Computer Vision with Deep Learning

Basics Day

Luis F. Teixeira

CV  DL

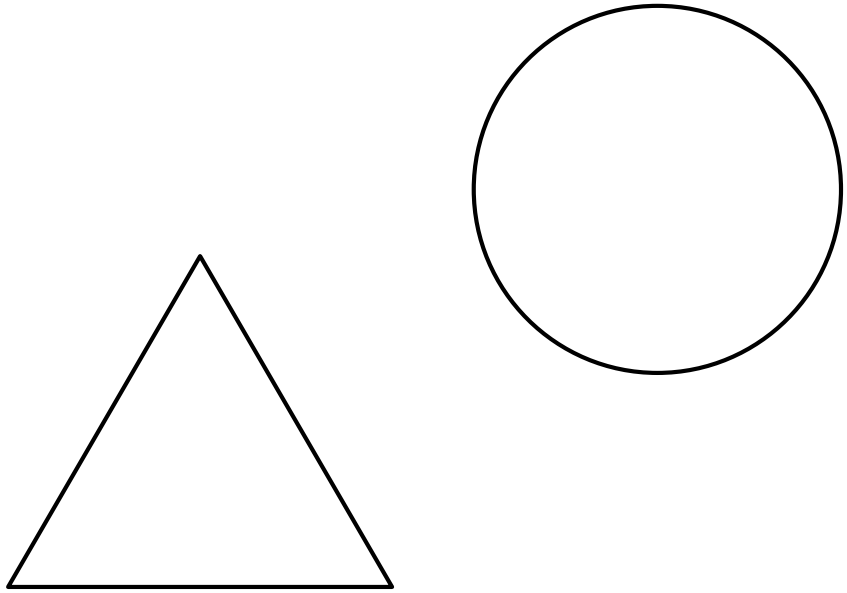
Let’s focus on a specific computer vision task → recognition
i.e. predict the correct label for an image

How to bridge the gap between pixels and “meaning”?

Variability is a problem

Typical computer vision pipeline

What features do we need?



What features do we need?

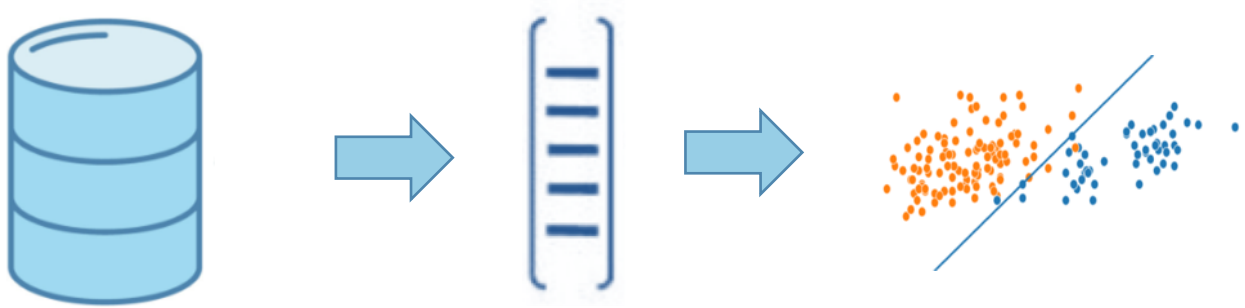
And the decision?

Fixed model (hand-designed)

Source: Computer Vision Project by Ivo Saavedra, Telmo Baptista, Tiago Silva

And the decision?

The Machine Learning approach



Collection of images with associated labels

Extracted features

Learned model

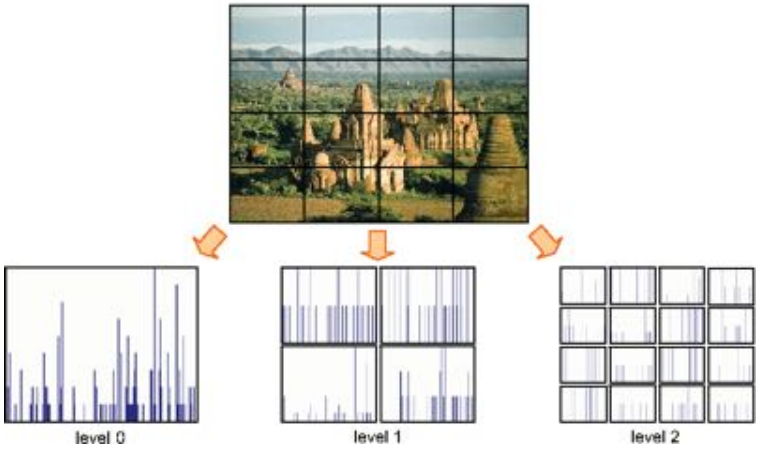

And, what features do we need?

Features

Extraction of relevant features from images is not an easy task

Traditional Computer Vision Fixed Approaches

- Hand-crafted features (colour, texture, shape)
- Local features (SIFT, SURF, HOG)
- Dictionary-based (Bag of Visual Words, Fisher Vectors)
- Spatial pyramid



Okay, nice…

But, can we also learn the features from the examples?

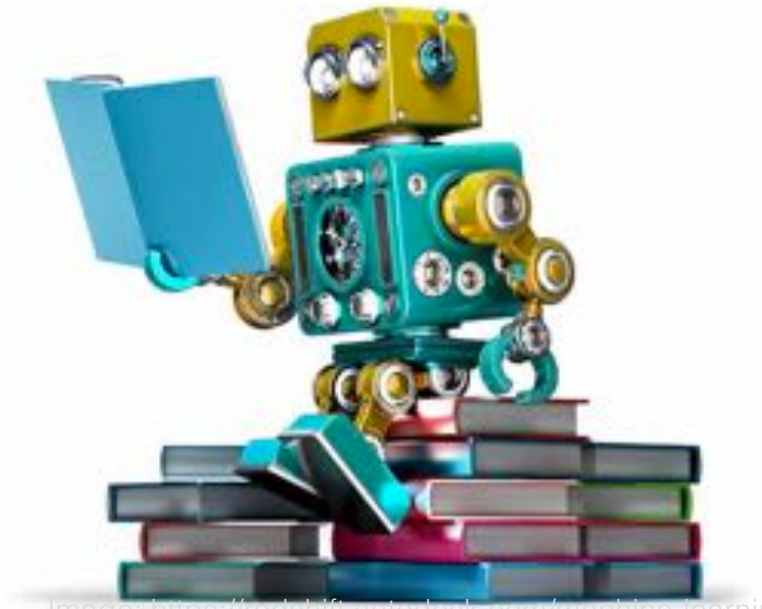
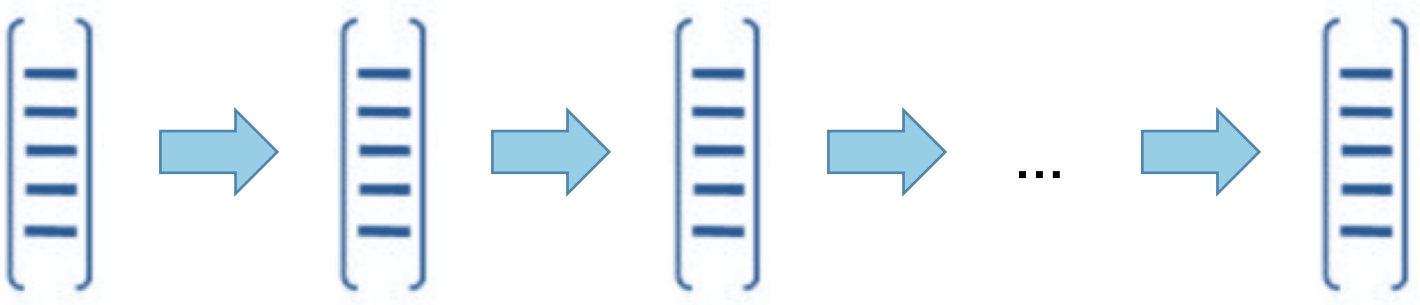


Image: https://redshift.autodesk.com/machine-learning/

Maybe using a model that can “construct” features from other features?



low-level features
(closer to the image content)

high-level features
(closer to the semantics)

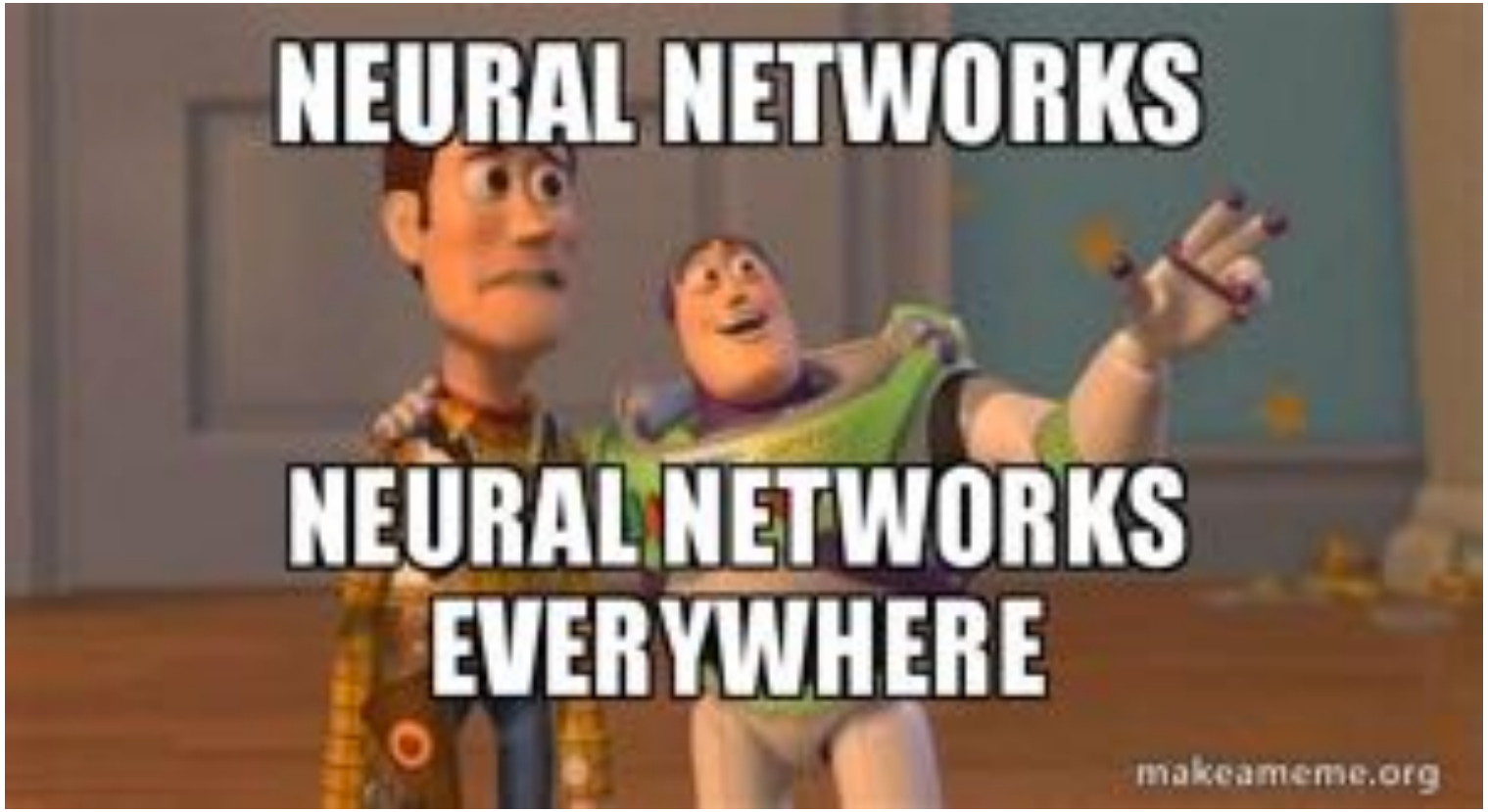
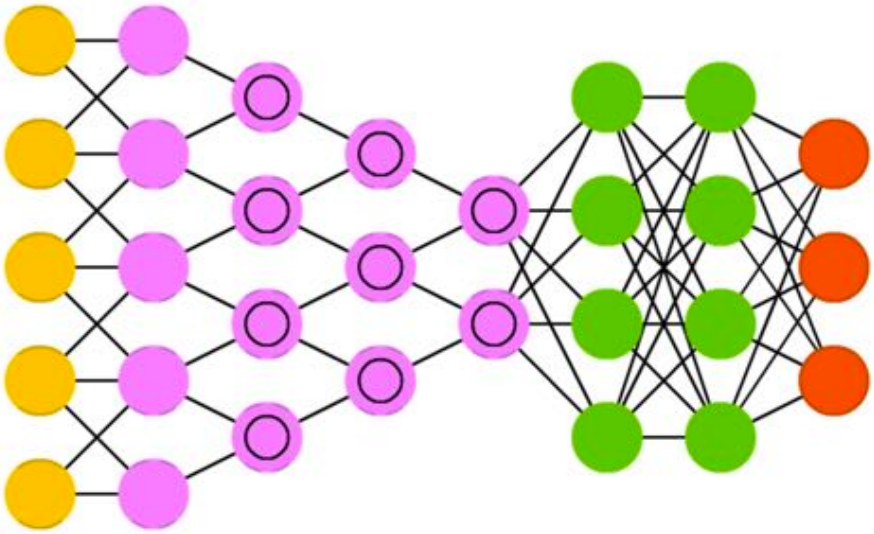
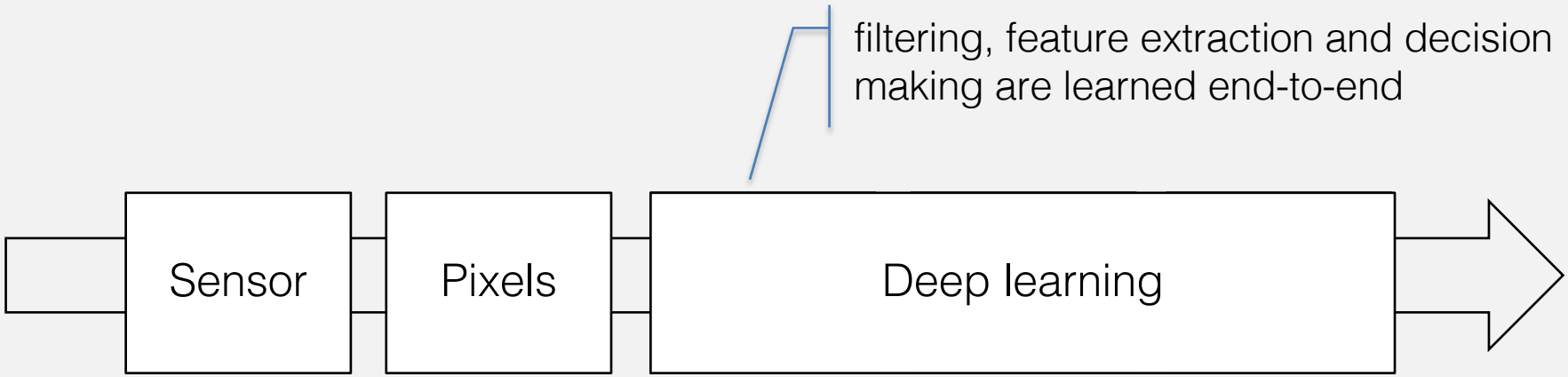


Image source: https://dudeperf3ct.github.io/

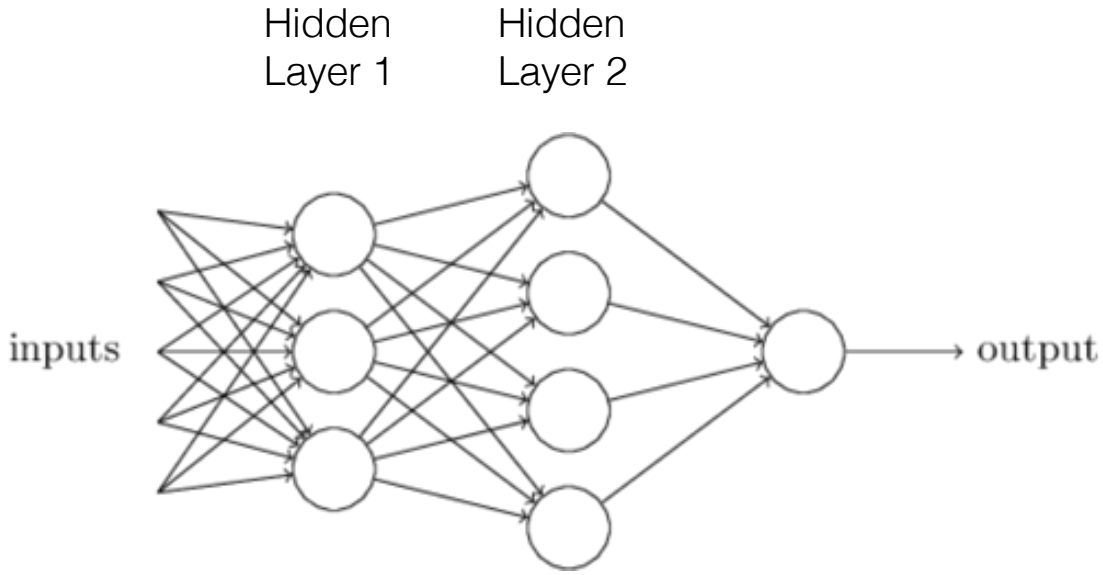


Deep learning → very large and deep neural networks



Deep learning pipeline

Multi-layer perceptron (MLP)



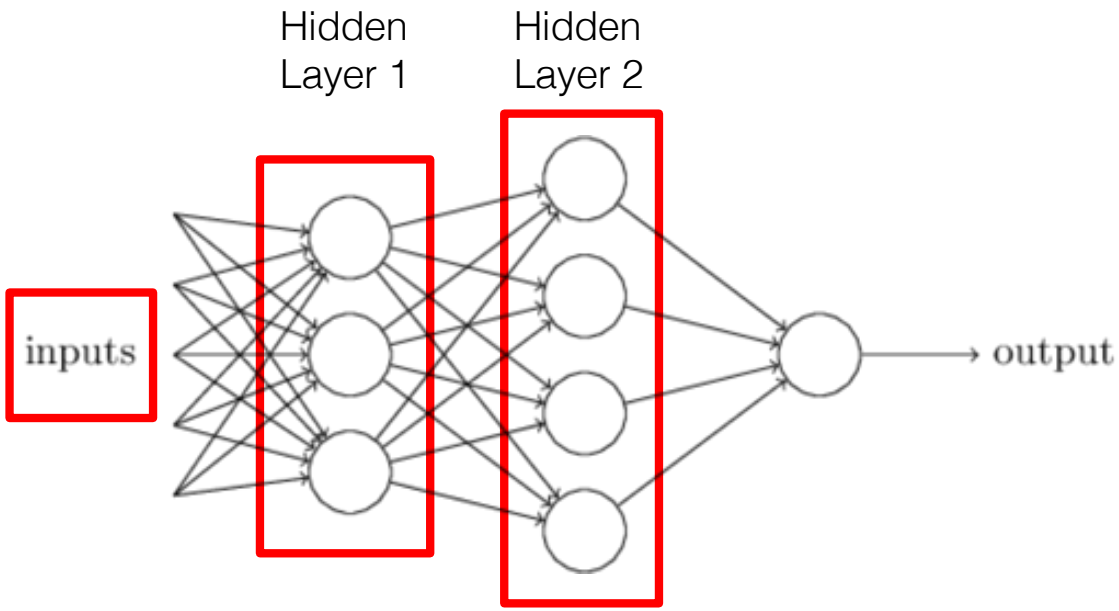
Sets of layers and the connections (weights) between them define the network architecture.

Each layer receives its inputs from the previous layer and forwards its outputs to the next layer

Explanation in 3Blue1Brown about the multi-layer perceptron
https://www.youtube.com/watch?v=aircAruvnKk

adapted from Nielsen

Multi-layer perceptron (MLP)



Multiple

Matrix Matrix Matrix

It's all just matrix multiplications!

Key-factor #1: GPUs -> special hardware for fast/large matrix multiplication.

adapted from Nielsen

Training an MLP

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial L}{\partial w_{ji}}$$

$\frac{\partial L}{\partial w_{ji}}$ → Error or Loss

η → Learning rate or step length

- Typically done using **sequential gradient descent**, i.e. in each iteration (step), calculate the error and update the weights
- A complete pass over the training set is called an epoch

- How can we compute the gradient efficiently given an arbitrary network structure?
- Answer: backpropagation algorithm
  - propagating errors backward into the network

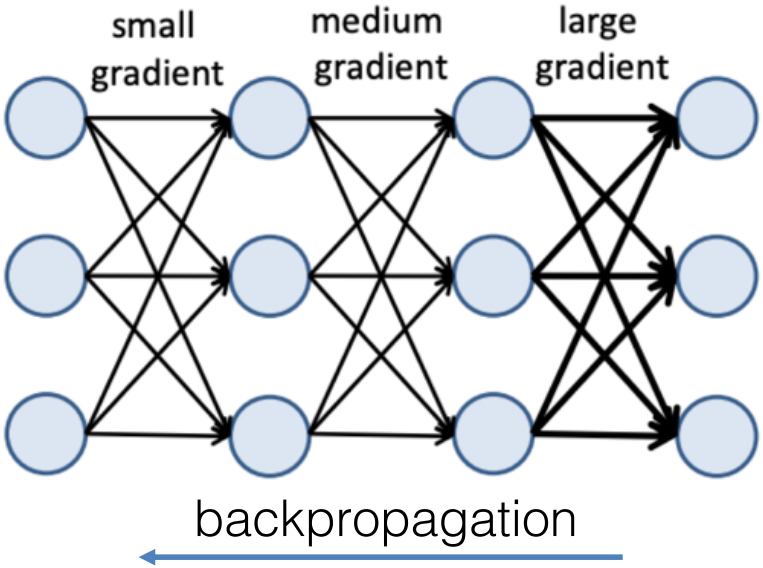
Explanation in 3Blue1Brown about the training process
https://www.youtube.com/watch?v=IHZwWFHWa-w

Deep Neural Networks

- Advantage: high expressivity
  - A NN with a single hidden layer could be enough with large (infinite) number of units can approximate any function
  - However, deep NNs allow for more compactness → with more layers, the number of units needed may decrease exponentially (with the number of layers)

Challenges in deep NNs

- Deep neural networks of sigmoid and hyperbolic units often suffer from **vanishing gradients**



- ReLU and other training strategies helped overcome this
- In fact, both the forward and backward passes have *L* matrices multiplications → can lead to numerical issues

Key-factor #2: new training training strategies

adapted from Pascal Poupart

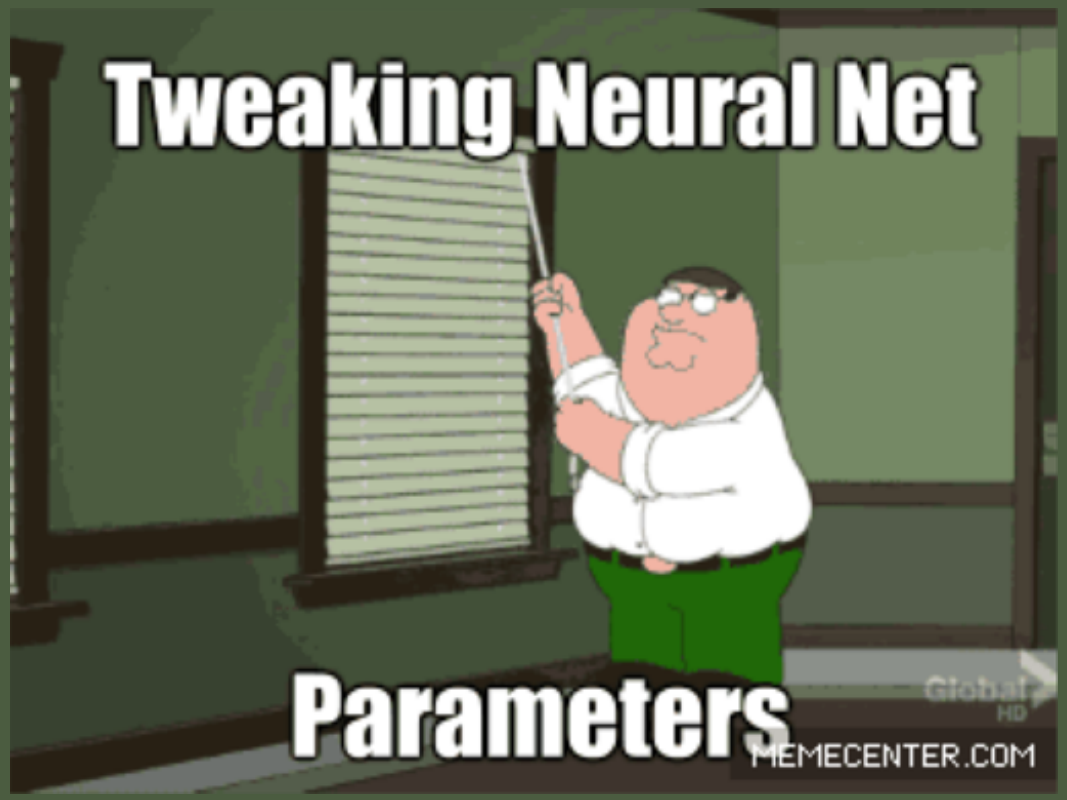
Overfitting

- There is serious risk of **overfitting**, since typically the number of parameters is **much larger** than the amount of data

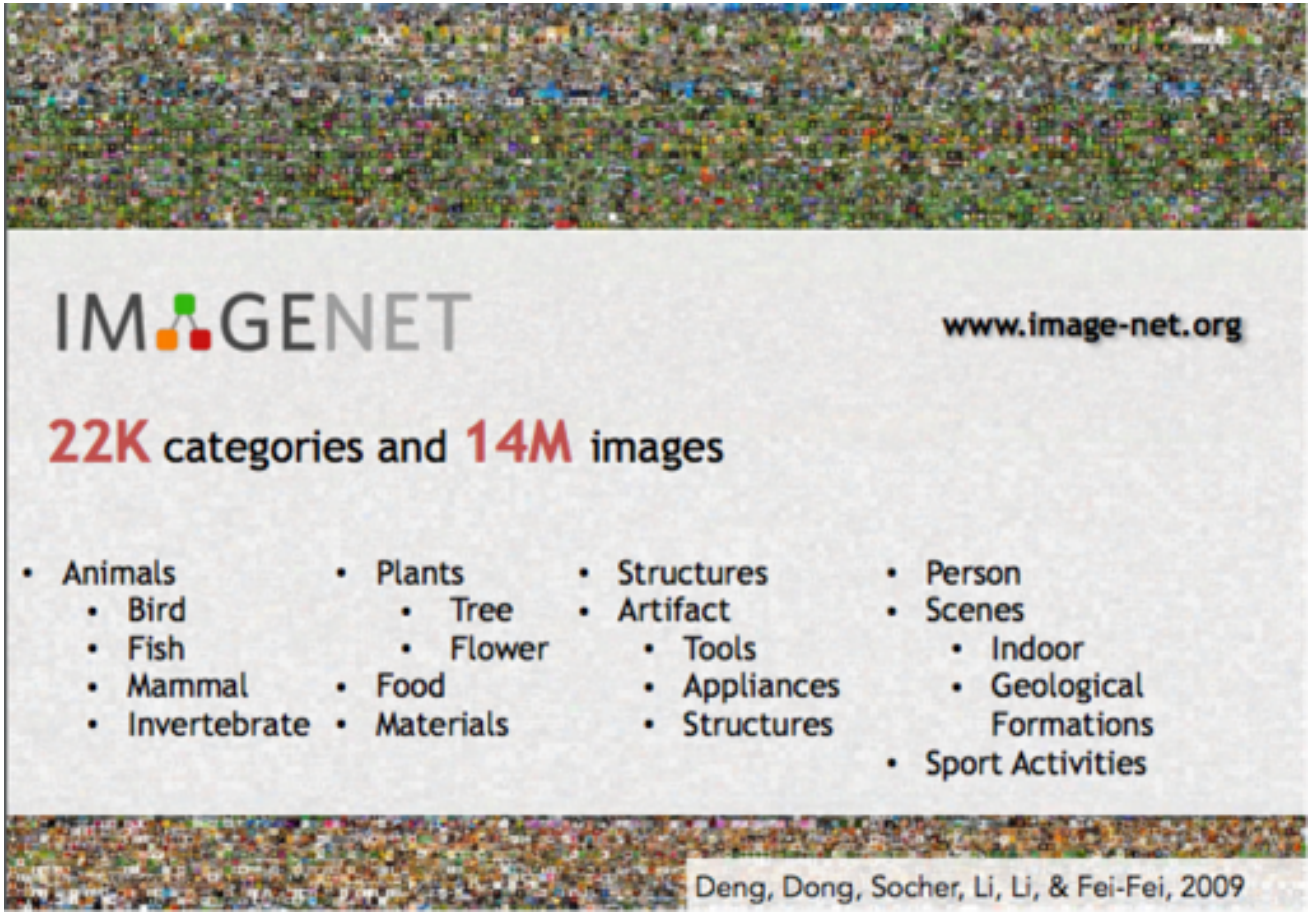
- Some solutions / regularization strategies:
  - Early stopping
  - Transfer learning - starting from another task with more samples
  - Multi-task learning
  - L1 or L2 penalty term in the Loss (penalizes large weights)
  - Dropout
  - Data augmentation

Overfitting

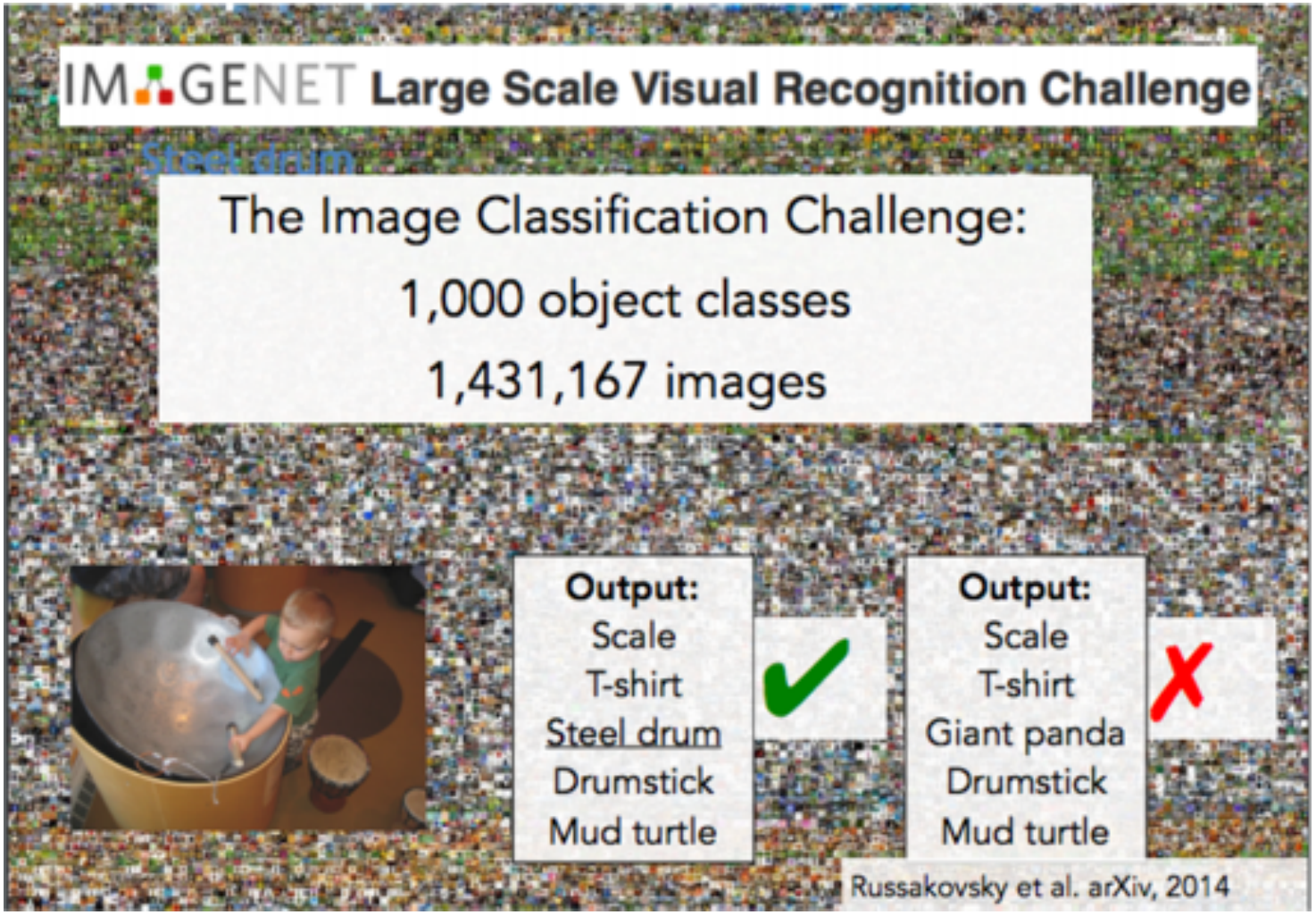
- Dropout Perturbations in the network
  - randomly “drop” some units from the network when training
- Data augmentation Perturbations in the input data
  - create new data by applying transformations to the given dataset


Tweaking Neural Net
Parameters
MEMECENTER.COM

What about data?

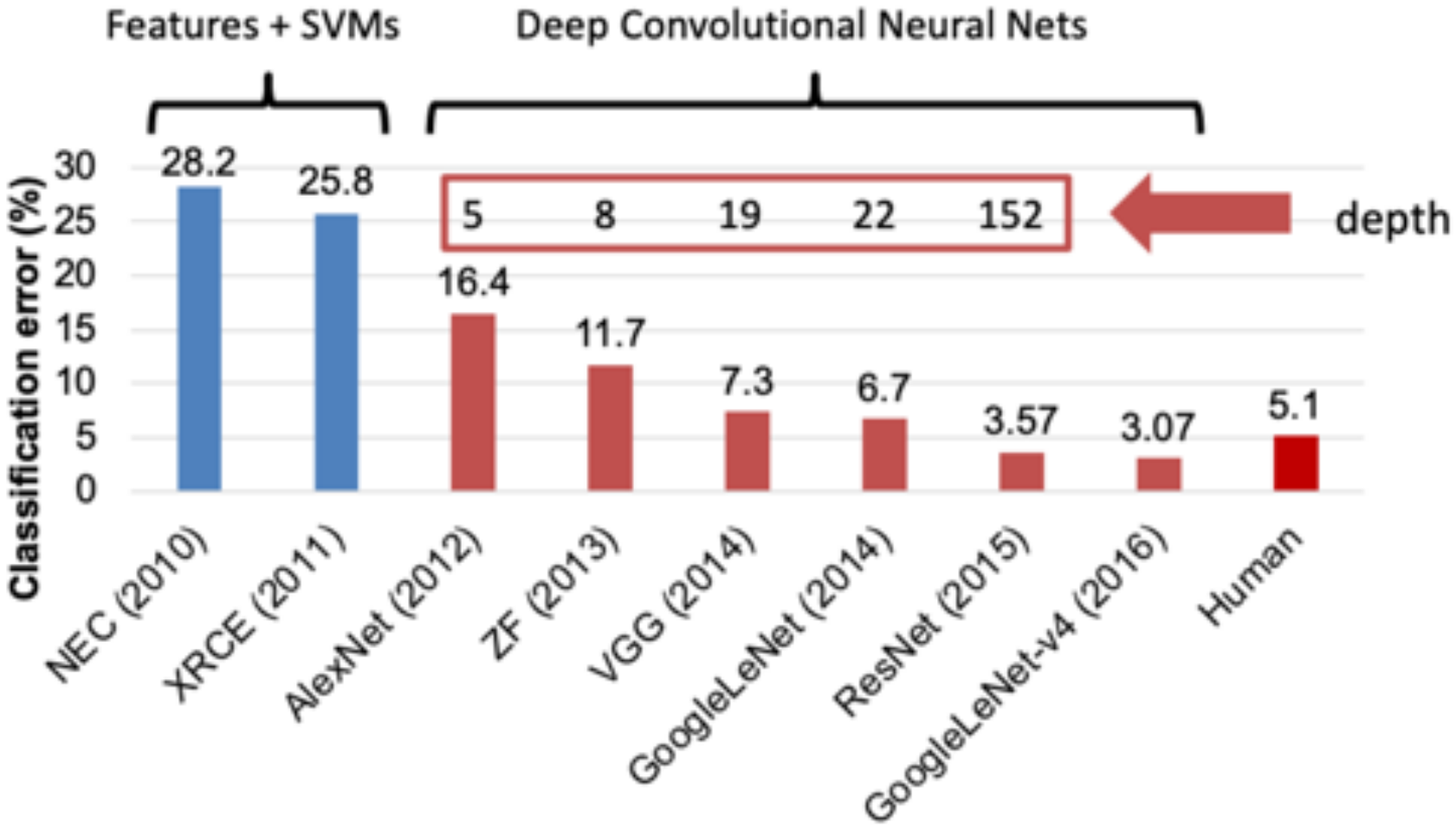


Key-factor #3: large amount of data to train the models

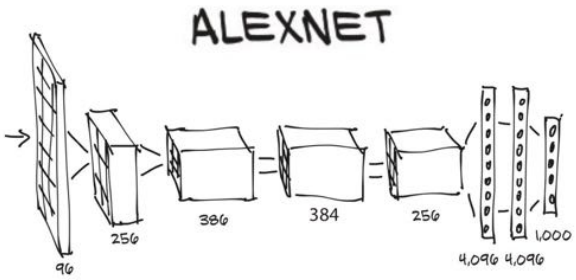

IMAGENET Large Scale Visual Recognition Challenge
Steel drum
The Image Classification Challenge:
1,000 object classes
1,431,167 images
Output:
Scale
T-shirt
Steel drum
Drumstick
Mud turtle
Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle
Russakovsky et al. arXiv, 2014

Impact of DL on image classification

ImageNet Large Scale Visual Recognition Challenge



Deep learning revolution



Better Architectures and Learning Strategies

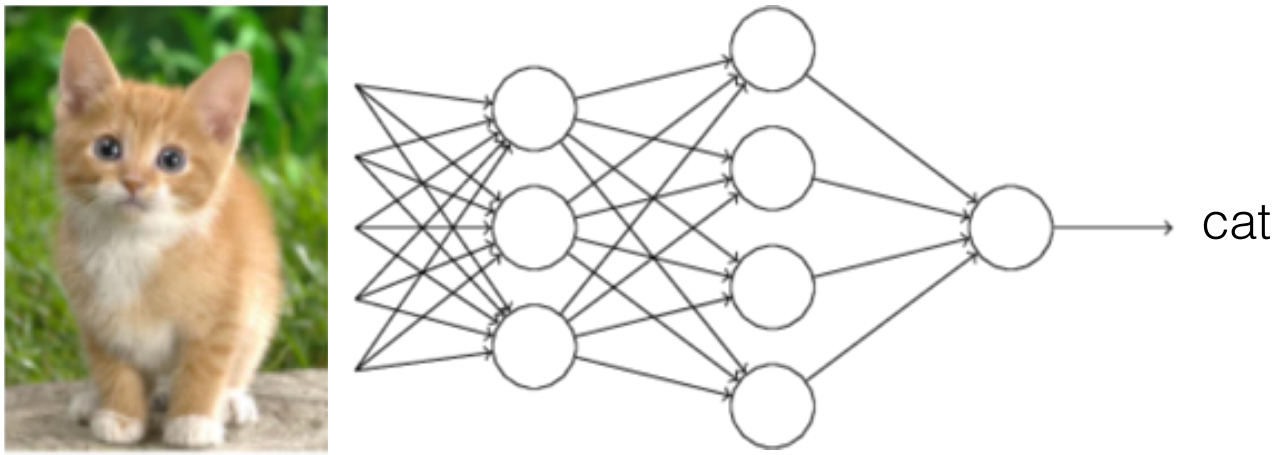
+

A Looooot More Data

+

Processing Power

Using MLPs with images



Can we make this work?

Using MLPs with images

- Image needs to be represented as a long vector…

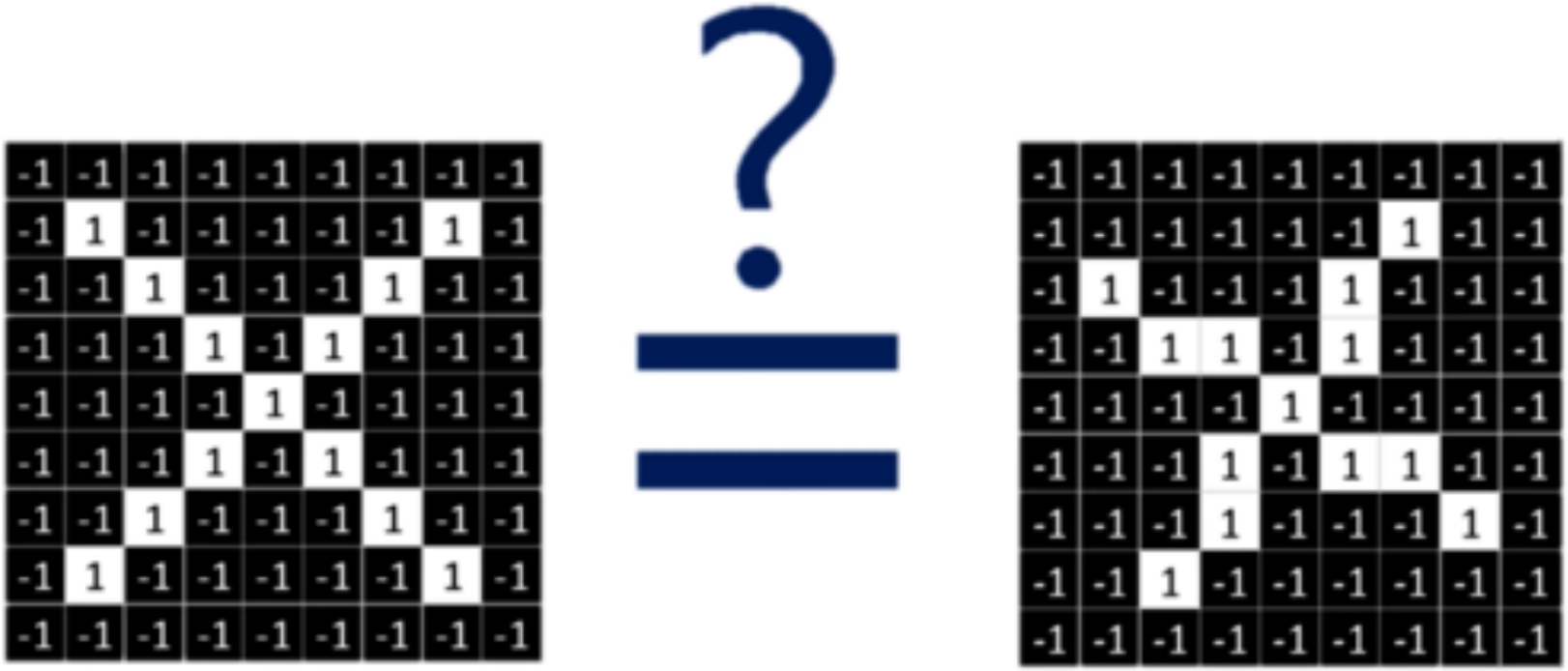
… losing important spatial information.

- A weight vector that learns to match a specific feature in cat images will not match the same feature in the same but shifted or scaled image

  → The MLP is not invariant to image translation or scaling

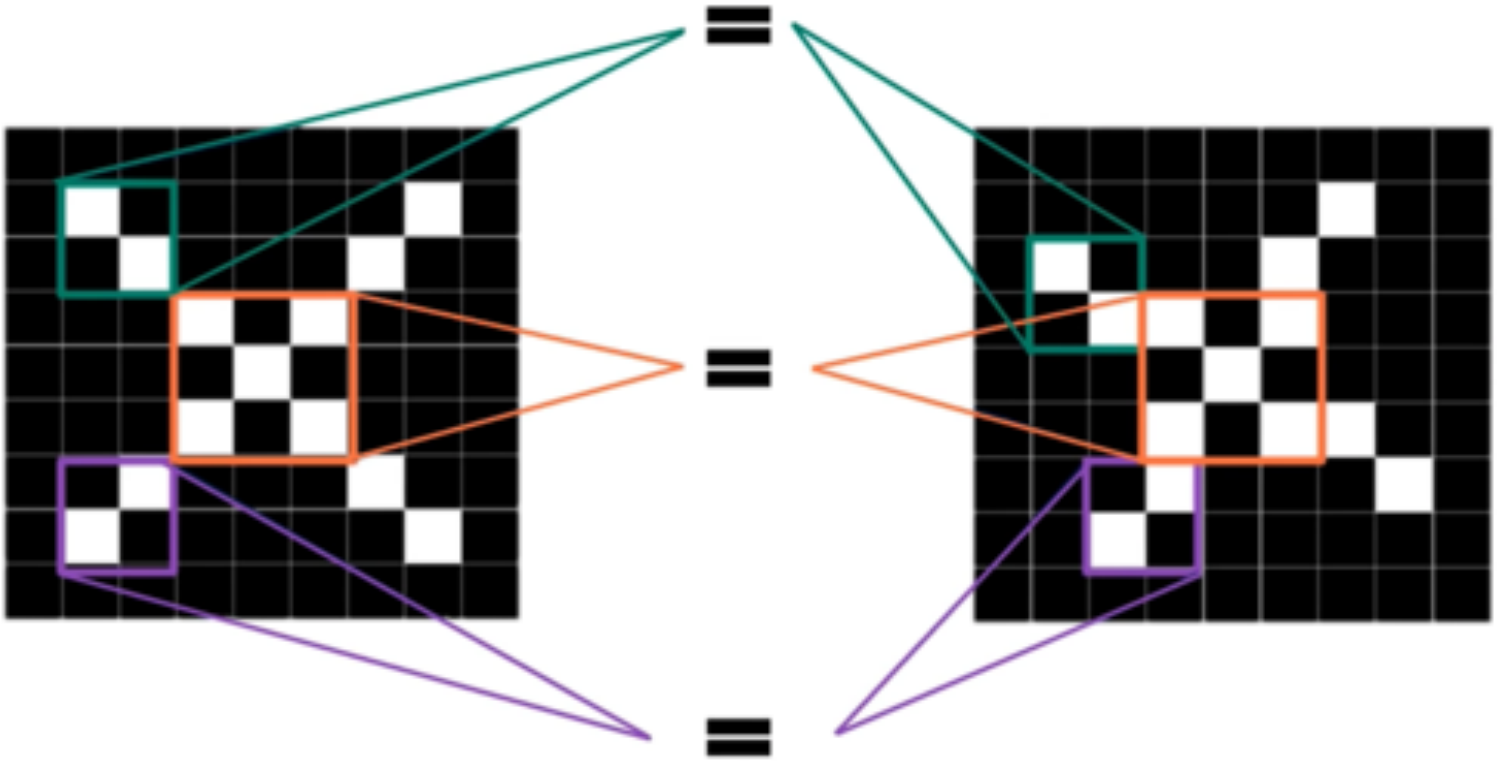
- Also, the MLP model has many weights that are not learning anything useful for images

Interpreting Images



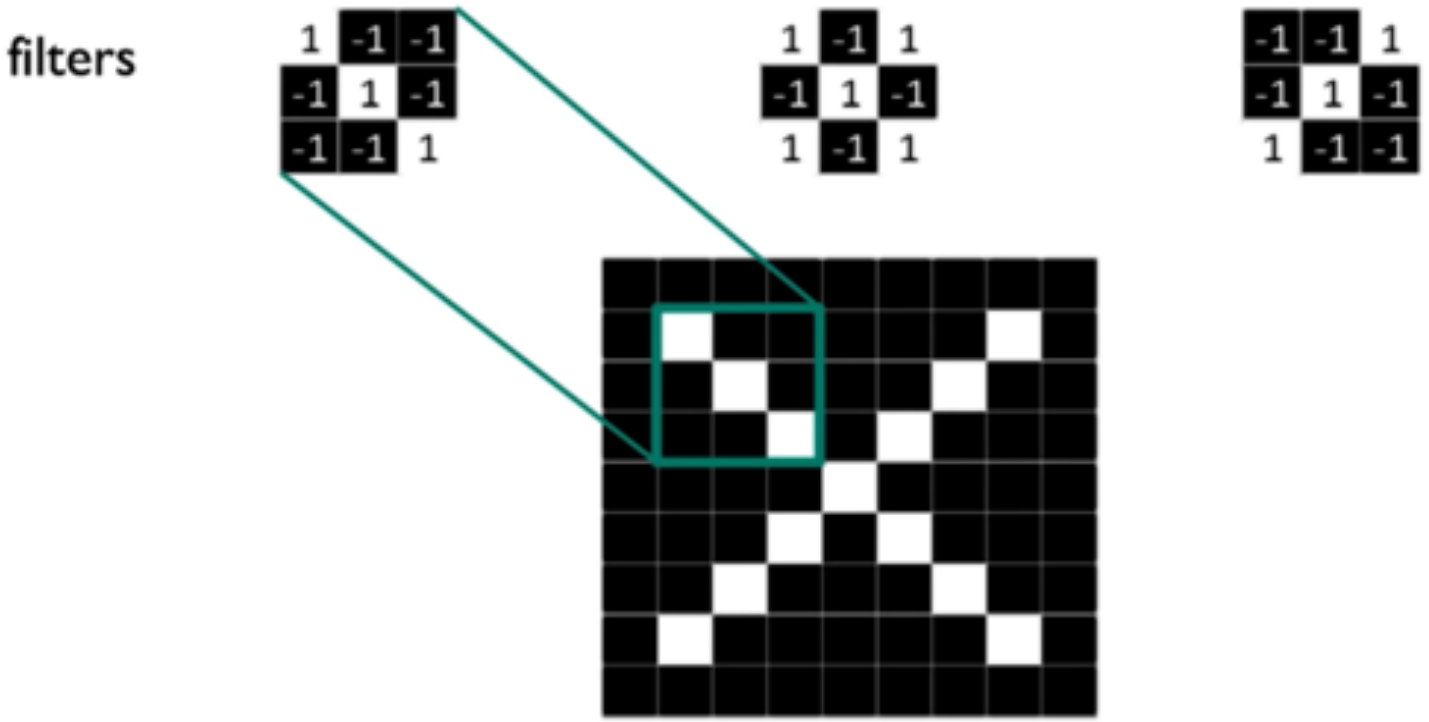
We want to be able to classify an object in the image, even if shifted, resized, rotated, deformed.

Interpreting Images



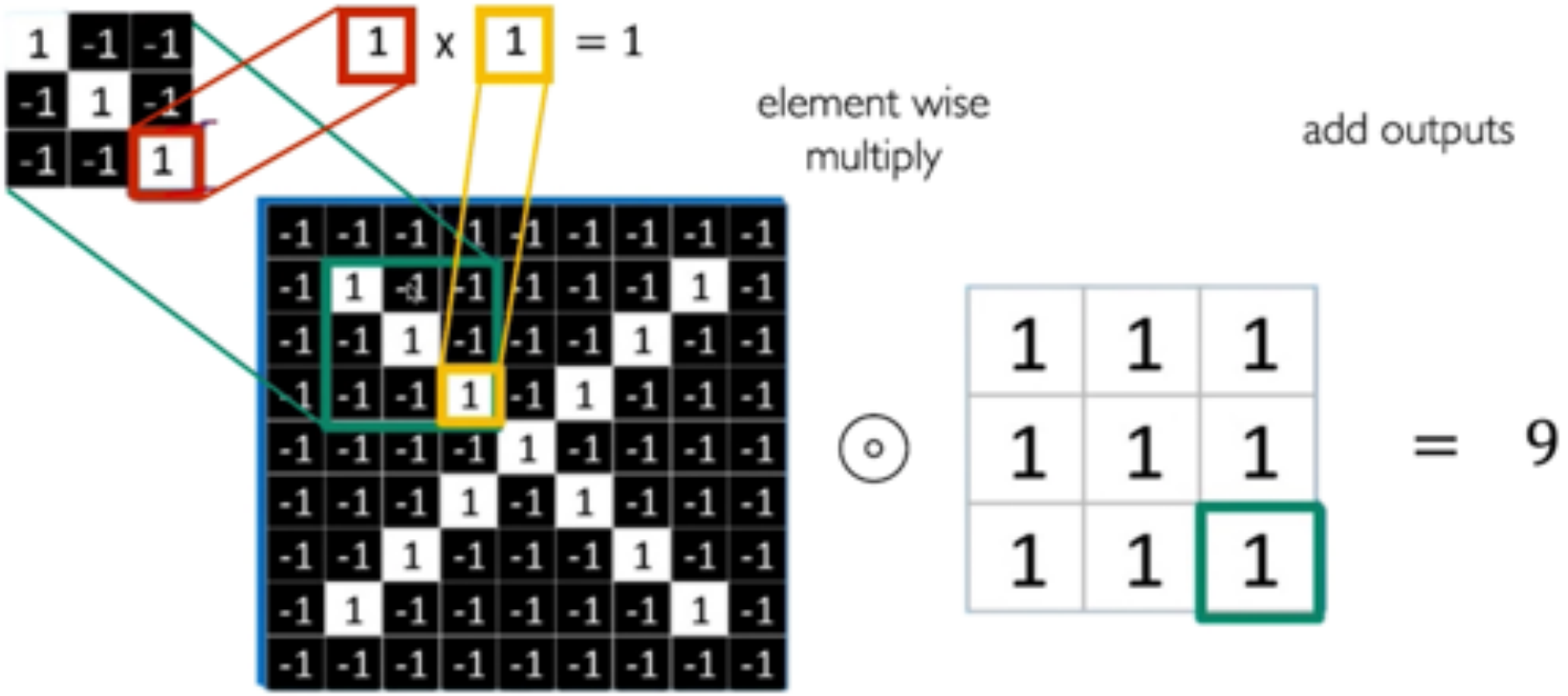
Instead of comparing the whole image, we should compare **local features**

Feature Extraction with Convolution



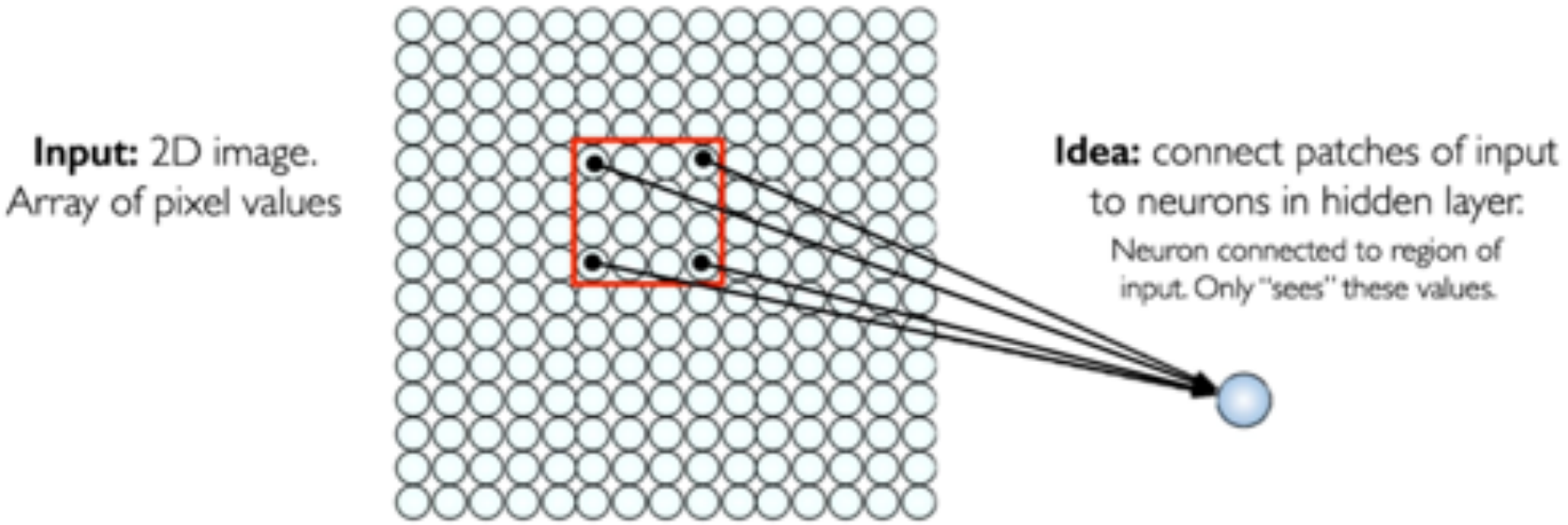
The features are extracted using filters based on a **convolution operation**

Convolution



Convolution

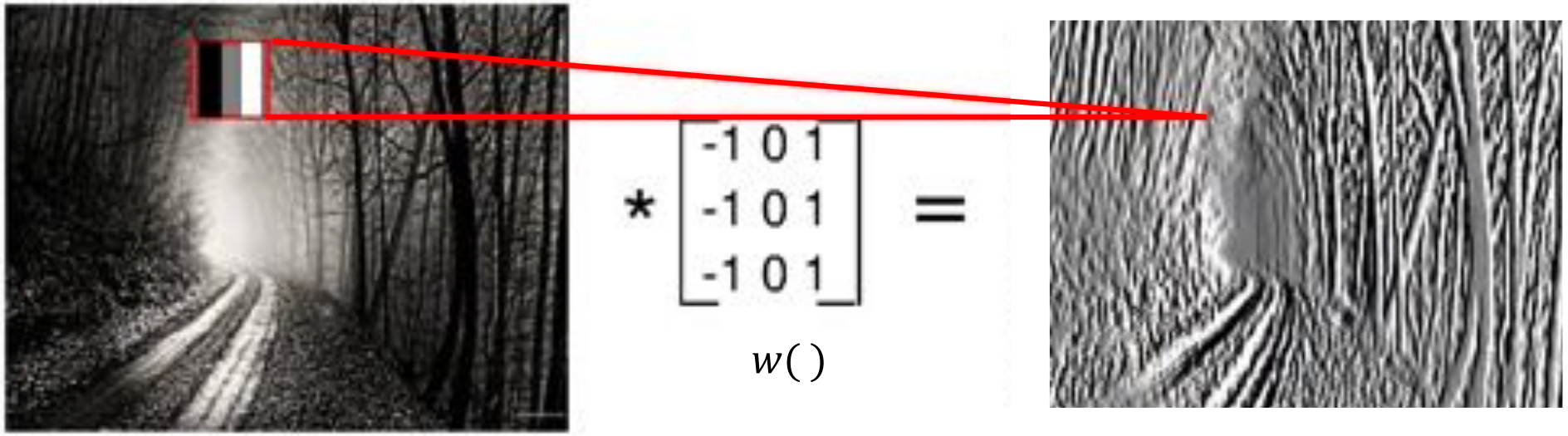
Using Spatial Structure



But, how do we analyse the complete image?

- Multiple neurons with shared weights
- In practice, we use a sliding window to go through the image

Convolution



$x()$ $w()$ $y()$

A filter for edge detection

Explanation in MIT 18.S191 about convolutions in images
https://www.youtube.com/watch?v=8rrHTtUzyZA

Convolution

- An example

$$x = \begin{bmatrix} 3 & 3 & 2 & 1 & 0 \\ 0 & 0 & 1 & 3 & 1 \\ 3 & 1 & 2 & 2 & 3 \\ 2 & 0 & 0 & 2 & 2 \\ 2 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad w = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

$$y = x * w = \begin{bmatrix} \quad ? \quad \end{bmatrix}$$

Convolutional Neural Network

- A Convolutional Neural Network (CNN) is a particular case of a NN with many layers.
- Two main types of layers:
  - Convolutional
  - Pooling
- CNNs have an inductive bias for images
  - closer to how information is structured in an image
  - or other data type where spatial relationship is relevant
- Depth allows to go from local to global

From Local to Global

- The **receptive field** is defined as the region in the input space that a particular CNN feature is looking at (i.e. be affected by)
- Increases as we go deeper in the network

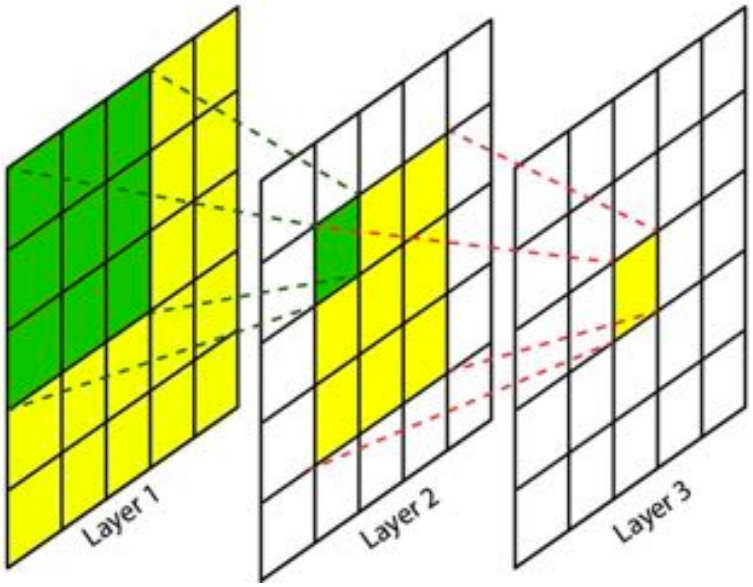
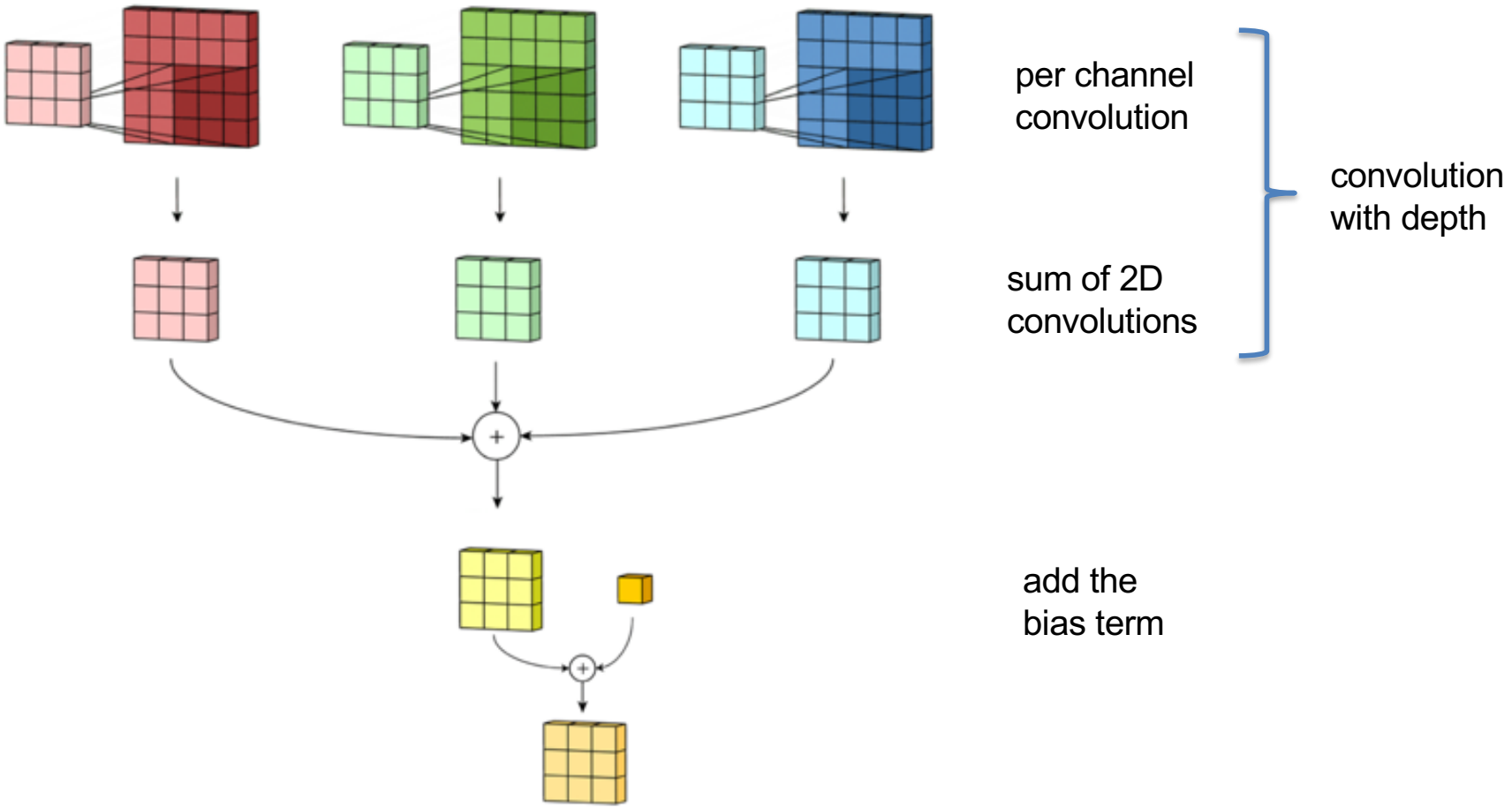


Image Source: 10.3390/rs9050480

Convolution Layer

Channels of the input image (usually RGB) are represented as depth of a 3D block
→ Filters' outputs are also represented as channels in the depth dimension

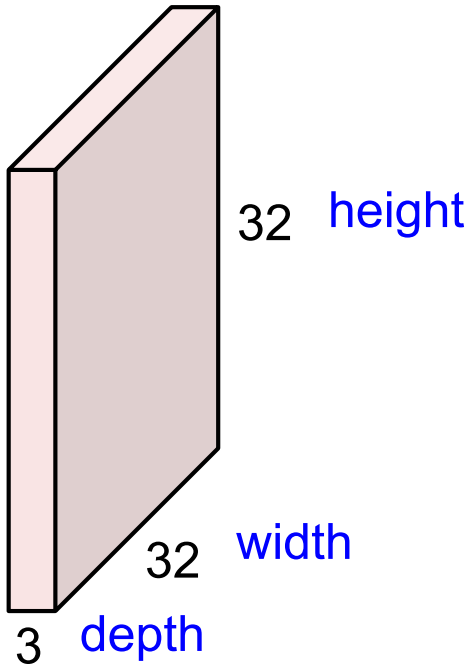


Source: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

adapted from José Luis Alba

Convolution Layer

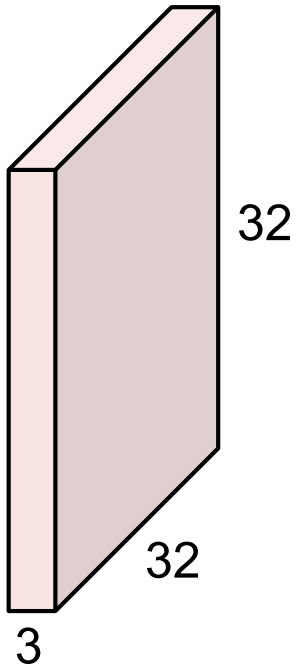
32x32x3 image



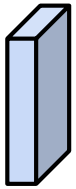
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

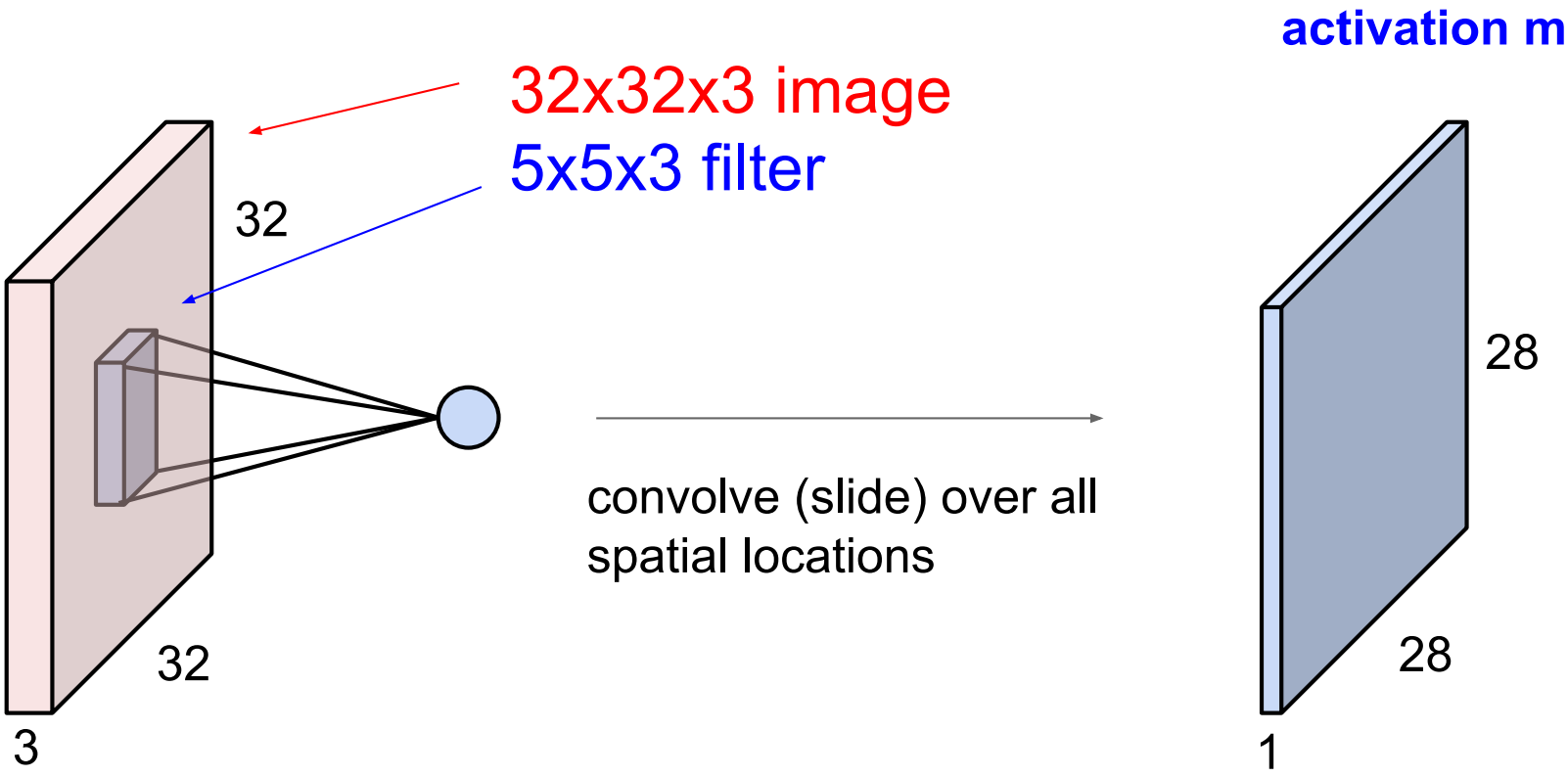


5x5x3 filter



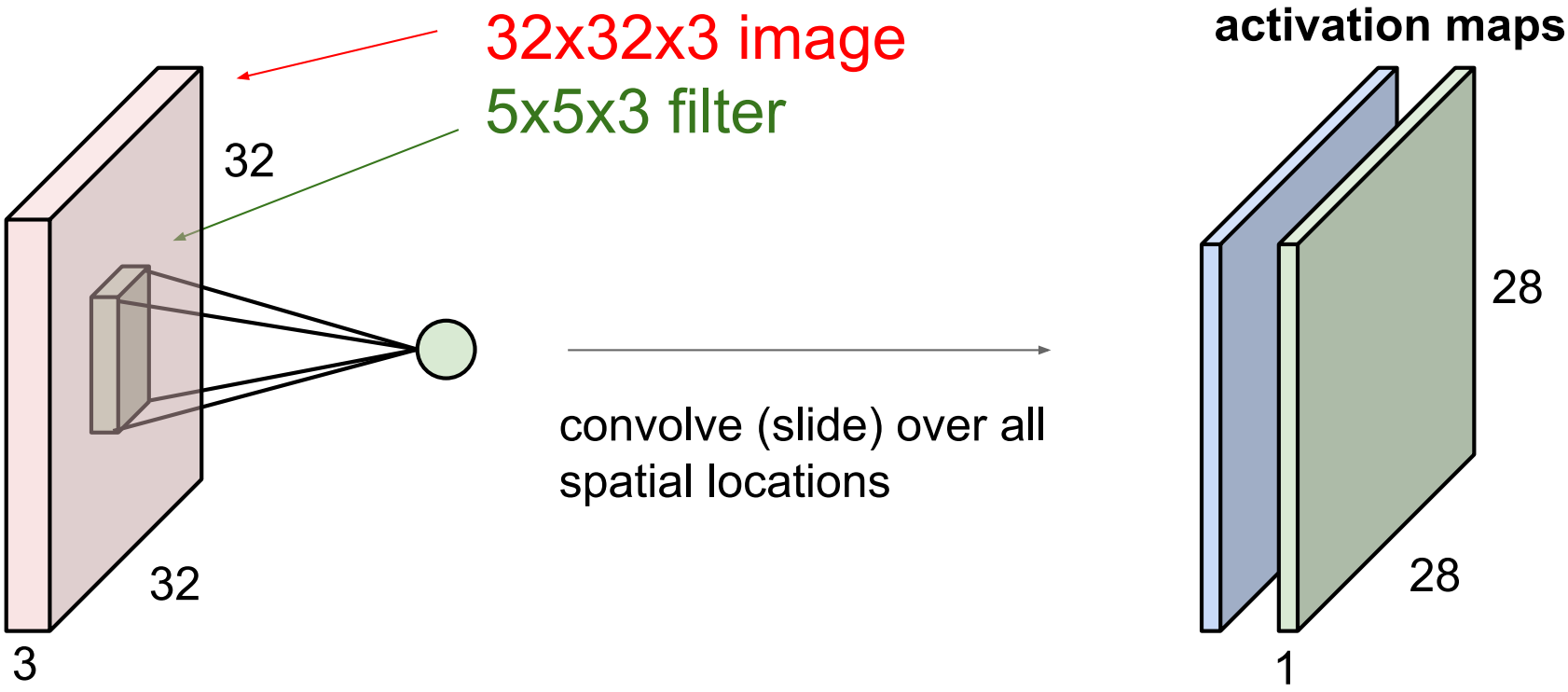
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

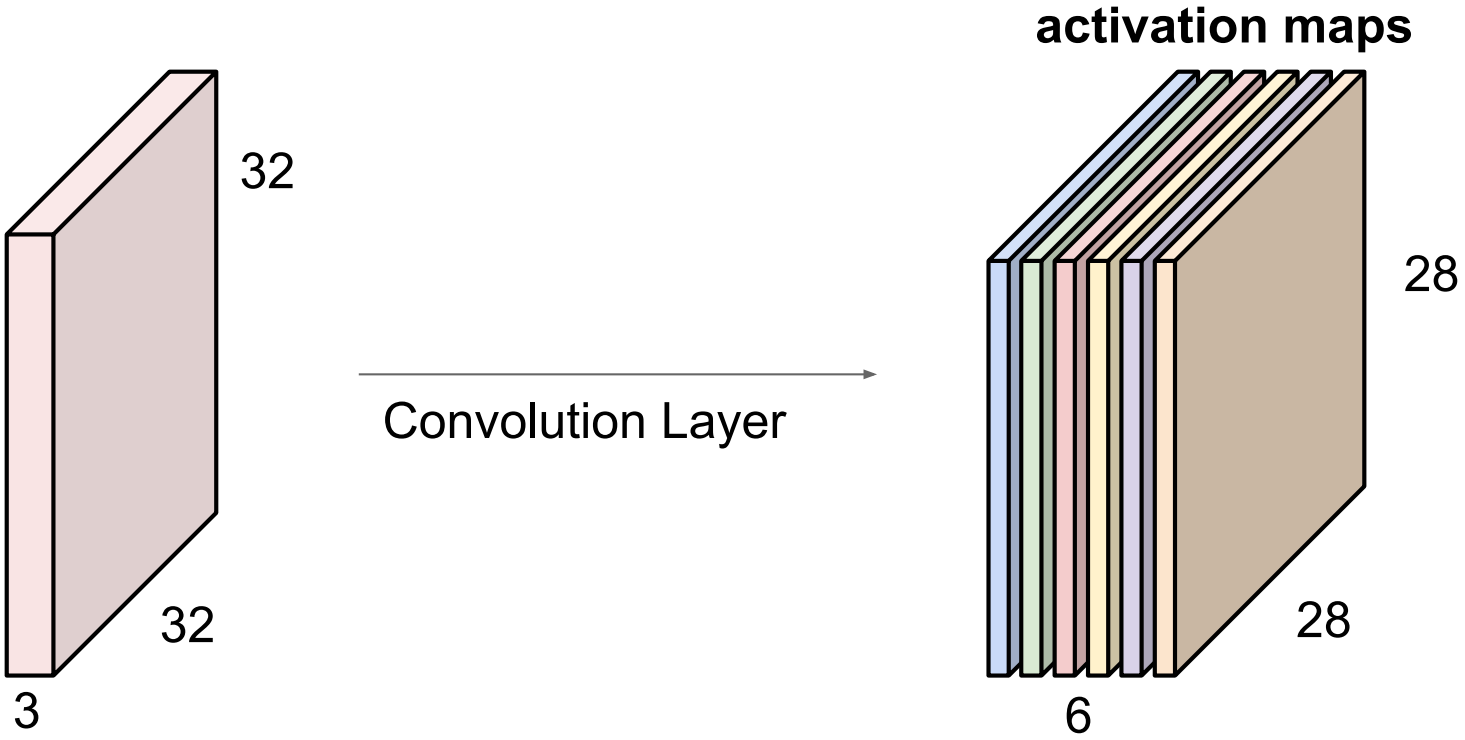


Convolution Layer

consider a second, green filter

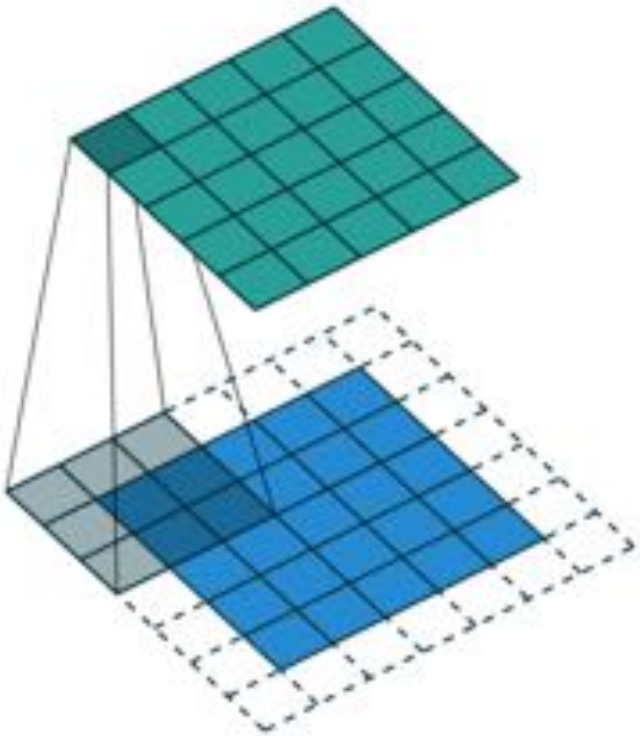


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

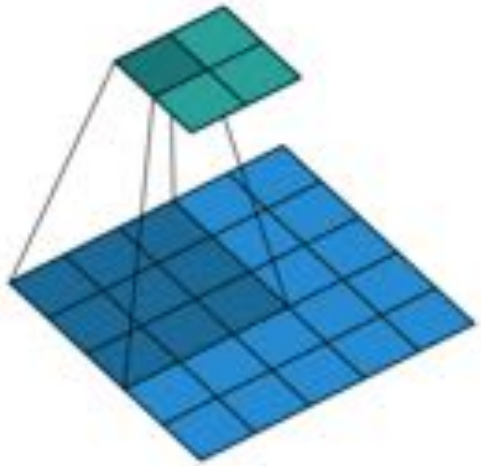


We stack these up to get a "new image" of size 28x28x6!

Padding and Stride



padding - avoids shrinking due to the border pixels



strided convolutions - use a step > 1 (usually the same for all dimensions, but mandatory)

Hyperparameters*

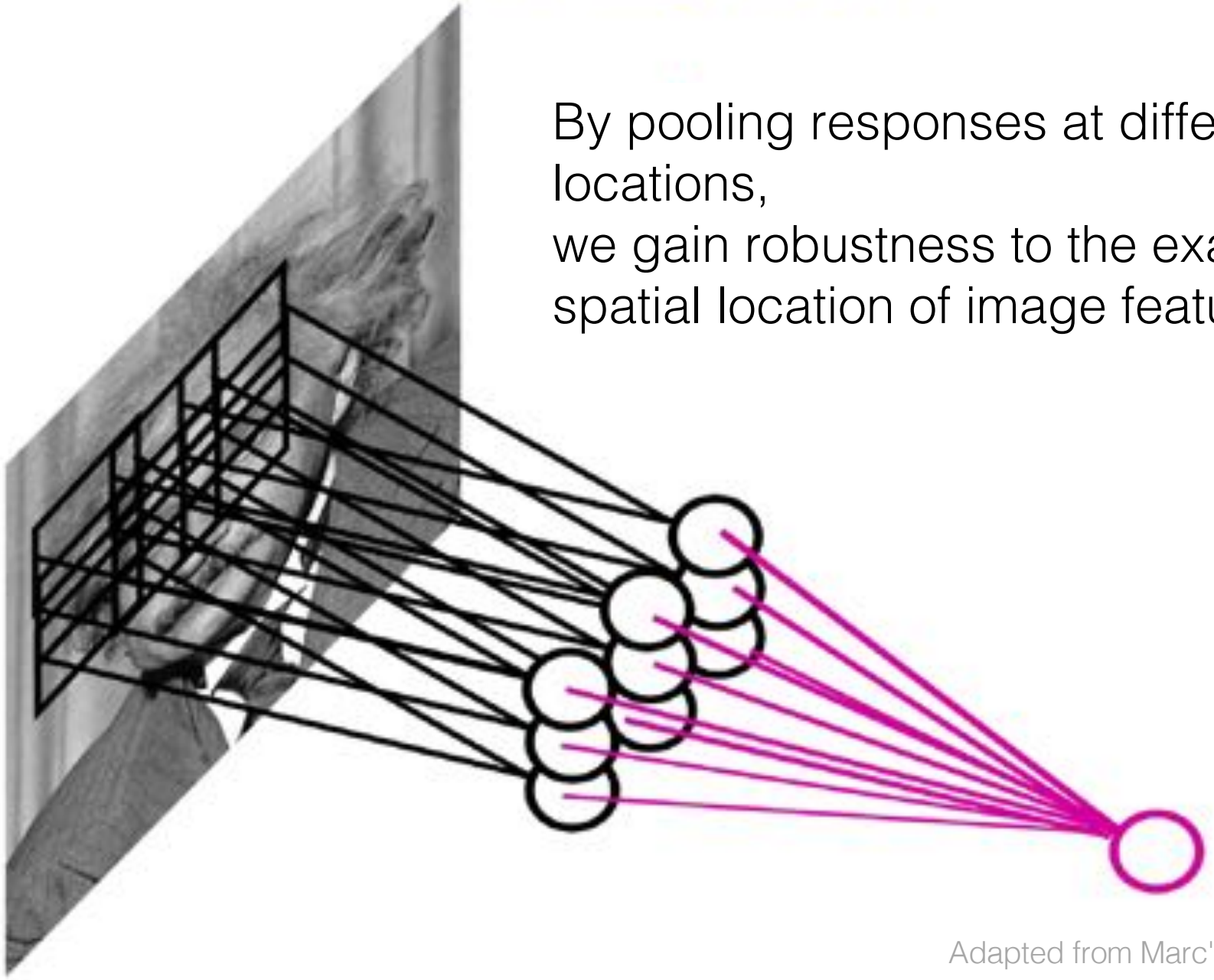
- **# of filters**: integer indicating the # of filters applied to each window
- **kernel size**: tuple (width, height) indicating the size of the window
- **stride**: tuple (horizontal, vertical) indicating the horizontal and vertical shift between each window.
- **padding**: boolean to indicate if the input is padded with a border of zeros to ensure that the output has the same size as the input.

*hyperparameter are used to control the learning process; parameters are learned

Exercise

- Calculate the **output volume size** {width, height, channels} of a convolutional layer composed of 10 filters of size 3x3 without padding and stride of 1, when applied to a color input image of size 6x6.
- Also, **how many parameters** (learnable) does this layer have?

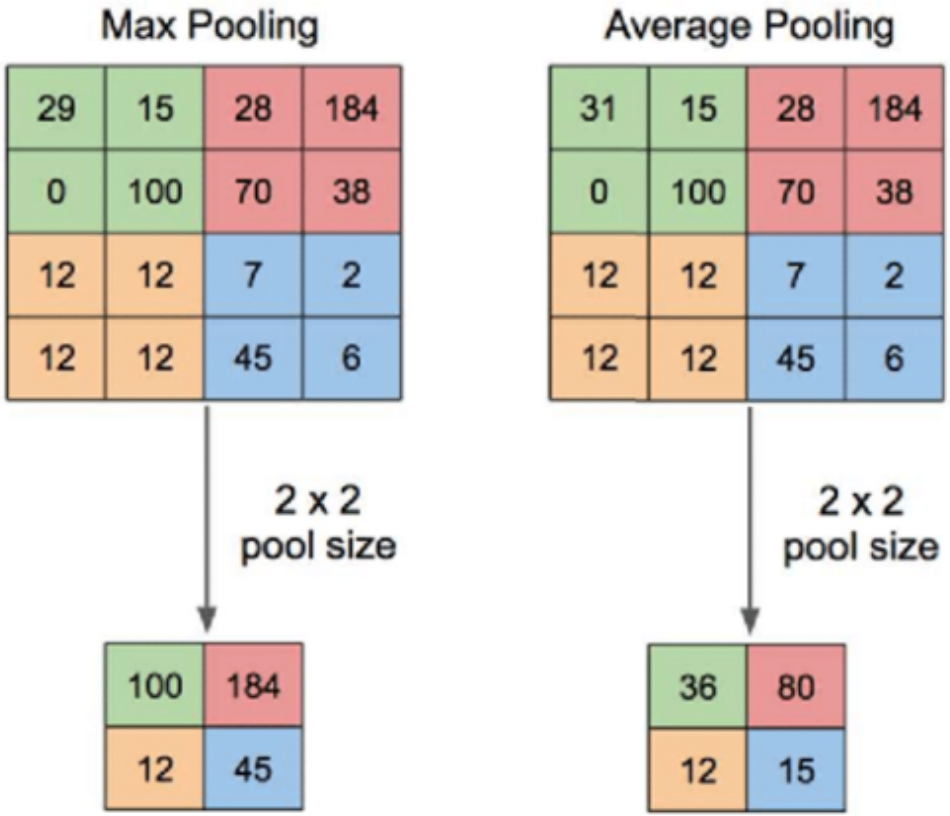
Pooling Layer



By pooling responses at different locations,
we gain robustness to the exact spatial location of image features.

Adapted from Marc'Aurelio Ranzato

Pooling Layer

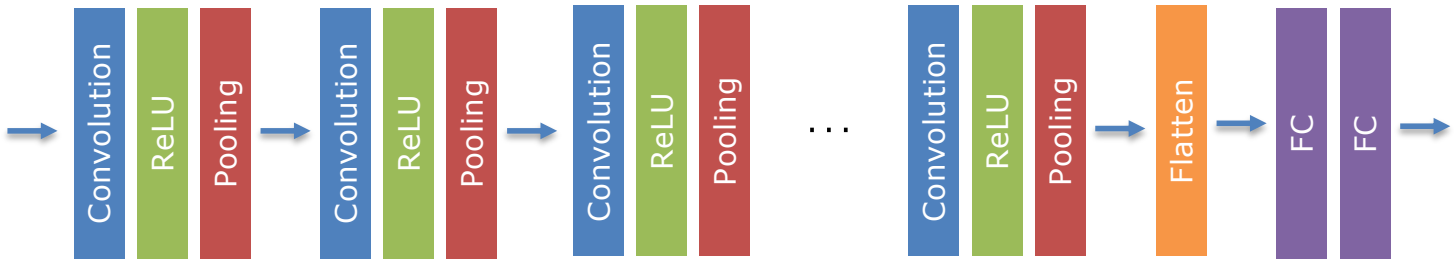


The operations are done per channel → same depth as input

How many parameters?
And how many hyperparameters?

Building blocks of a CNN

- Stacking Convolutional and Pooling layers we can build a NN with several interesting properties.

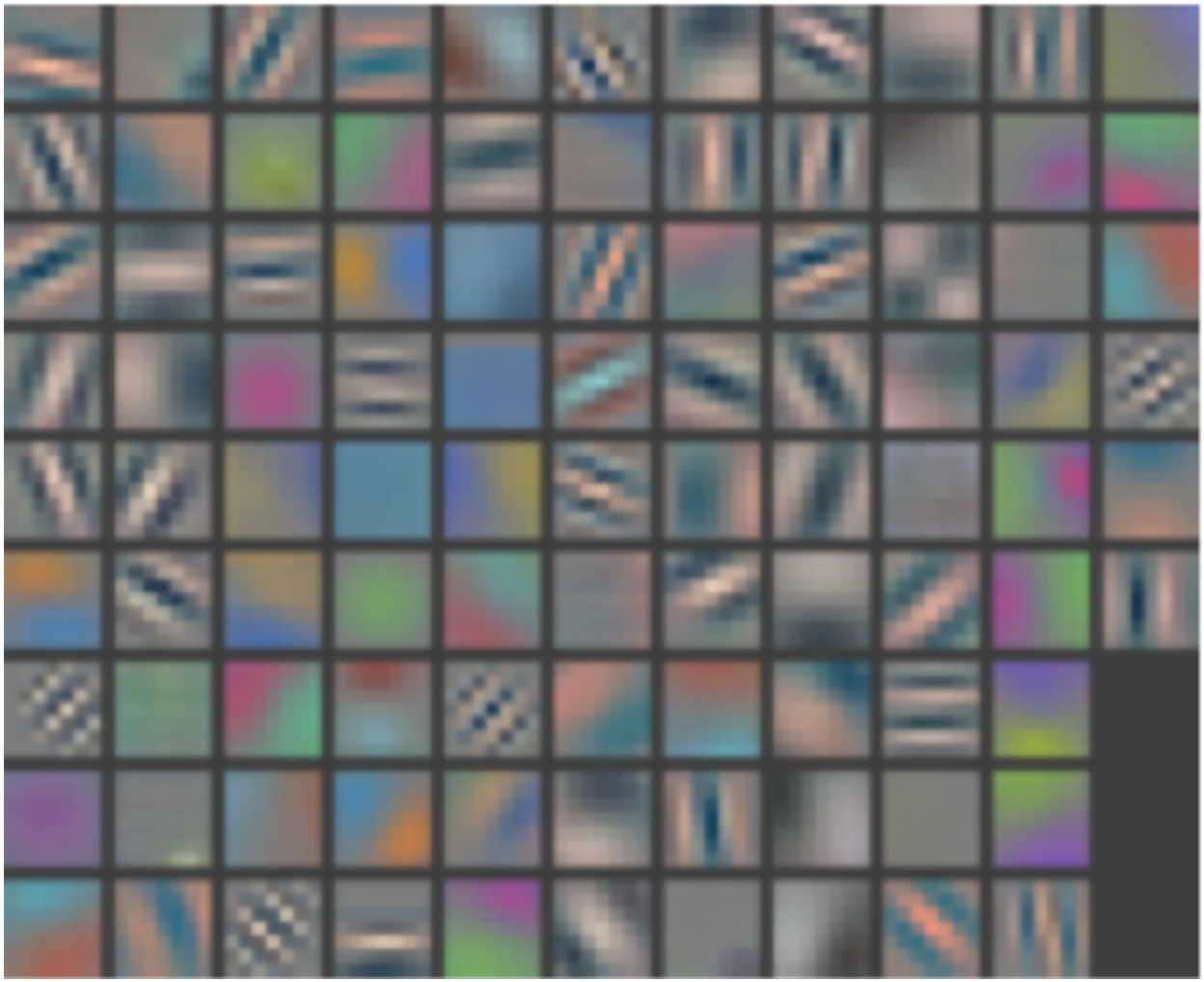


- Inserting a non-linear activation function we achieve non-linear **complex transformations**.
- Inserting pooling layers we can downsample the spatial information and **trade with more feature filters** learnt during training.
- Each new layer learns on top of previously filtered images with **less spatial information and more extracted features**.
- Inserting a final classifier (e.g. fully-connected MLP) over a "flattened" image allows to do classification or regression, not on the raw pixels but on **learnt features of the image**.

adapted from José Luis Alba

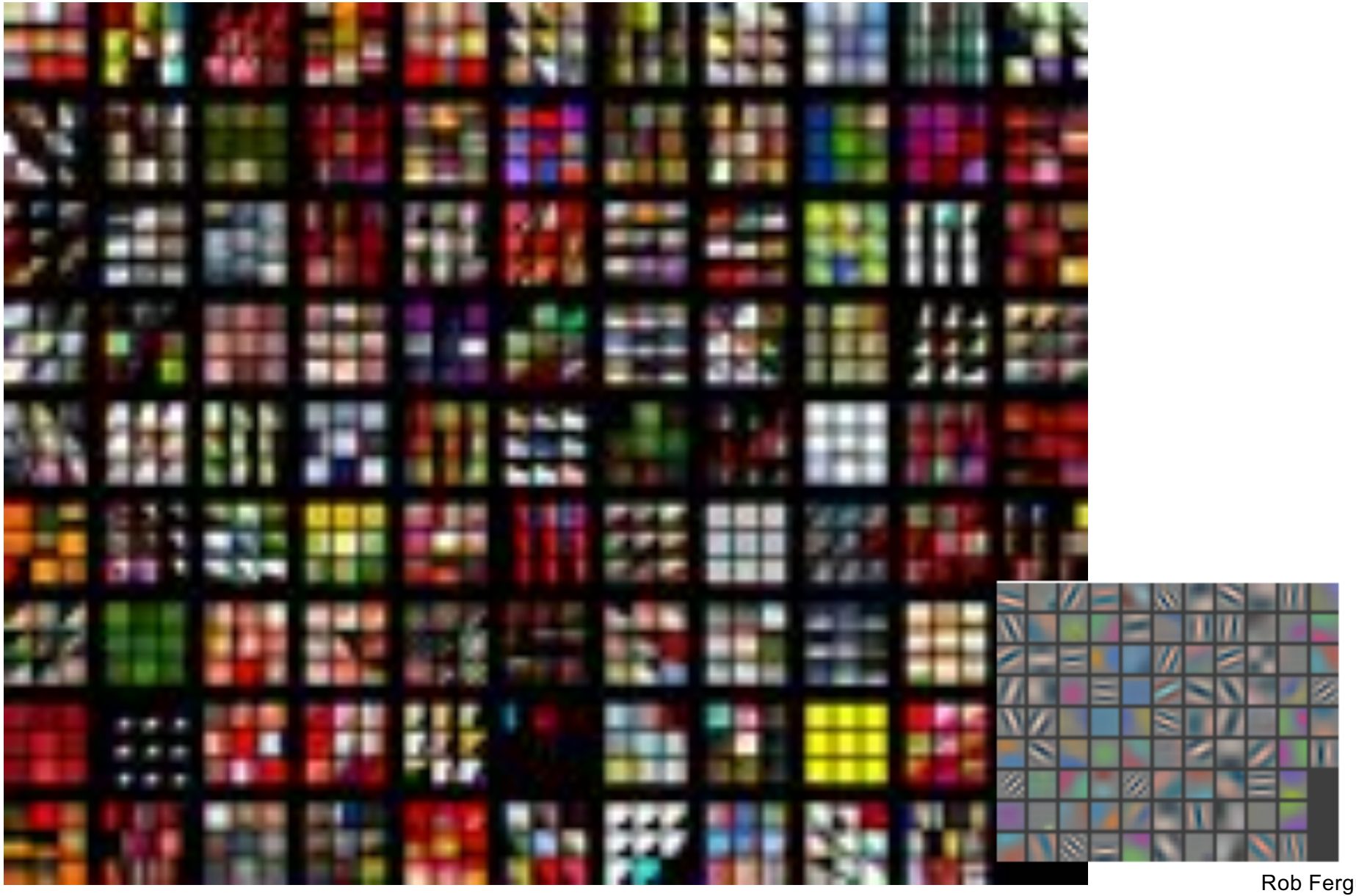
What is a CNN learning?

Layer 1: Filters



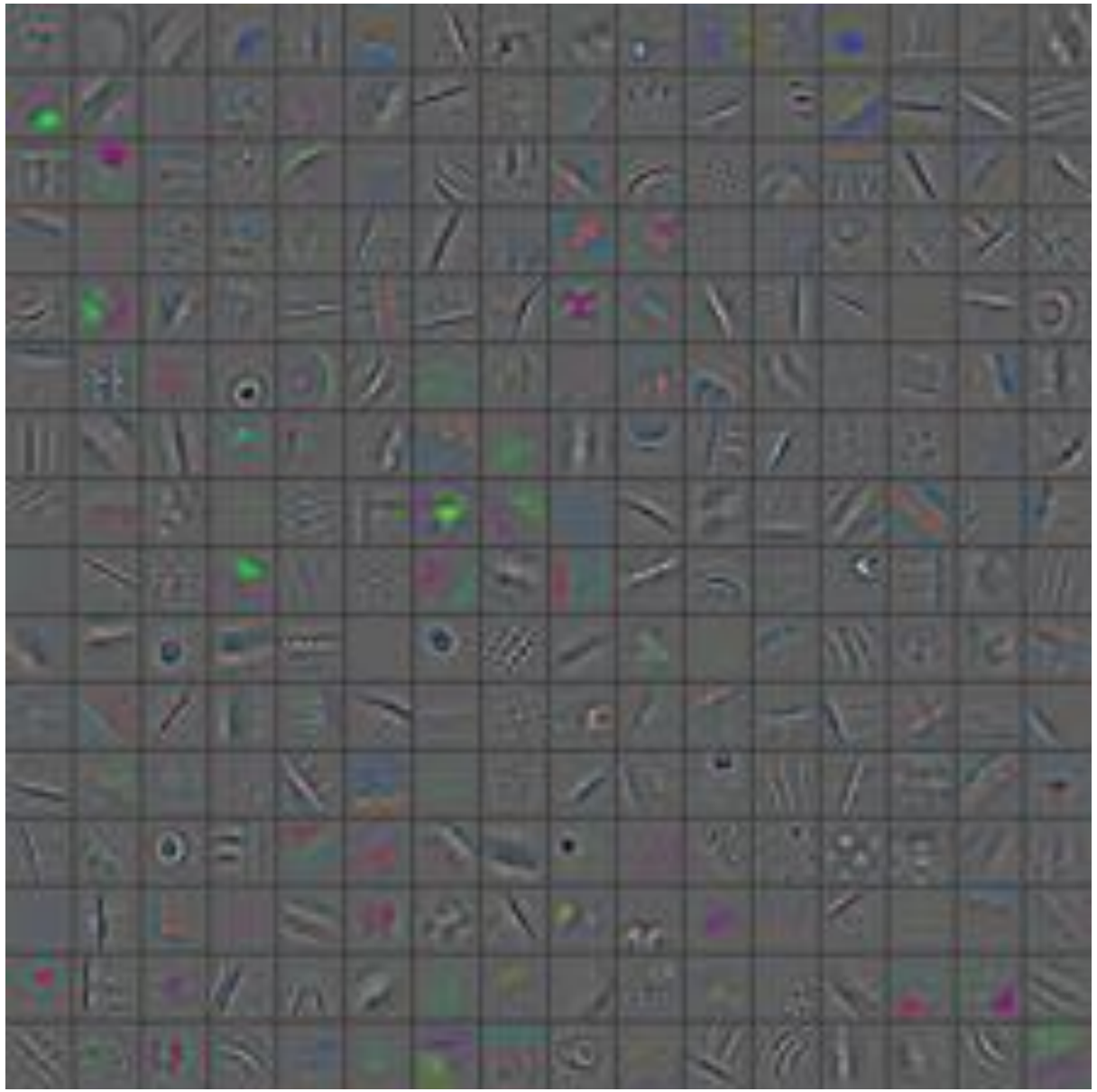
Rob Fergus

Layer 1: Top-9 Patches

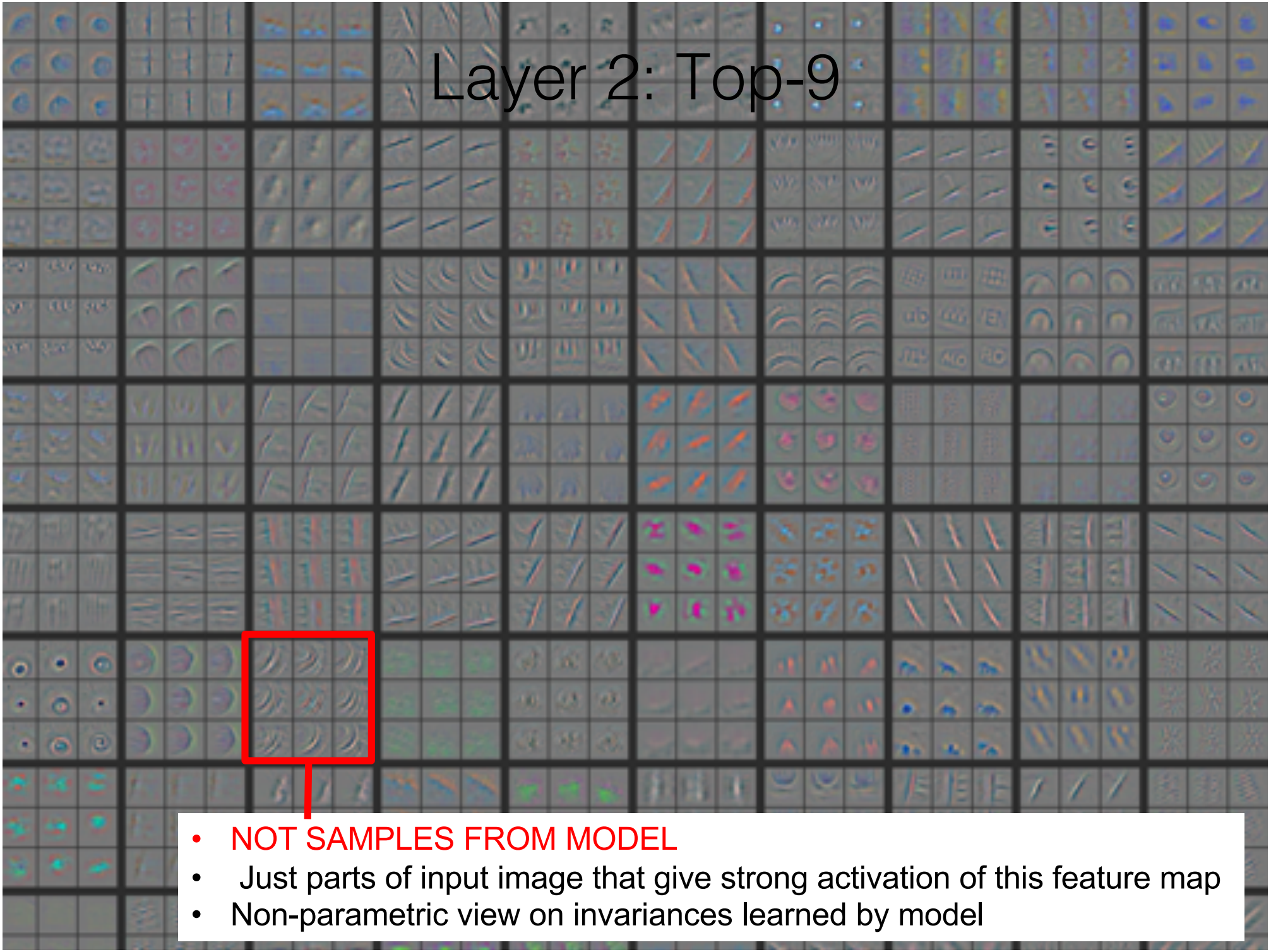


Rob Fergus

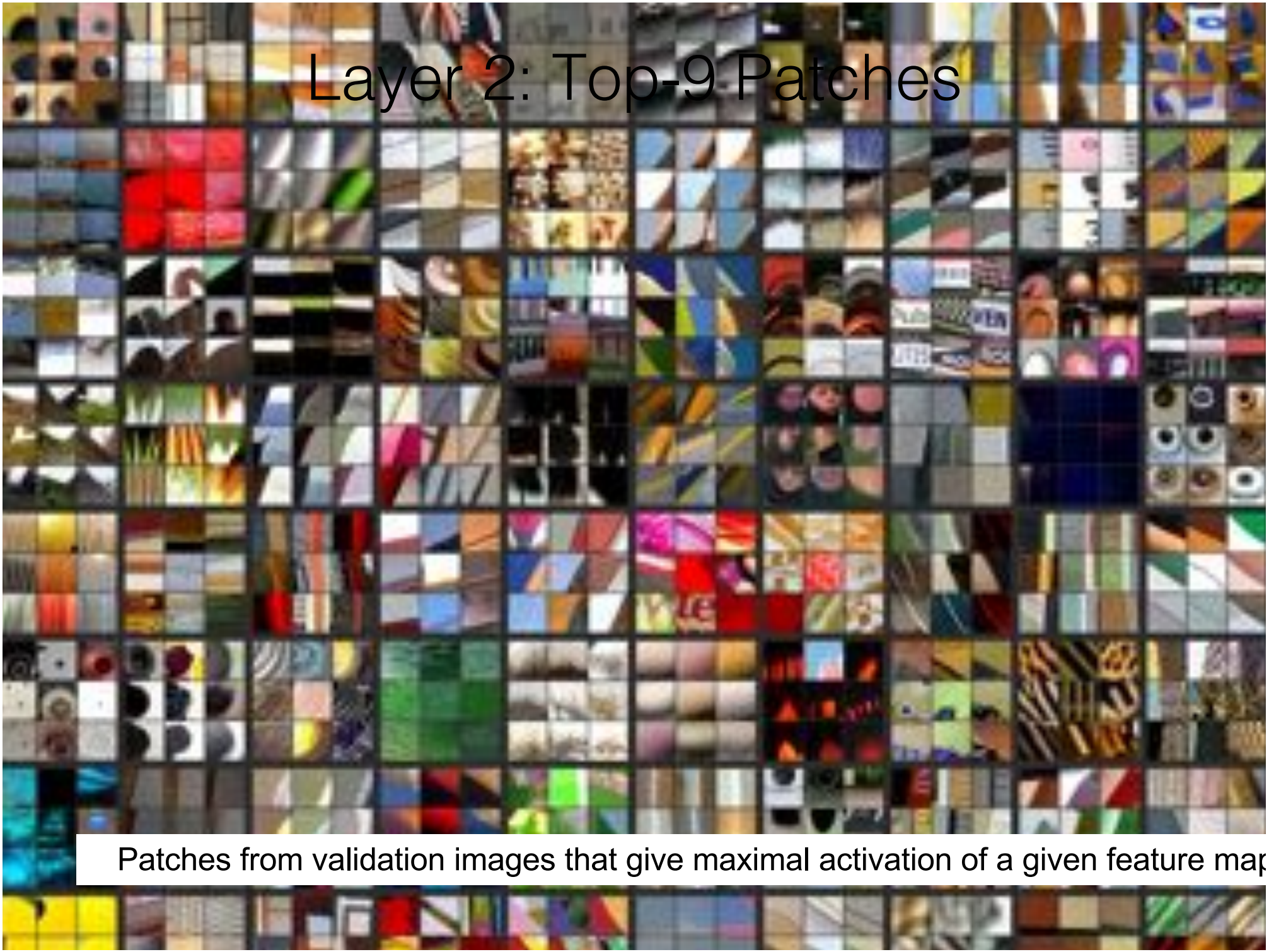
Layer 2: Top-1



Rob Fergus

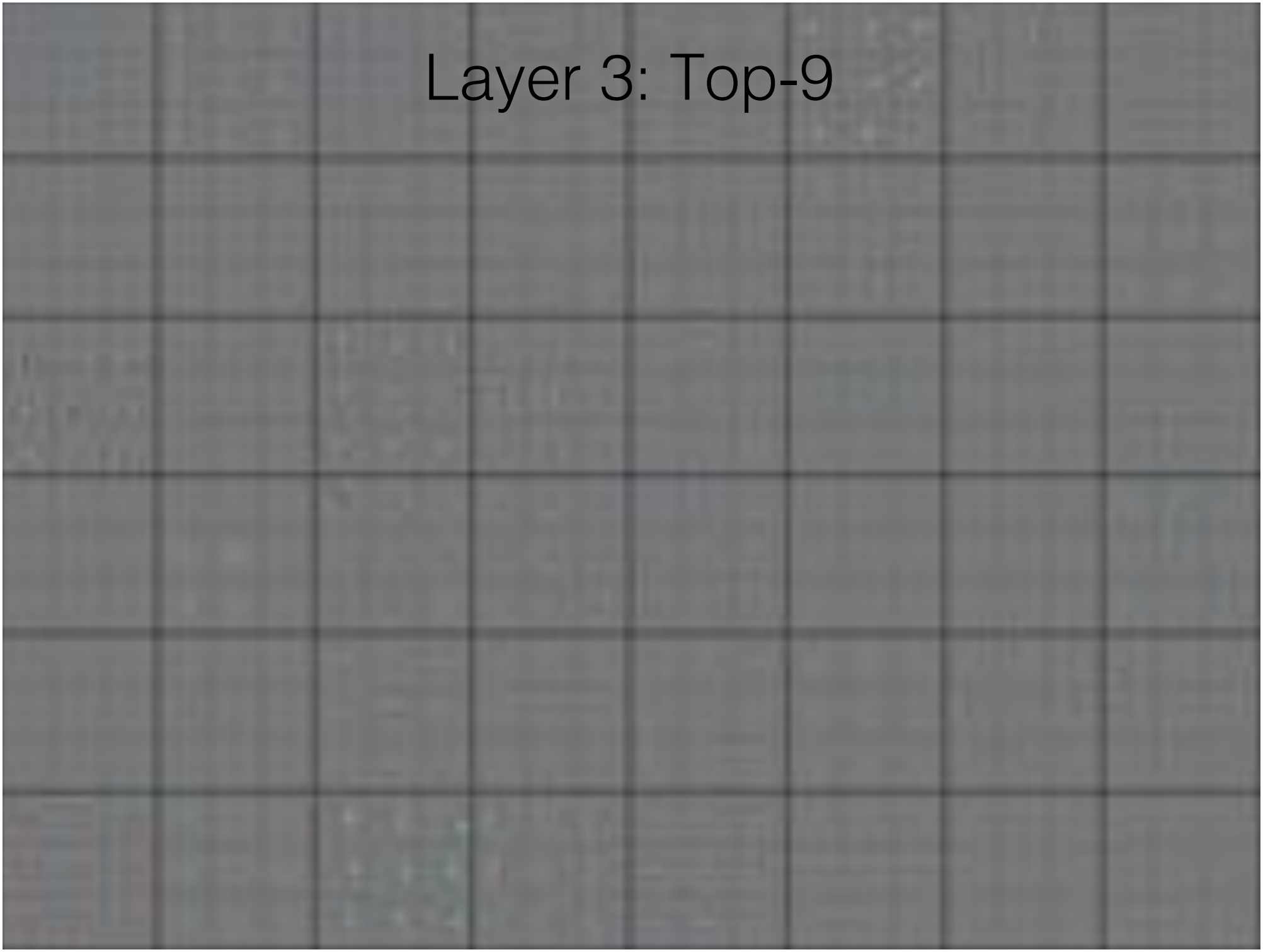

Layer 2: Top-9
• NOT SAMPLES FROM MODEL
• Just parts of input image that give strong activation of this feature map
• Non-parametric view on invariances learned by model

Layer 2: Top-9 Patches



Patches from validation images that give maximal activation of a given feature map

Layer 3: Top-9



Layer 5: Top-9

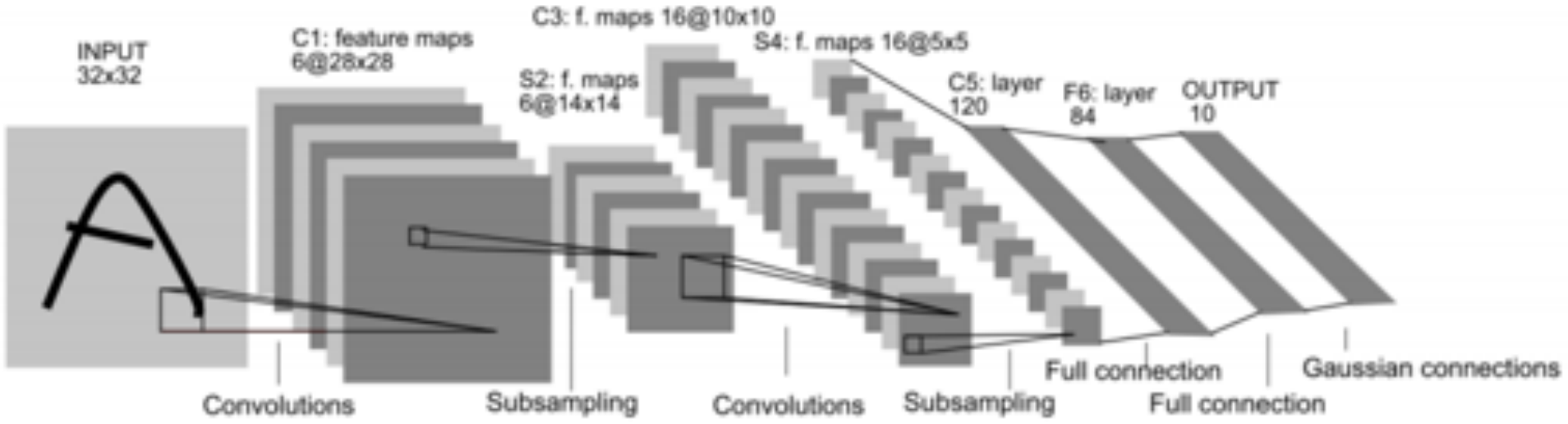

Layer 5: Top-9 Patches

Pretrained models as feature extractors

- Convolutional layers as first block for another trainable computer vision problem → **transfer learning**
- Change the model slightly for a similar problem with same or different classes → **fine tuning**
- CNN trained as a vector representation of the image (**embedded vector**) and use it as input in other vision or non-vision related tasks, e.g.: captioning, translating, etc.
- Nowadays there are many **pretrained models** in all the development platforms

Extra Information

LeNet



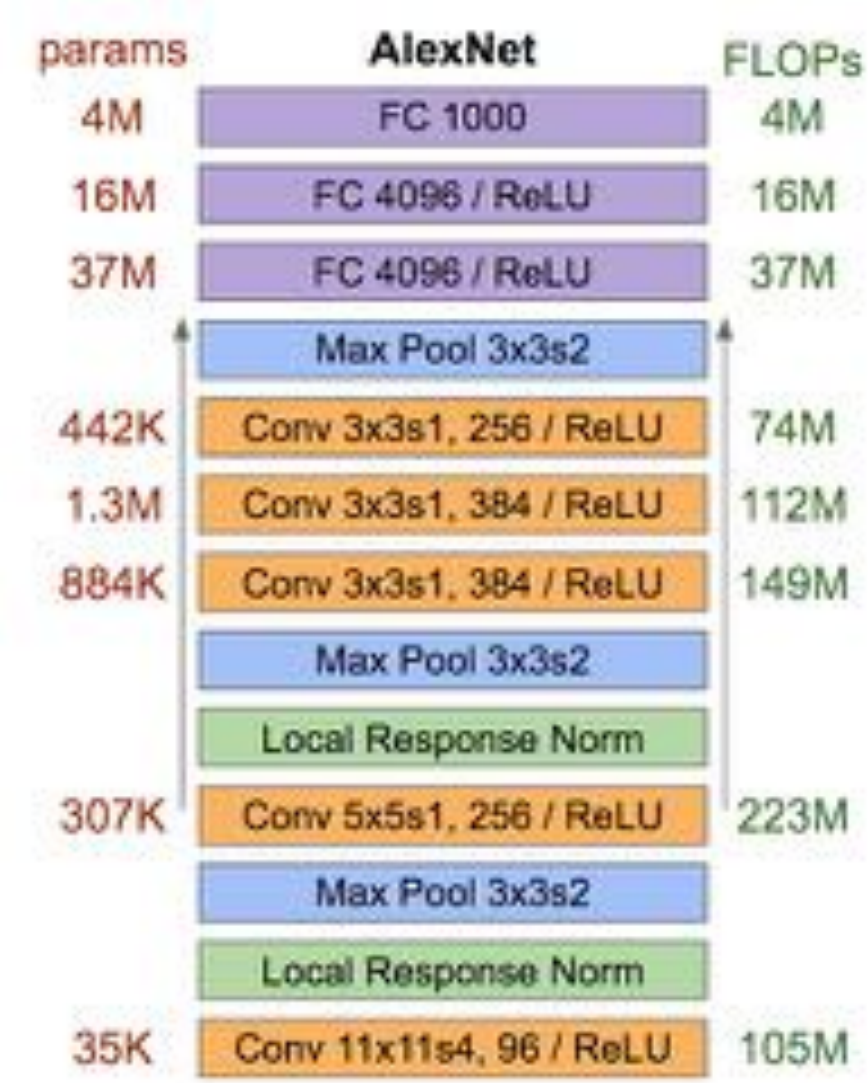
- Input: 32x32 pixel image. Largest character is 20x20 (All important info should be in the center of the receptive field of the highest level feature detectors)
- Cx: Convolutional layer
- Sx: Subsample (pooling) layer
- Fx: Fully connected layer

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., & others. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324

AlexNet

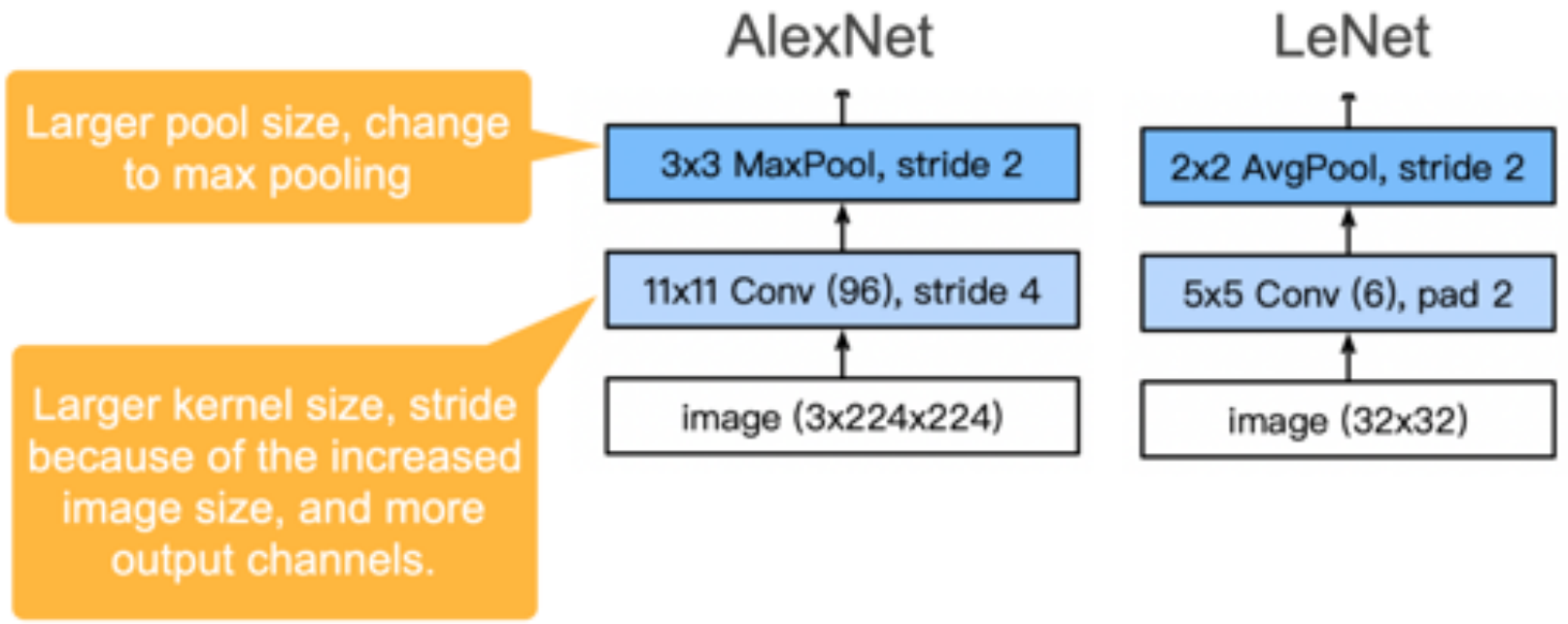
Total parameters: 61M

vs. LeNet
Total parameters: 60k

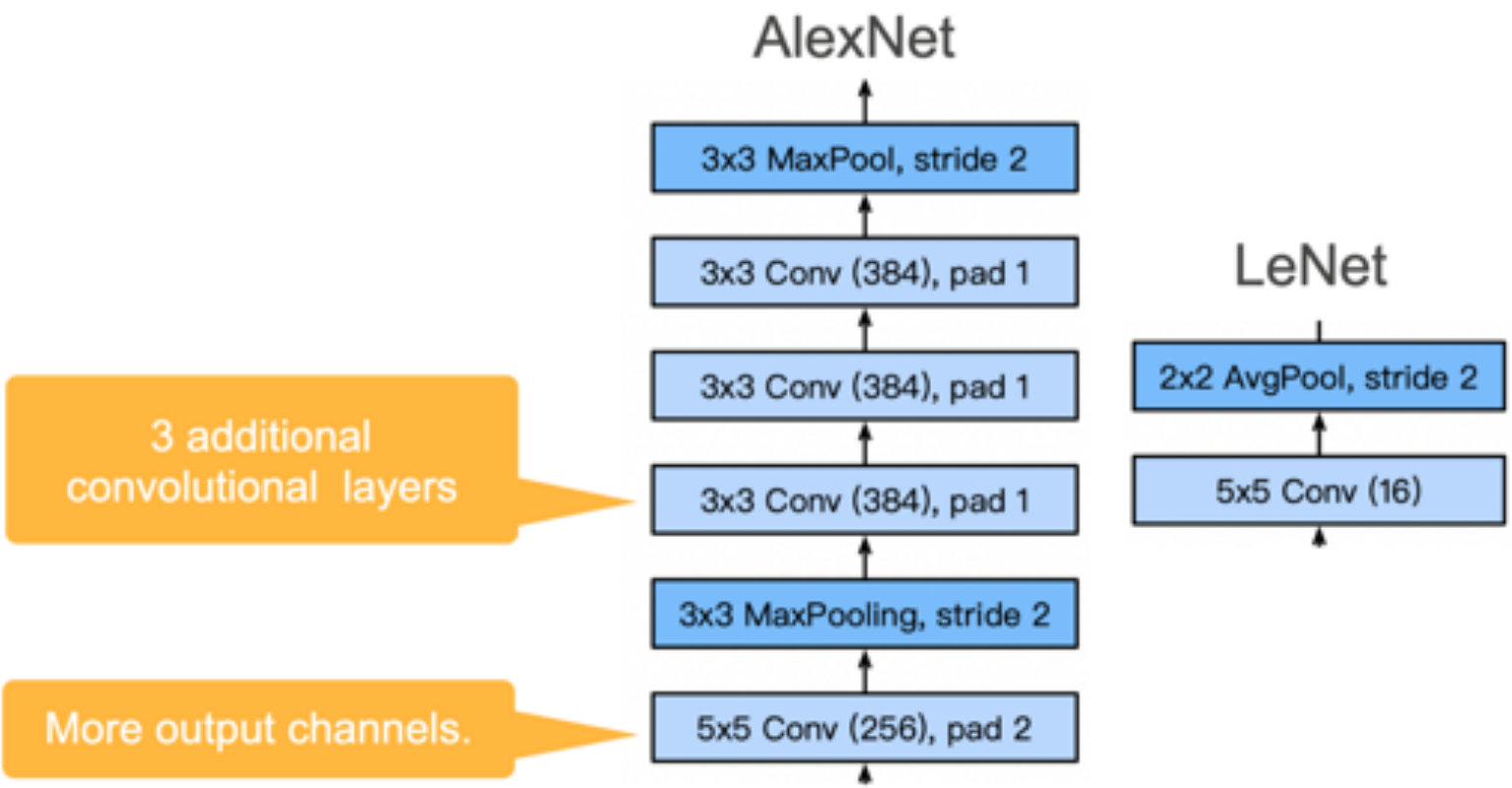


Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. "ImageNet classification with deep convolutional neural networks" . Communications of the ACM. 60 (6): 84–90.

AlexNet


AlexNet
LeNet
Larger pool size, change to max pooling
3x3 MaxPool, stride 2
2x2 AvgPool, stride 2
11x11 Conv (96), stride 4
5x5 Conv (6), pad 2
Larger kernel size, stride because of the increased image size, and more output channels.
image (3x224x224)
image (32x32)

AlexNet


AlexNet
3x3 MaxPool, stride 2
3x3 Conv (384), pad 1
3x3 Conv (384), pad 1
3x3 Conv (384), pad 1
3x3 MaxPooling, stride 2
5x5 Conv (256), pad 2
3 additional convolutional layers
More output channels.
LeNet
2x2 AvgPool, stride 2
5x5 Conv (16)

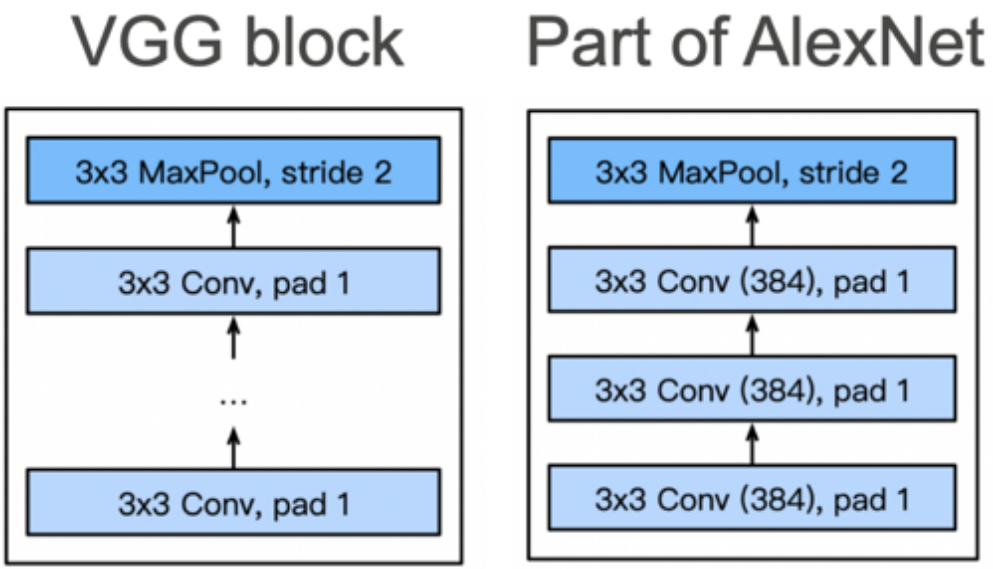
AlexNet

More tricks:

- Change activation function from sigmoid to ReLU (reduce the vanishing gradient problem)
- Add a dropout layer after two hidden dense layers (better robustness / regularization)
- Data augmentation
- GPUs

VGG

- Deeper vs. wider?
  - 7x7 convolutions
  - 3x3 convolutions (more)
  - Deep & narrow better
- VGG block
  - 3x3 convolutions (pad 1) (n layers, m channels)
  - 2x2 max-pooling (stride 2)
- 138M parameters



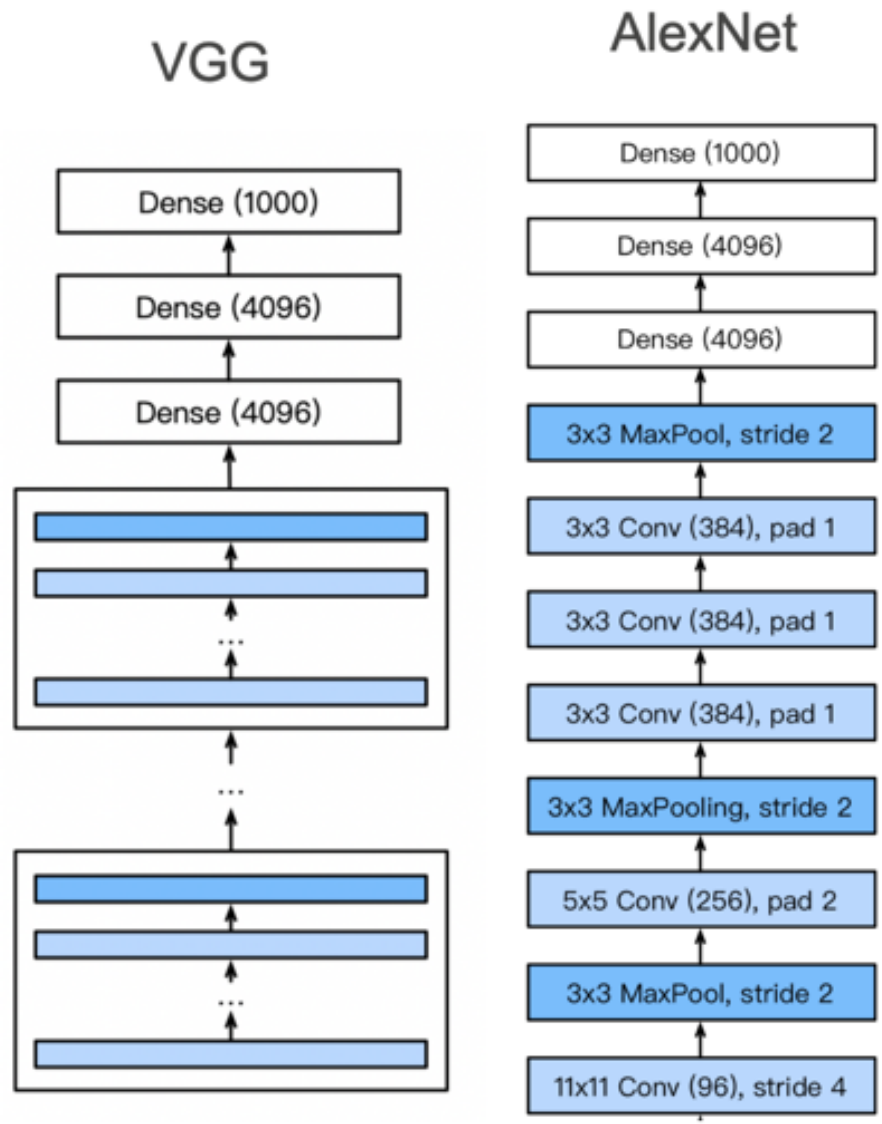
Note:

- 2 stacked layers of 3x3 cover an area of 5x5
- 3 stacked layers of 3x3 cover 7x7
- Number of parameters is reduced

K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition International Conference on Learning Representations, 2015

VGG

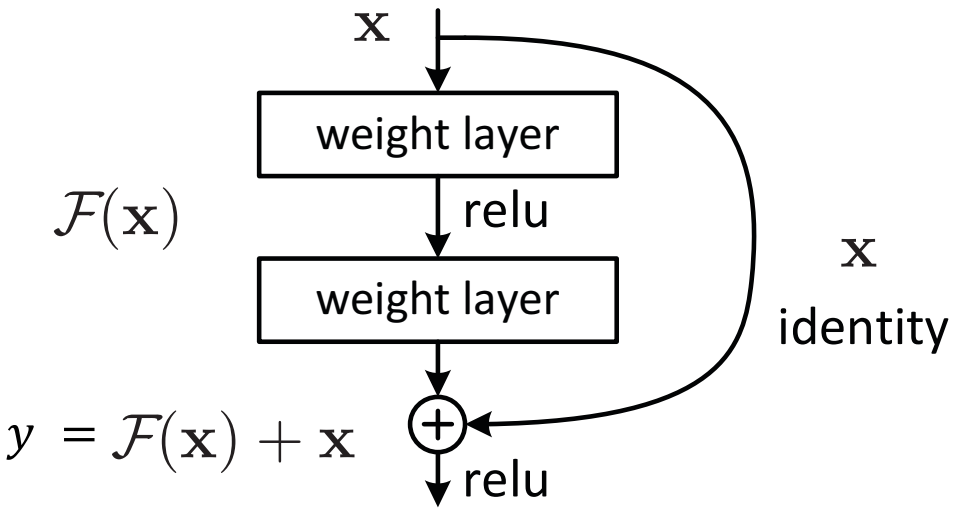
- Multiple VGG blocks followed by dense layers
  - Increasing number of filters 64-128-256-512
  - Decreasing image size 224-112-56-28-14-7
- Vary the repeating number to get different architectures, such as VGG-16, VGG-19, ...



Explanation in CS 152 about VGG
https://www.youtube.com/watch?v=hEpPYypNo4s

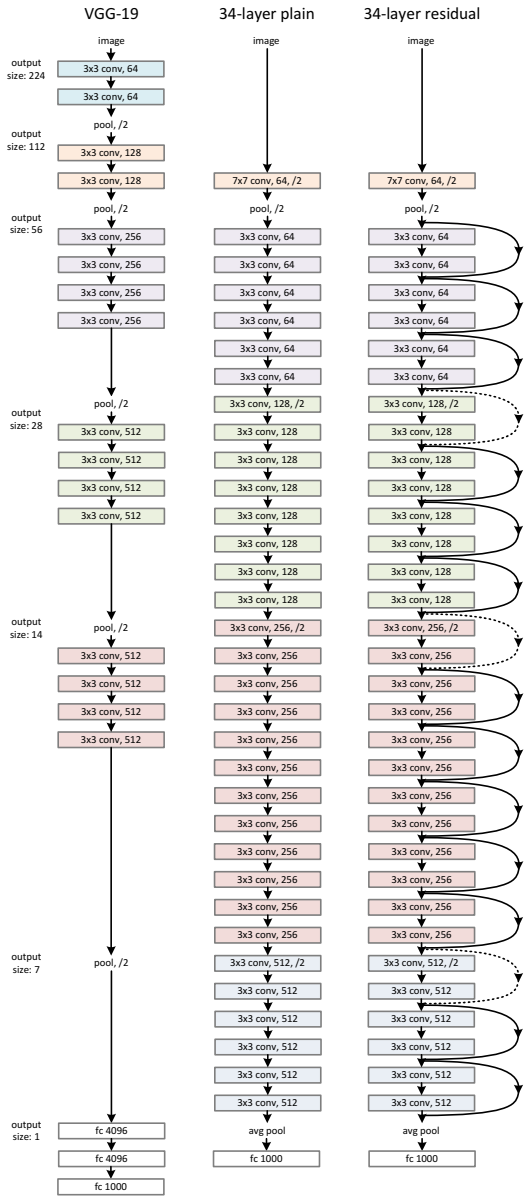
ResNet

- Really deep CNNs don't train well
- Key idea: introduce "pass through" into each layer (**skip connections**)
- Gradient can backpropagate more easily
- Learning **residuals** instead of full mapping → $\mathcal{F}(\mathrm{x}) = y - x$



$\mathcal{F}(\mathbf{x})$

$y = \mathcal{F}(\mathbf{x}) + \mathbf{x}$

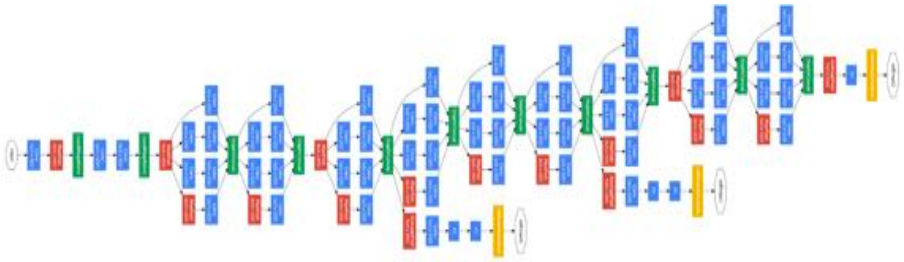


Explanation in CS 152 about ResNet
https://www.youtube.com/watch?v=d6vkM_xCJko

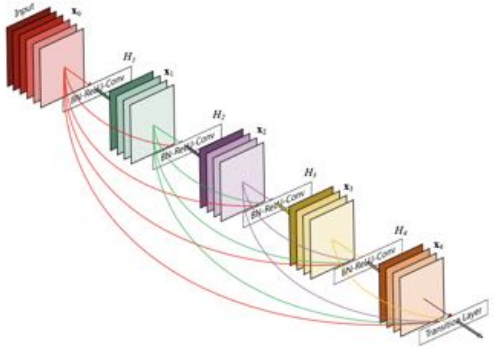
Many architectures

VGG and ResNet are (probably) the most well-known, but there are many more:

- Inception (in particular v3)
- DenseNet
- MobileNet
- ShuffleNet
- SqueezeNet
- Xception
- …



Inception v3



DenseNet

Many architectures

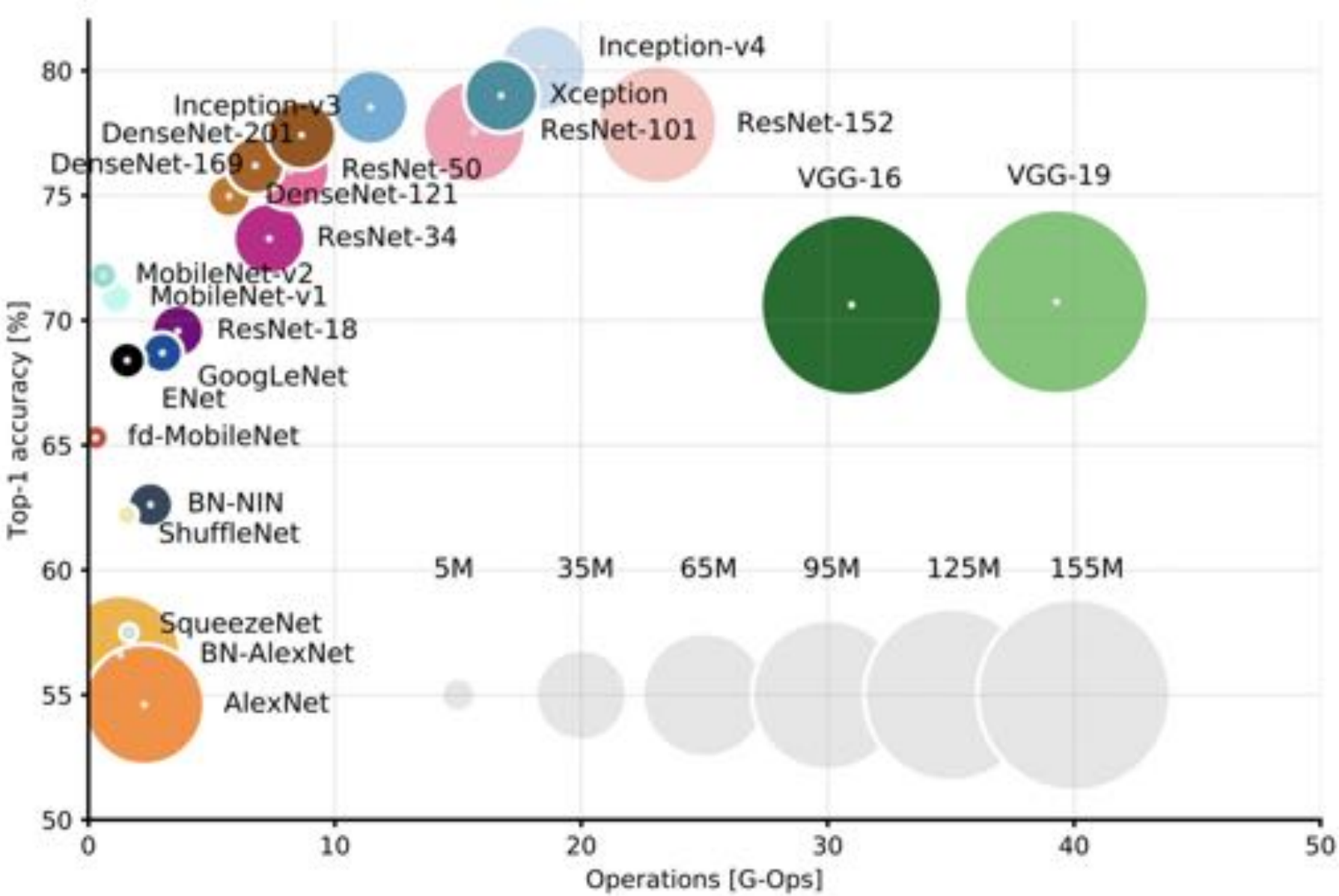
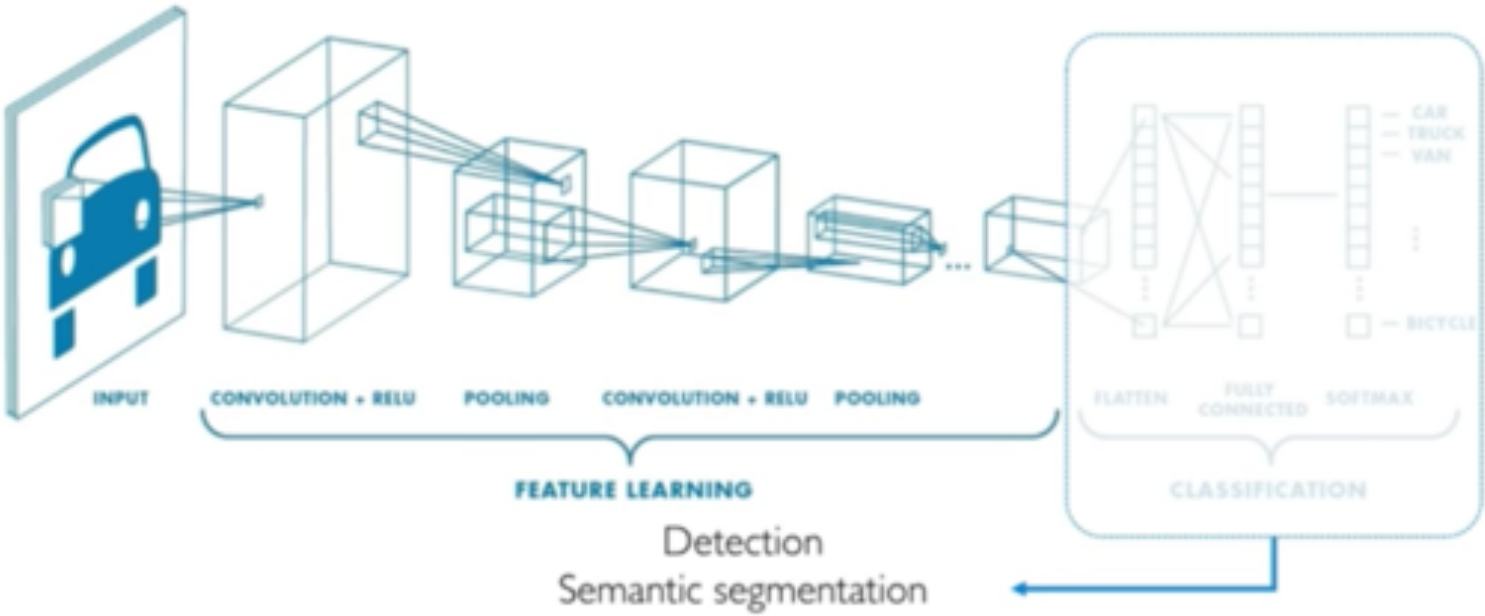


Image Source (2017): https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

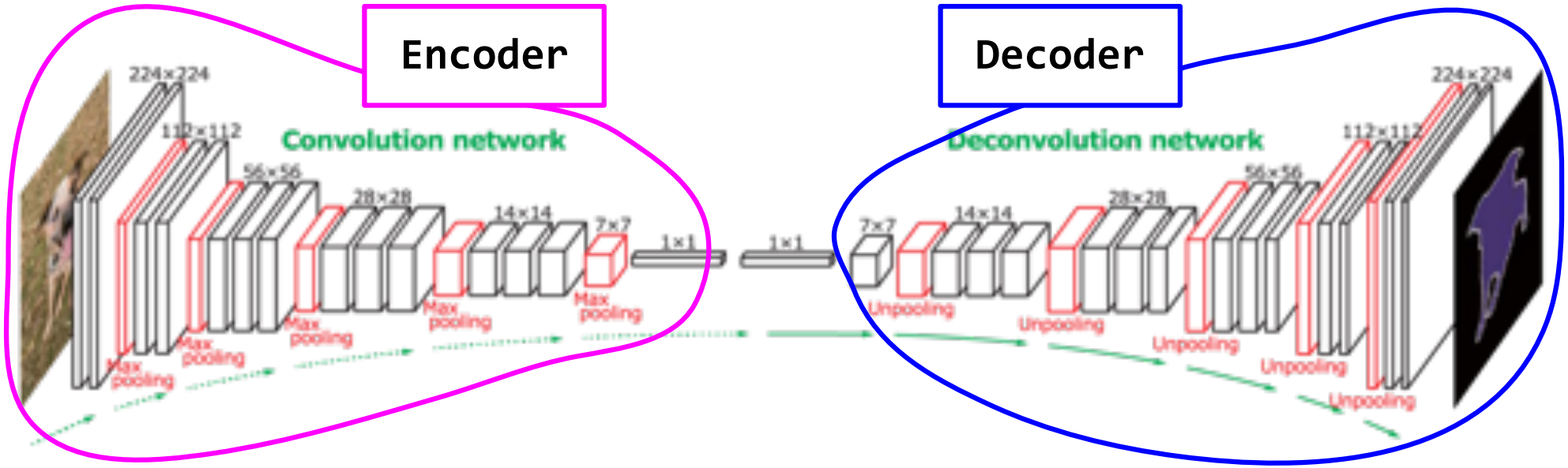
Beyond Image Classification



Source: MIT 6.S191 – Alexander Amini and Ava Soleimany

Semantic Segmentation

Encoder-decoder architecture

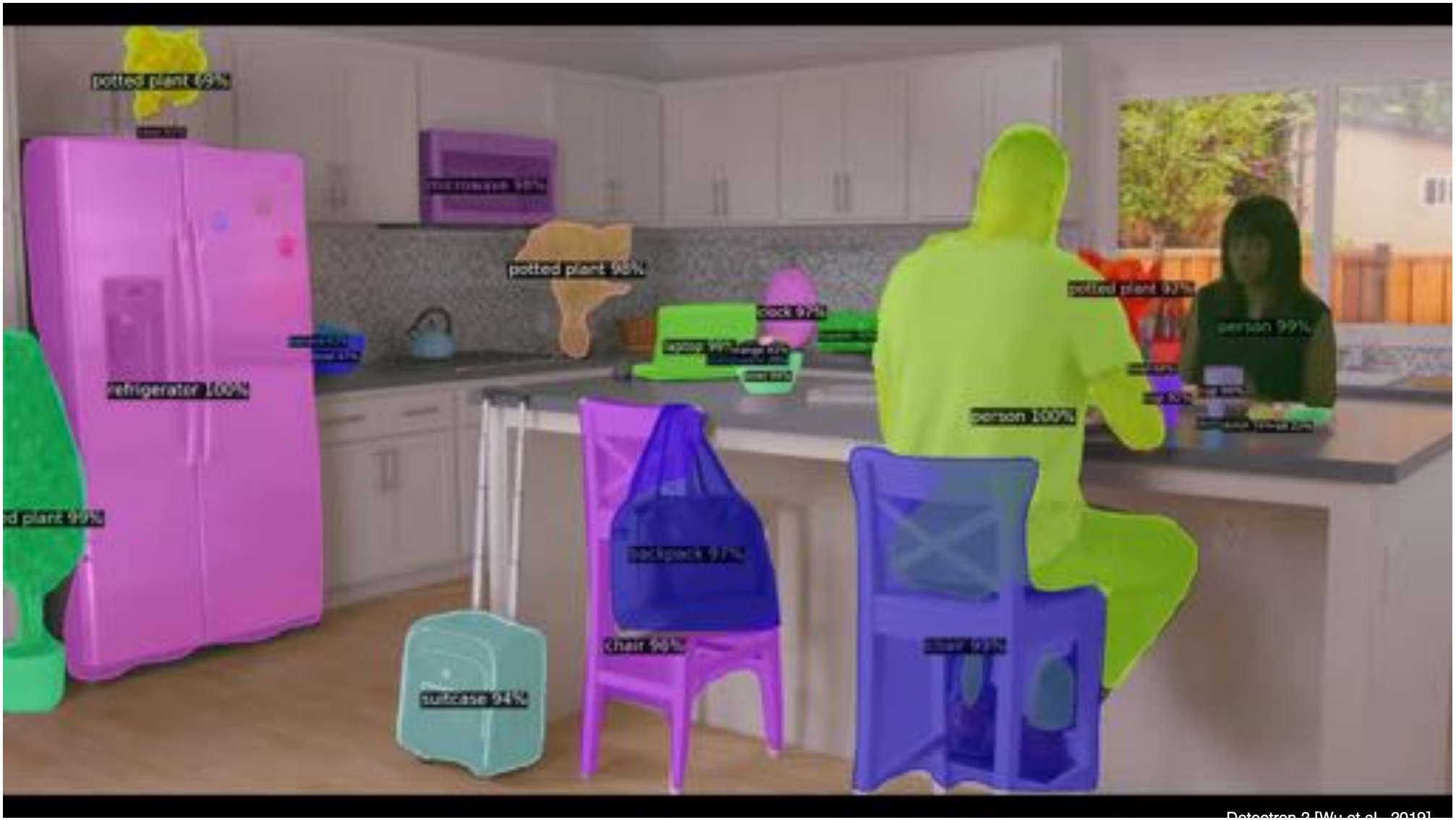


Zeiler et al., Deconvolutional Networks, CVPR 2010

Long et al., Fully Convolutional Models for Semantic Segmentation, CVPR 2015

Noh et al., Learning Deconvolution Network for Semantic Segmentation, ICCV 2015

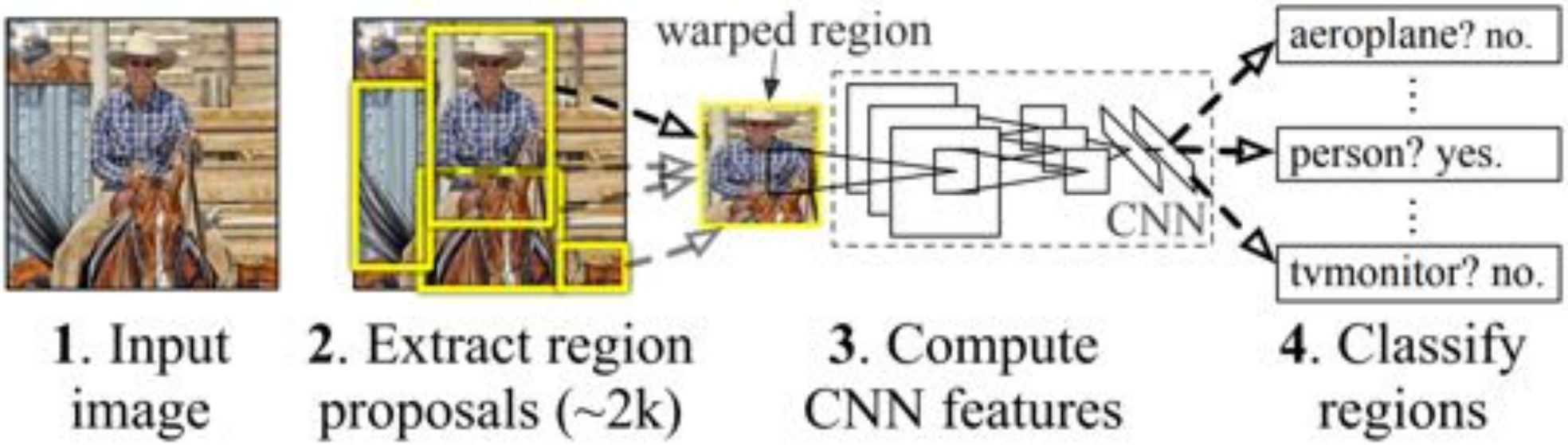
Image source: https://arxiv.org/pdf/1505.04366.pdf



Detectron 2 [Wu et al., 2019]

Object Detection

R-CNN: Region-based CNN



+bbox regression

Girshick et al.

Object Detection

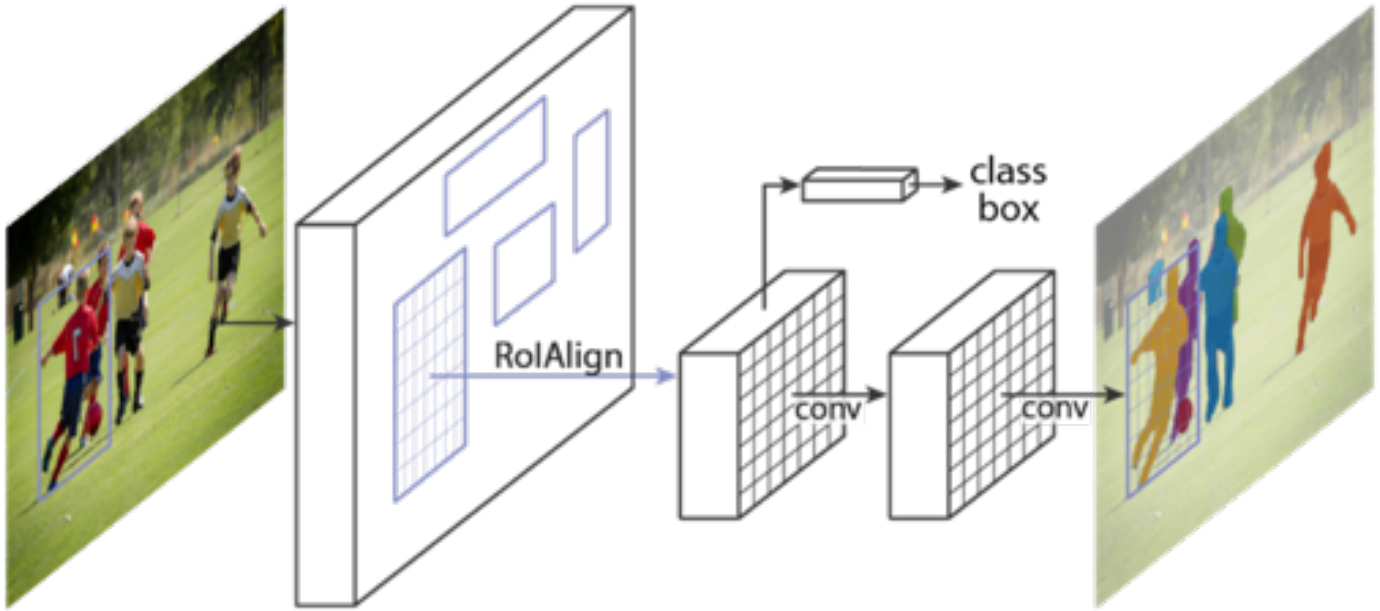
R-CNN is slow → Fast R-CNN → Faster R-CNN

It still requires two stages, and other architectures adopt a one-stage approach:

- Single Shot Detector (SSD)
- You Only Look Once (YOLO)
  - many different versions v3, v4, v5, v6…
- RetinaNet
- EfficientDet
- …

Instance Segmentation

Mask R-CNN predicts mask *as well as* bounding box

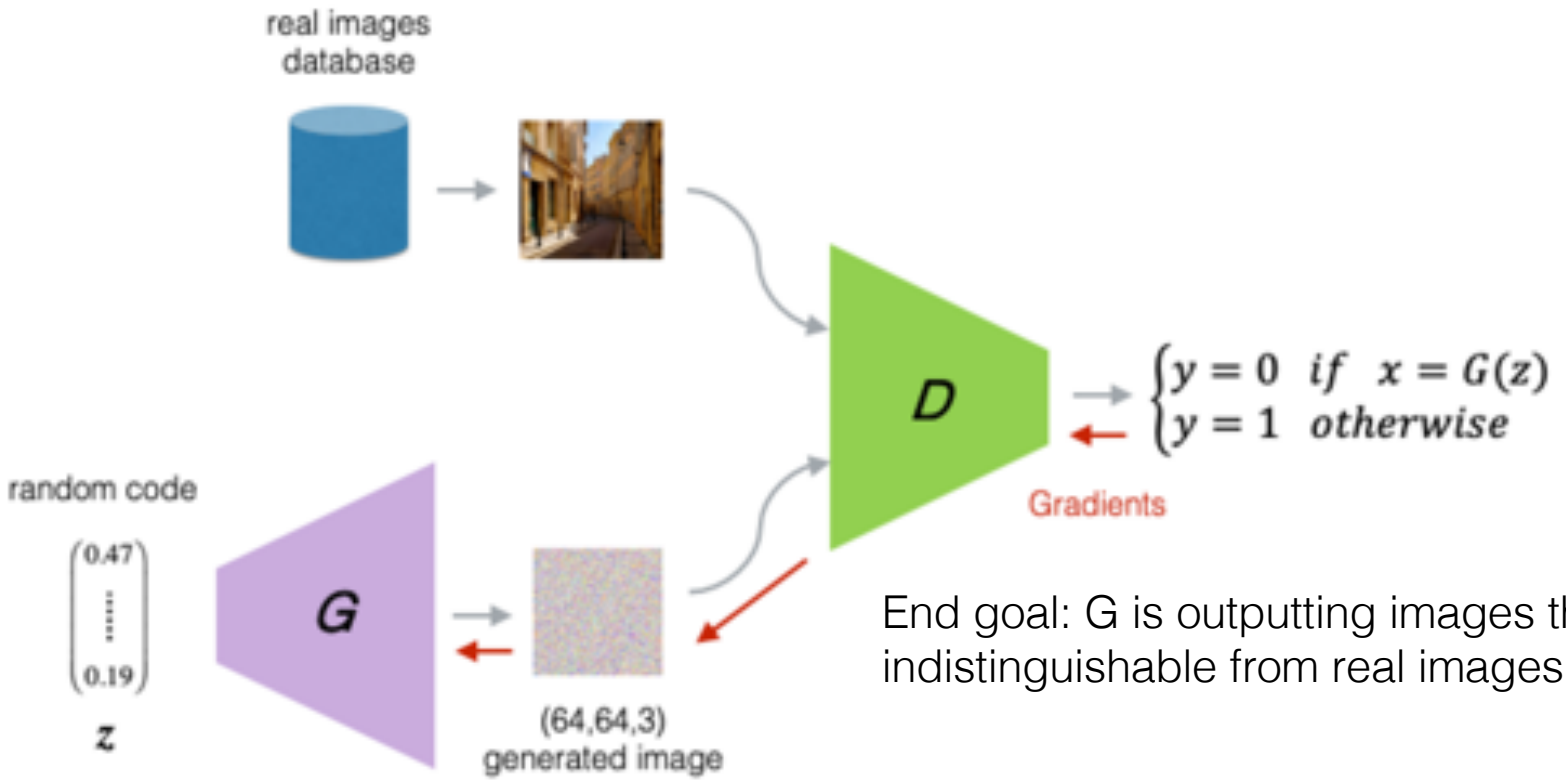


He et al., Mask R-CNN
https://arxiv.org/pdf/1703.06870.pdf


traffic light.96
traffic light.95
traffic light.92
truck1.00
person1.00
person.83
cart.00
traffic light.84
person1.00
person.89
person.86
BUS

Image Generation/Synthesis

Generative Adversarial Networks (GANs)

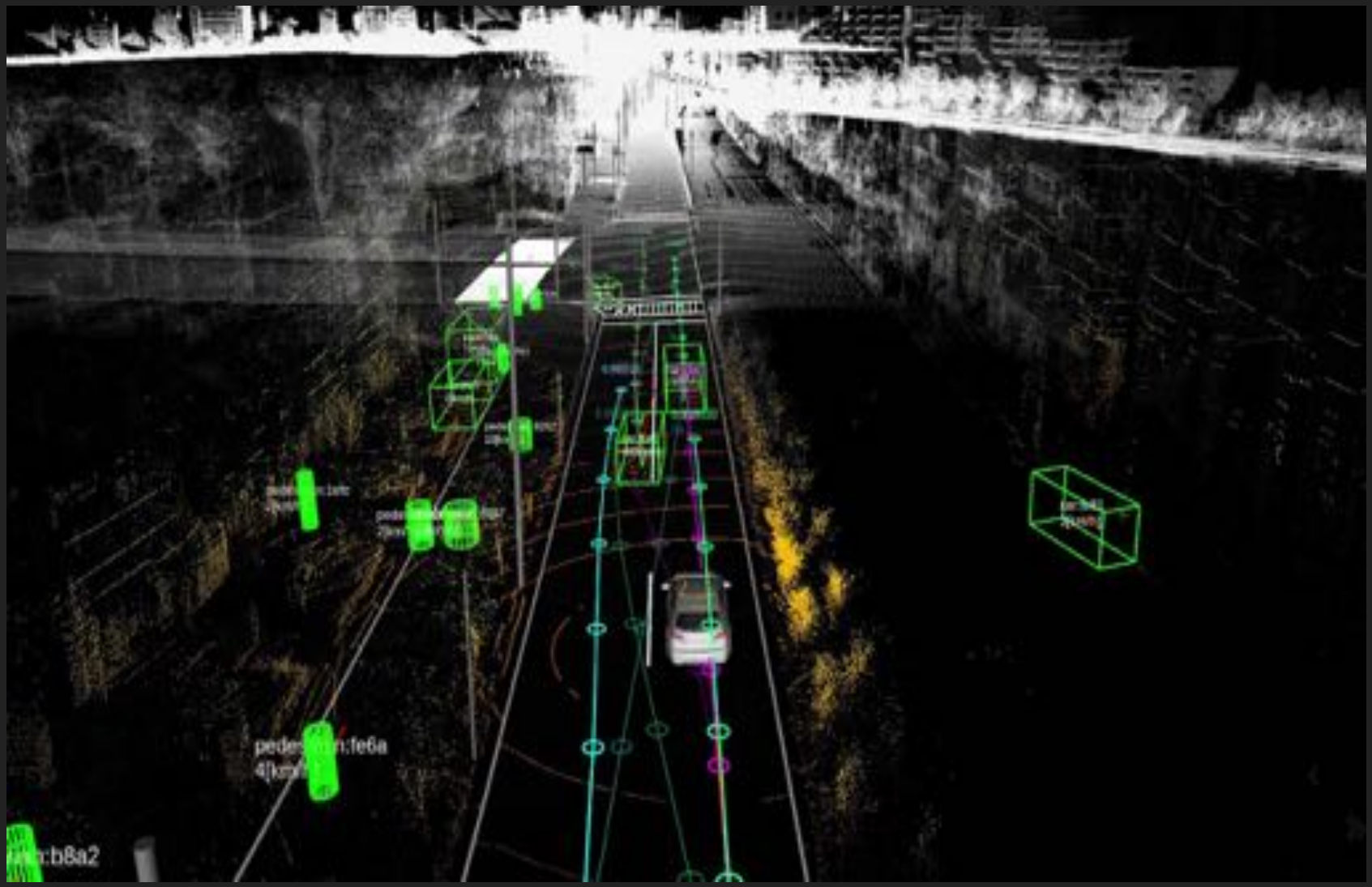


End goal: G is outputting images that are indistinguishable from real images for D

Generative Adversarial Networks
Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, Bengio 2014

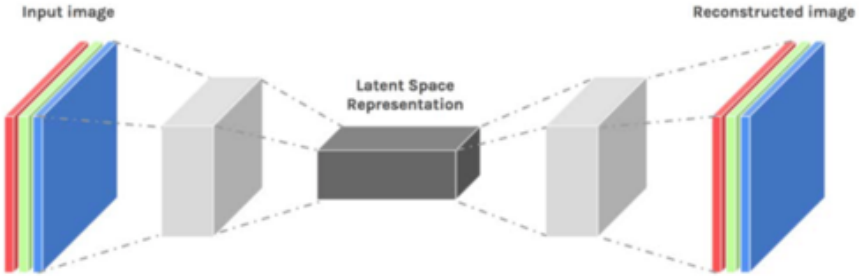
Style GAN 2 [Karras et al., 2019]

Beyond Images

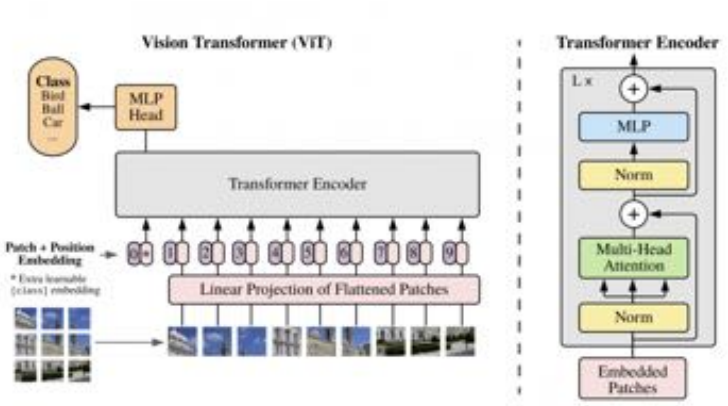


A General Framework

- There are many neural network **architectures** (e.g. CNN, GAN, autoencoder, transformer, diffusion model) that can be by built combining different types of layers (e.g. fully-connected, convolution, attention, etc)



Autoencoder



Transformer

- Can be seen as a paradigm: differentiable programming


python


TensorFlow


K
Keras


OpenCV


PyTorch

Acknowledgments: This presentation contains slides from too many authors to list here. Thank you for all the incredible work by the computer vision and machine learning communities.