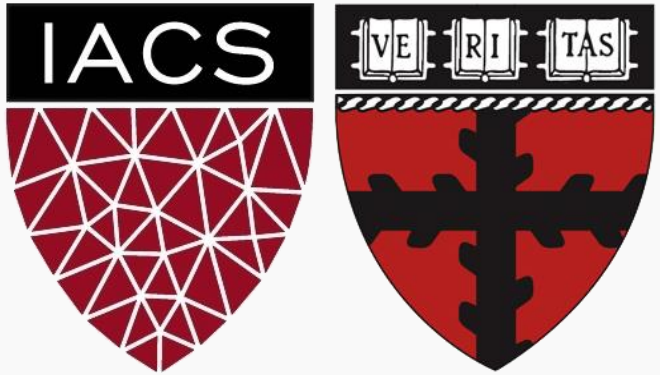
Advanced Section #8:
Neural Networks for Image Analysis

Camilo Fosco

CS109A Introduction to Data Science

Pavlos Protopapas and Kevin Rader



Outline

- Image analysis: why neural networks?
- Multi Layer Perceptron refresher
- Convolutional Neural Networks
  - How they work
  - How to build them
- Building your own image classifier
- Evolution of CNNs

Image analysis – why neural networks?

Imagine that we want to recognize swans in an image:

Cases can be a bit more complex…

Round, elongated head with orange or black beak

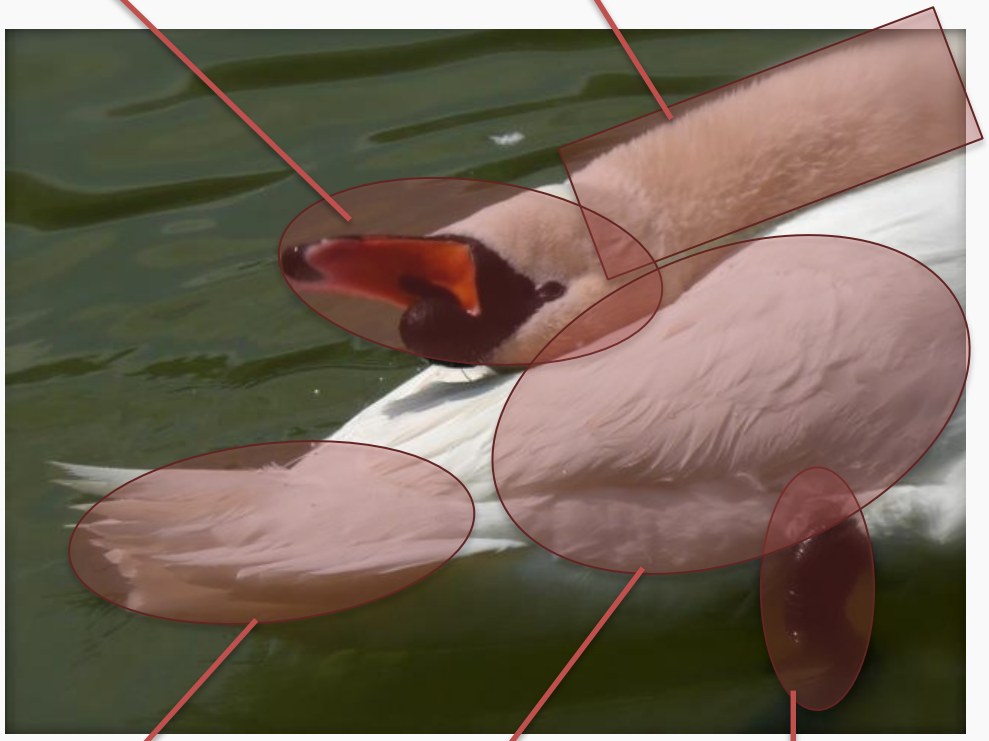
Long white neck, square shape

Oval-shaped white body with or without large white symmetric blobs (wings)

Now what?

Round, elongated head with orange or black beak, can be turned backwards

Long white neck, can bend around, not necessarily straight



White tail, generally far from the head, looks feathery

White, oval shaped body, with or without wings visible

Black feet, under body, can have different shapes

Small black circles, can be facing the camera, sometimes can see both

Black triangular shaped form, on the head, can have different sizes

White elongated piece, can be squared or more triangular, can be obstructed sometimes

Luckily, the color is consistent…

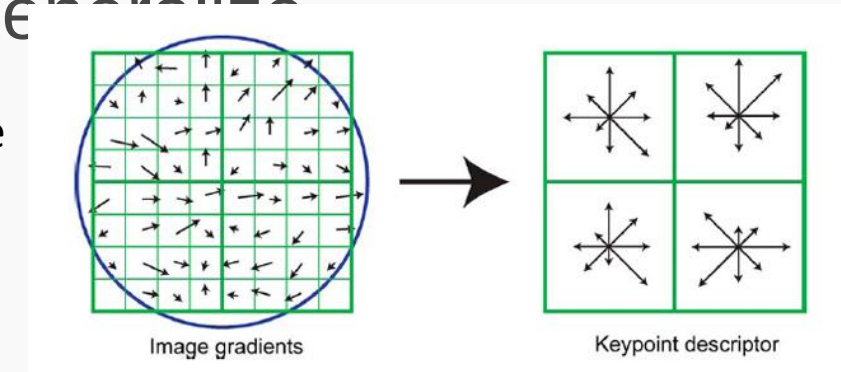
We need to be able to deal with these cases.

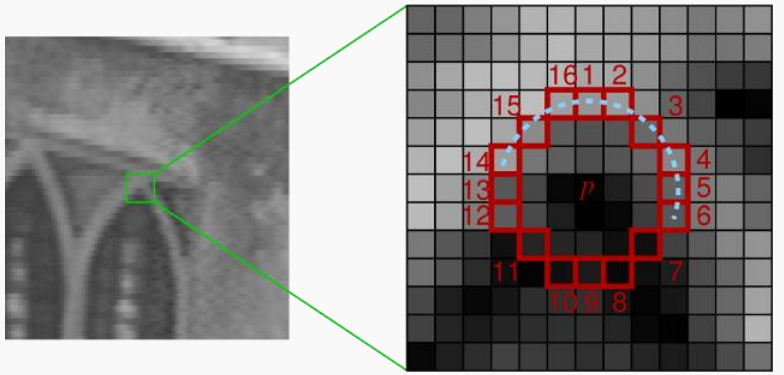
Man in swan tent photographing swans

Image features

- We've been basically talking about detecting features in images, in a very naïve way.
- Researchers built multiple computer vision techniques to deal with these issues: SIFT, FAST, SURF, BRIEF, etc.
- However, similar problems arose: the detectors where either too general or too over-engineered. Humans were designing these feature detectors, and that made them either too simple or hard to generalize

SIFT feature descriptor





FAST corner detection algorithm

- What if we learned the features to detect?
- We need a system that can do Representation Learning (or Feature Learning).

Representation Learning: technique that allows a system to automatically find relevant features for a given task. Replaces manual feature engineering.

Multiple techniques for this:

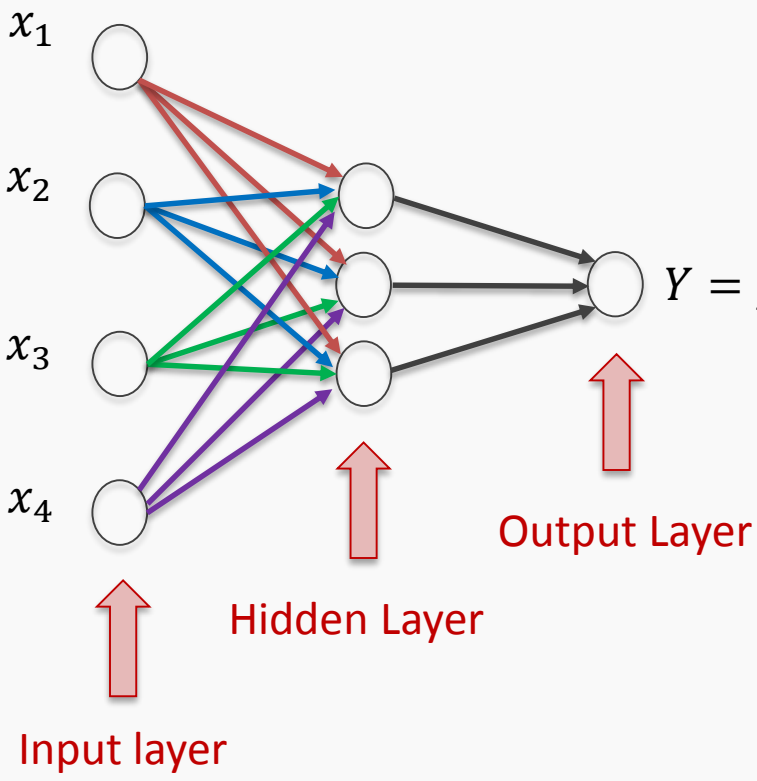
- Unsupervised (K-means, PCA, …).
- Supervised (Sup. Dictionary learning, Neural Networks!)

MULTILAYER PERCEPTRON

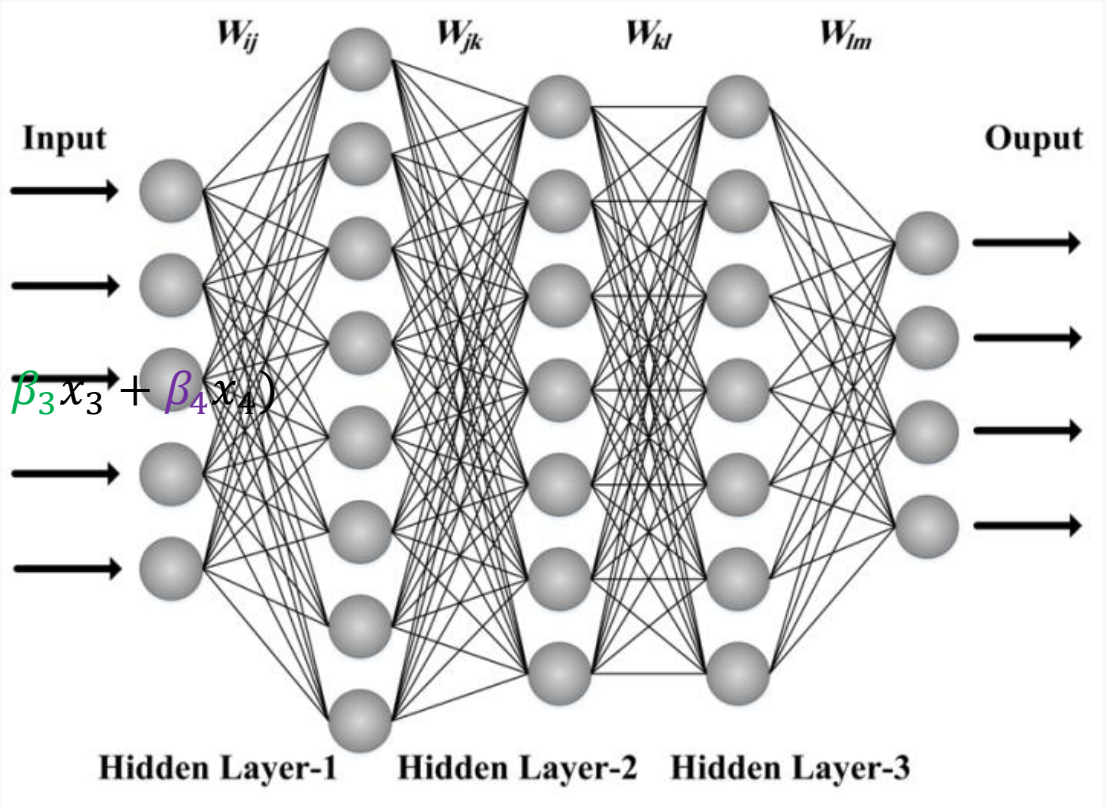
Or Fully Connected Network (FCN)

Perceptron to MLP

Multilayer Perceptron



$$Y = f(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4)$$



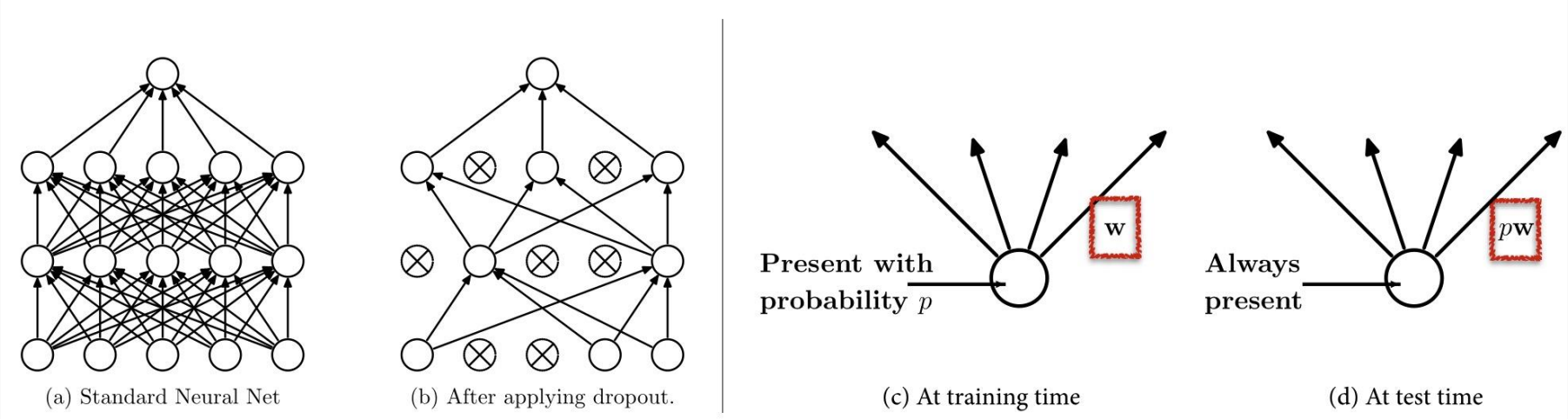
They can be more complex…

Main advantages of MLP

- Ability to find patterns in complex and messy data.
- Network with one hidden layer and sufficient hidden nodes has been proven to be an universal approximator.
- Can take the raw data as input, and learn its own features internally to better classify.
- Amount of human involvement is low: we only prepare and feed the data. No feature engineering needed.
- MLP makes no assumption on the distribution of input data.

Combatting overfitting: Dropout

- Method of regularization consisting of randomly dropping nodes during training.
- Similar to bagging.
- We re-randomize our network at each training iteration.
- During test time, we use the full network where nodes are scaled by their probability of appearing.



(a) Standard Neural Net (b) After applying dropout. (c) At training time (d) At test time

Multilayer perceptron - visualization

Let’s have a look at a cool tool to play with MLPs:

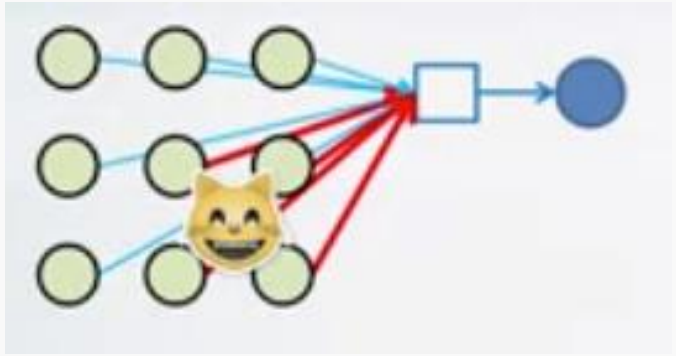
https://playground.tensorflow.org/

Drawbacks

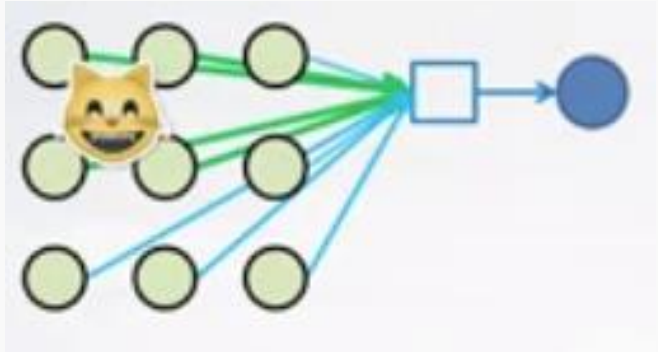
- MLPs use one perceptron for each pixel in an image, multiplied by 3 in RGB case. the amount of weights rapidly becomes unmanageable for large images.
- Training difficulties arise, overfitting can appear.
- MLPs react differently to an image and its shifted version – they are not translation invariant.

Drawbacks

Imagine we want to build a cat detector with an MLP.



In this case, the red weights will be modified to better recognize cats



In this case, the green weights will be modified.

We are learning redundant features. Approach is not robust, as cats could appear in yet another position.

Drawbacks

Example: CIFAR10

Simple 32x32 color images (3 channels)

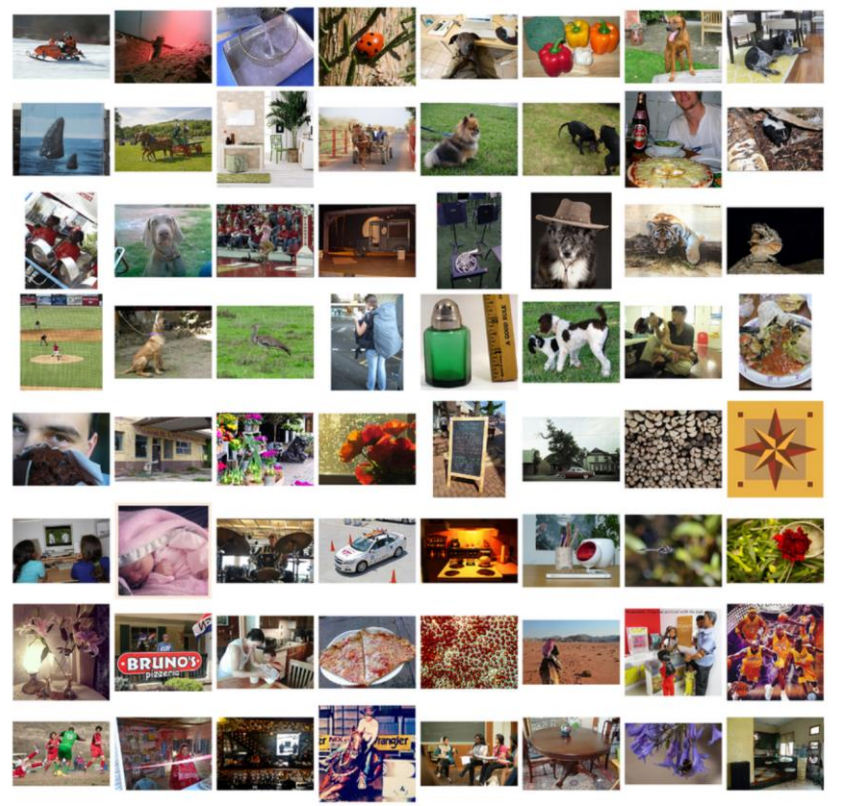
Each pixel is a feature: an MLP would have 32x32x3+1 = 3073 weights per neuron!

Drawbacks

Example: ImageNet

Images are usually 224x224x3: an MLP would have 150129 weights per neuron. If the first layer of the MLP is around 128 nodes, which is small, this already becomes very heavy to calculate.

Model complexity is extremely high: overfitting.



CONVOLUTIONAL NEURAL NETWORKS

The smart way of looking at images

Basics of CNNs

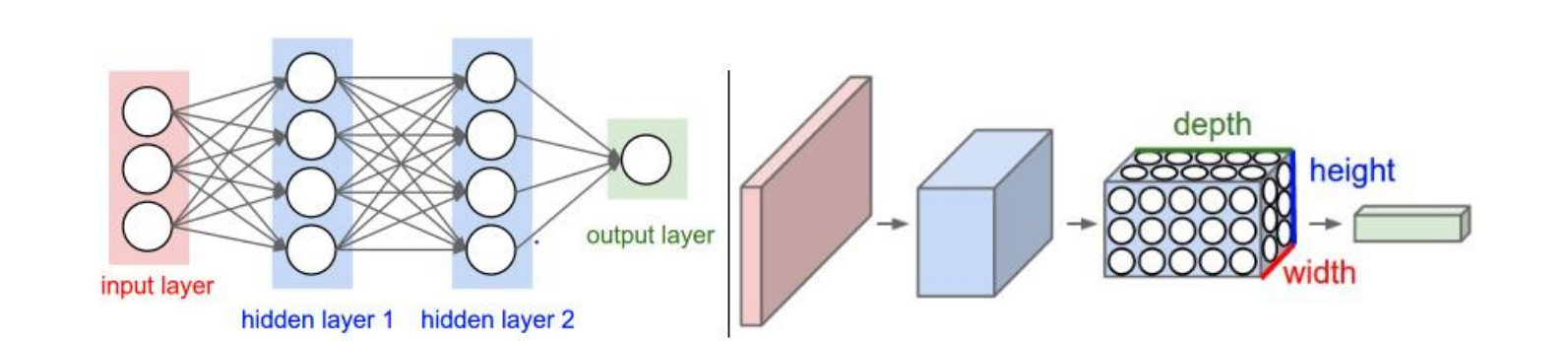
We know that MLPs:

- Do not scale well for images
- Ignore the information bought by pixel position and correlation with neighbors
- Cannot handle translations

The general idea of CNNs is to intelligently adapt to properties of images:

- Pixel position and neighborhood has semantic meaning.
- Elements of interest can appear anywhere in the image.

Basics of CNNs



MLP CNN

CNNs are also composed of layers, but those layers are not fully connected: they have filters, sets of cube-shaped weights that are applied throughout the image. Each 2D slice of the filters are called kernels.

These filters introduce translation invariance and parameter sharing.

How are they applied? Convolutions!

Convolution and cross-correlation

- Convolution of f and g $(f * g)$ is defined as the integral of the product, having one of the functions inverted and shifted:

$$(f * g)(t) = \int_a f(a)g(t-a)da$$

Function is inverted and shifted left by t

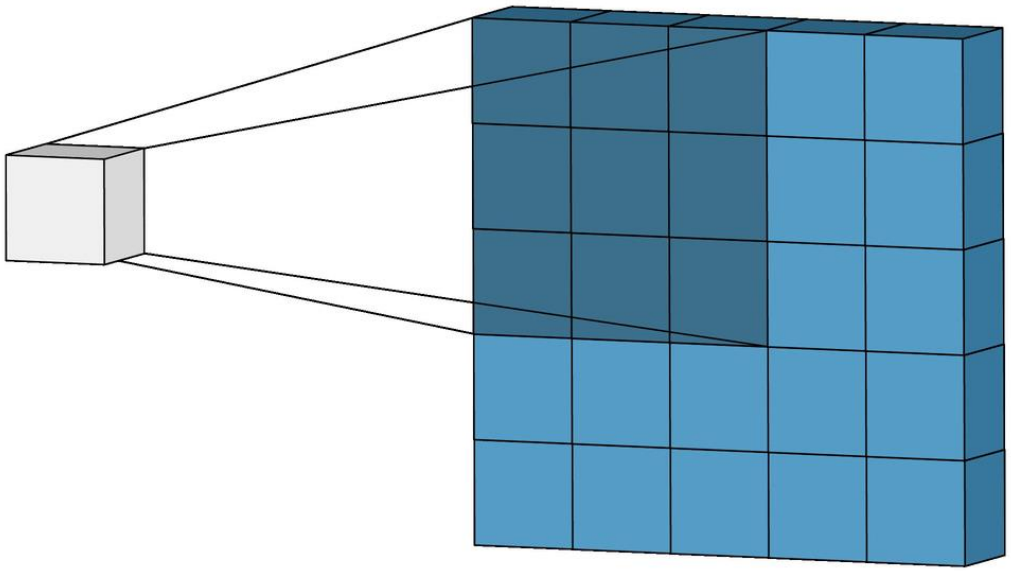
- Discrete convolution:

$$(f * g)(t) = \sum_{a=-\infty}^{\infty} f(a)g(t-a)$$

- Discrete cross-correlation:

$$(f \star g)(t) = \sum_{a=-\infty}^{\infty} f(a)g(t+a)$$

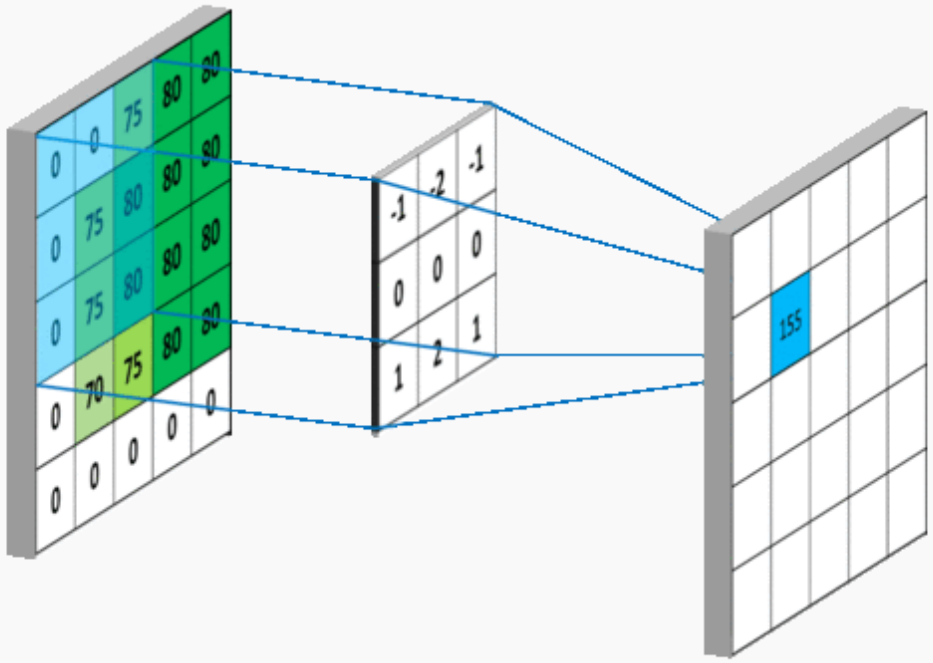
Convolutions – step by step



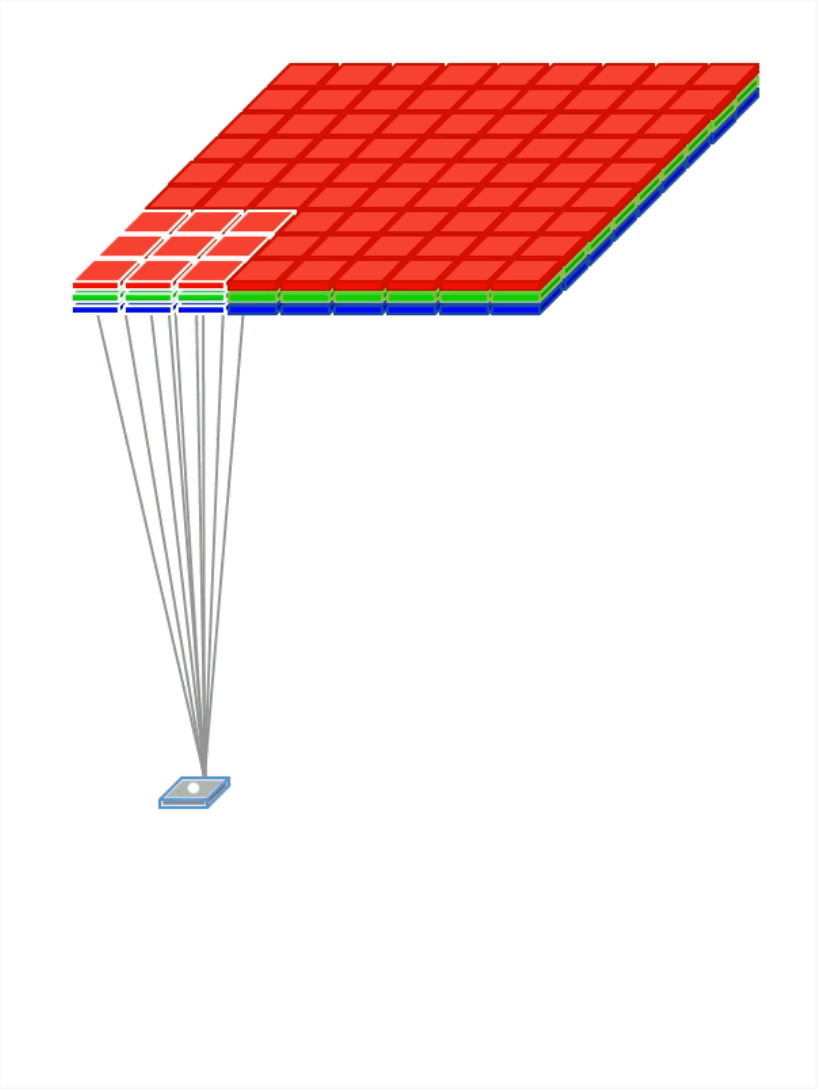
3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12	12	17
10	17	19
9	6	14

Convolutions – another example

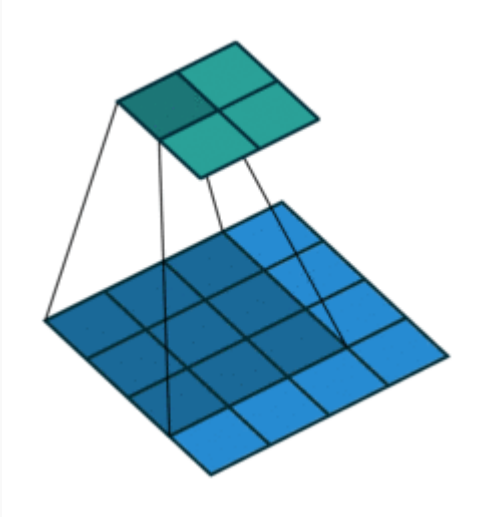


Convolutions – 3D input



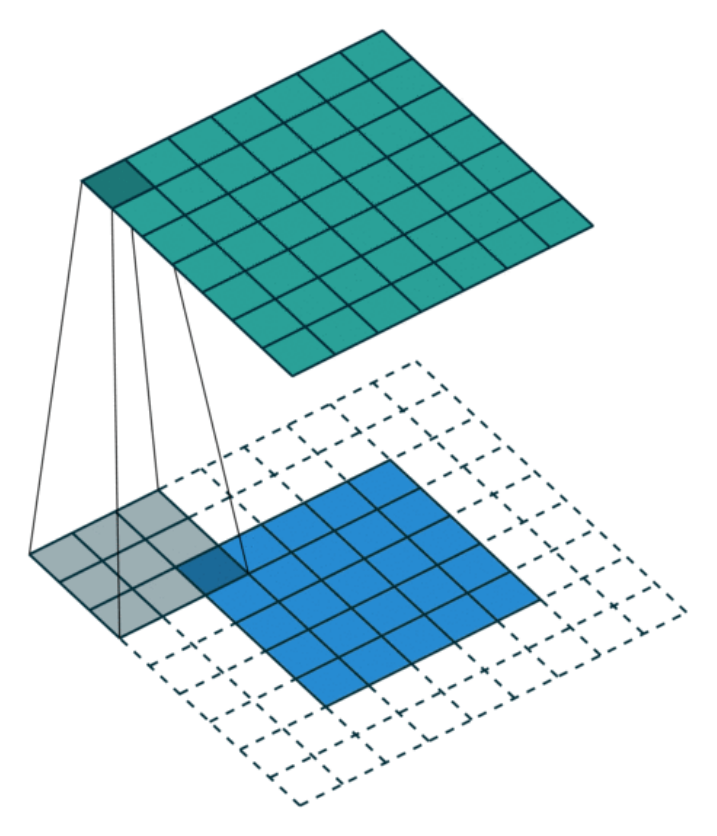
Convolutions – what happens at the edges?

If we apply convolutions on a normal image, the result will be downsampled by an amount depending on the size of the filter.

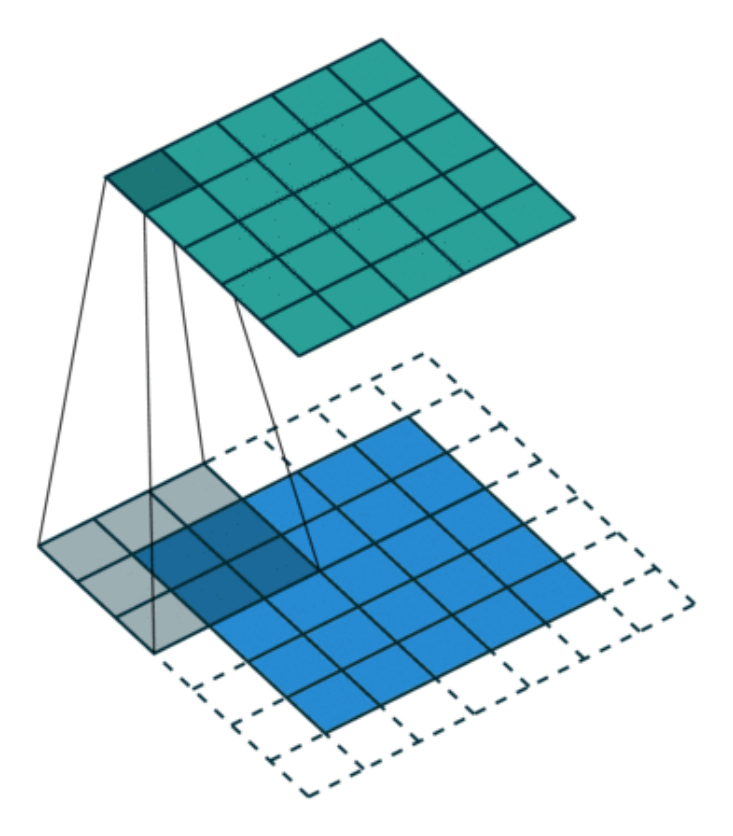


We can avoid this by padding the edges in different ways.

Padding

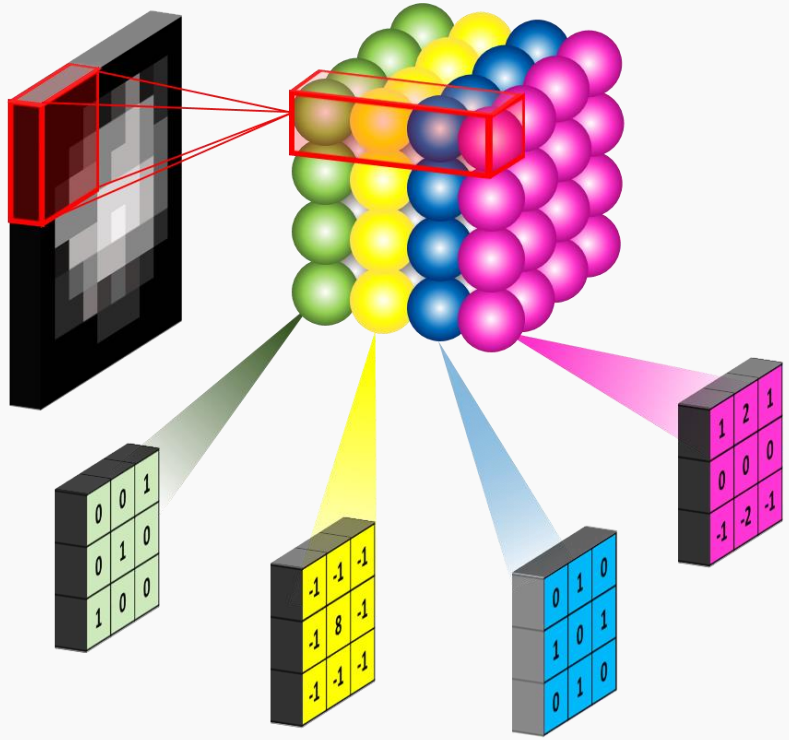


Full padding. Introduces zeros such that all pixels are visited the same amount of times by the filter. Increases size of output.

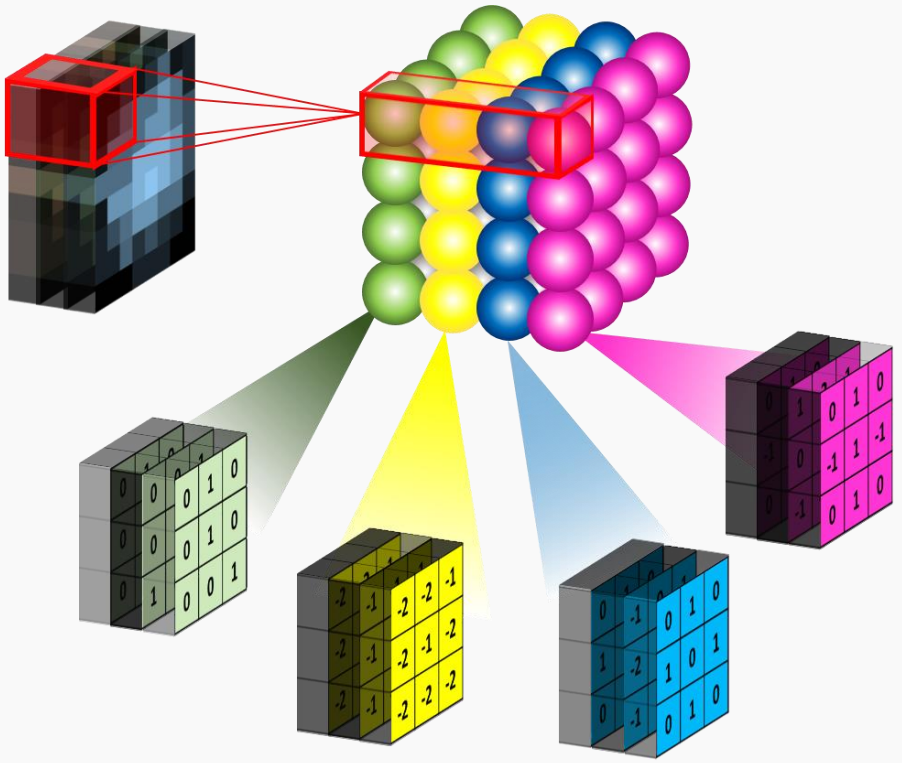


Same padding. Ensures that the output has the same size as the input.

Convolutional layers



Convolutional layer with four 3x3 filters on a black and white image (just one channel)

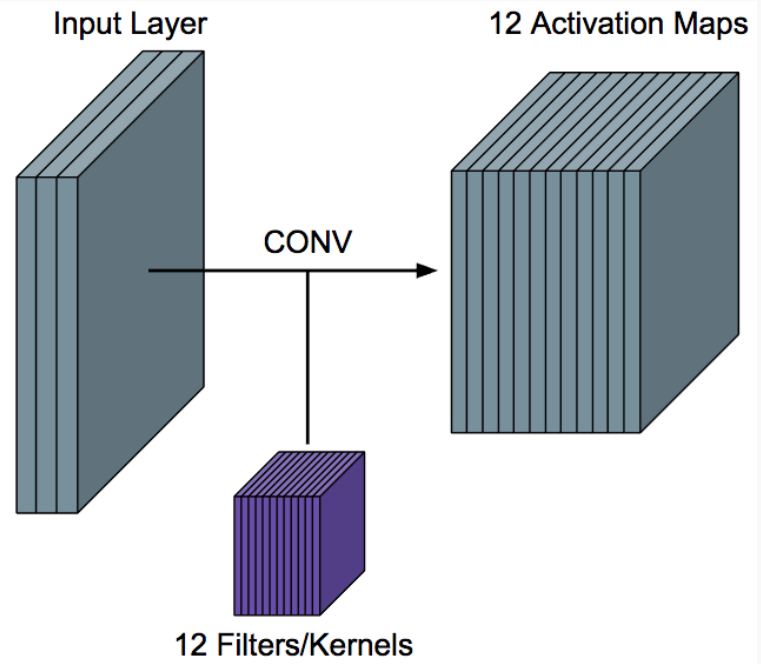


Convolutional layer with four 3x3 filters on an RGB image. As you can see, the filters are now cubes, and they are applied on the full depth of the image..

- To be clear: each filter is convolved with the entirety of the 3D input cube, but generates a 2D feature map.

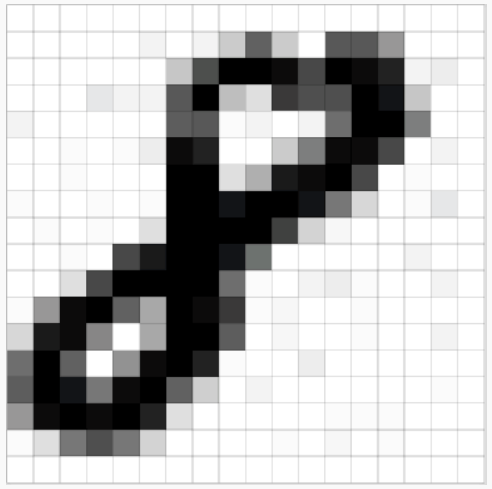
- Because we have multiple filters, we end up with a 3D output: one 2D feature map per filter.

- The feature map dimension can change drastically from one conv layer to the next: we can enter a layer with a 32x32x16 input and exit with a 32x32x128 output if that layer has 128 filters.



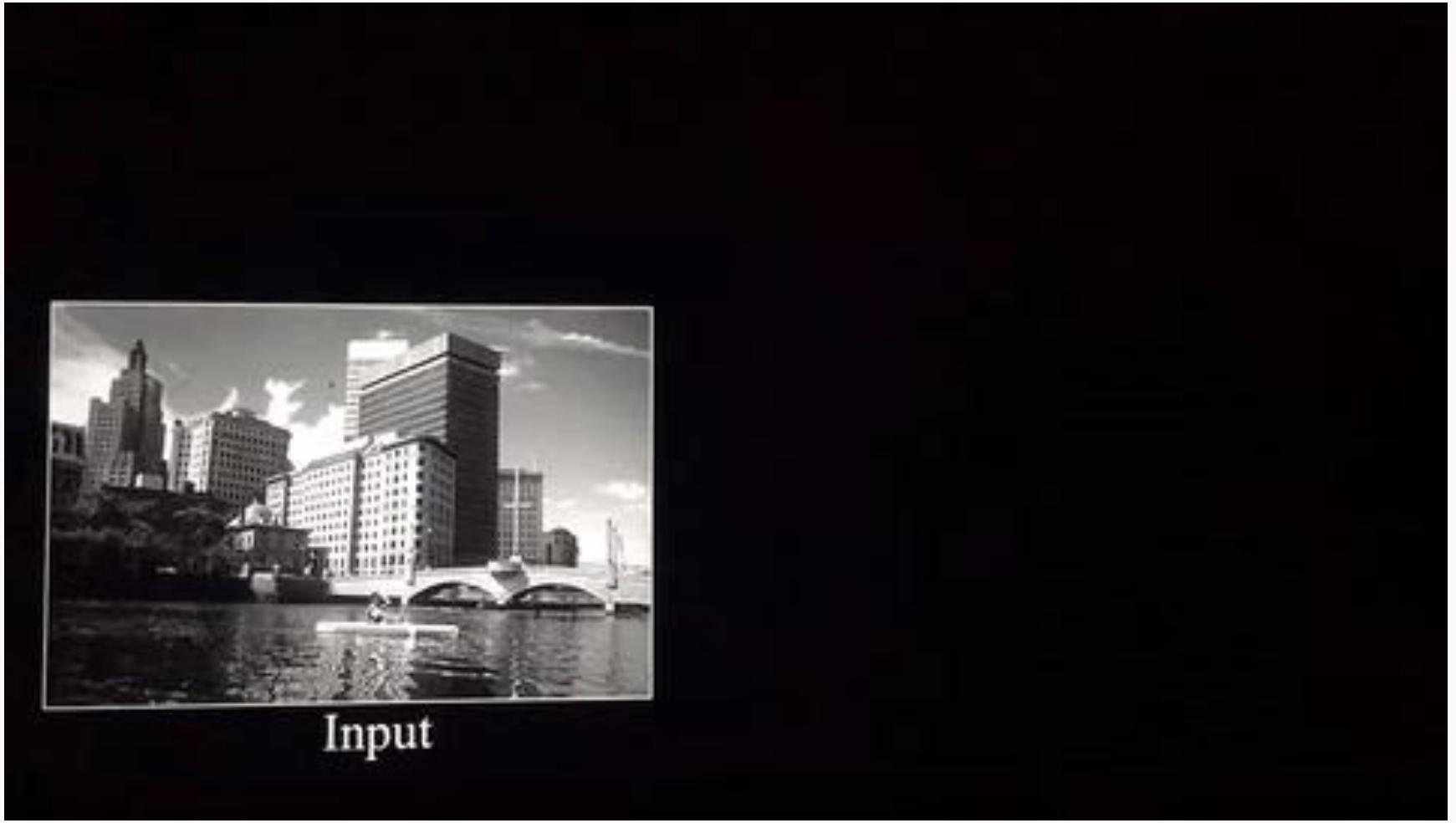
Why does this make sense?

In image is just a matrix of pixels.



Convolving the image with a filter produces a feature map that highlights the presence of a given feature in the image.

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	


Input

In a convolutional layer, we are basically applying multiple filters at over the image to extract different features.

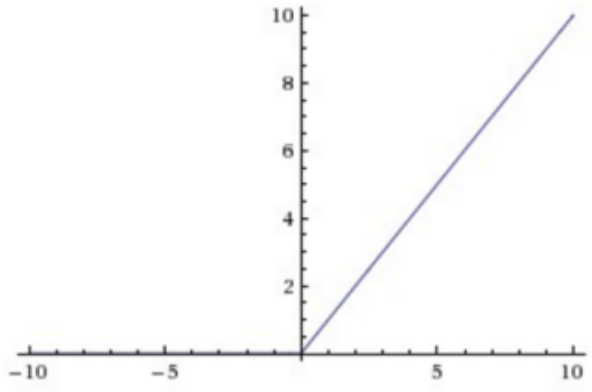
But most importantly, we are learning those filters!

One thing we're missing: non-linearity.

Introducing ReLU

The most successful non-linearity for CNNs is the Rectified Non-Linear unit (ReLU):

Output = Max(zero, Input)



Combats the vanishing gradient problem occurring in sigmoids, is easier to compute, generates sparsity (not always beneficial)

Convolutional layer so far

- A convolutional layer convolves each of its filters with the input.
- Input: a 3D tensor, where the dimensions are Width, Height and Channels (or Feature Maps)
- Output: a 3D tensor, with dimensions Width, Height and Feature Maps (one for each filter)
- Applies non-linear activation function (usually ReLU) over each value of the output.
- Multiple parameters to define: number of filters, size of filters, stride, padding, activation function to use, regularization.

Building a CNN

A convolutional neural network is built by stacking layers, typically of 3 types:

Convolutional Layers

Pooling Layers

Fully connected Layers

Building a CNN

A c
typ

Convolutional Layers

Action

- Apply filters to extract features
- Filters are composed of small kernels, learned.
- One bias per filter.
- Apply activation function on every value of feature map

Parameters

- Number of kernels
- Size of kernels (W and H only, D is defined by input cube)
- Activation function
- Stride
- Padding
- Regularization type and value

I/O

- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Building a CNN

A convolutional neural network is built by stacking layers, typically of 3 types:

Convolutional Layers

Pooling Layers

Fully connected Layers

Building a CNN

A c
typ

Pooling Layers

Action

- Reduce dimensionality
- Extract maximum of average of a region
- Sliding window approach

Parameters

- Stride
- Size of window

I/O

- Input: 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filte, reduced spatial dimensions

Building a CNN

A convolutional neural network is built by stacking layers, typically of 3 types:

Convolutional Layers

Pooling Layers

Fully connected Layers

Building a CNN

A c
typ

Fully connected Layers

Action

- Aggregate information from final feature maps
- Generate final classification

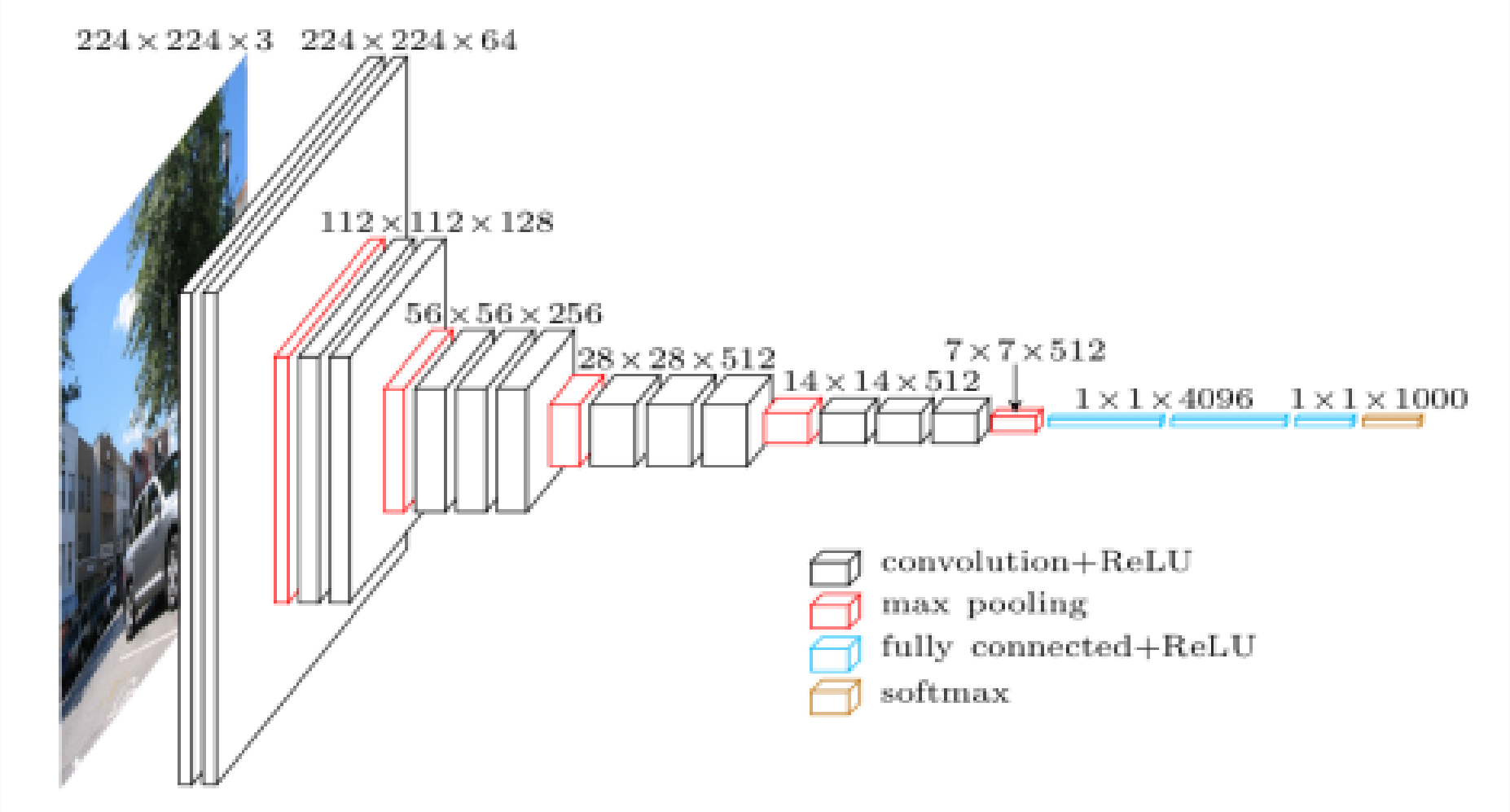
Parameters

- Number of nodes
- Activation function: usually changes depending on role of layer. If aggregating info, use ReLU. If producing final classification, use Softmax.

I/O

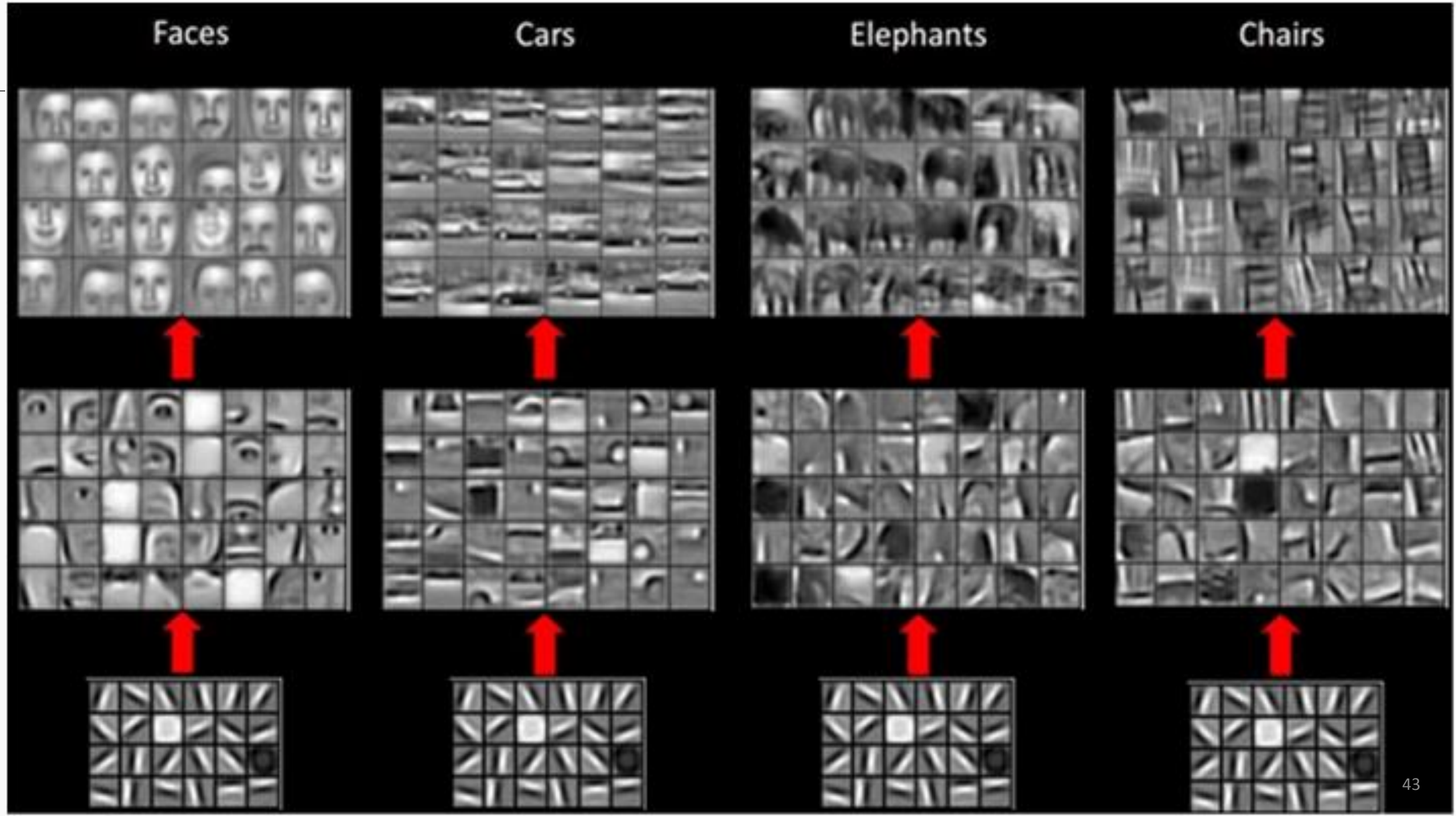
- Input: FLATTENED 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

Fully built CNN (VGG)



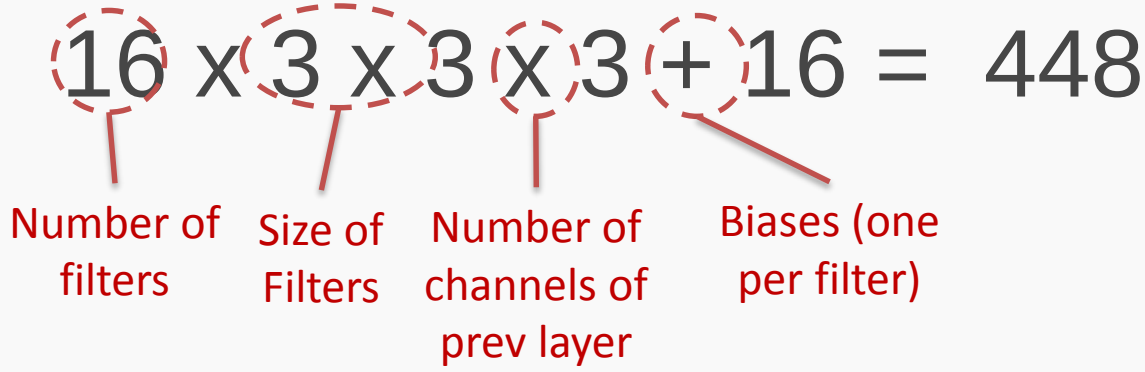
What do they learn?

- Each CNN layer learns filters of increasing complexity.
- The first layers learn basic feature detection filters: edges, corners, etc.
- The middle layers learn filters that detect parts of objects. For faces, they might learn to respond to eyes, noses, etc.
- The last layers have higher representations: they learn to recognize full objects, in different shapes and positions.


Faces
Cars
Elephants
Chairs

Examples

- I have a convolutional layer with 16 3x3 filters that takes an RGB image as input.
  - What else can we define about this layer?
    - Activation function
    - Stride
    - Padding type
  - How many parameters does the layer have?



Examples

- Let C be a CNN with the following disposition:
  - Input: 32x32x3 images
  - Conv1: 8 3x3 filters, stride 1, padding=same
  - Conv2: 16 5x5 filters, stride 2, padding=same
  - Flatten layer
  - Dense1: 512 nodes
  - Dense2: 4 nodes
- How many parameters does this network have?

$$(8 \times 3 \times 3 \times 3 + 8) + (16 \times 5 \times 5 \times 8 + 16) + (16 \times 16 \times 16 \times 512 + 512) + (512 \times 4 + 4)$$

Conv1 Conv2 Dense1 Dense2

3D visualization of networks in action

http://scs.ryerson.ca/~aharley/vis/conv/

https://www.youtube.com/watch?v=3JQ3hYko51Y

BUILDING YOUR OWN IMAGE CLASSIFIER

Keras, Tensorflow, Pytorch?

Machine Learning libraries

Machine Learning is growing, and so are the libraries.

Language that has grown the most in this field: Python.

Popular libraries for machine learning:

- Tensorflow (Google)
- Pytorch (Facebook)
- Keras (initially independent, now part of TF)
- Theano (MILA, University of Montreal)
- Scikit-learn (Started as Google summer project, now backed by INRIA)
- Caffe, Caffe2 (Berkeley AI Research)
- MXNet (Amazon)
- CNTK (Microsoft)

Keras

- High-level API built in Python.
- Focused on Neural Networks.
- Runs seamlessly on CPU and GPU.
- Runs on top of Tensorflow, CNTK or Theano.
- Very intuitive and simple to use: building a net, training and testing is straightforward.
- Developed with focus on fast experimentation.

Keras Guiding principles

- **User friendliness:** designed for human beings. Focuses on good user experience. Consistent and simple APIs. Clear and actionable feedback upon error.
- **Modularity.** Model is sequence or graph of standalone blocks that can be connected to each other with as few restrictions as possible. Layers, losses, activations, regularizations, etc. are all modules that can be plugged in and out of a model easily.
- **Easy extensibility.** New modules are simple to add. Abundance of examples to adapt your mdel to new ideas.
- **Work with Python**. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

Different ways to build our image classifier

Two ways of building models in Keras: Sequential or Functional APIs.

- Sequential:
  - Create a model with *model = Sequential()*
  - Add layers one after the other with *model.add(layer)*
  - Simple to understand, but rigid and basic.
  - Cannot create complex models that require parallel branches or multipe input/outputs
- Functional:
  - Main concept: A layer instance is callable (same as a model), outputs a tensor
  - Connect layers one after the other with *next_output = Layer(params)(previous_output)*
  - Build the final model by joining input an output with *model = Model(in, out)*
  - Keras builds computational graph in the background
  - Very flexible, allows for multiple input/outputs, shared layers, residual

Compiling and training a model

Once our model is built, we need to compile it.

Compilation in Keras links the model with its loss function, optimizer and metrics to compute.

Simple syntax:

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9, nesterov=True))
```

Training a model is similar to Sklearn:

```
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

Evaluating and predicting is also simple:

```
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
```

```
classes = model.predict(x_test, batch_size=128)
```

EVOLUTION OF CNNs

A bit of history

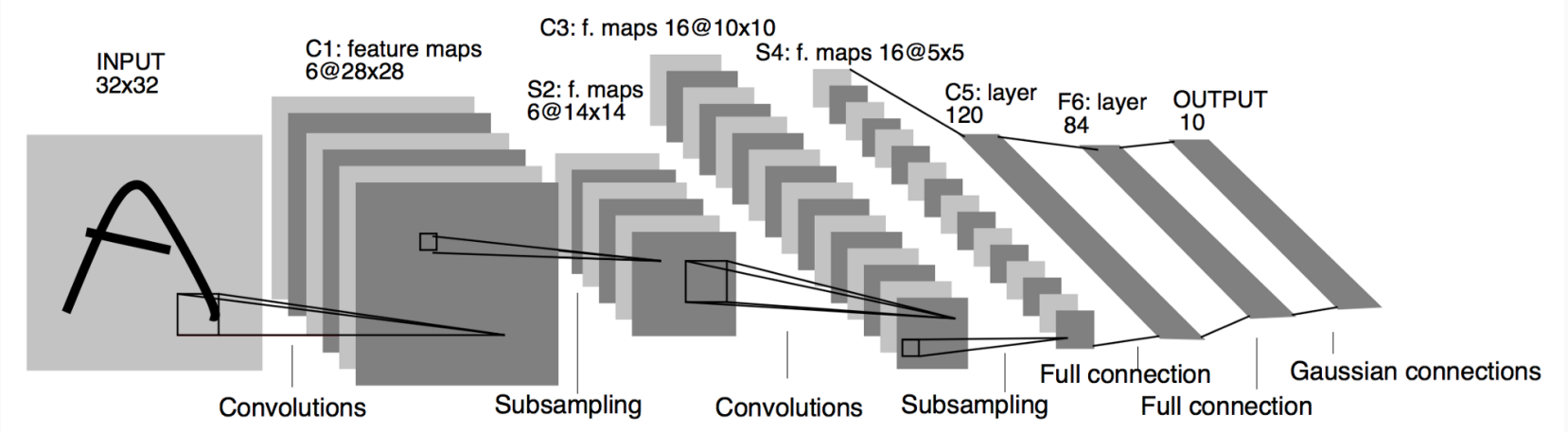
Initial ideas

- The first piece of research proposing something similar to a Convolutional Neural Network was authored by Kunihiko Fukushima in 1980, and was called the NeoCognitron[1].
- Inspired by discoveries on visual cortex of mammals.
- Fukushima applied the NeoCognitron to hand-written character recognition.
- End of the 80's: several papers advancing the field
  - Backpropagation published in French by Yann LeCun in 1985 (independently discovered by other researchers as well)
  - TDNN by Waiber et al., 1989 - Convolutional-like network trained with backprop.
  - Backpropagation applied to handwritten zip code recognition by LeCun et al., 1989

[1] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4): 93-202, 1980.

LeNet

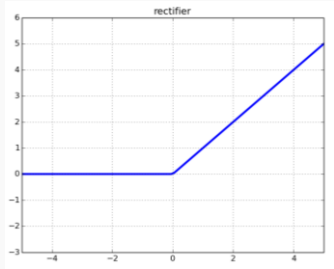
- November 1998: LeCun publishes one of his most recognized papers describing a “modern” CNN architecture for document recognition, called LeNet[1].
- Not his first iteration, this was in fact LeNet-5, but this paper is the commonly cited publication when talking about LeNet.

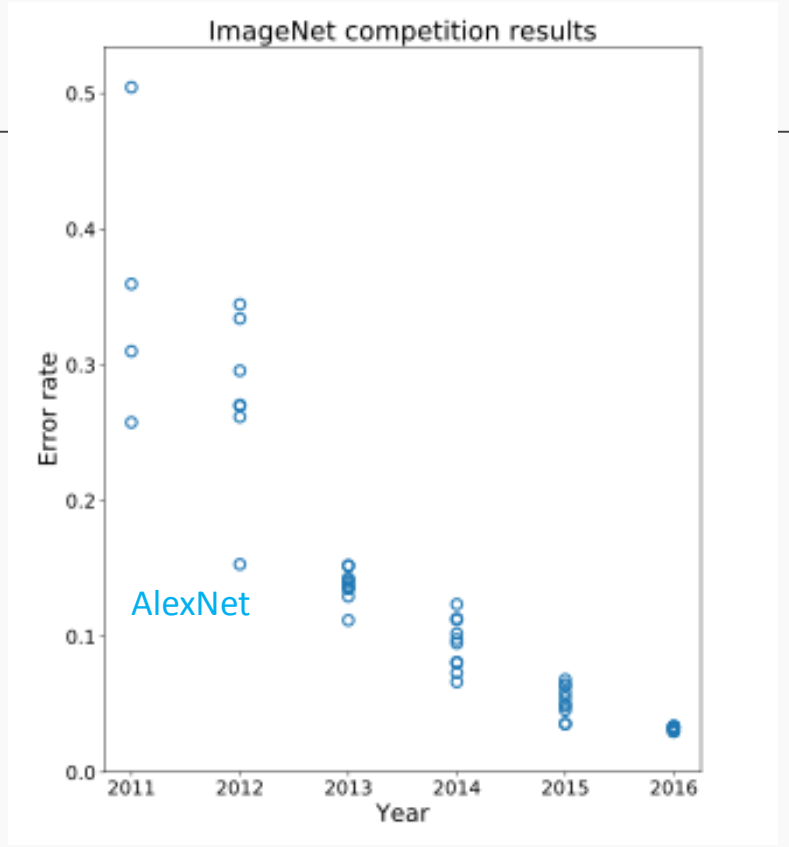


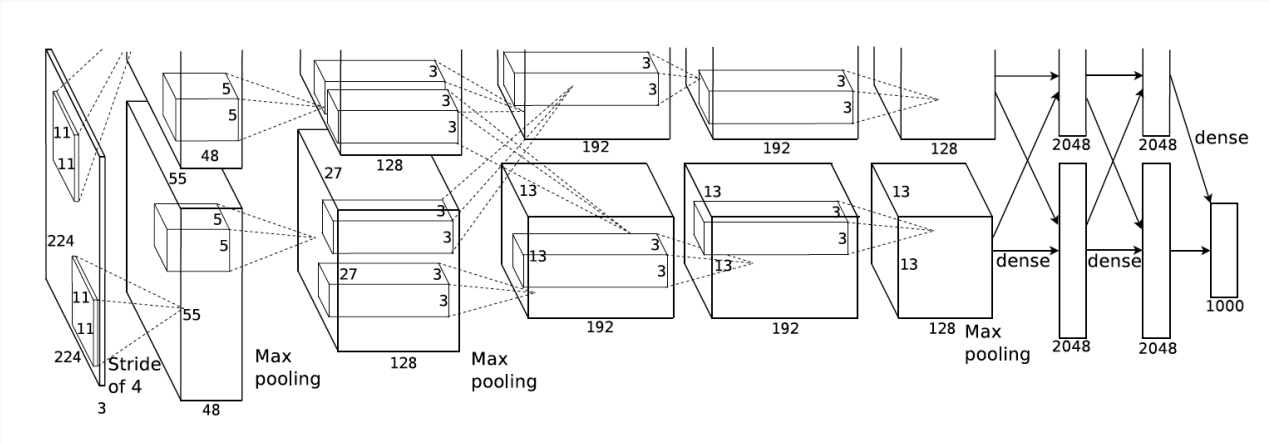
[1] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

AlexNet

- Developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton at Utoronto in 2012. More than 25000 citations.
- Destroyed the competition in the 2012 ImageNet Large Scale Visual Recognition Challenge. Showed benefits of CNNs and kickstarted AI revolution.
- top-5 error of 15.3%, more than 10.8 percentage points lower than runner-up.
- Main contributions:
  - Trained on ImageNet with data augmentation
  - Increased depth of model, GPU training (*five to six days*)
  - Smart optimizer and Dropout layers
  - ReLU activation!

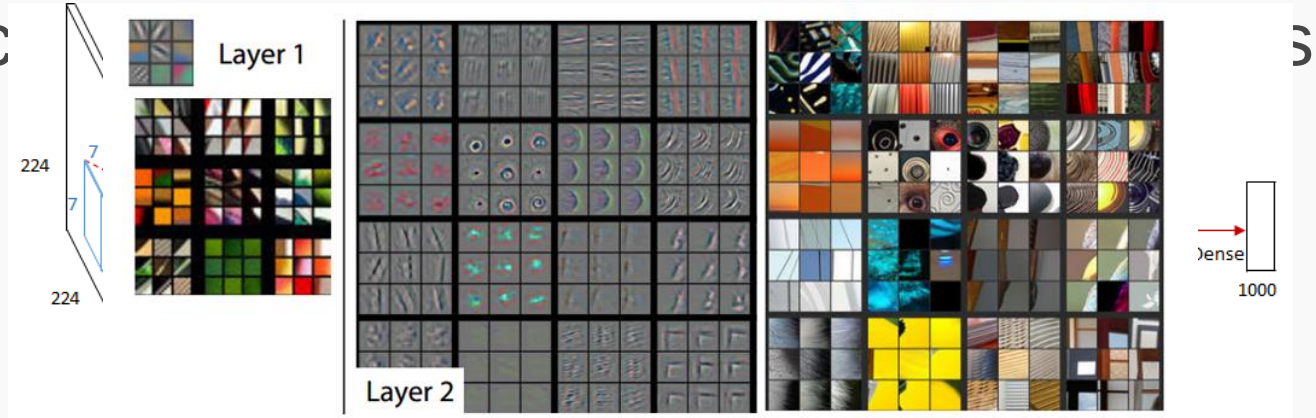






ZFNet

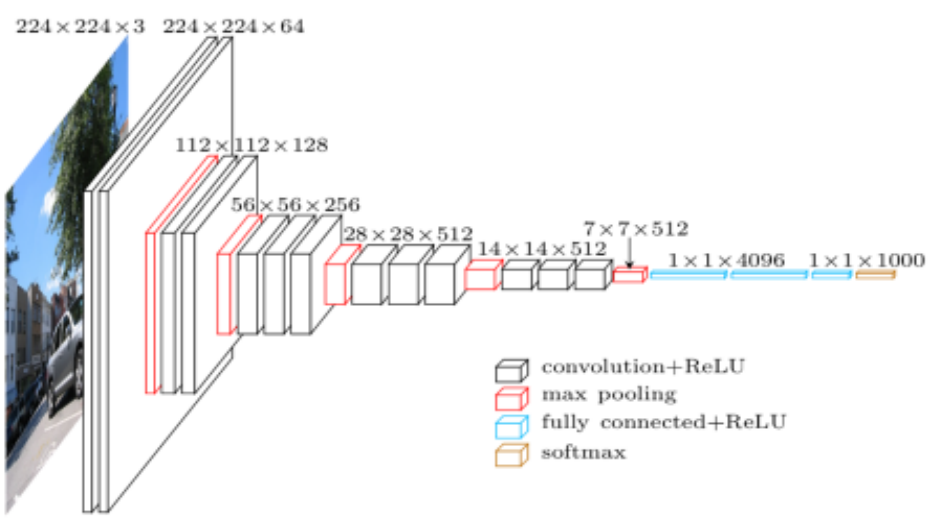
- Introduced by Matthew Zeiler and Rob Fergus from NYU, won ILSVRC 2013 with 11.2% error rate. Decreased sizes of filters.
- Trained for 12 days.
- Paper presented a visualization technique named Deconvolutional Network, which helps to examine different feature ac space.



Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labled Layer 2, we have representations of the 16 different filters (on the left)

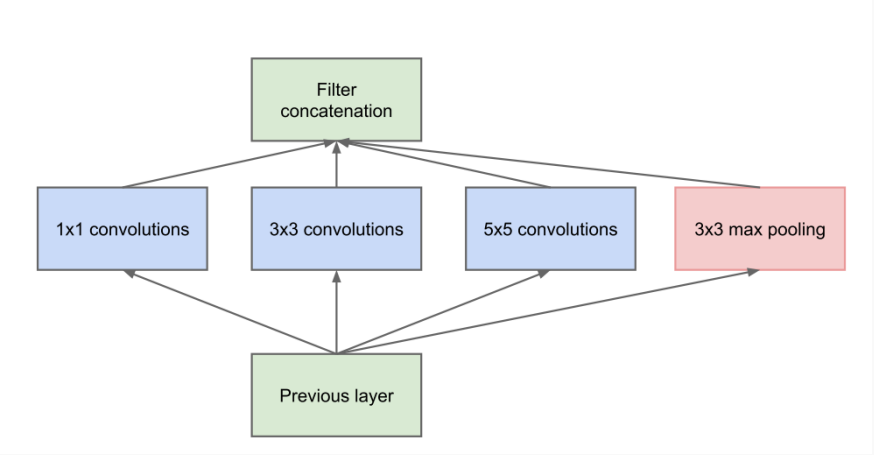
VGG

- Introduced by Simonyan and Zisserman (Oxford) in 2014
- Simplicity and depth as main points. Used 3x3 filters exclusively and 2x2 MaxPool layers with stride 2.
- Showed that two 3x3 filters have an effective receptive field of 5x5.
- As spatial size decreases, depth increases
- Trained for *two to three weeks*.
- Still used as of today.

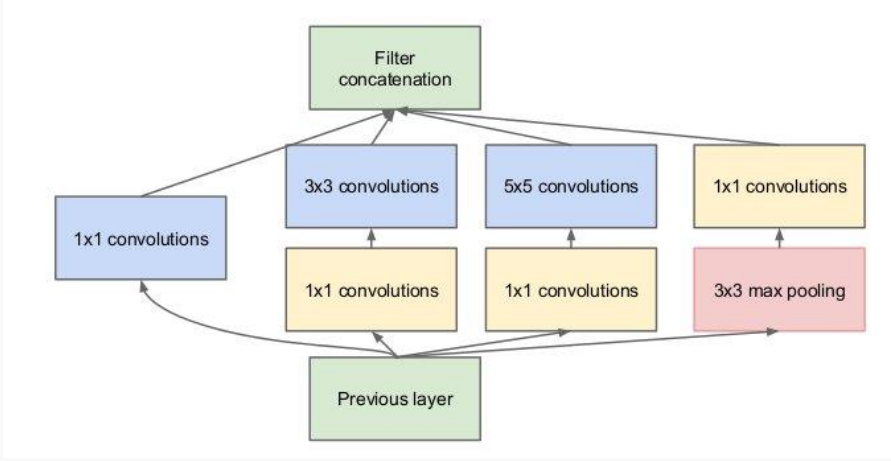


GoogLeNet (Inception-v1)

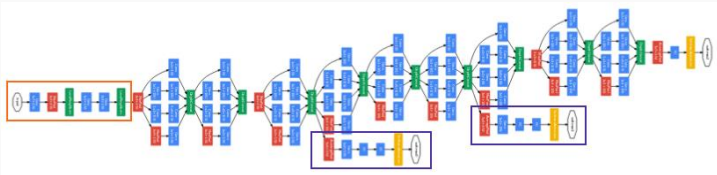
- Introduced by Szegedy et al. (Google), 2014. Winners of ILSVRC 2014.
- Introduces inception module: parallel conv. layers with different filter sizes. Motivation: we don't know which filter size is best – let the network decide. Key idea for future archs.
- No fully connected layer at the end. AvgPool instead. 12x fewer params than A



1x1 convs to Reduce number of parameters



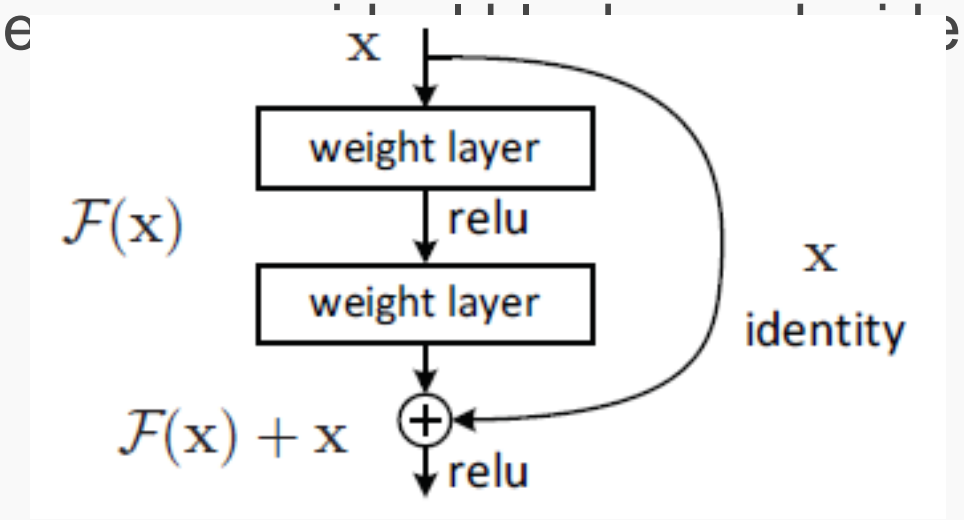
Proto Inception module



Inception module

ResNet

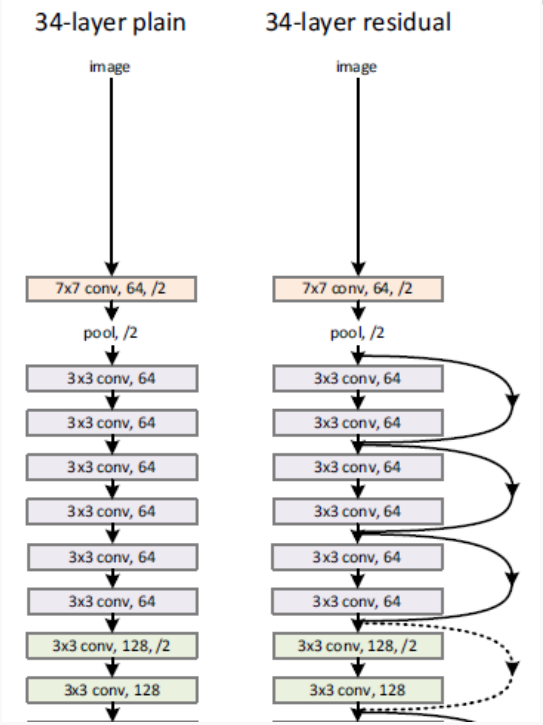
- Presented by He et al. (Microsoft), 2015. Won ILSVRC 2015 in multiple categories.
- Main idea: Residual block. Allows for extremely deep networks.
- Authors believe that it is easier to optimize the residual mapping than the original one. Furthe[illegible] to "shut itself down" if needed.

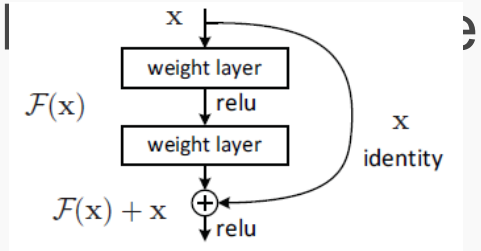


Residual Block

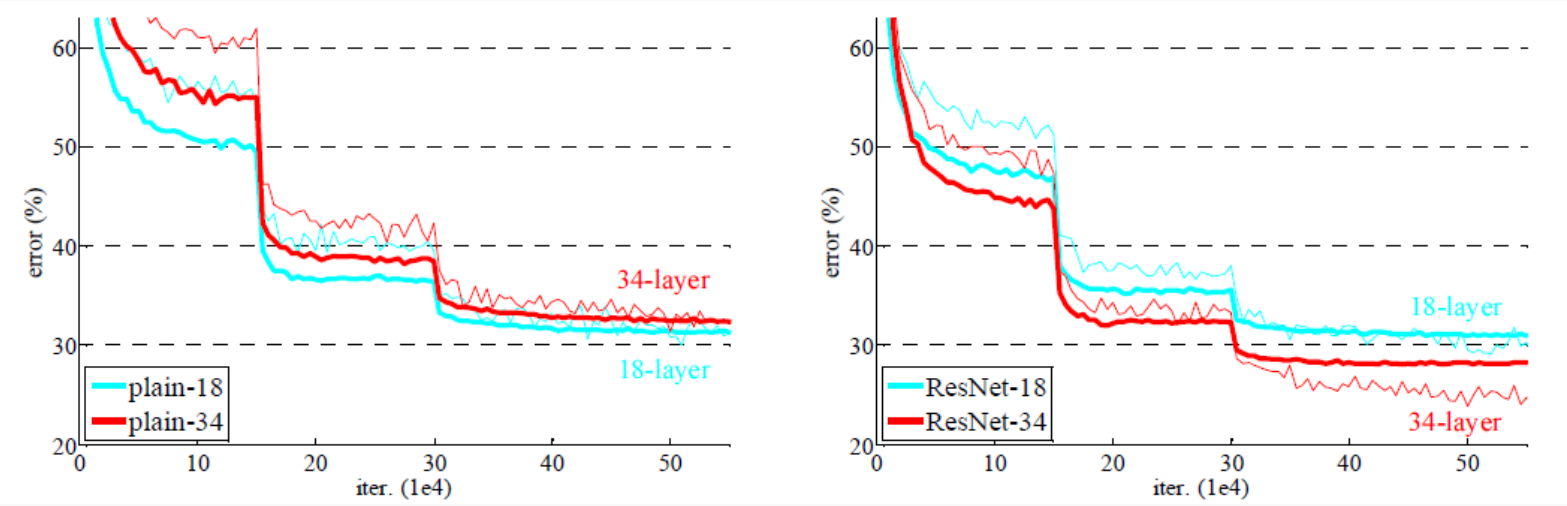
ResNet

- Presented by He et al. (Microsoft), 2015. Won ILSVRC 2015 in multiple categories.
- Main idea: Residual block. Allows for extremely deep networks.
- Authors believe that it is easier to optimize the residual mapping than the original one. Furthermore, residual l e to "shut itself down" if n





Residual Block



DenseNet

- Proposed by Huang et al., 2016. Radical extension of ResNet idea.
- Each block uses every previous feature map as input.
- Idea: n computation of redundant features. All the previous information is available at each point.
- Counter-intuitively, it reduces the number of parameters needed.

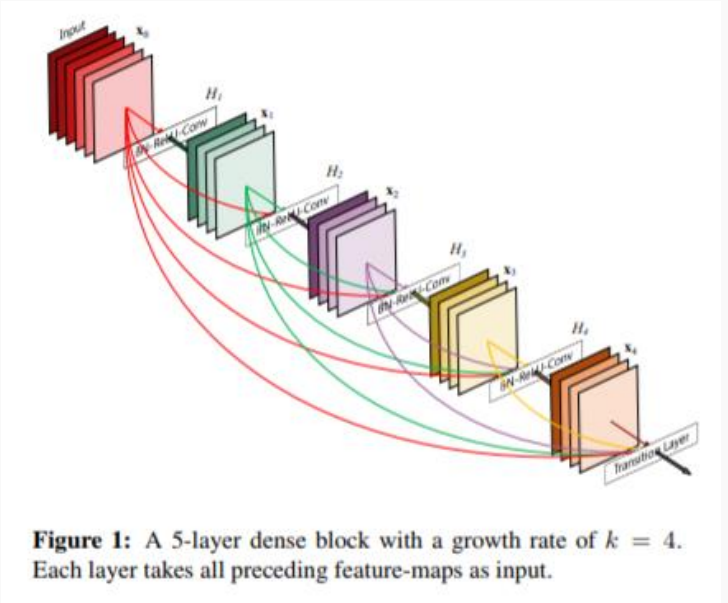


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

DenseNet

- Proposed by Huang et al., 2016. Radical extension of ResNet idea.
- Each block uses every previous feature map as input.
- Idea: n computation of redundant features. All the previous information is available at each point.
- Counter-intuitively, it reduces the number of parameters needed.

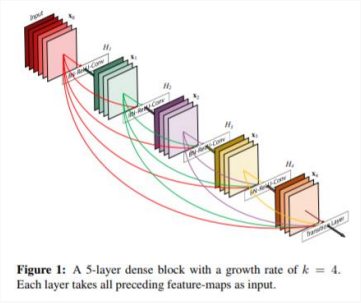
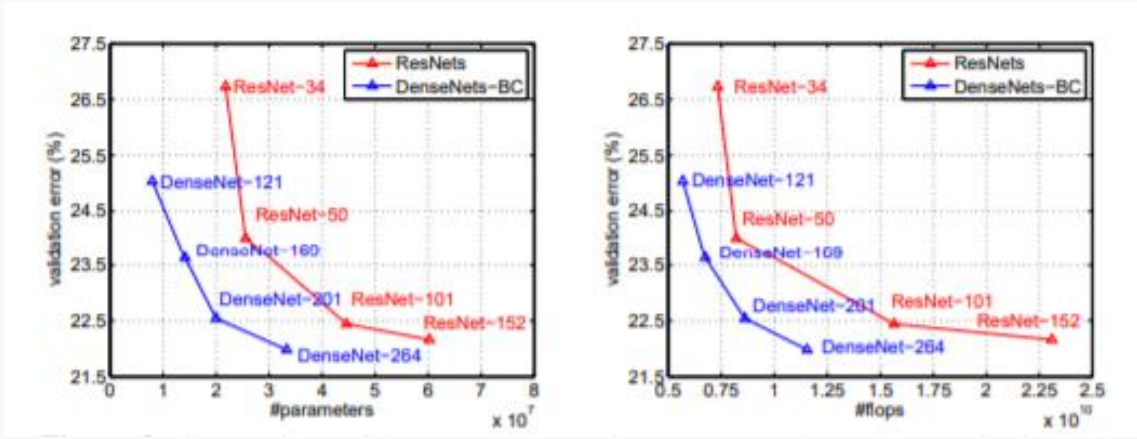
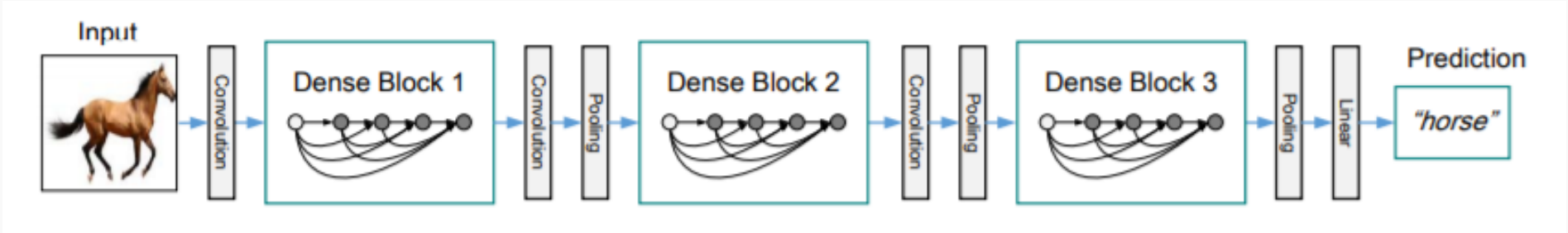


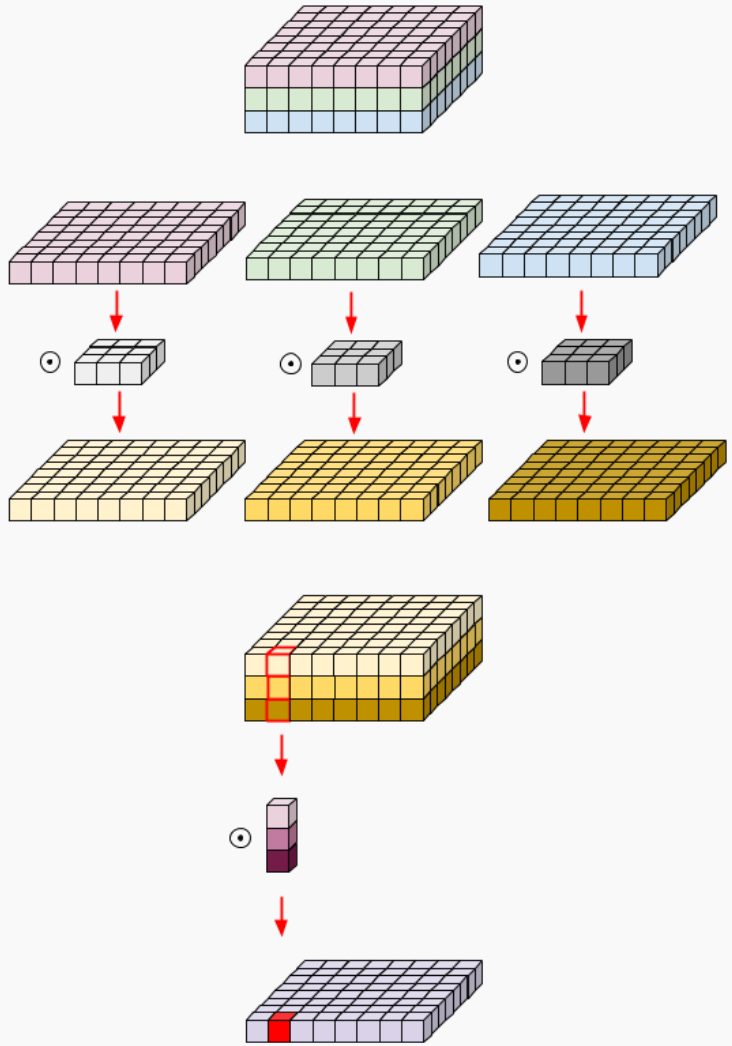
Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.





MobileNet

- Published by Howard et al., 2017.
- Extremely efficient network with decent accuracy.
- Main concept: depthwise-separable convolutions. Convolve each feature maps with a kernel, then use a 1x1 convolution to aggregate the result.
- This approximates vanilla convolutions without having to convolve large kernels through channels.



Beyond

- MobileNetV2 (https://arxiv.org/abs/1801.04381)
- Inception-Resnet, v1 and v2 (https://arxiv.org/abs/1602.07261)
- Wide-Resnet (https://arxiv.org/abs/1605.07146)
- Xception (https://arxiv.org/abs/1610.02357)
- ResNeXt (https://arxiv.org/pdf/1611.05431)
- ShuffleNet, v1 and v2 (https://arxiv.org/abs/1707.01083)
- Squeeze and Excitation Nets (https://arxiv.org/abs/1709.01507)

The world of image analysis

Image classification is just one task. There are many other interesting tasks that use the networks presented here and more:

- Object detection and localization
- Image denoising
- Semantic Segmentation
- Saliency prediction
- Captioning
- Style transfer
- …

CS209b!

THANK YOU!

Let’s now take a look at how to build very simple models in practice.

Notebook examples!