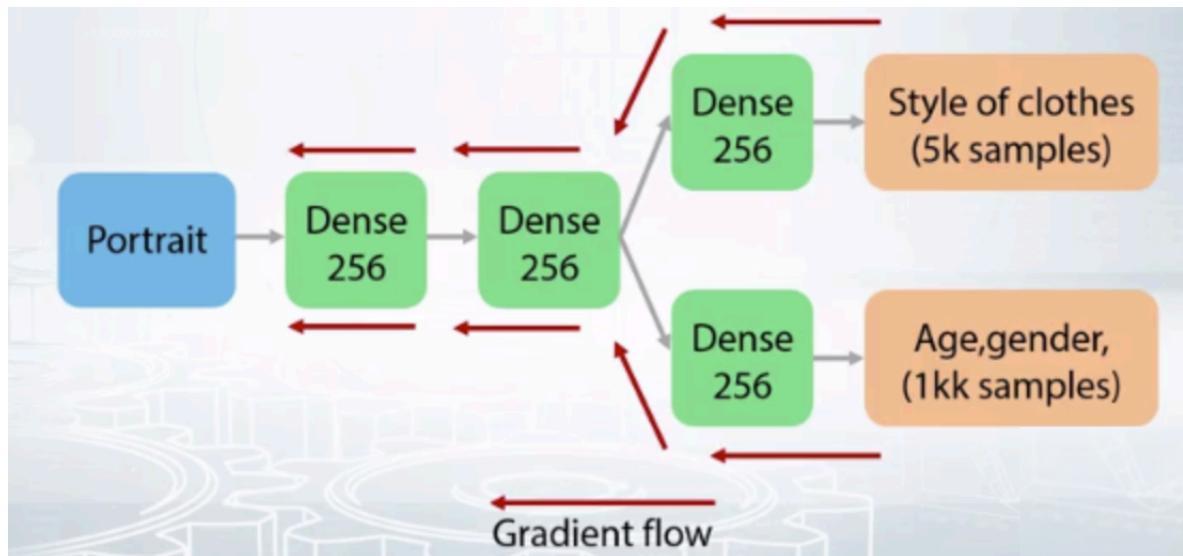
Deep learning as a language

- You have a small dataset
- And a large dataset for similar task



Deep learning as a language

- Learn features for style classification that also help determine age & gender.



Deep learning as a language

For images:

- Need to classify cats and dogs so they are invariant to translation and scale

For text:

- Want to reconstruct the underlying language generation process

Tabular data:

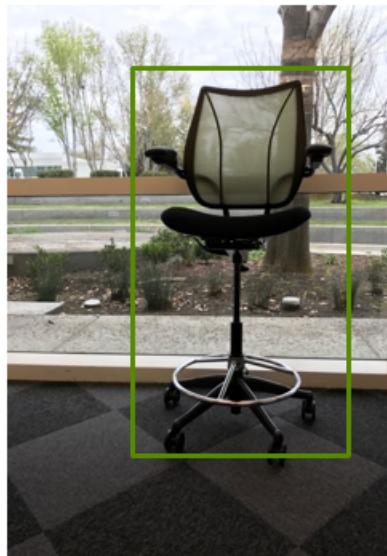
- Want to capture the interactions between variable at different levels of abstraction.

COMPUTER VISION TASKS

Image Classification



Image Classification + Localization



Object Detection

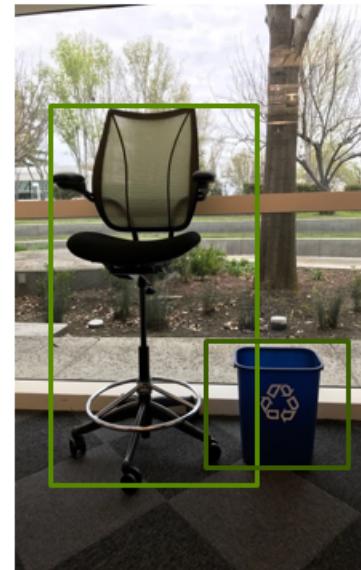


Image Segmentation



(inspired by a slide found in cs231n lecture from Stanford University)

Inputs and Outputs

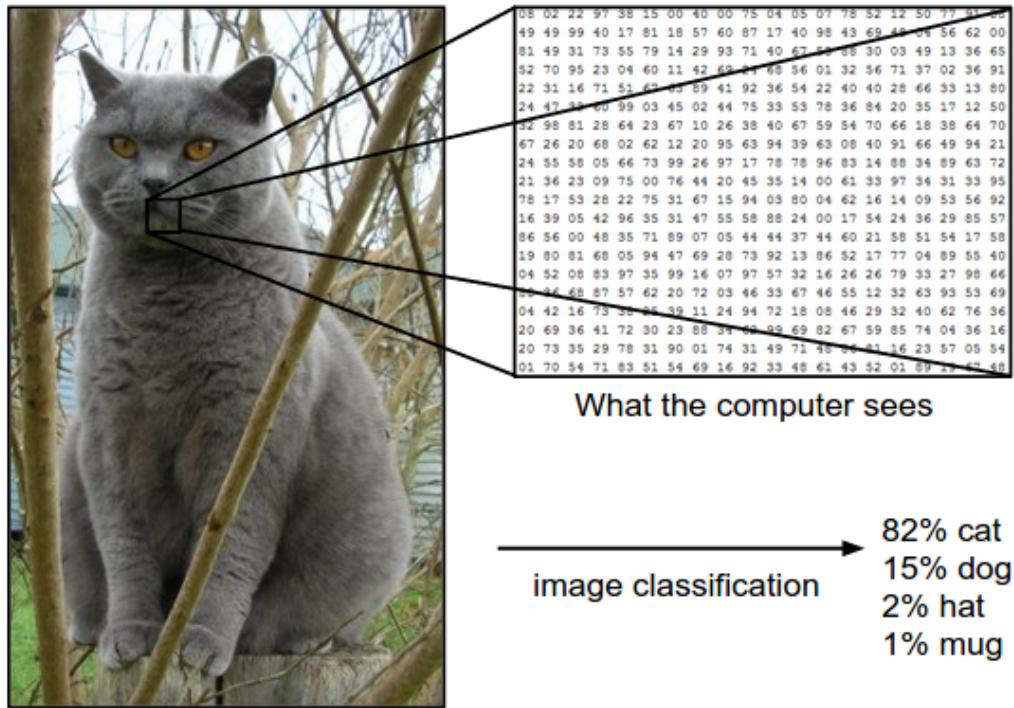


Image from the [Stanford CS231 Course](#)

Digital representation of an image

- Grayscale image is a matrix of pixels (picture elements)
- Dimensions of this matrix are called resolution, e.g., 1024 x 1024
- Each pixel stores its intensity 0 (black) to 255 (white)
- Color images store pixel intensities for 3 channels: red, green, blue.
- Image is just a matrix of numbers.

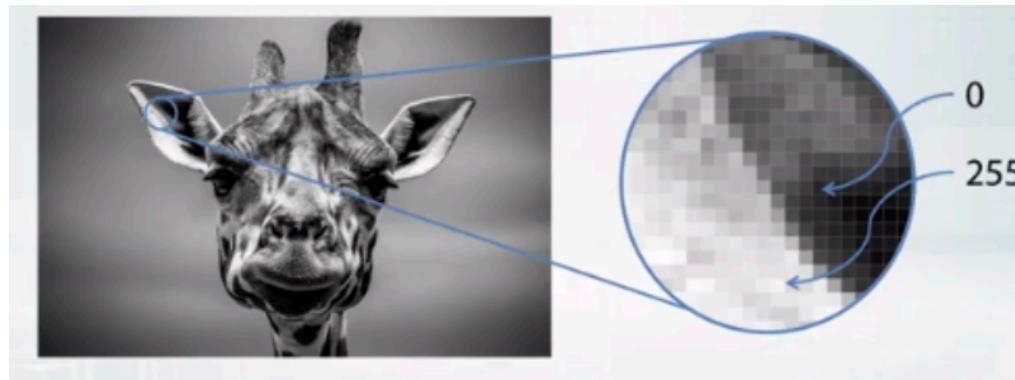
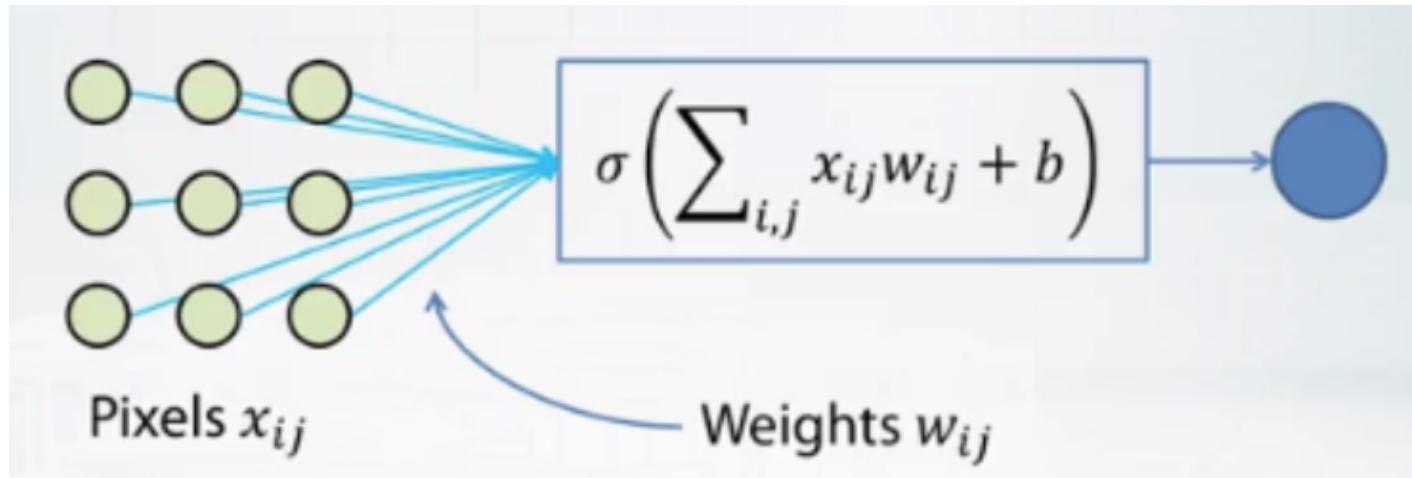


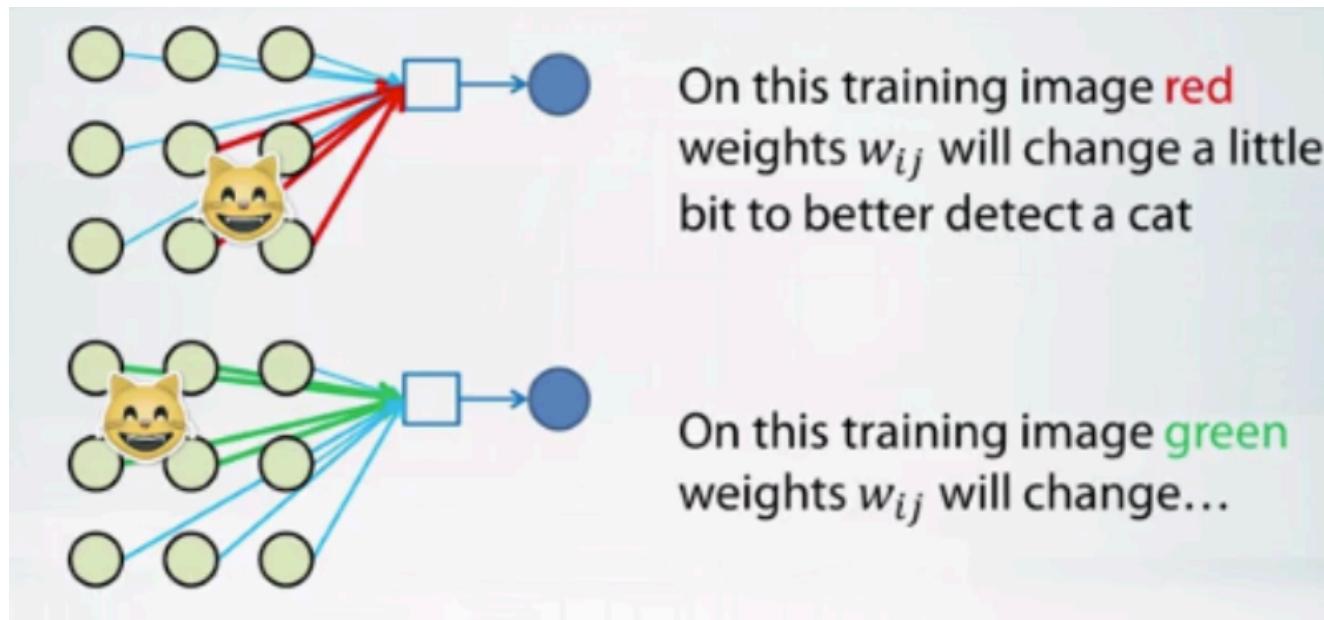
Image as a neural network input

- Normalize input pixels: $x_{norm} = x/255$ or $x_{norm} = x/255 - 0.5$
- *Will multilayer perceptron work?*



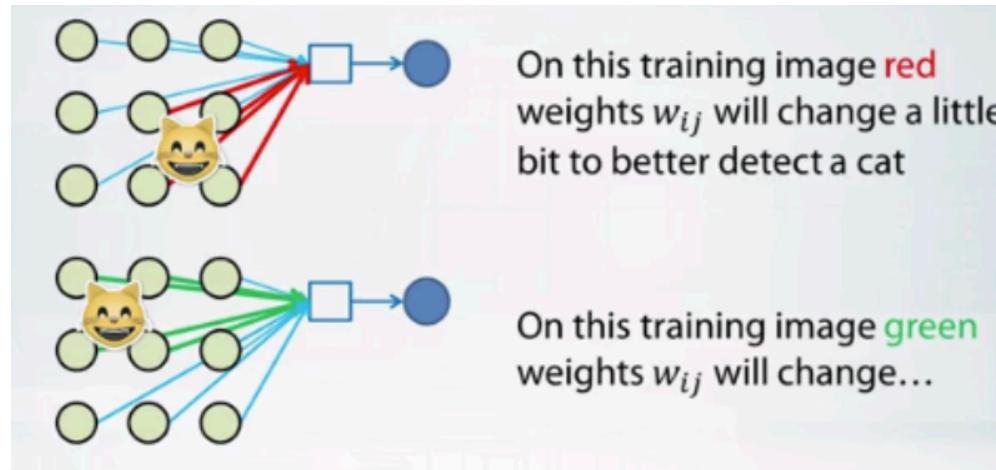
Why not MLP?

- Train a cat detector



Why not MLP?

- Train a cat detector

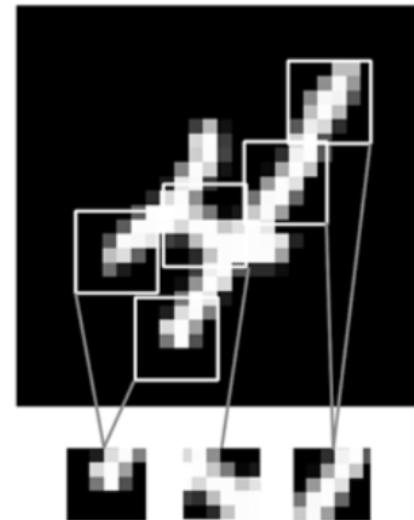


- We learn the same “cat features” in different areas and don’t fully utilize the training set.
- What if the cats in the test set appear in different places?

Convolution Operation

Fundamental difference between densely connected layer and a convolution:

- Dense layers learn global patterns in their input feature space.
- Convolution layers learn local patterns.



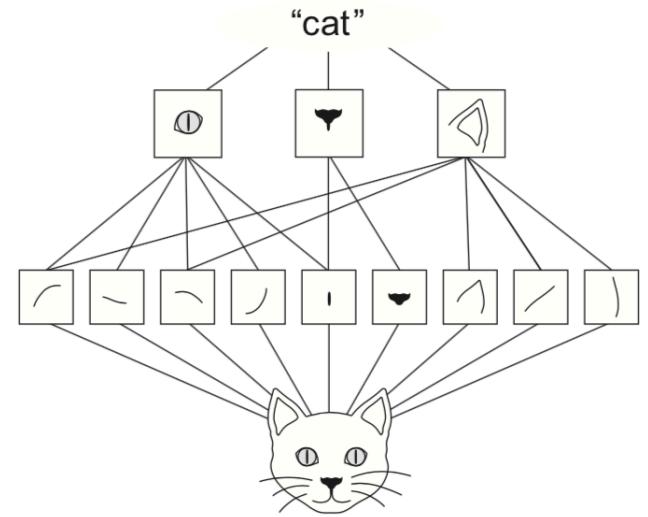
Benefits of convolution layers learning local patterns.

The patterns they learn are translation invariant.

- After learning a certain pattern in the lower-right corner of a picture, a convnet can recognize it anywhere: for example, in the upper-left corner.
- A densely connected network would have to learn the pattern anew if it appeared at a new location.
- This makes convnets data efficient when processing images.

They can learn spatial hierarchies of patterns.

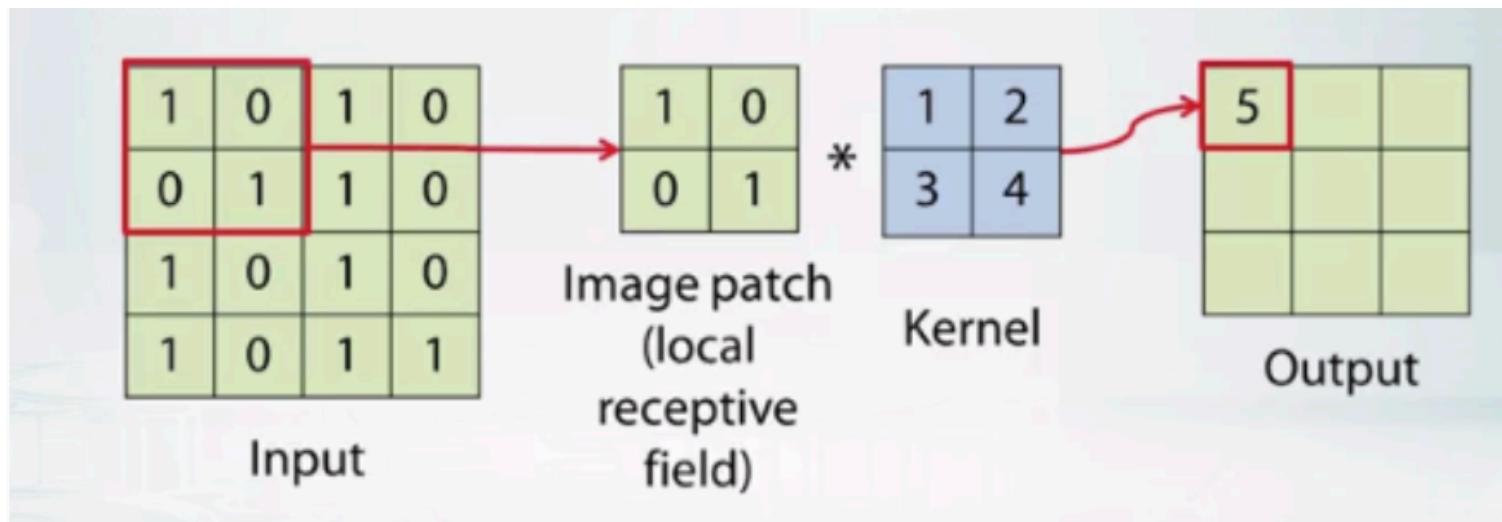
- A first convolution layer will learn small local patterns such as edges.
- A second convolution layer will learn larger patterns made of the features of the first layers, and so on.
- Allows convnets to efficiently learn increasingly complex and abstract visual concepts.



World is made of hierarchical representations that are spatially invariant.

Convolutions

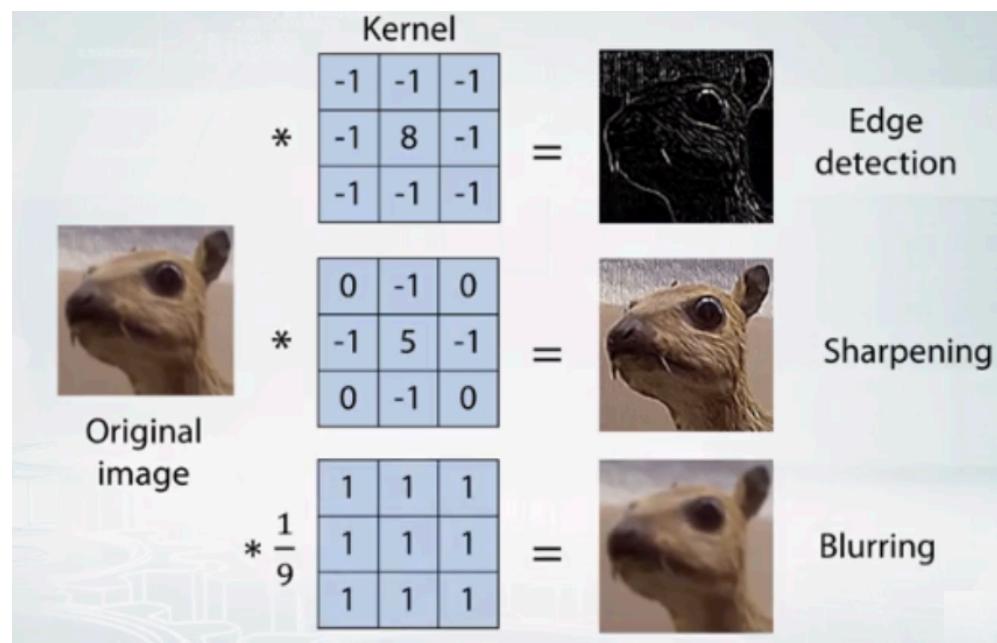
- Convolution is a dot product of a kernel (or filter) and a patch of an image (local receptive field) of the same size.



Dot product = sum of element wise product (scalar)

Convolutions have been used for a long time

- Convolution is a dot product of a kernel (or filter) and a patch of an image (local receptive field) of the same size.



Convolutions

What is the convolution output for the following image using the given 2x2 filter?

- Stride = 1 (sliding window size)
- No padding (add zero – valued edges)

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

Input Kernel Output



Convolutions

What is the convolution output for the following image using the given 2x2 filter?

- Stride = 1 (sliding window size)
- No padding (add zero – valued edges)

0	0	0	0
0	0	0	0
0	0	0	1
0	0	1	0

Input

*

1	0
0	1

Kernel

=

0	0	0
0	0	1
0	1	0

Output

Convolution is similar to correlation

The diagram illustrates the convolution process using two examples. Both examples show an input matrix of size 4x4, a kernel of size 2x2, and the resulting output matrix.

Example 1:

0	0	0	0
0	0	0	0
0	0	1	0
0	0	0	1

Input

Kernel:

1	0
0	1

Output:

0	0	0
0	1	0
0	0	2

Example 2:

0	0	0	0
0	0	0	0
0	0	0	1
0	0	1	0

Input

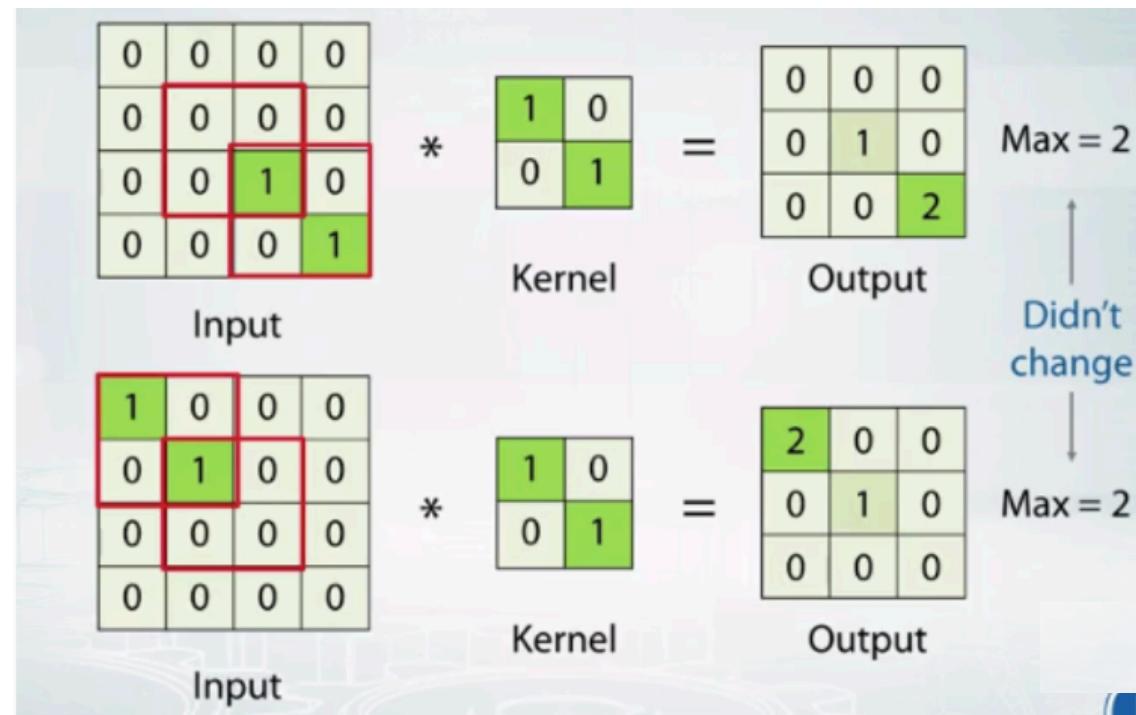
Kernel:

1	0
0	1

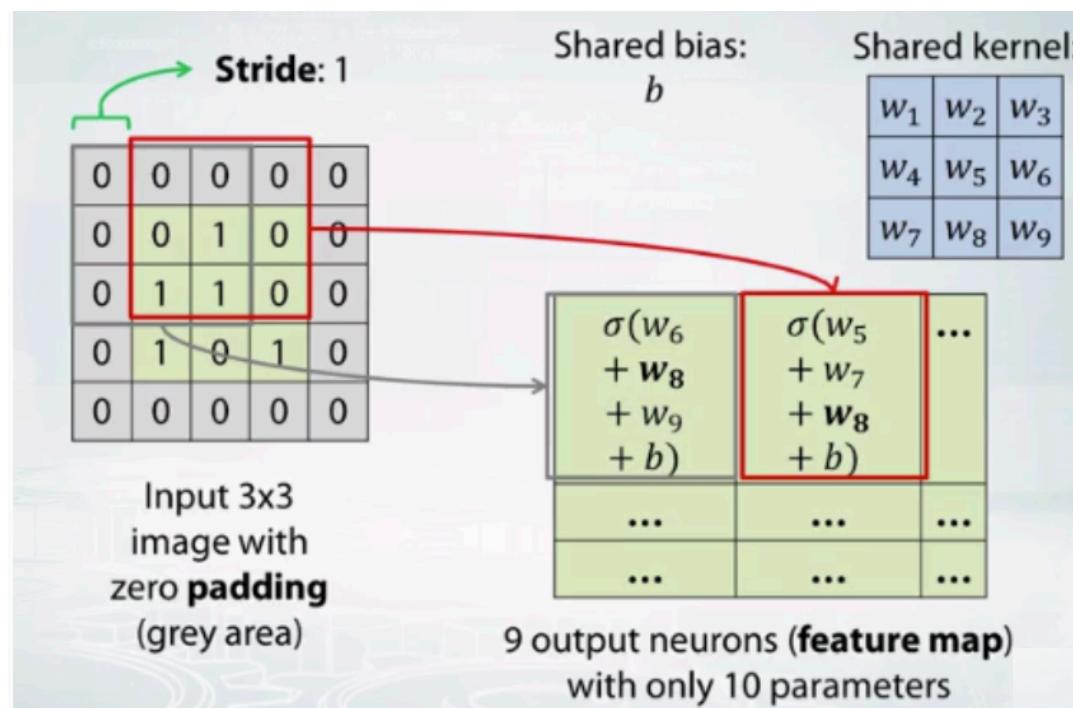
Output:

0	0	0
0	0	1
0	1	0

Convolution is translation invariant

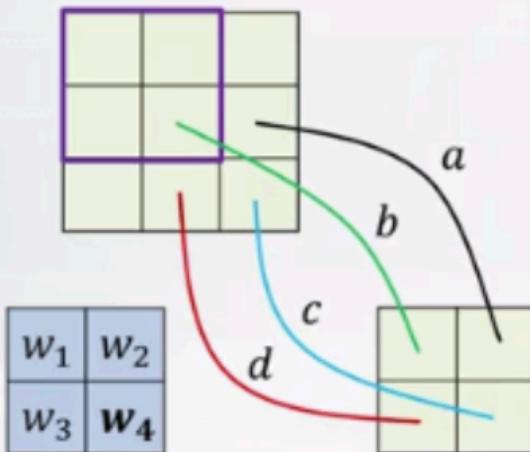


Convolution layer in neural network



Backpropagation for CNN

Gradients are first calculated as if the kernel weights were not shared:



$$a = a - \gamma \frac{\partial L}{\partial a} \quad b = b - \gamma \frac{\partial L}{\partial b}$$

$$c = c - \gamma \frac{\partial L}{\partial c} \quad d = d - \gamma \frac{\partial L}{\partial d}$$

$$w_4 = w_4 - \gamma \left(\frac{\partial L}{\partial a} + \frac{\partial L}{\partial b} + \frac{\partial L}{\partial c} + \frac{\partial L}{\partial d} \right)$$

Gradients of the same shared weight are summed up!

Question

- Suppose we have a 300×300 input and a 300×300 output.
- How many parameters (weights) do you train with a convolutional layer (filter) 5×5 ?

Question

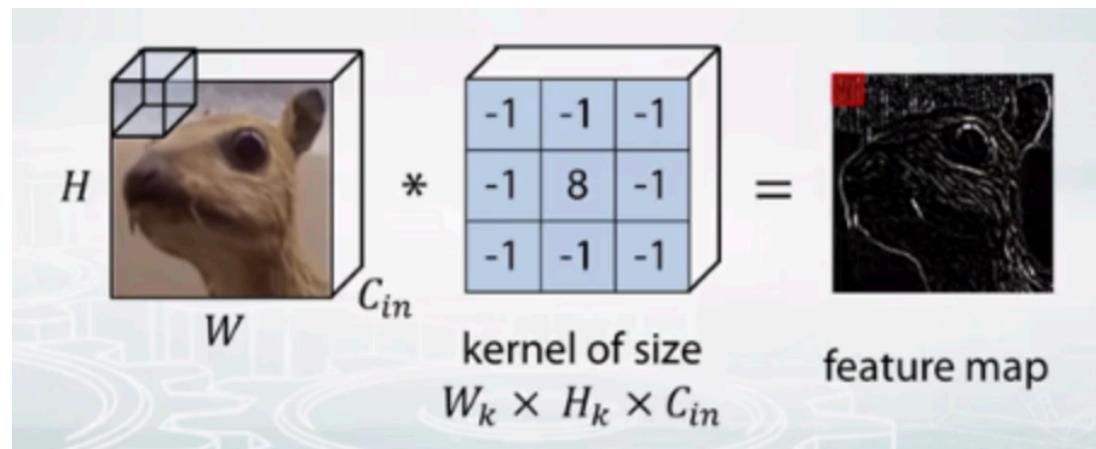
- Suppose we have a 300×300 input and a 300×300 output.
- How many parameters (weights) do you train with a convolutional layer (filter) 5×5 ?
- $26 = (5 \times 5 = 25 \text{ kernel parameters}) + 1 \text{ bias term}$

Convolutional vs full connected layer

- In a conv layer the same kernel is used for every output neuron, this way we share parameters of the network and train a better model - provides translational invariance.
- 300×300 input, 300×300 output – 5×5 kernel = 26 parameters in conv layer.
- **8.1×10^9** parameters in fully connected network where each output is a perceptron.
- Conv layer can be viewed as a special case of a fully connected layer when all the weights outside the local **receptive field** of each neuron equal **0** and kernel parameters are shared between neurons.

Color image input

- Color image input: $W \times H \times C_{in}$ tensor (multidimensional array)
- W – image width
- H – image height
- C_{in} – is a number of input channels (e.g., 3 RGB channels)

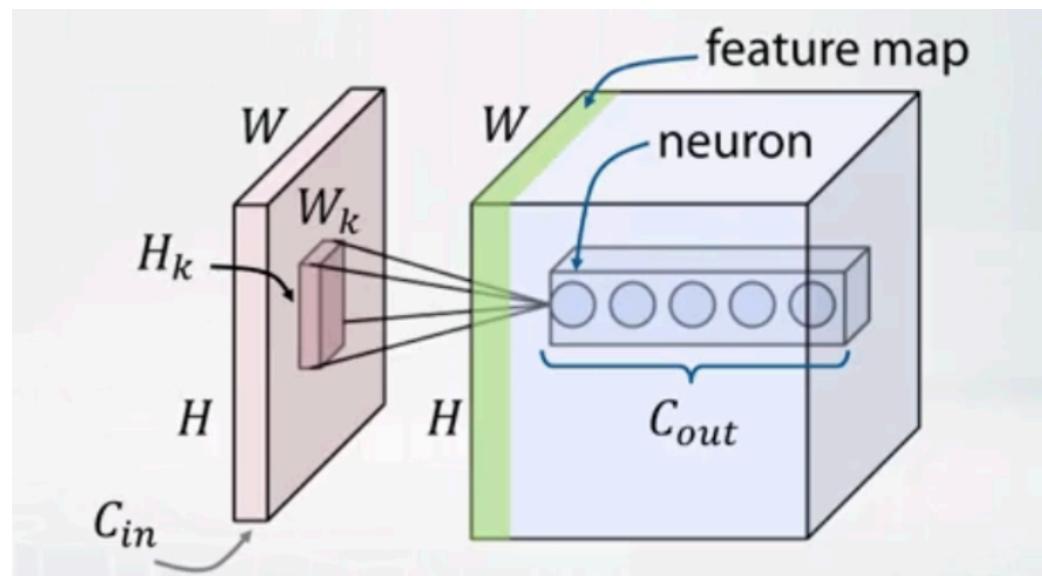


One kernel is not enough

- Every filter in your convolutional layer extracts a specific feature from the input.
- One filter could be sensitive to horizontal edges while another is sensitive to vertical edges.
- A third filter may be sensitive to a triangular shape.
- You want the feature maps to be as different from each other as possible to avoid redundancy.
- Avoiding redundancy improves the network's capacity to as many variations in the data as possible.
- Random initialization prevents learning duplicate filters.

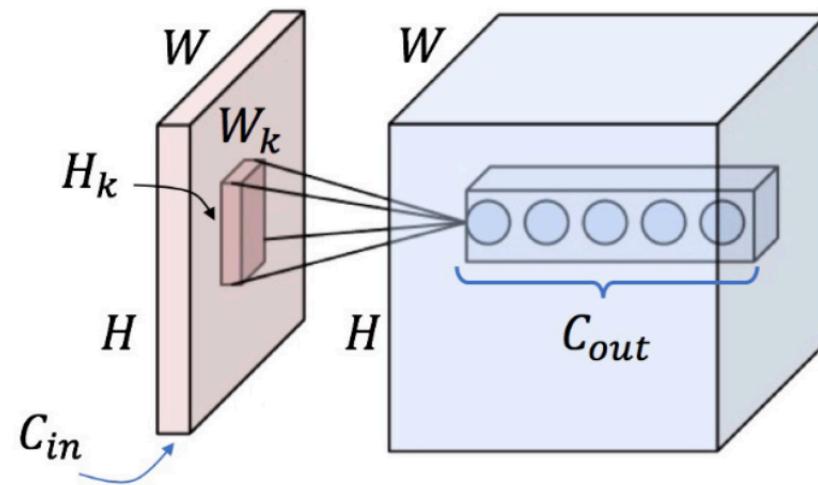
One kernel is not enough

- Need to train C_{out} kernels of size $W_k \times H_k \times C_{in}$
- Having a stride of 1 and enough zero padding we can have $W \times H \times C_{out}$ output neurons.



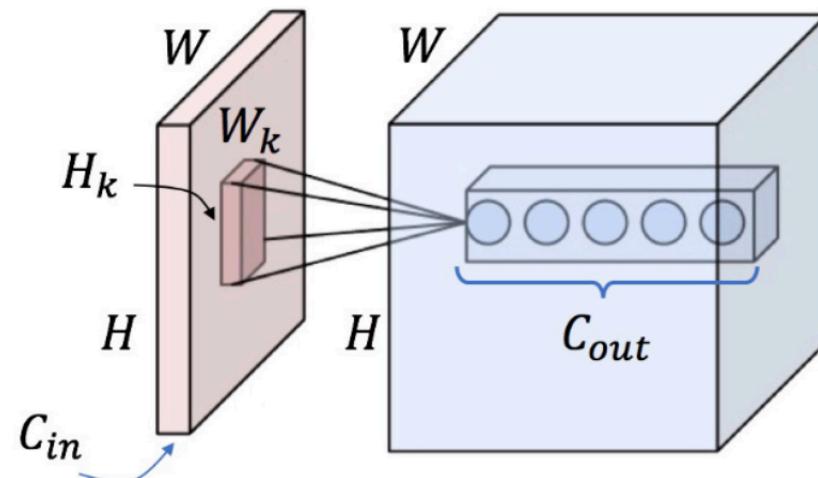
Question

- How many parameters do we need to train this convolutional layer with C_{out} output channels?



Question

- How many parameters do we need to train this convolutional layer with C_{out} output channels?



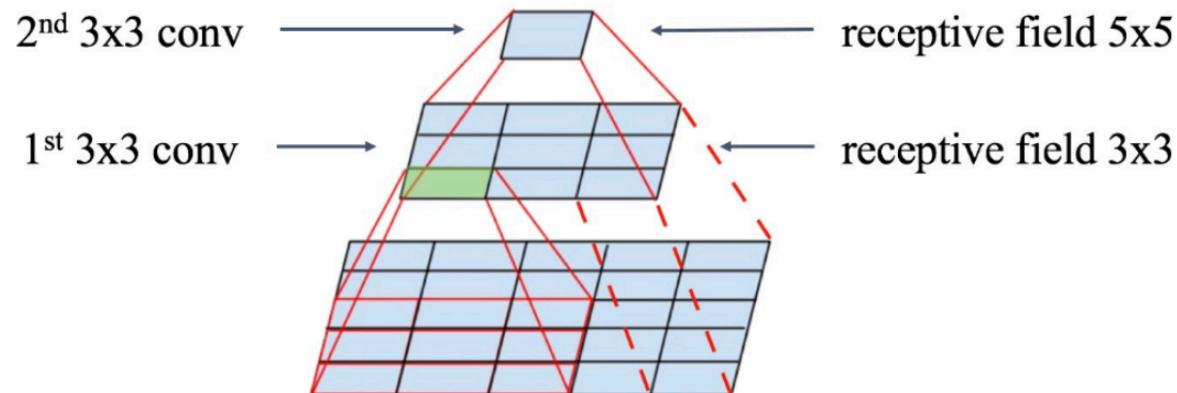
$$(W_k \cdot H_k \cdot C_{in} + 1) \cdot C_{out}$$

Selecting the number of feature maps (kernels)

- 6 feature maps works well for C1 on MNIST.
- This is a result of trying out other numbers of filters.
- Increasing the number of filters results in higher computational overhead and possibly overfitting.
- Another intuition for 6 is that there's not that much variation in pixels, you'll eventually extract more complex features in subsequent layers.

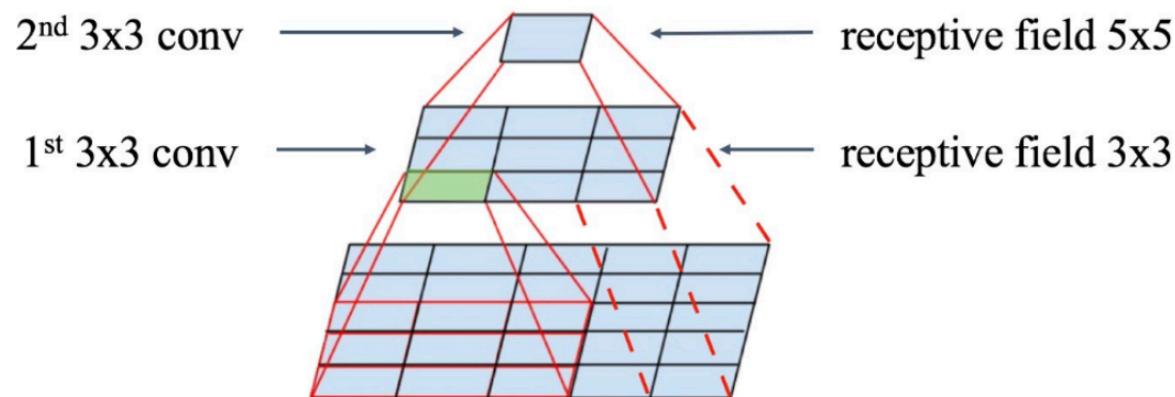
One conv layer is not enough

- Neurons of the 1st conv layer look at patches of the image of size 3 x 3.
- What if the object of interest is larger than that?
- Need a 2nd conv layer on top of the 1st, etc.



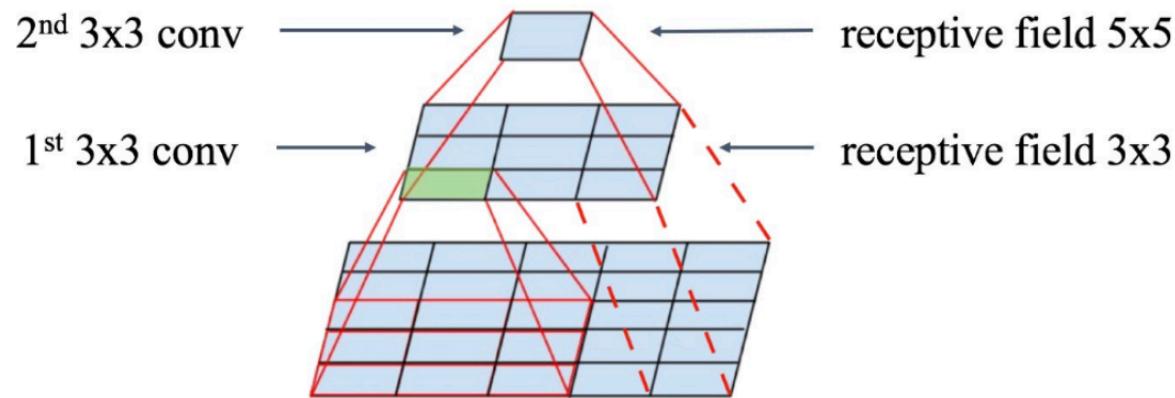
One conv layer is not enough

- What is the receptive field for the 4th conv layer (assume all layers have 3 x 3 kernels and stride 1?)



One conv layer is not enough

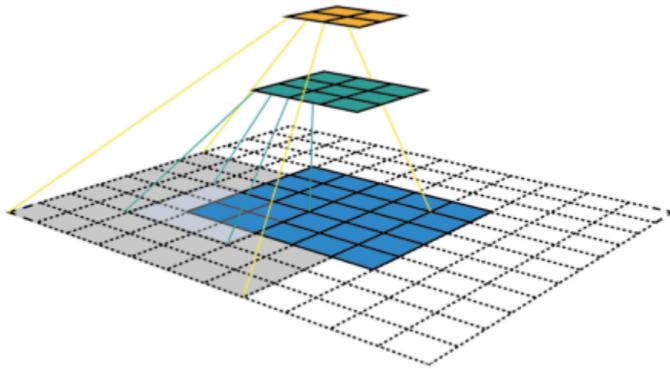
- What is the receptive field for the 4th conv layer (assume all layers have 3 x 3 kernels and stride 1?)



9 x 9 – add 2 to receptive field with each layer.

Receptive field after N convolutional layers

- More generally, if we stack N convolutional layers with kernel size 3×3 , the receptive field on the N^{th} layer = $(2N + 1) * (2N + 1)$.
- You need to stack a lot of convolutional layers to identify objects as big as the input image.
- To cover the receptive field of a 300x300 input image, need ~ 75 layers.



Understanding the Effective Receptive Field in Deep Convolutional Neural Networks

Wenjie Luo, Yujia Li, Raquel Urtasun, Richard Zemel

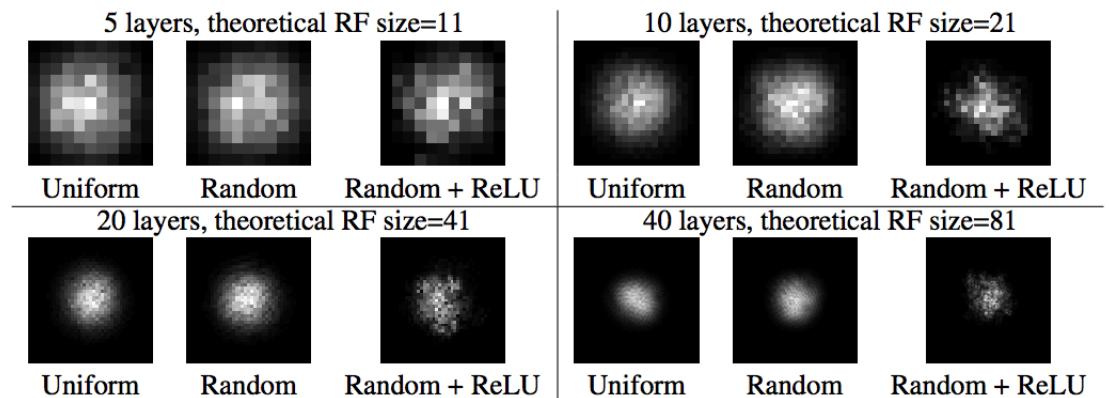
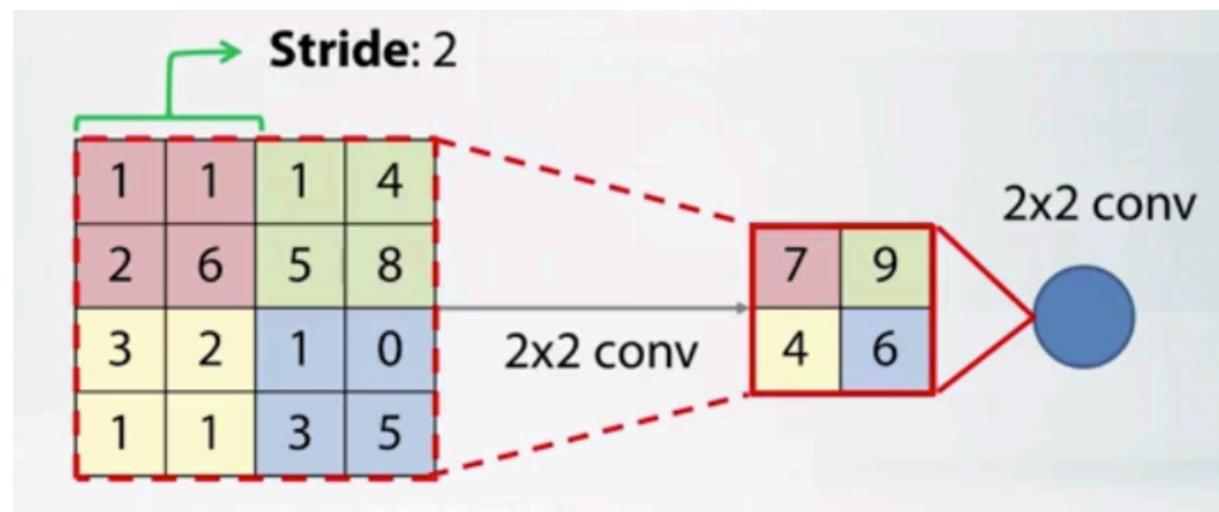


Figure 1: Comparing the effect of number of layers, random weight initialization and nonlinear activation on the ERF. Kernel size is fixed at 3×3 for all the networks here. Uniform: convolutional kernel weights are all ones, no nonlinearity; Random: random kernel weights, no nonlinearity; Random + ReLU: random kernel weights, ReLU nonlinearity.

Need to grow receptive field faster

- We can increase the stride in our conv layer to reduce the output dimensions.
- Further convolutions will effectively double their receptive field.



Need to maintain translation invariance

The diagram illustrates two examples of convolution operations to demonstrate the need for translation invariance.

Example 1:

0	0	0	0
0	0	0	0
0	0	1	0
0	0	0	1

Input

Kernel:

Max = 2
↓
Didn't change
↓
Max = 2

Output:

0	0	0
0	1	0
0	0	2

Output

Example 2:

1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0

Input

Kernel:

0	1
---	---

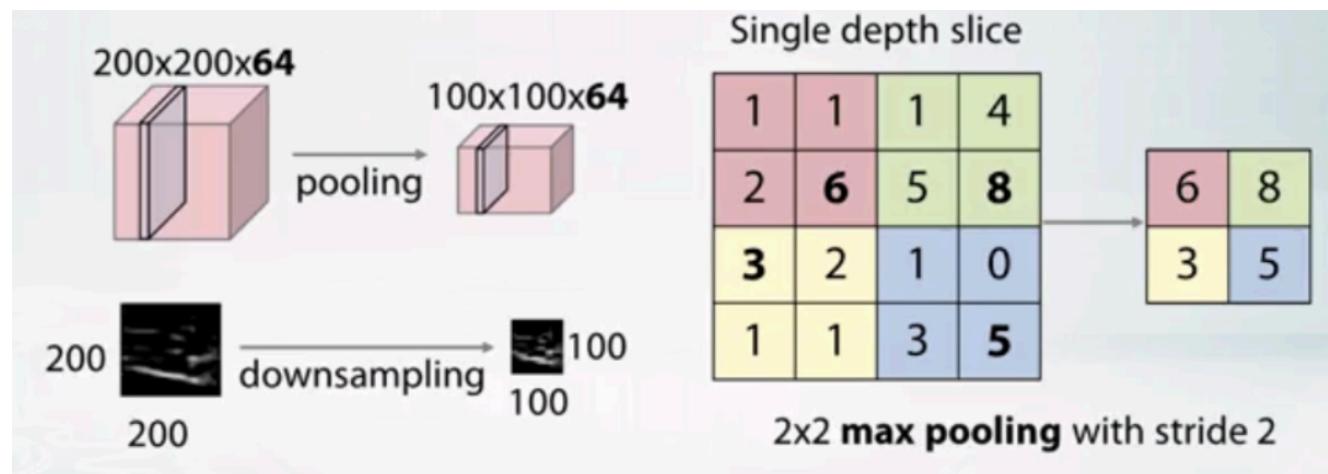
Output:

2	0	0
0	1	0
0	0	0

Output

Pooling layer

- Works like a convolutional layer but doesn't have a kernel.
- Calculates max or average of input patch values.



Backpropagation for max pooling layer

- Works like a convolutional layer but doesn't have a kernel.
- Calculates max or average of input patch values.
- For the max path neuron you have a gradient of 1.

Strictly speaking: maximum is not a differentiable function!

6	8
3	5

Maximum = 8

7	8
3	5

Maximum = 8

There is no gradient with respect to non maximum patch neurons, since changing them slightly does not affect the output.

6	8
3	5

Maximum = 8

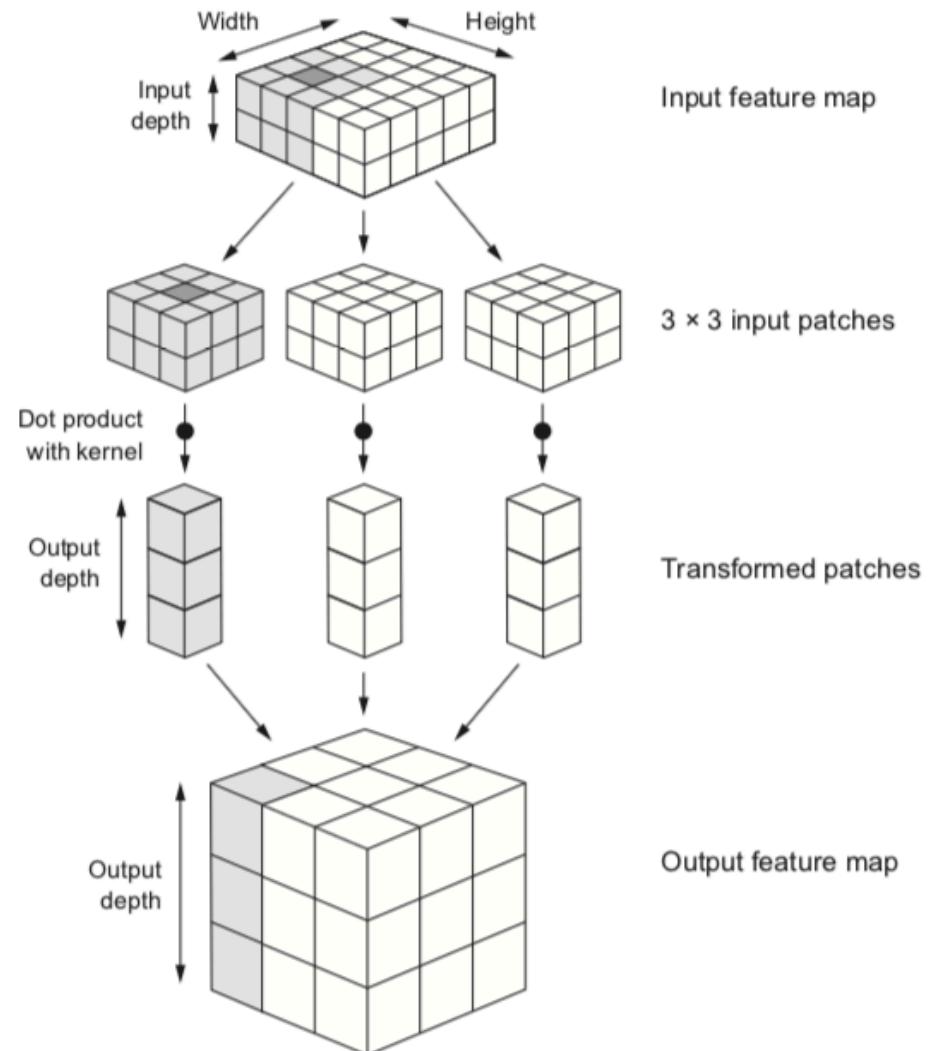
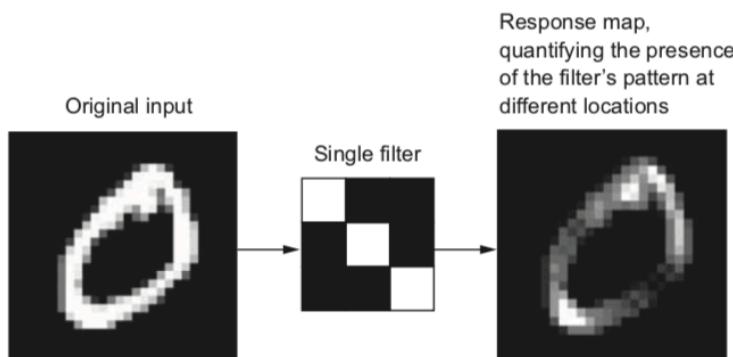
7	9
3	5

Maximum = 9

Conv Net tensor flow

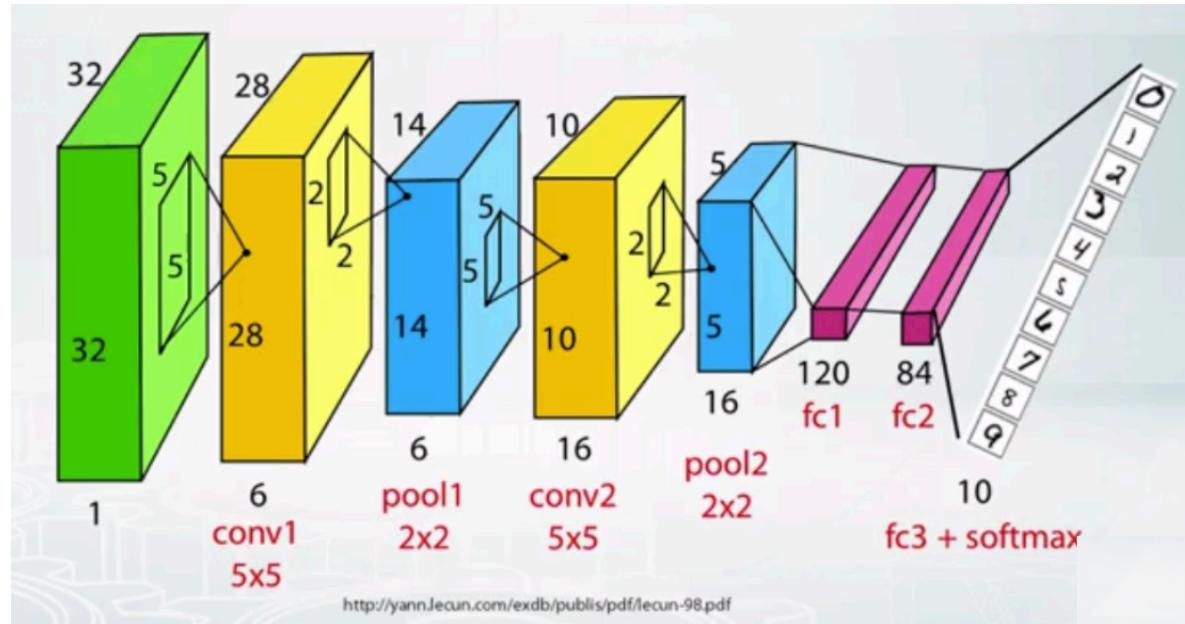
Convolutions are defined by two key parameters:

- Size of the patches extracted from the input:
- Depth of the output feature map.



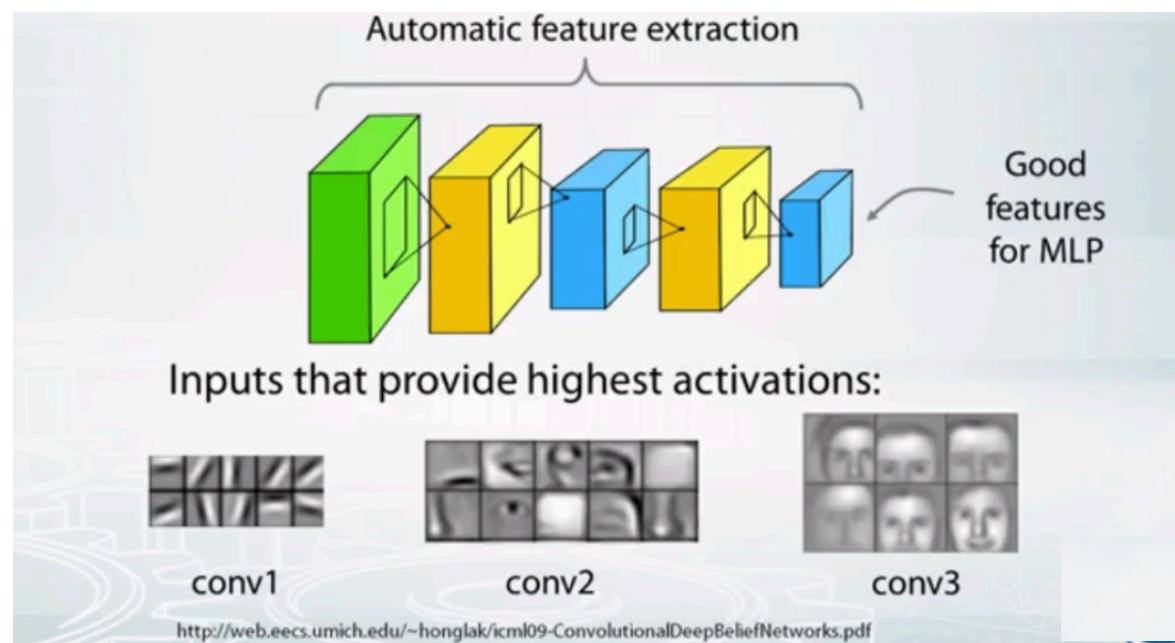
Putting together a simple CNN

- LeNet for handwritten digits recognition on MNIST dataset.



Learning Deep Representations

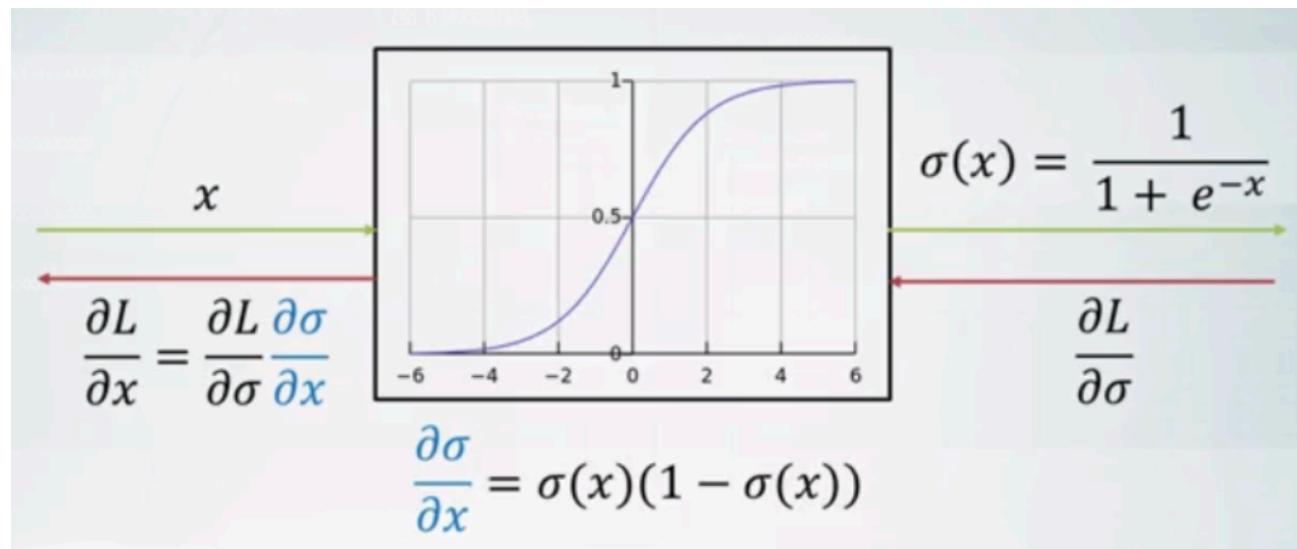
- Neurons of deep convolutional layers learn complex representations that can be used as features for classification with MLP.



Activations - Sigmoid

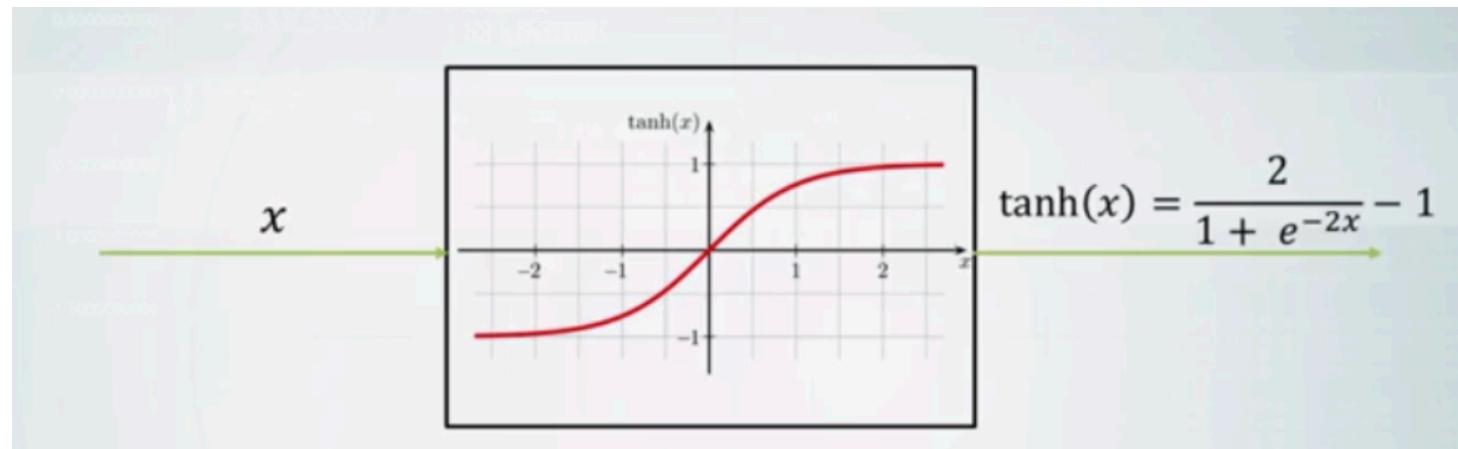
<https://keras.io/layers/advanced-activations/>

- Sigmoid neurons can saturate and lead to vanishing gradients.
- Not zero centered
- e^x is computationally expensive.



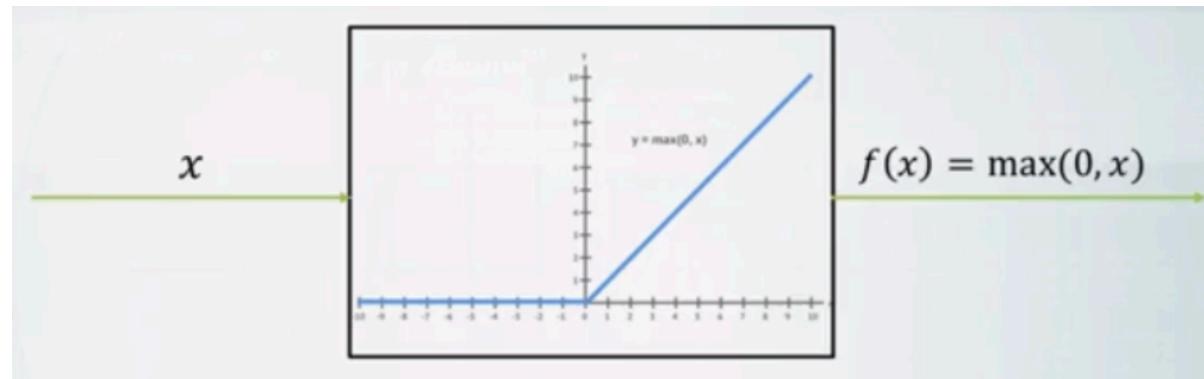
Activations - Tanh

- Zero centered
- Otherwise pretty much like sigmoid.



Activations - ReLU

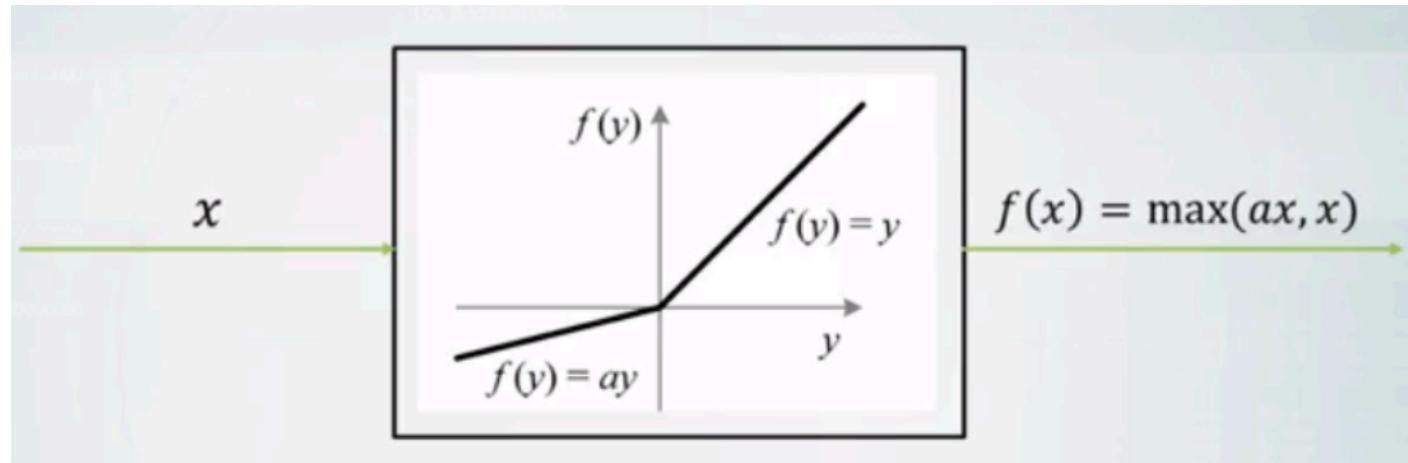
- Fast to compute
- Gradients do not vanish for $x > 0$
- Provides faster convergence in practice
- Not zero centered
- Can die: if not activated, never updates..



$$f(x) = \alpha * x \text{ for } x < 0, f(x) = x \text{ for } x \geq 0$$

Activations – Leaky ReLU

- Will not die if $a \neq 1$



$$f(x) = \max_{\text{value}} \text{ for } x \geq \text{max_value}, f(x) = x \text{ for threshold} \leq x < \text{max_value}, f(x) = \text{alpha} * (x - \text{threshold})$$

Important practices

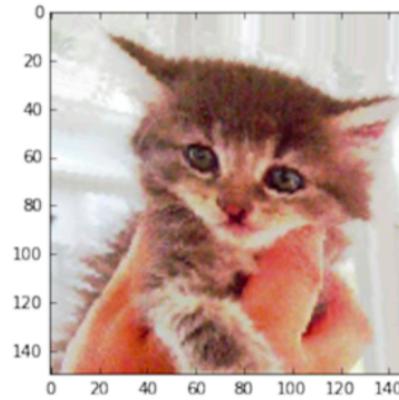
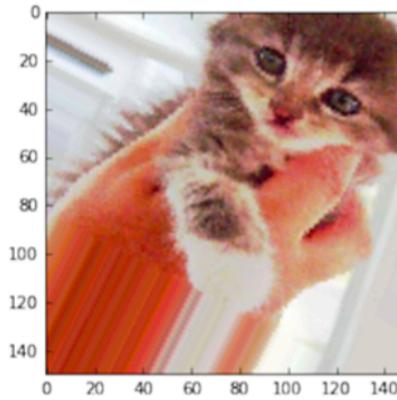
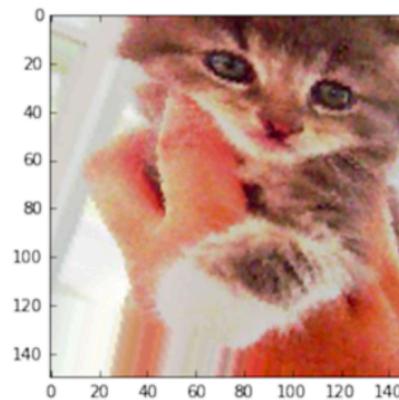
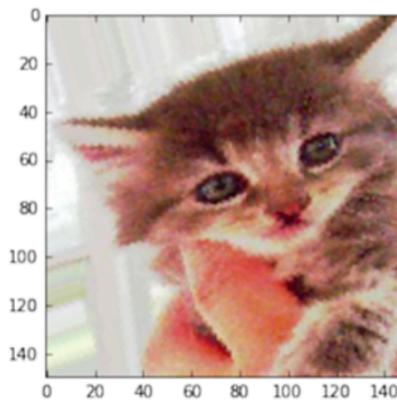
- Random initialization centered at zero – Xavier
- Batch training
- Batch normalization
- Dropout
- Data augmentation
- Transfer learning

Important takeaways

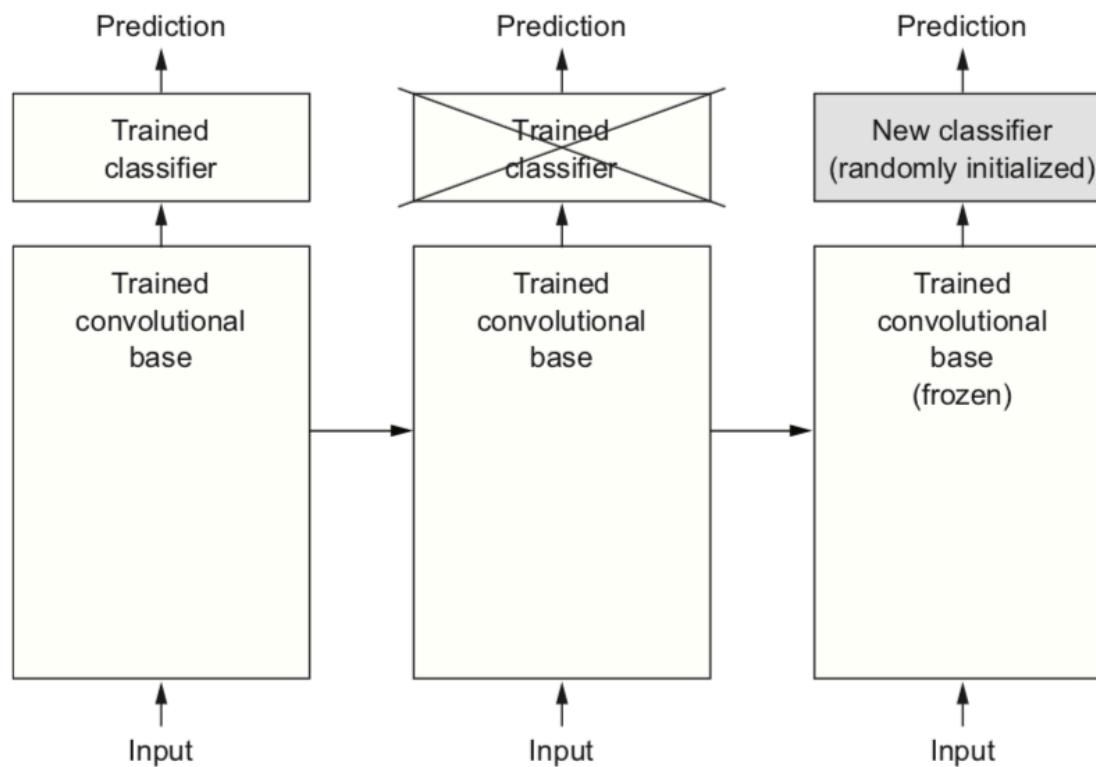
- Use ReLU or leaky ReLU activation
- Use He et al., Xavier initialization or similar
- Add batch normalization and/or dropout
- Augment training data set
- Use transfer learning
- Transfer learning + fine tuning

Data augmentation

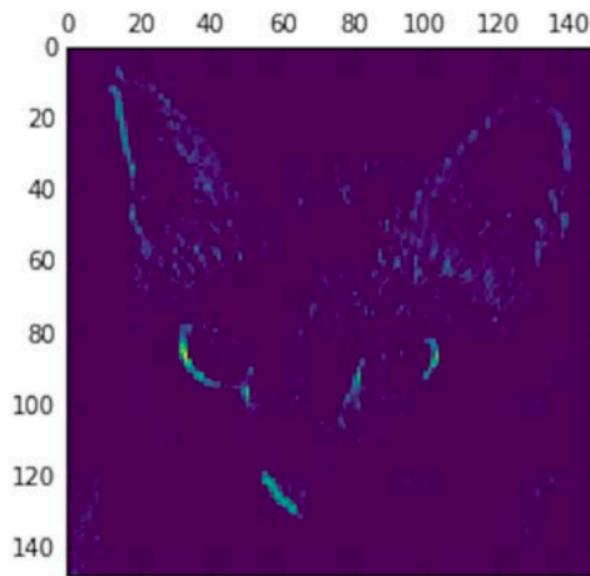
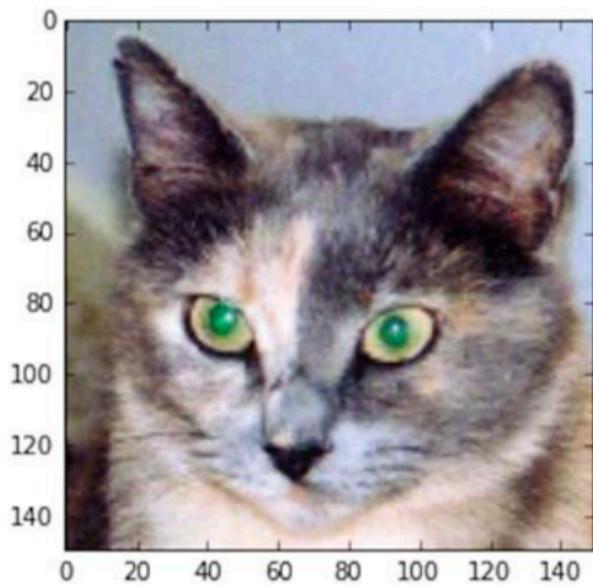
- Scale
- Translation
- Flip
- Small rotations



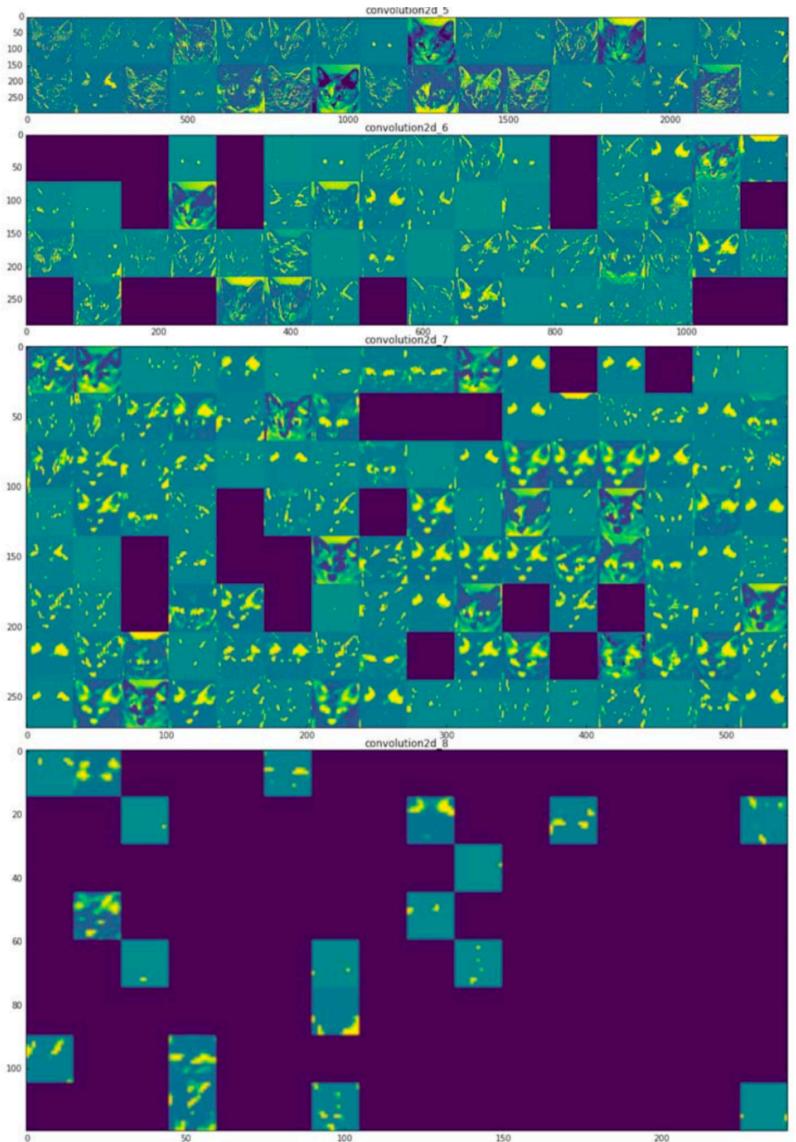
Transfer learning



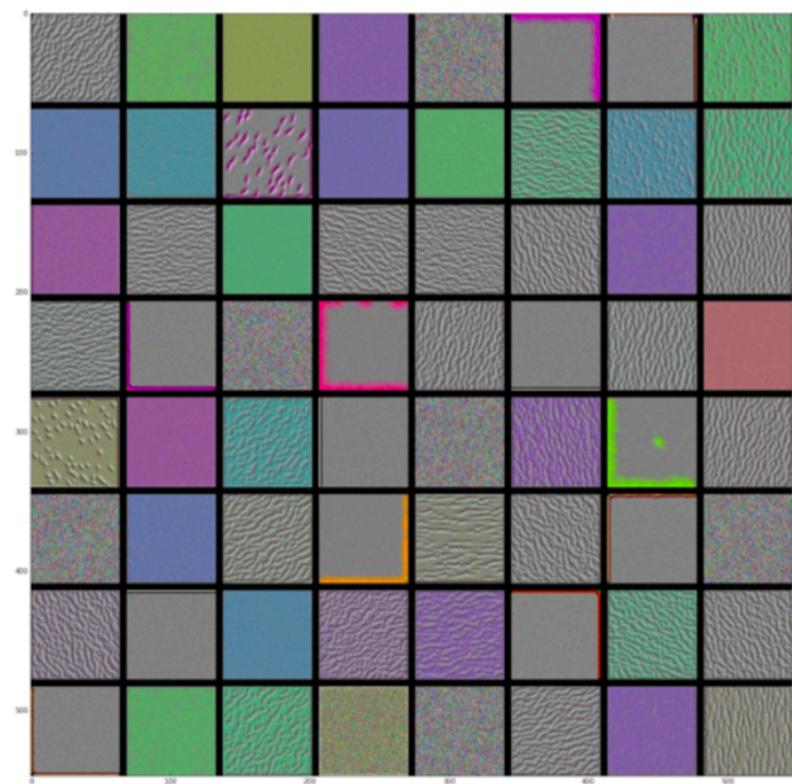
Visualizing activations



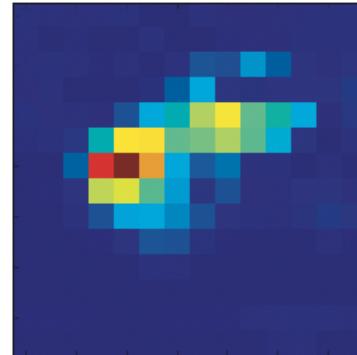
Visualizing every channel of every layer



Visualizing filter patterns



Heat maps

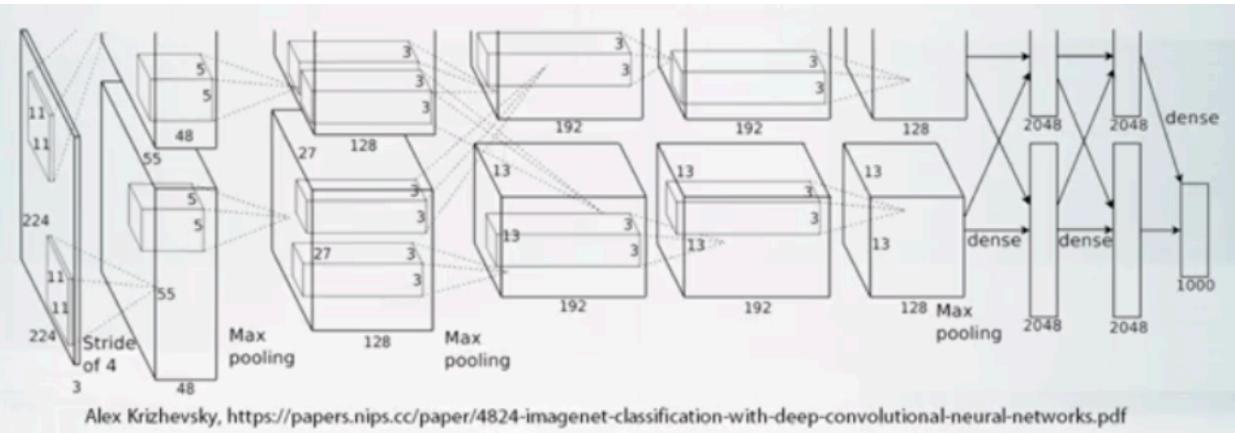


Modern CNN Architectures



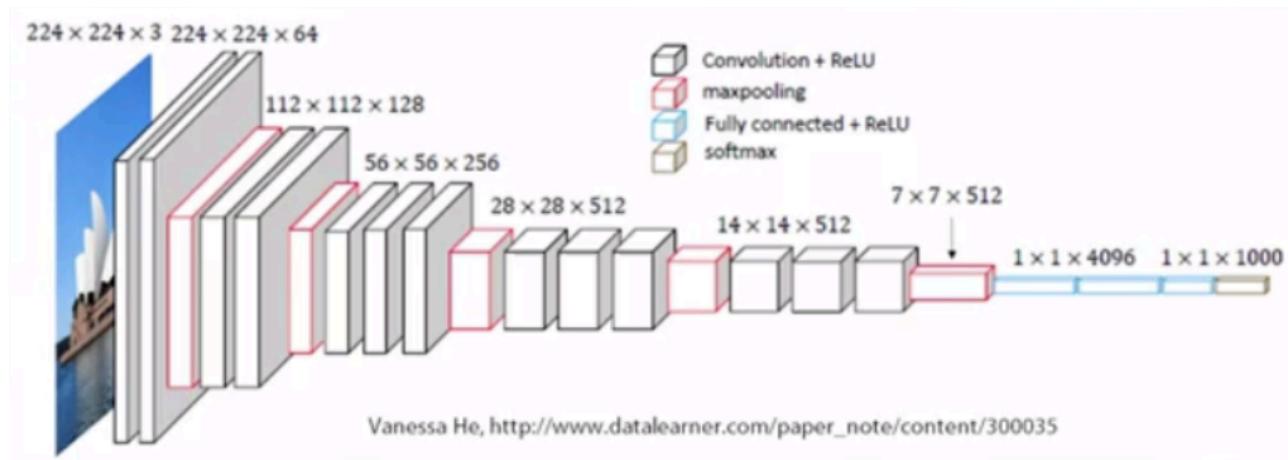
AlexNet (2012)

- First deep conv net for ImageNet
- Significantly reduced top 5 error from 26% to 15%
- 11x11, 5x5, 3x3 convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum
- 60M parameters
- Trained on 2 GPUs for 6 days



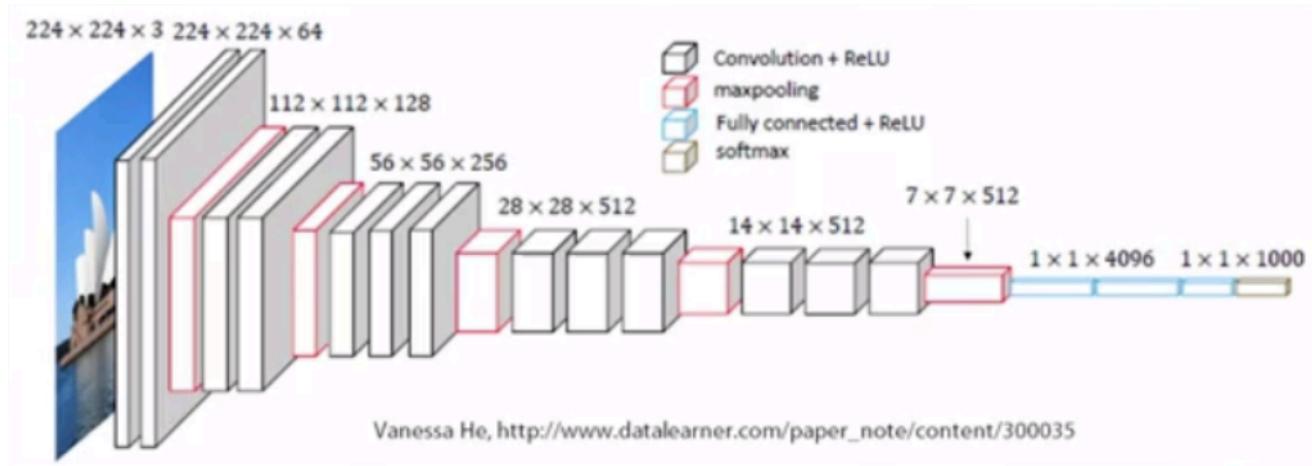
VGG (2015)

- Similar to AlexNet, only 3x3 convolutions, but lots of filters
- Significantly reduced top 5 error 8% (single model)
- Training similar to AlexNet with additional multi-scale cropping.
- 138M parameters
- Trained on 4 GPUs for 2-3 weeks



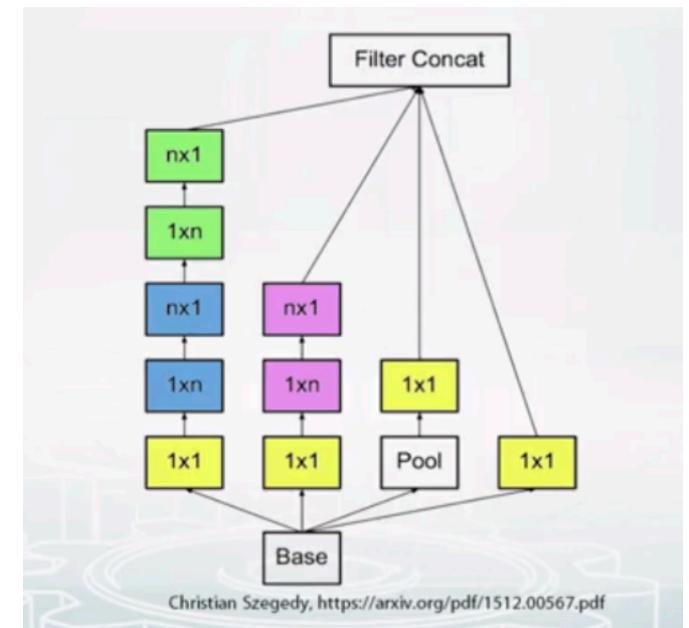
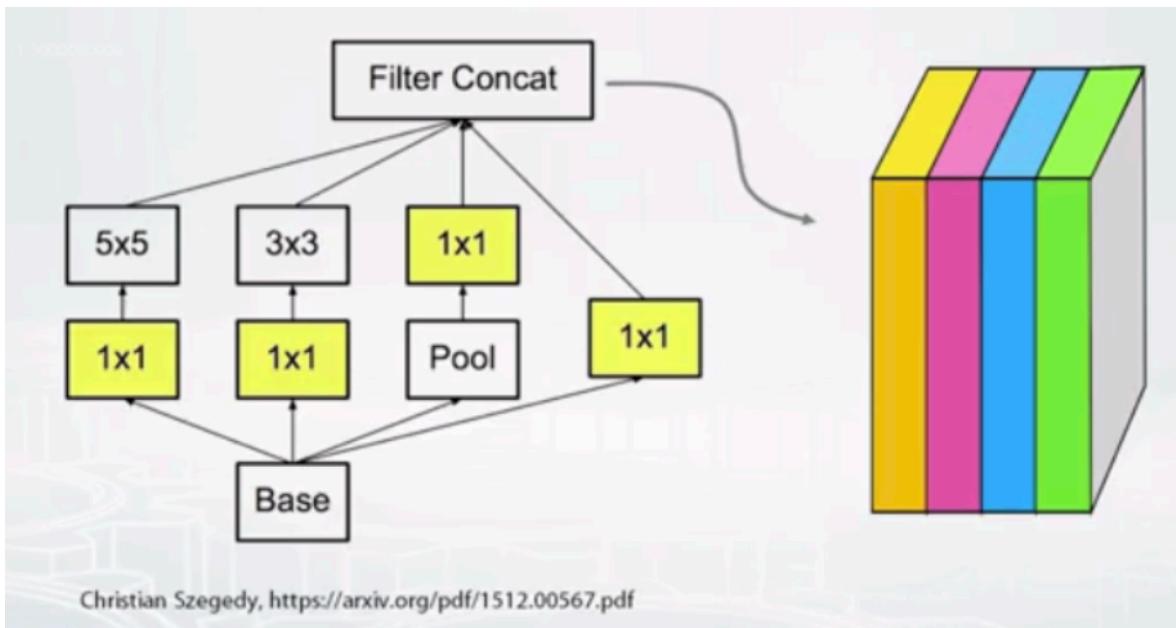
Inception V3 (2015)

- Uses inception block
- ImageNet top 5 error 5.6% (single model), 3.6% ensemble
- Batch normalization, image distortions, RMSProp
- 25M parameters
- Trained on 8 GPUs for 2 weeks



Basic Inception Block

- All operations inside a block use stride 1 and enough padding to output the same spatial dimensions ($W \times H$) of feature map.
- 4 different feature maps are concatenated on depth at the end.



ResNet (2015)

- Introduces residual connections
- ImageNet top 5 error: 4.5% (single model), 3.5% (ensemble)
- 152 layers, few 7x7 conv layers, the rest are 3x3, batch normalization, max and average pooling
- 60M parameters
- Trains on 8 GPUs for 2-3 weeks

