Overview of Computer Vision Architectures:
from 1998 to 2018

Ekapol Chuangsuwanich
Nvidia IVA workshop

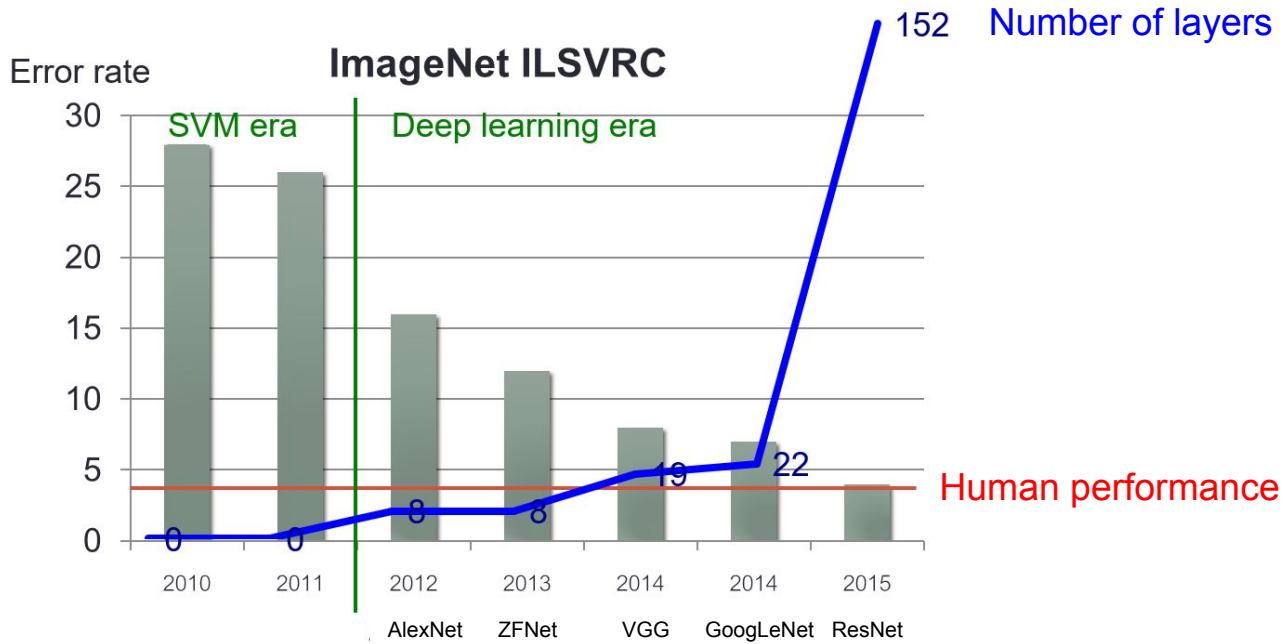
CHULA ENGINEERING
Foundation toward Innovation
COMPUTER

Deep learning

Artificial Neural Networks rebranded

- Deeper models
- Bigger data
- Larger compute

A brief history of ImageNet



Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

Statistics > Machine Learning

Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, Jeffrey Pennington

(Submitted on 14 Jun 2018)

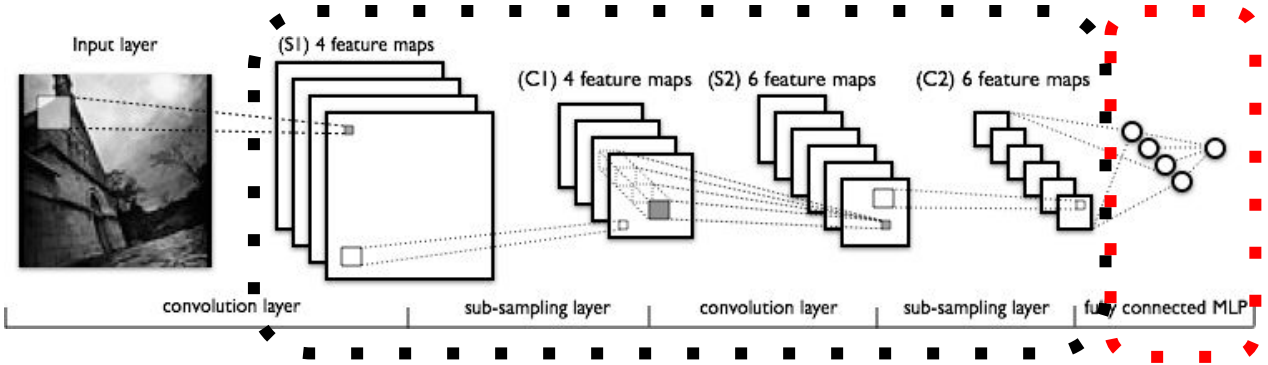
In recent years, state-of-the-art methods in computer vision have utilized increasingly deep convolutional neural network architectures (CNNs), with some of the most successful models employing hundreds or even thousands of layers. A variety of pathologies such as vanishing/exploding gradients make training such deep networks challenging. While residual connections and batch normalization do enable training at these depths, it has remained unclear whether such specialized architecture designs are truly necessary to train deep CNNs. In this work, we demonstrate that it is possible to train vanilla CNNs with ten thousand layers or more simply by using an appropriate initialization scheme. We derive this initialization scheme theoretically by developing a mean field theory for signal propagation and by characterizing the conditions for dynamical isometry, the equilibration of singular values of the input-output Jacobian matrix. These conditions require that the convolution operator be an orthogonal transformation in the sense that it is norm-preserving. We present an algorithm for generating such random initial orthogonal convolution kernels and demonstrate empirically that they enable efficient training of extremely deep architectures.

Comments: ICML 2018 Conference Proceedings
Subjects: **Machine Learning (stat.ML)**; Machine Learning (cs.LG)
Cite as: **arXiv:1806.05393 [stat.ML]**
(or **arXiv:1806.05393v1 [stat.ML]** for this version)

Submission history

From: Samuel Schoenholz [view email]
[v1] Thu, 14 Jun 2018 07:04:15 GMT (6734kb,D)

Convolutional Neural Network (CNN)

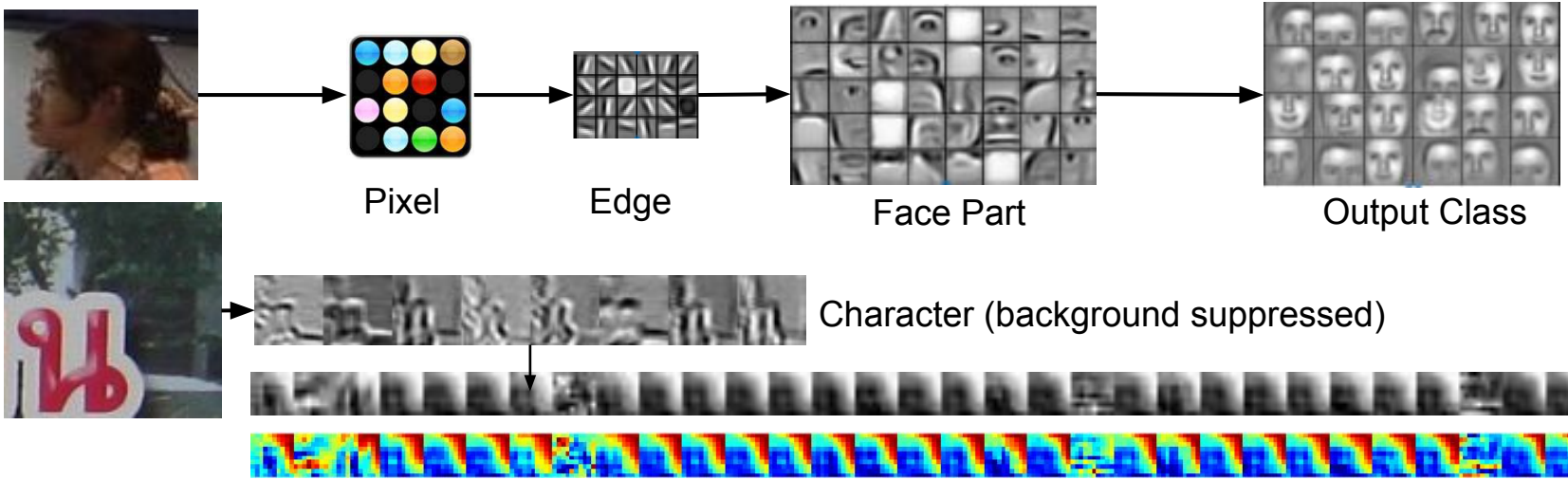


Trainable Feature Extractor Layer **Classification Layer**

Traditional CV: fixed representation but learnable classifier
Deep learning: jointly learn representations with the classifier

Convolutional Neural Network (CNN)

- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition: Pixel →edge → texture → part → object



Agenda

Building blocks

Object classifiers

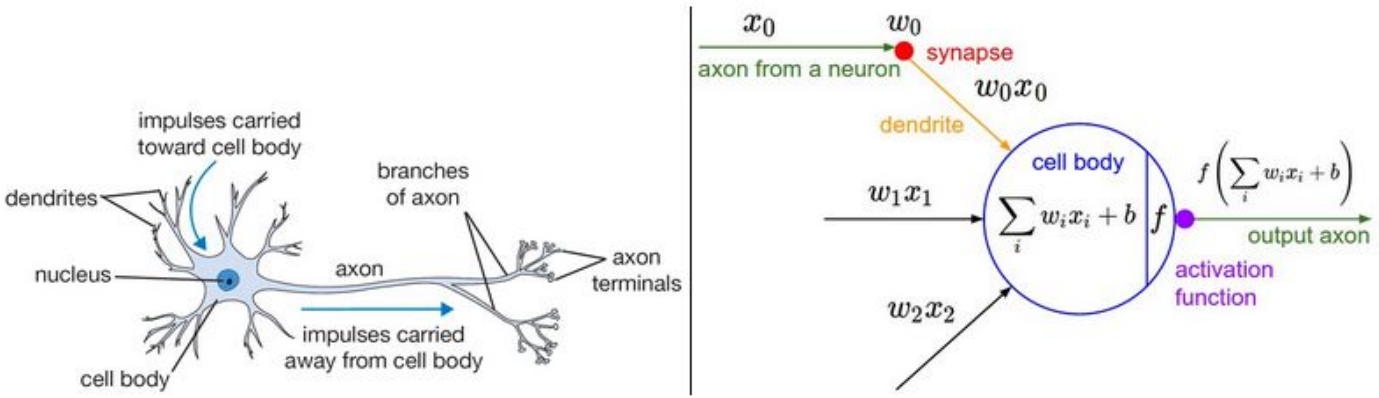
- From LeNet to NASNet

Object detectors

- Region proposal based detectors
- Single stage detectors

Fully connected networks

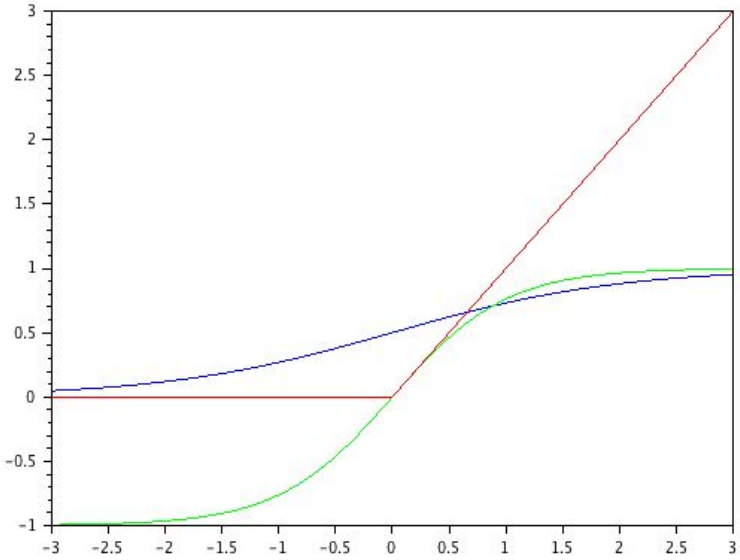
- Many names: feedforward networks or deep neural networks or multilayer perceptron or artificial neural networks
- Composed of multiple neurons



http://cs231n.github.io/neural-networks-1/

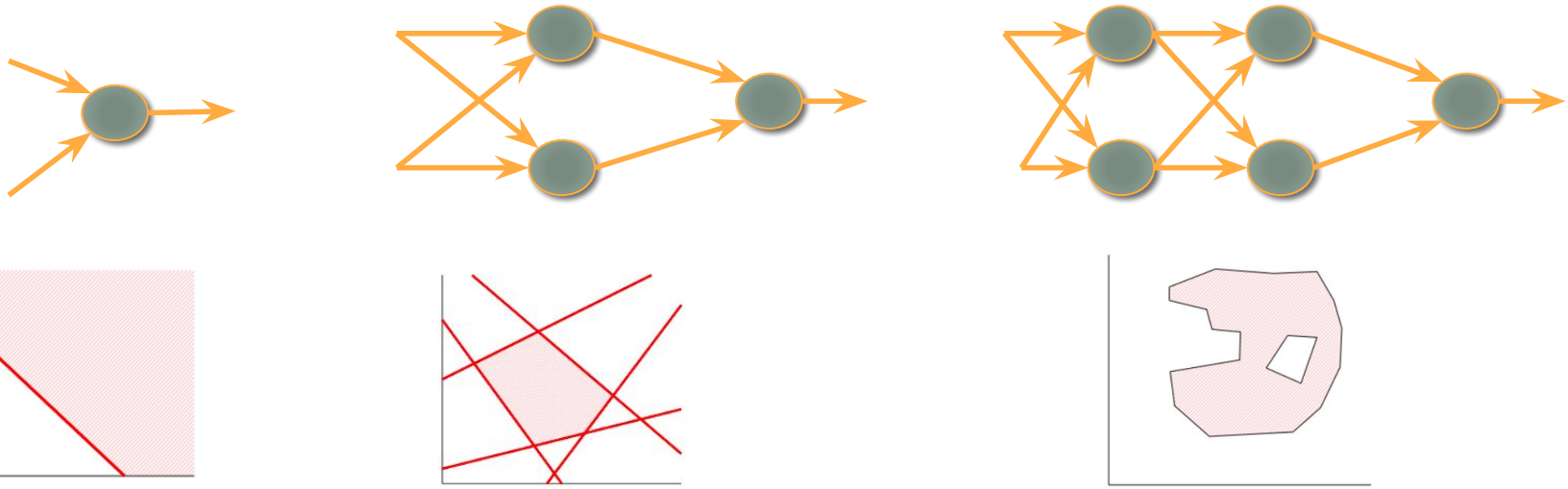
Non-linearity (Activation Function)

- The non-linearity is important in order to stack neurons
- If linear, a multi layered network can be collapsed to a single layer (by just multiplying weights together)
  - Sigmoid or logistic function
  - Tanh
  - Rectified Linear Unit (ReLU)
- Most popular is ReLU and its variants (Fast to train, and more stable)



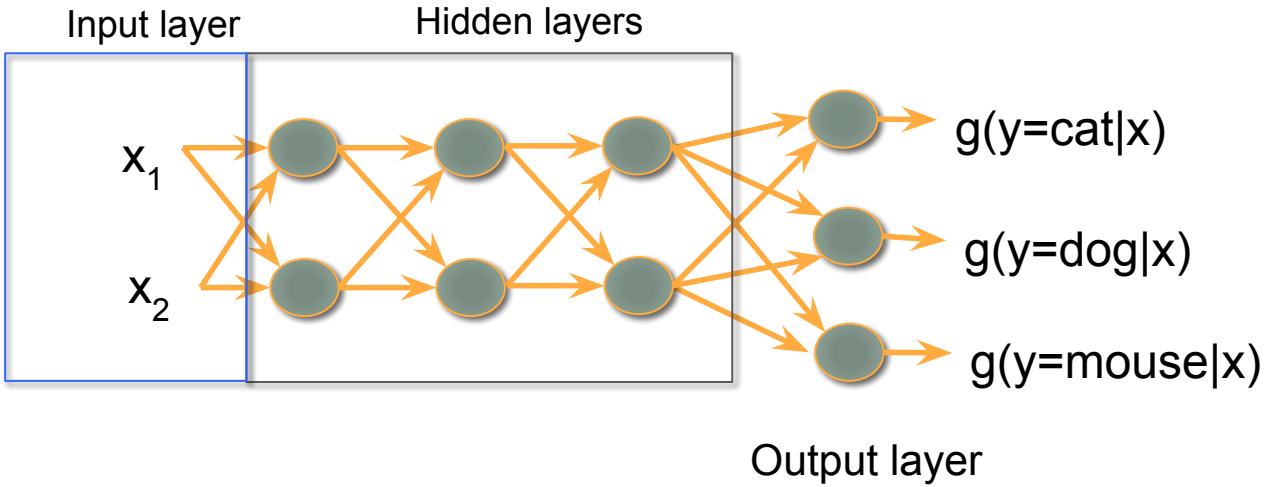
Combining neurons

- Each neuron splits the feature space with a hyperplane
- Stacking neuron creates more complicated decision boundaries
- More powerful but prone to overfitting



Terminology

Deep in Deep neural networks means many hidden layers

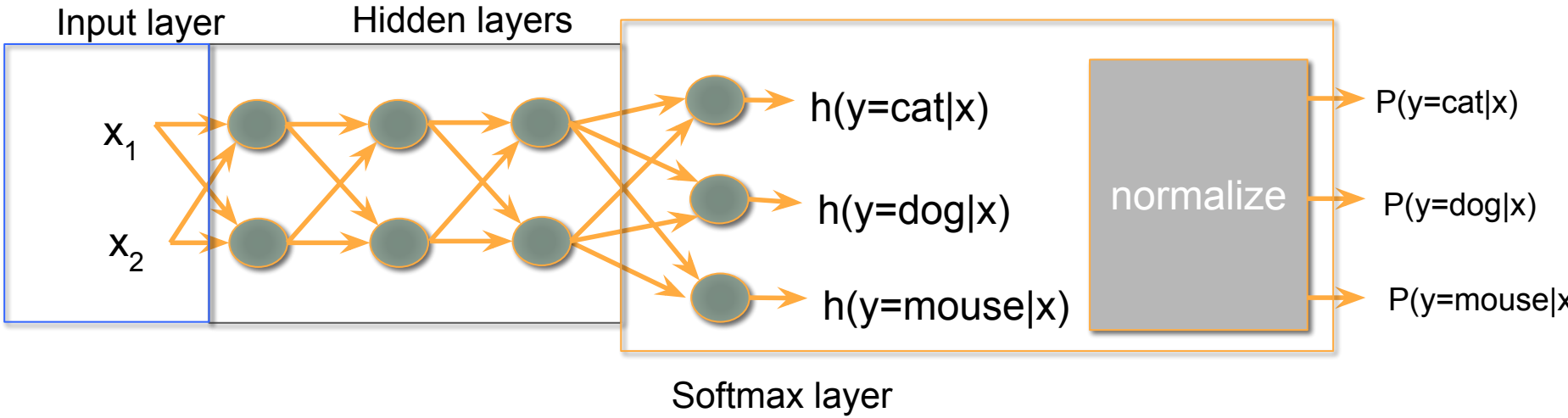


Function that map input to scores of being a particular class

Output layer – Softmax layer

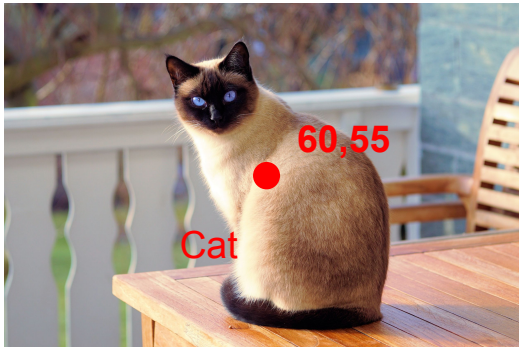
- We usually wants the output to mimic a probability function (0<=P<=1,sums to 1)
- Add a normalization layer called “softmax”

$$P(y=j|x) = \frac{e^{h(y=j|x)}}{\sum_y e^{h(y|x)}}$$



Objective function (Loss function)

- Can be any function that summarizes the performance into a single number
  - Cross entropy
    - Categorical tasks (cat vs dog, age range)
  - Sum of squared errors
    - Numerical tasks (location of cat, age)



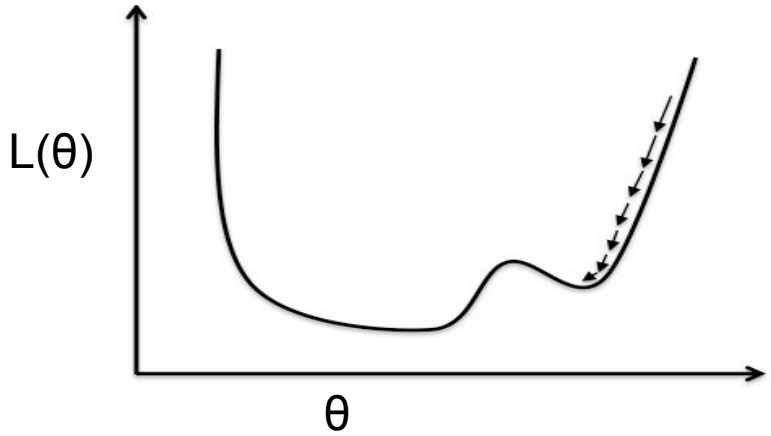
$q_1(x, \theta)$

$q_n(x, \theta)$

	Output type	
	Categorical	Numerical
	$y_1 = 1$	$y_x = 60$
	$y_n = 0$	$y_y = 55$

Minimization using gradient descent

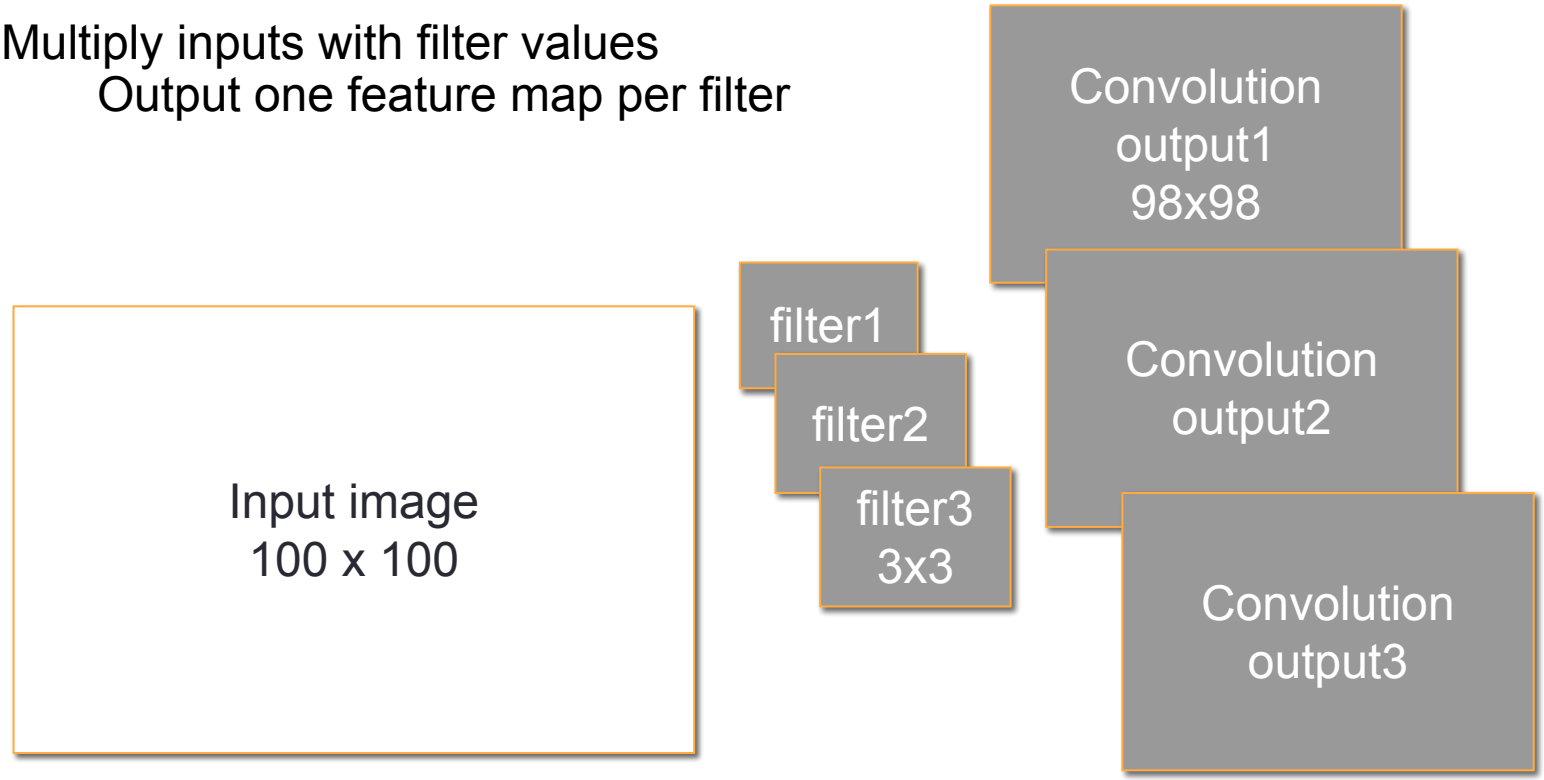
- We want to minimize L with respect to θ (weights and biases)
  - Differentiate with respect to θ
  - Gradients passes through the network by Back Propagation



Convolutional filters

Multiply inputs with filter values

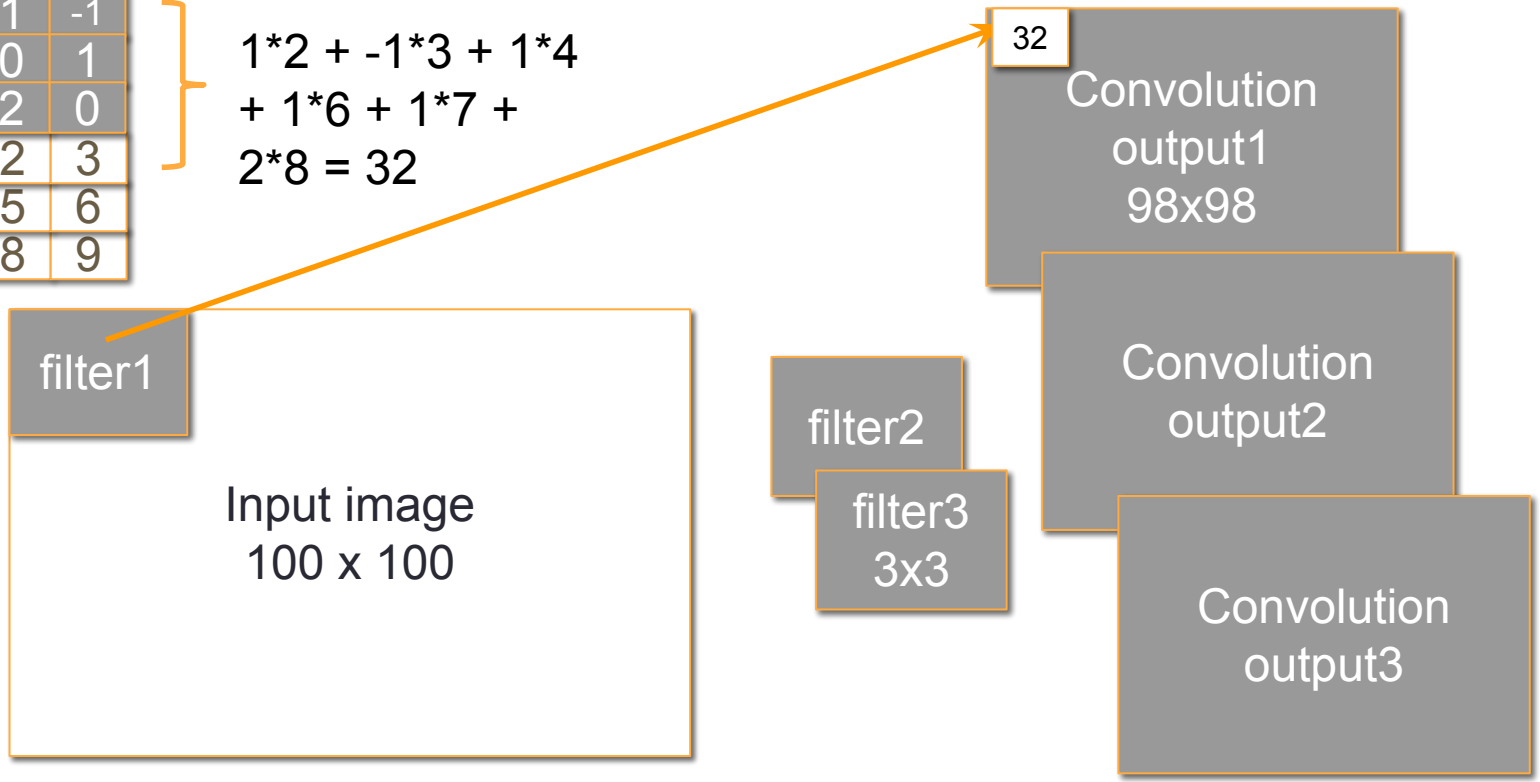
Output one feature map per filter



Convolutional filters

0	1	-1
1	0	1
1	2	0
1	2	3
4	5	6
7	8	9

1*2 + -1*3 + 1*4
+ 1*6 + 1*7 +
2*8 = 32



Convolutional filters

0	1	-1
1	0	1
1	2	0
2	3	1
5	6	3
8	9	8

1*3 + -1*1 + 1*5
+ 1*3 + 1*8 +
2*9 = 36

Stride of 1

33 36

Convolution output1 98x98

Convolution output2

Convolution output3

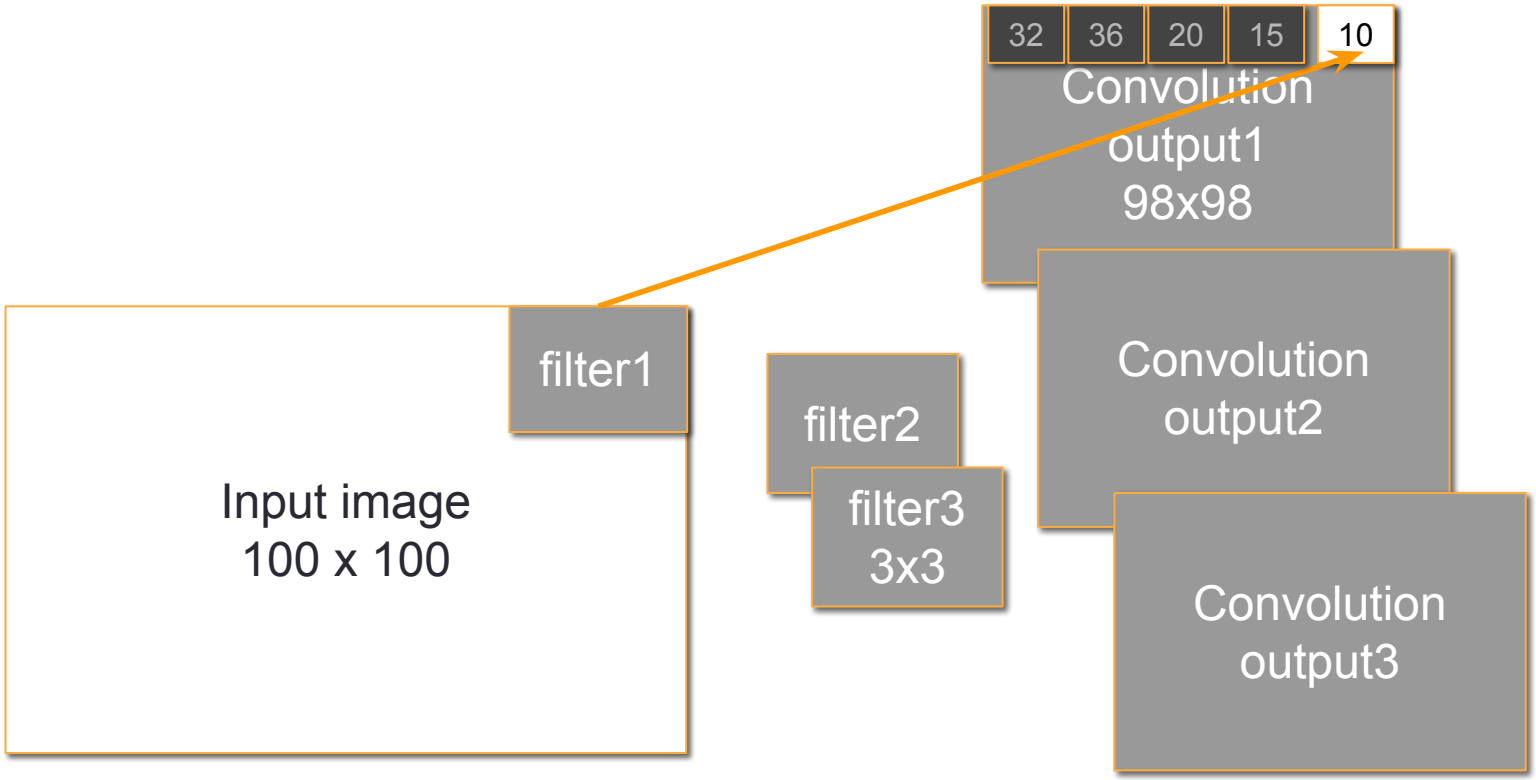
filter1

filter2

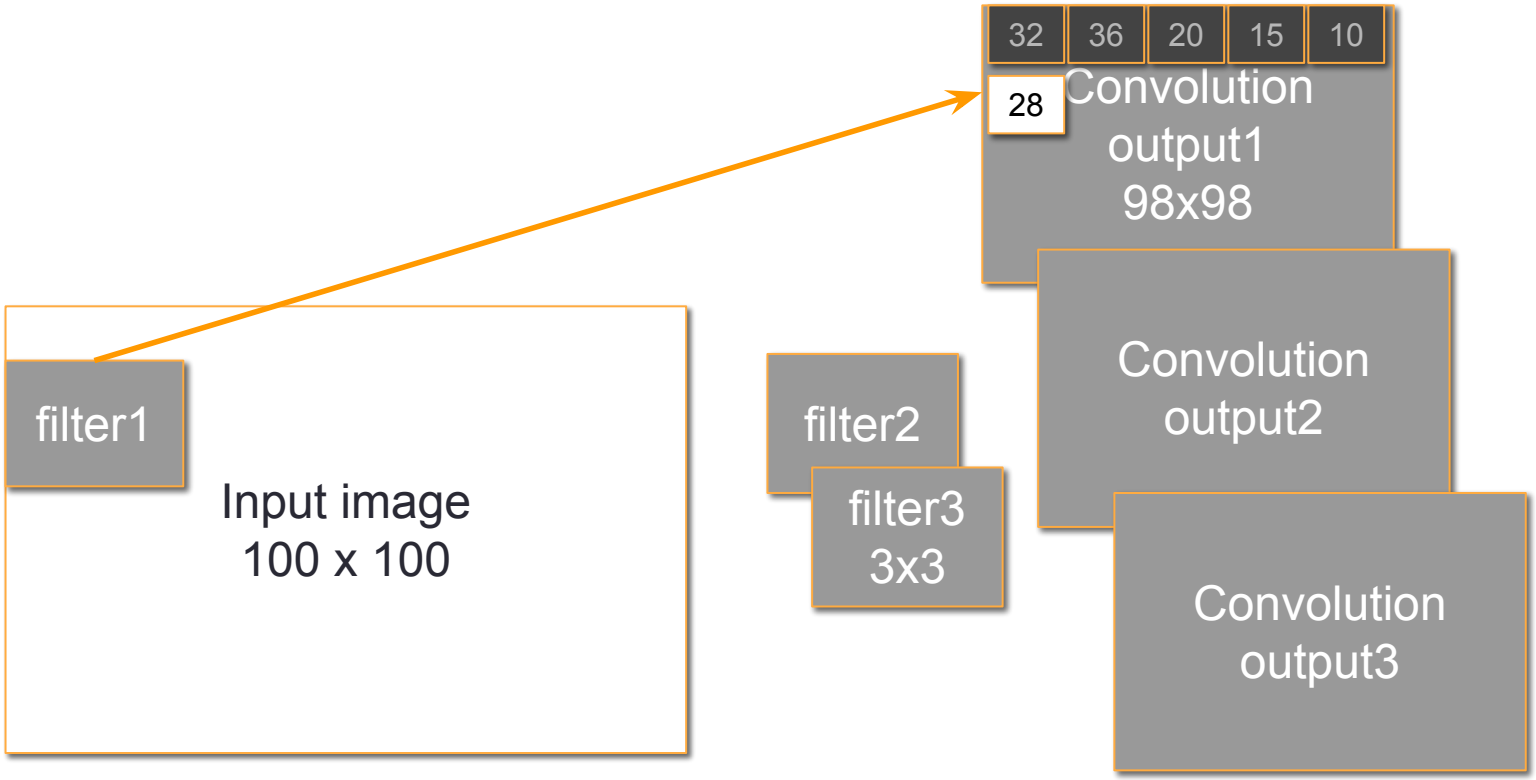
filter3 3x3

Input image
100 x 100

Convolutional filters

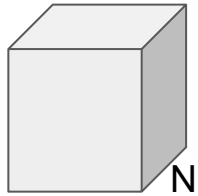


Convolutional filters



Convolutional filters

N filters means N feature maps
You get a 3 dimensional output



32	36	20	15	10
28	72	0	12	50
32	36	18	9	2
17	6	2	17	1

16

filter2

Input image
100 x 100

filter1

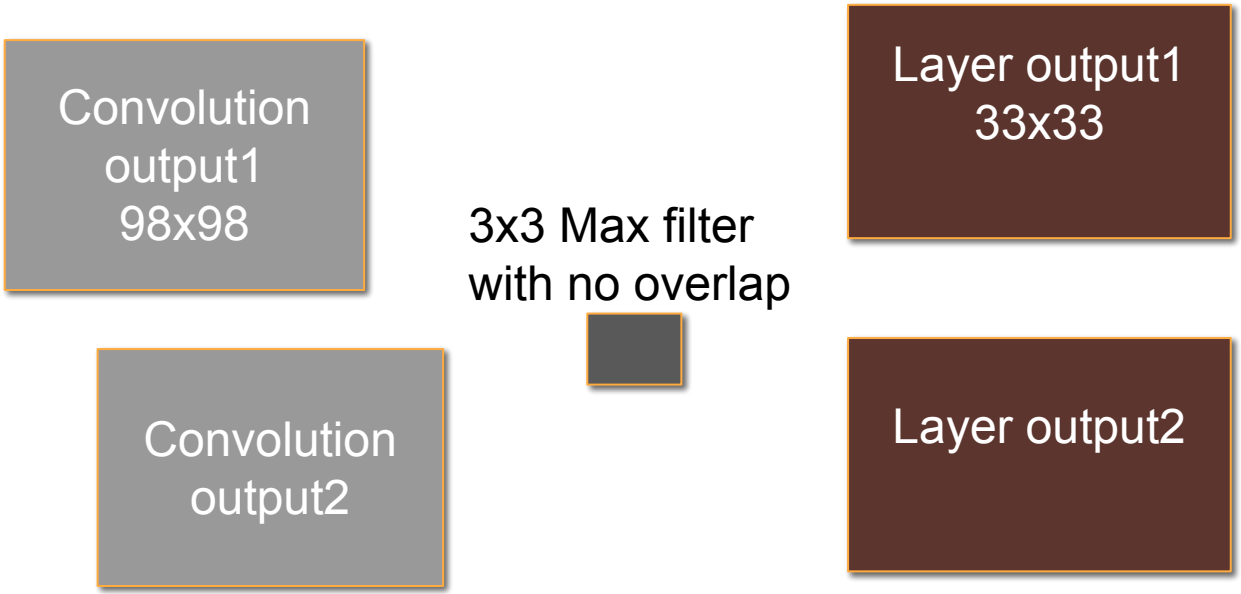
filter3
3x3

Convolution
output2

Convolution
output3

Pooling/subsampling

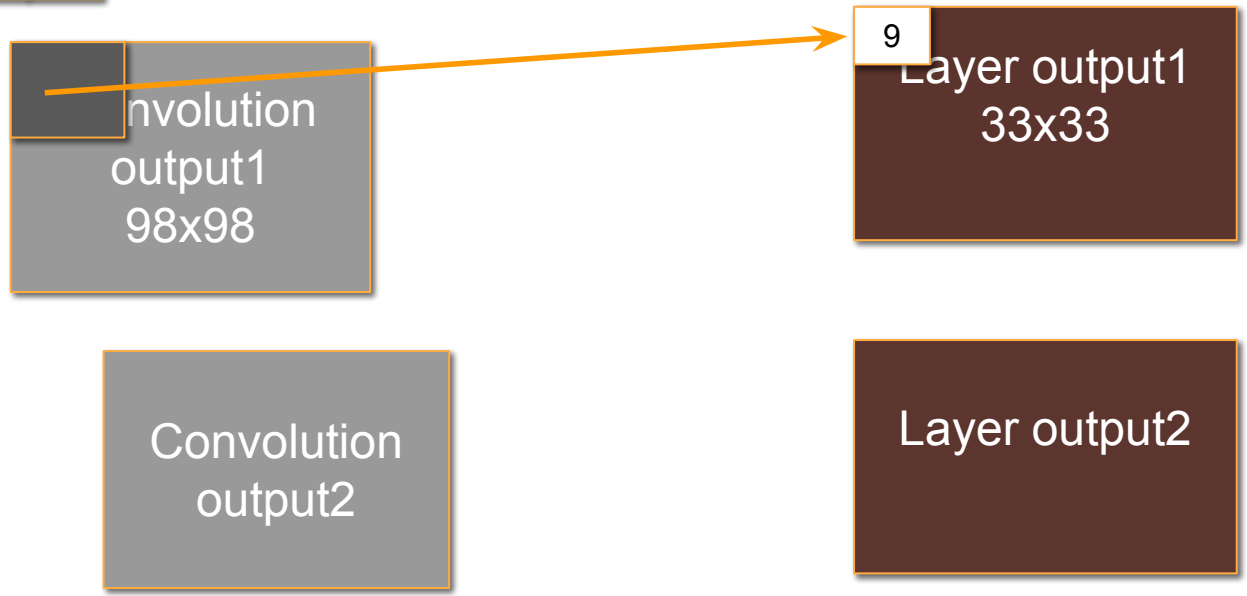
Reduce dimension of the feature maps



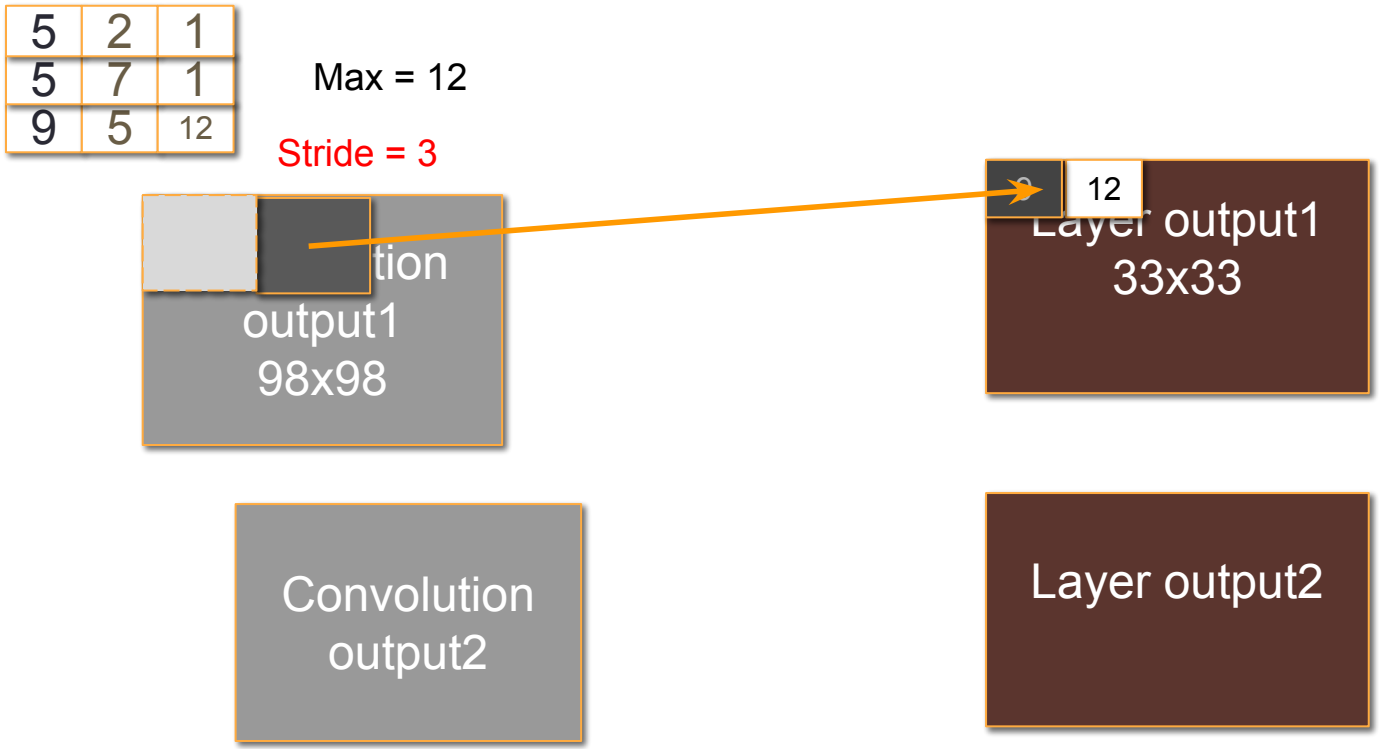
Pooling/subsampling

1	2	3
4	5	6
7	8	9

Max = 9

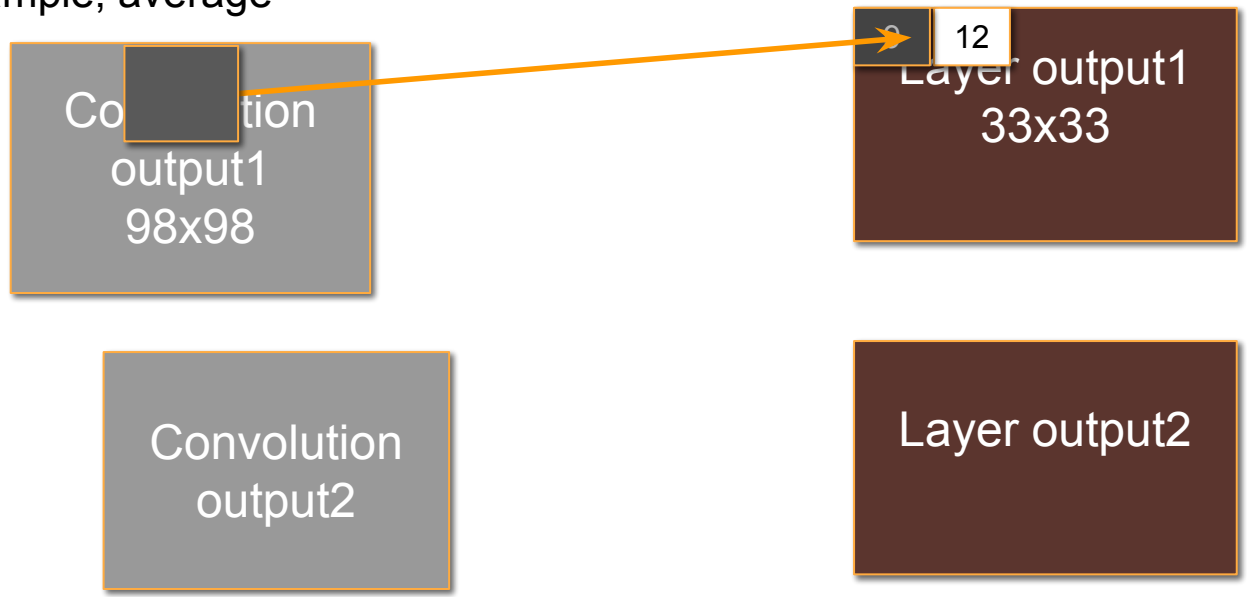


Pooling/subsampling



Pooling/subsampling

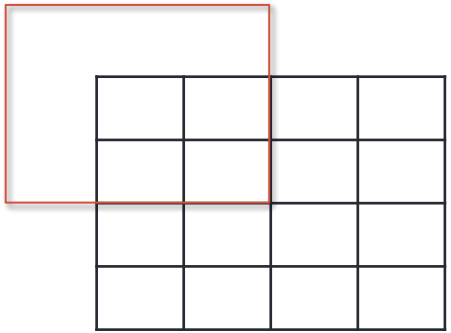
Can use other functions besides max
Example, average



Convolution puzzle

5 filters 3x3 filter pad, stride 1, pad 1

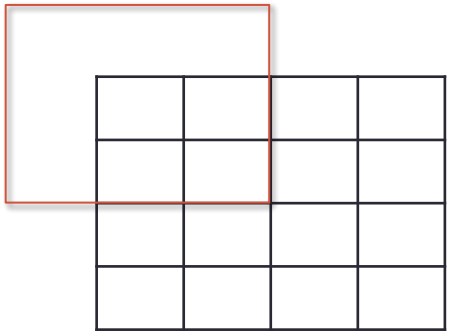
What is the output size?

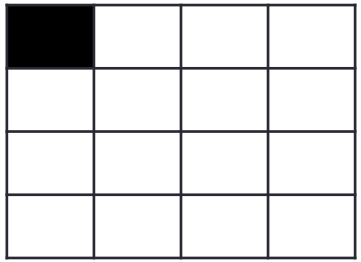


Convolution puzzle

5 filters 3x3 filter pad, stride 1, pad 1

What is the output size?

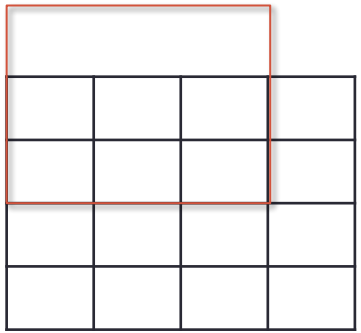


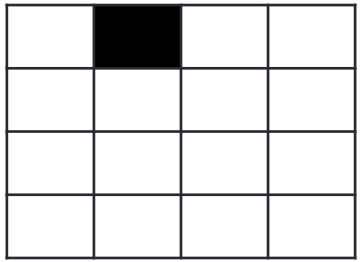


Convolution puzzle

5 filters 3x3 filter pad, stride 1, pad 1

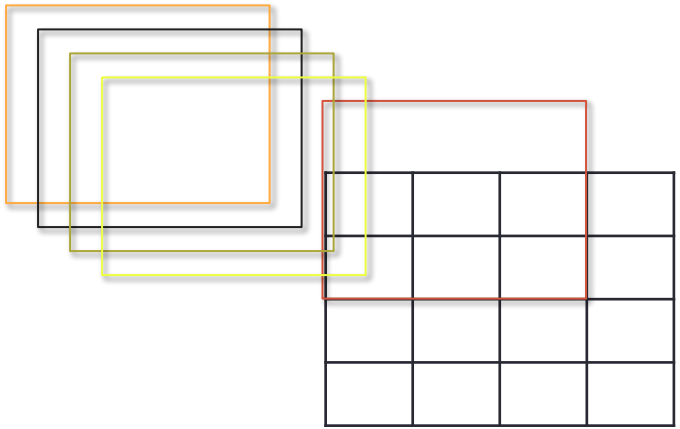
What is the output size?

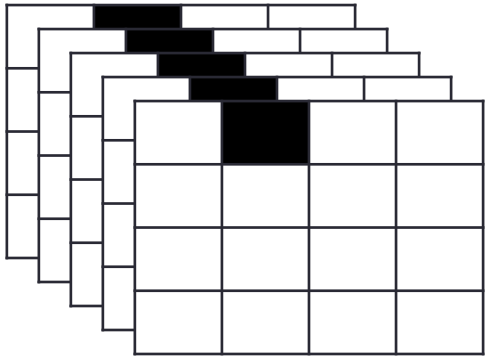




Convolution puzzle

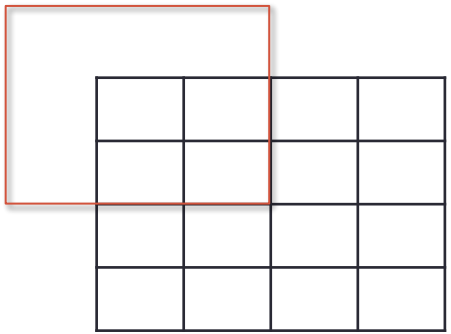
5 filters 3x3 filter pad, stride 1, pad 1





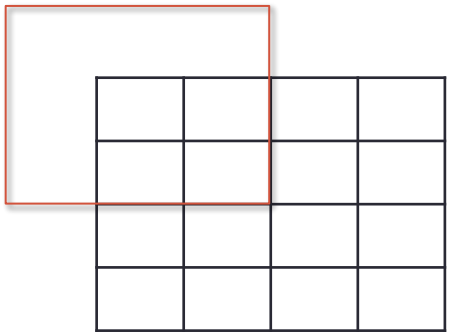
Convolution puzzle

3x3 filter pad, stride 2, pad 1



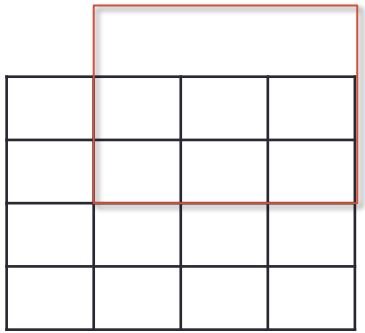
Convolution puzzle

3x3 filter pad, stride 2, pad 1



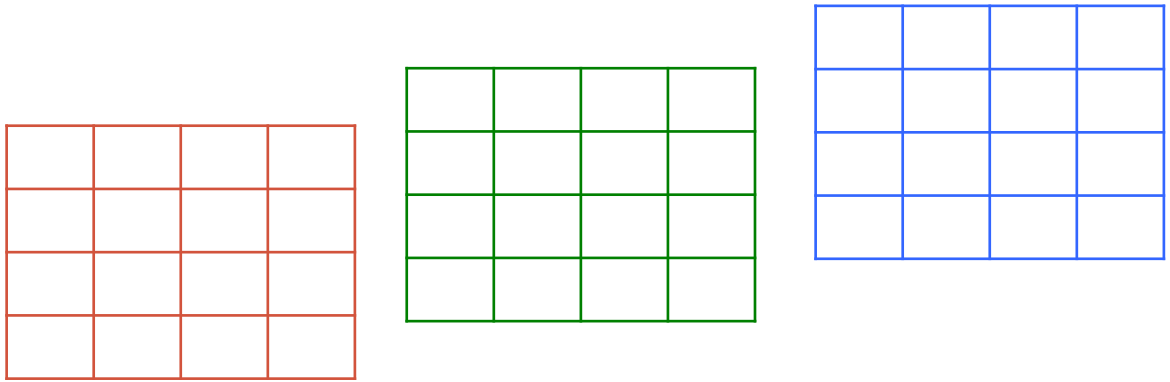
Convolution puzzle

3x3 filter pad, stride 2, pad 1



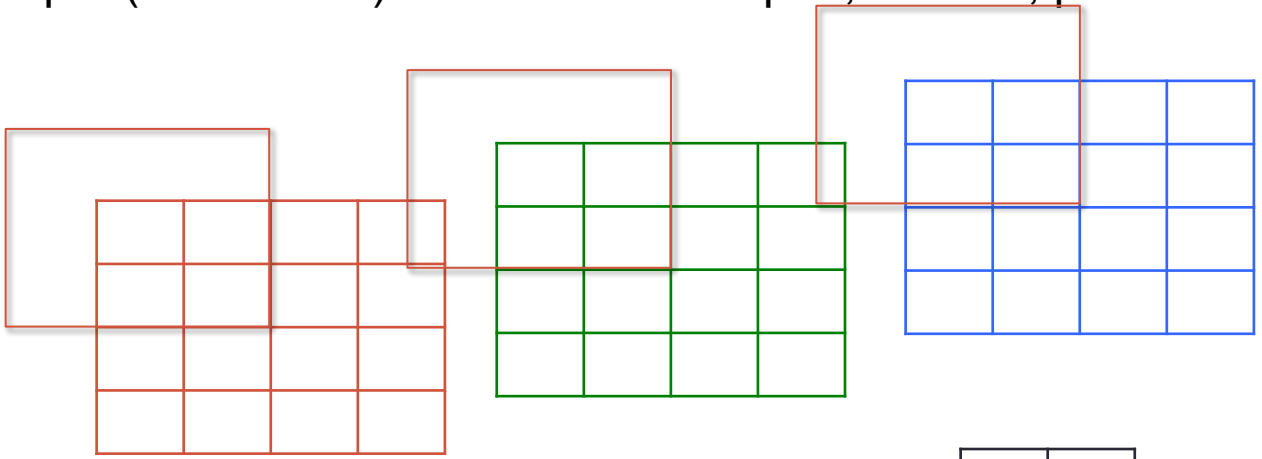
Convolution puzzle

RGB input (3 channels) 5 filters 3x3 filter pad, stride 2, pad 1

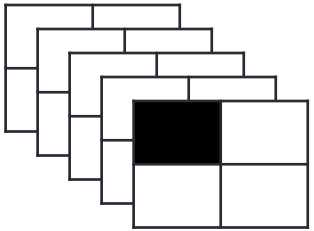


Convolution puzzle

RGB input (3 channels) 5 filters 3x3 filter pad, stride 2, pad 1

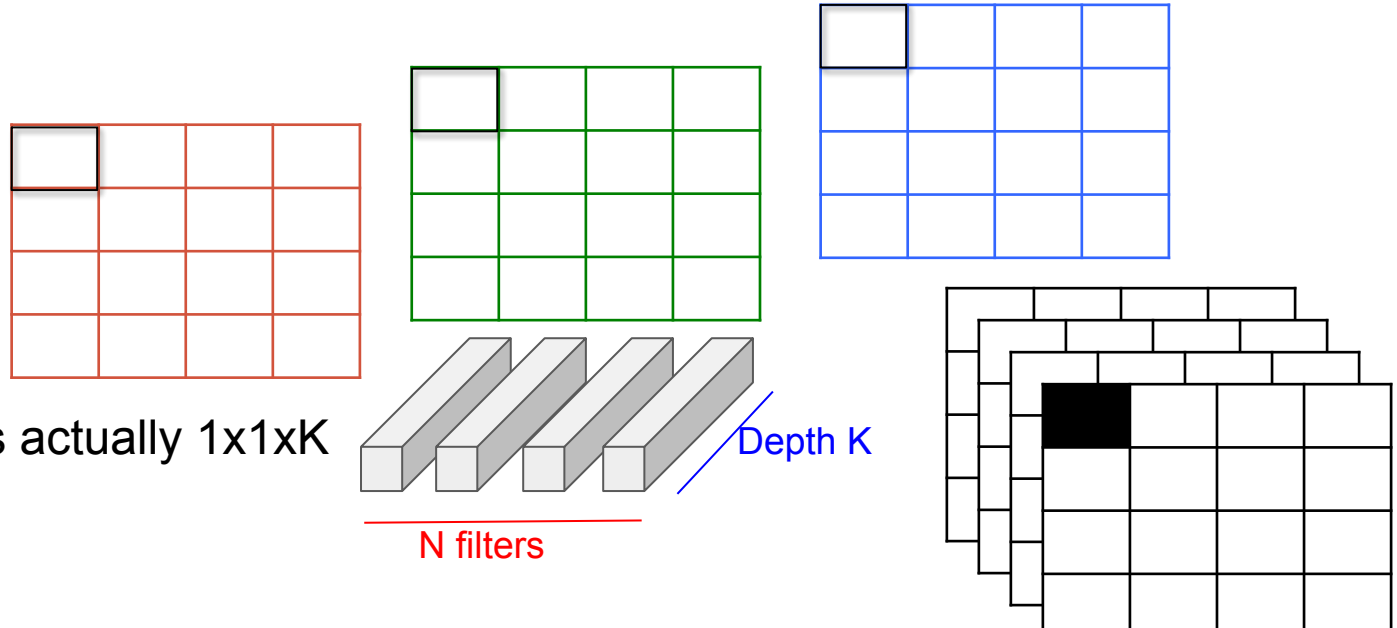


The filter is actually 3x3x3



1x1 convolution

Reduces the dimension of feature maps



The filter is actually 1x1xK

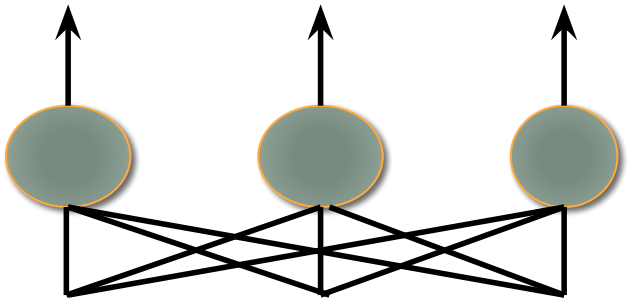
Dropout

A regularization technique for reducing overfitting

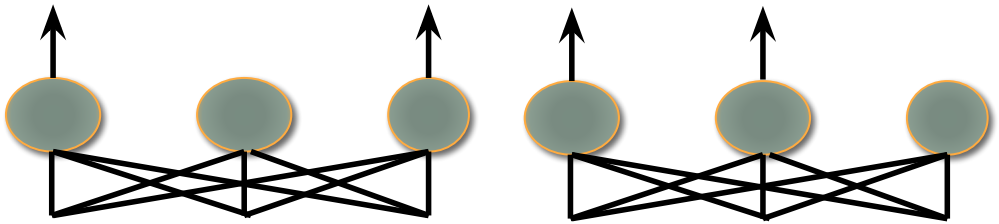
Randomly turn off different subset of neurons during training

- Network no longer depend on any particular neuron
- Force the model to have redundancy – robust to any corruption in input data

A form of performing model averaging (ensemble of experts)



Model



Dropout rate of ⅓

Hinton, Geoffrey "Improving neural networks by preventing co-adaptation of feature detectors" 2012

Batch norm

Normalizes the activations in the network to be “uniform”

For each mini-batch that goes through batch norm

1. Normalize by the mean and variance of the mini-batch for each dimension
2. Shift and scale by learnable parameters

Replaces dropout in some networks

$$\hat{x} = \frac{x - \mu_b}{\sigma_b}$$

$$y = \alpha\hat{x} + \beta$$

Agenda

Building blocks

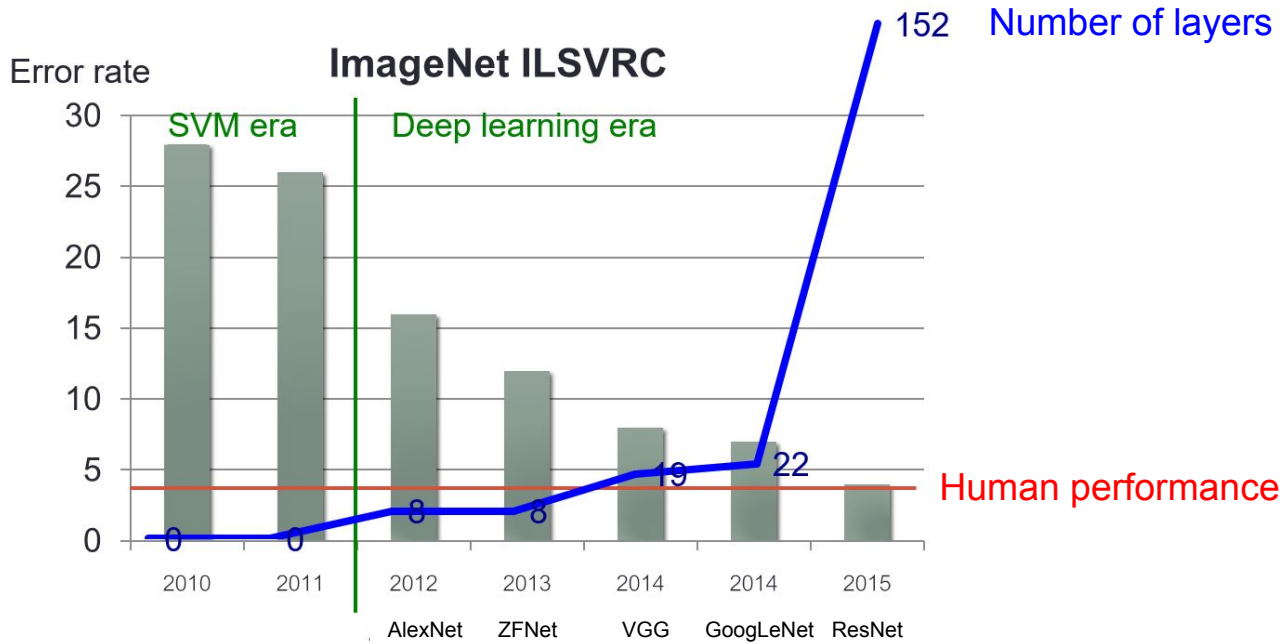
Object classifiers

- From LeNet to NASNet

Object detectors

- Region proposal based detectors
- Single stage detectors

A brief history of ImageNet



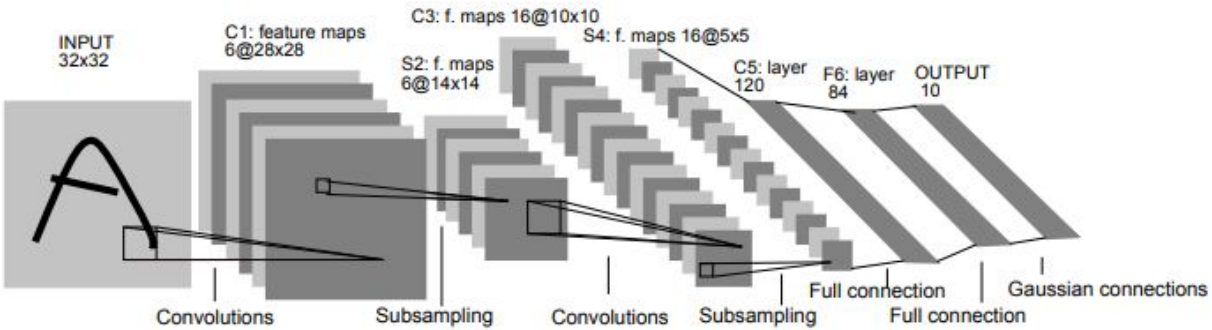
Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

LeNet

Convolutions and poolings followed by fully connected layers

Tanh activations

Ability to handle larger images limited by compute

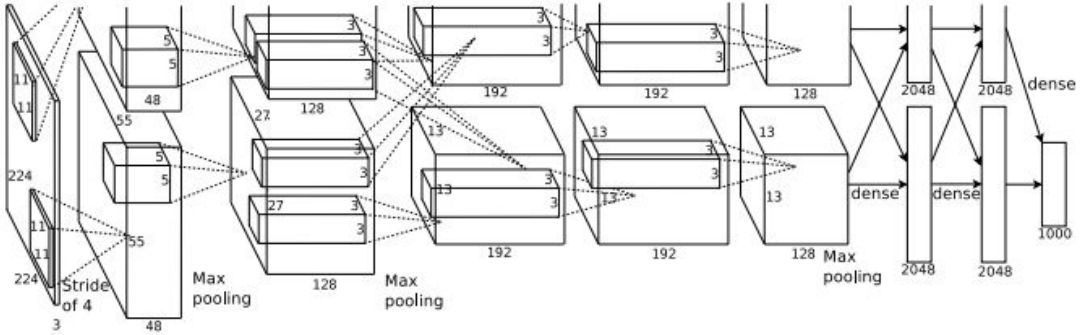


Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition,” 1998.

AlexNet

Convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum

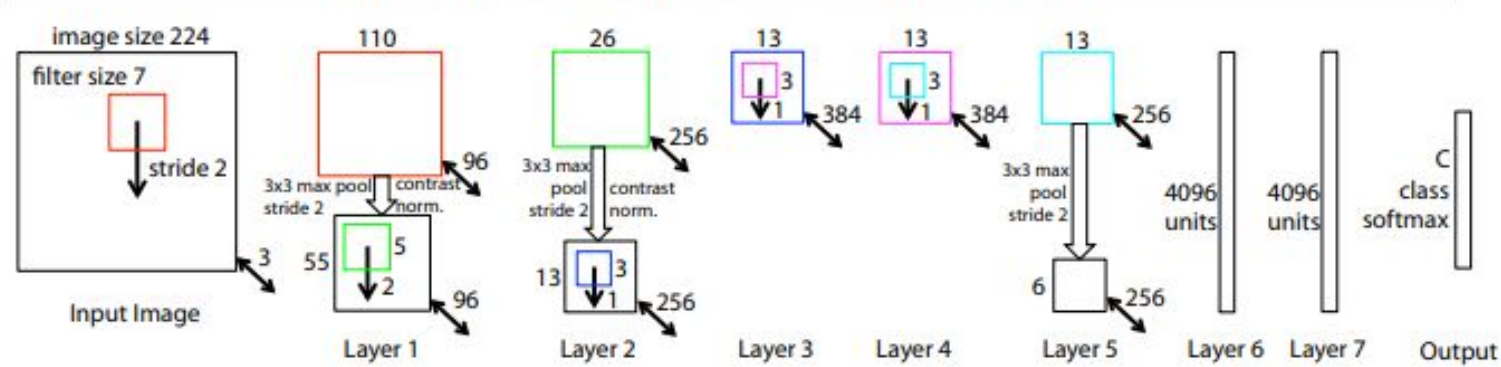
Two pipelines to fit into two GPUs



Alex Krizhevsky, et al. “ImageNet Classification with Deep Convolutional Neural Networks”, 2012.

ZFNet

Tweaking hyperparameters



Matthew D. Zeiler, Rob Fergus, "Visualizing and Understanding Convolutional Networks," 2013

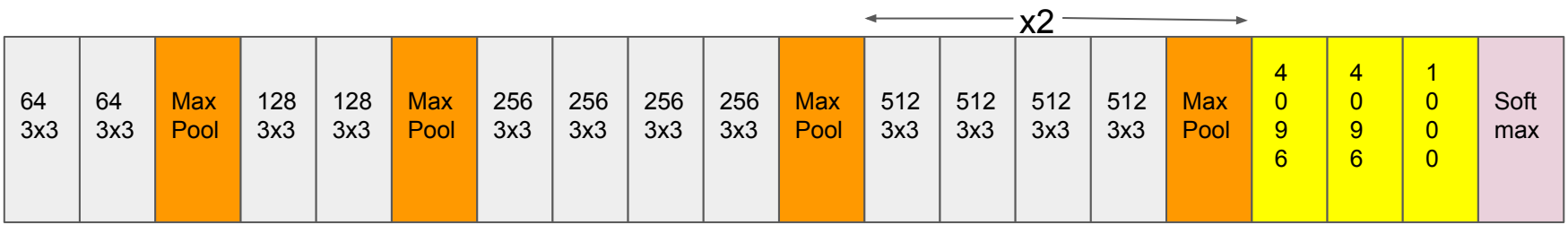
VGG

Uniform 3x3 convolutional filters

19 layers! Pushing the limits of conventional wisdom at that time.

Used by many since pre-train weights are publically available

VGG19



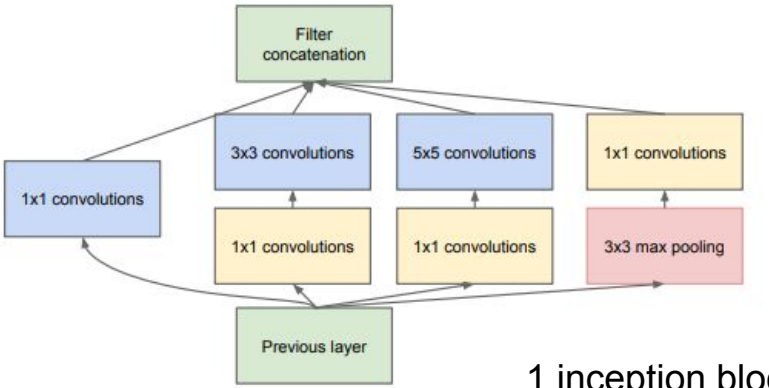
Karen Simonyan, Andrew Zisserman“Very Deep Convolutional Networks for Large-Scale Image Recognition” 2014

GoogLeNet (Inception v1)

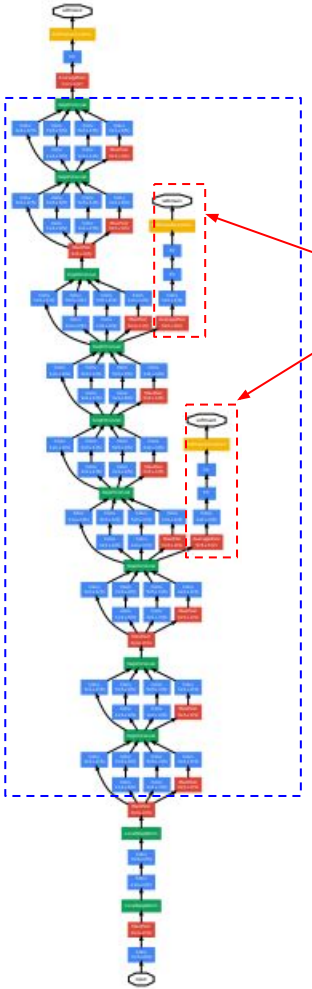
Multiple filter sizes per layer (objects come in different scales)

Dimensionality reduction via 1x1 convolution

Multiple softmax losses to help the gradient problem



1 inception block

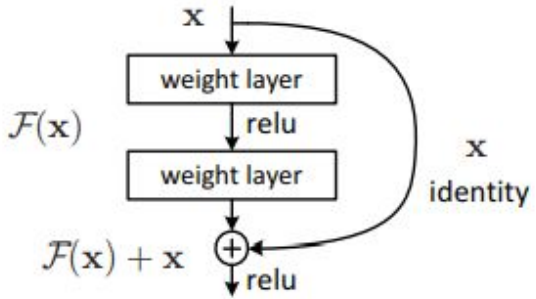


Christian Szegedy, et al. “Going Deeper with Convolutions,” 2014

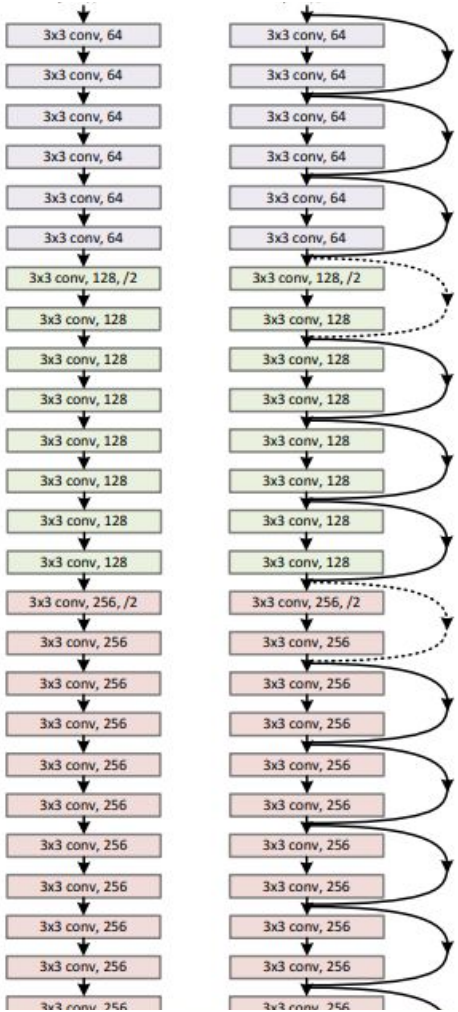
ResNet (Residual Network)

Batch norm

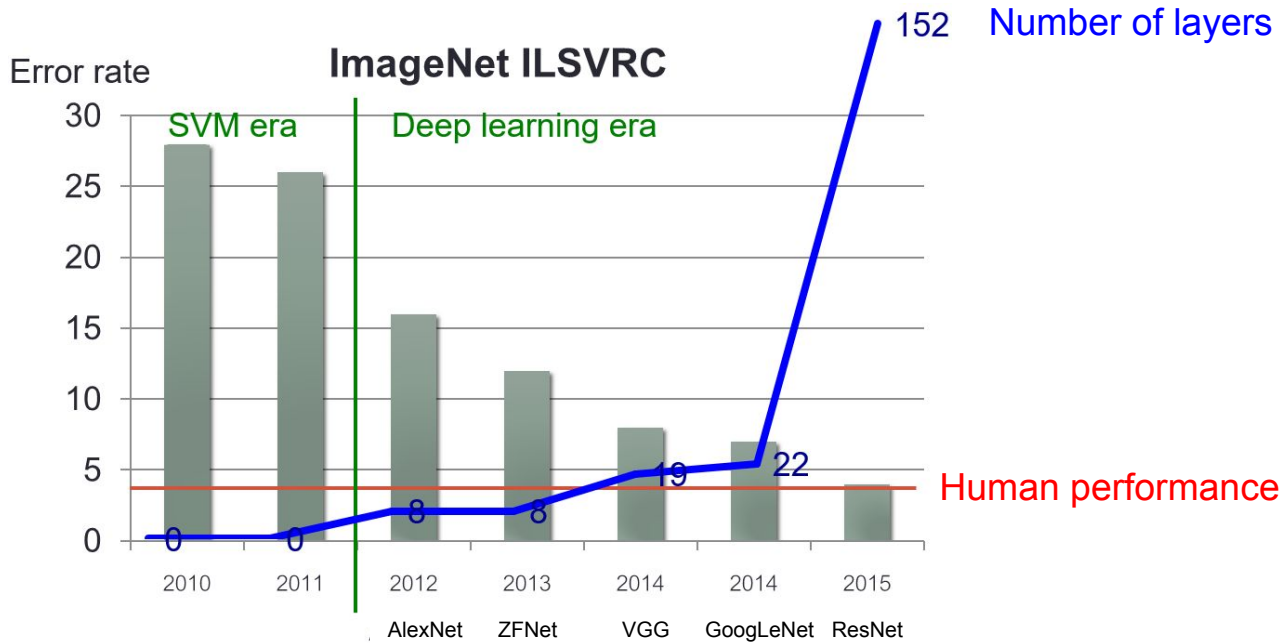
Extra “skip connections” to reduce the vanishing gradient problem



Kaiming He, et al. “Deep Residual Learning for Image Recognition” 2015



A brief history of ImageNet



Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

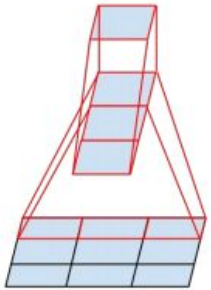
Inception v2+v3

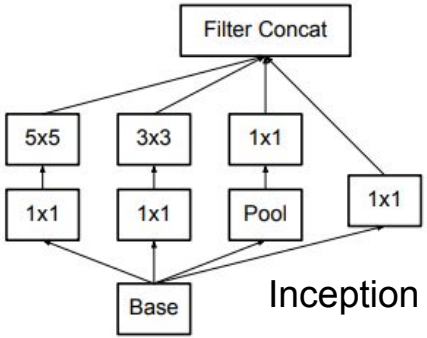
Factorized convolution

3x3 -> 3x1 and 1x3

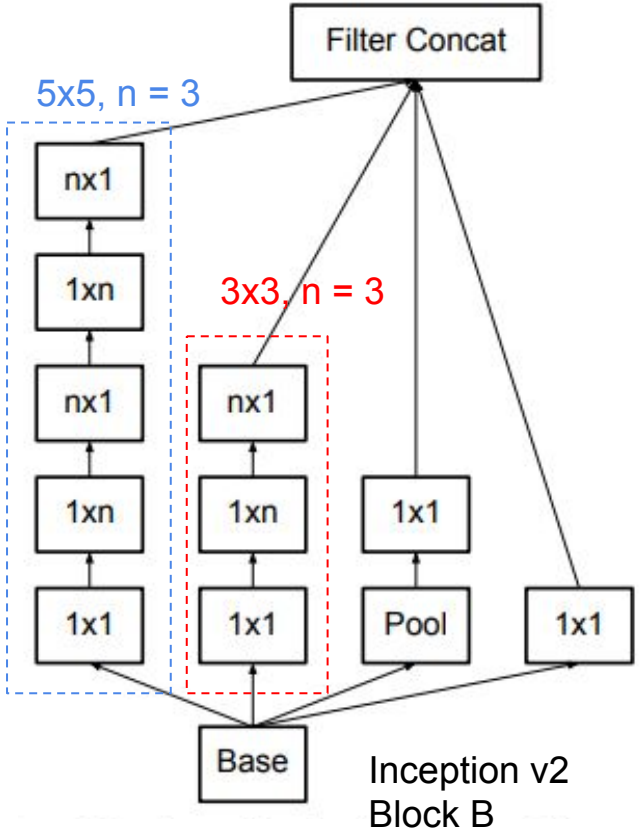
3 types of inception blocks

RMSprop, Batch norm, label smoothing





Inception v1



Inception v2
Block B

Christian Szegedy, “Rethinking the Inception Architecture for Computer Vision,” 2015

Xception

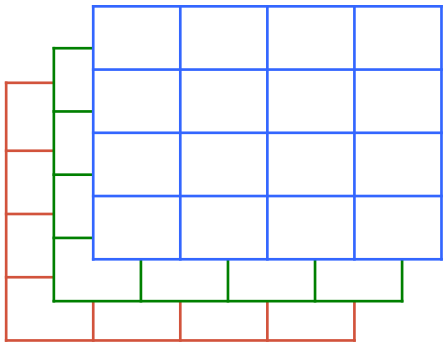
Depthwise separable convolution: two-step convolution

1. Depthwise convolution
2. 1x1 convolution

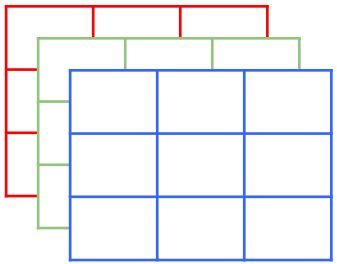
Typical convolution 3x3 filter is 3x3xinput channel

Depthwise convolution 3x3 filter is 3x3x1
1 filter per input channel

input



One regular convolution filter



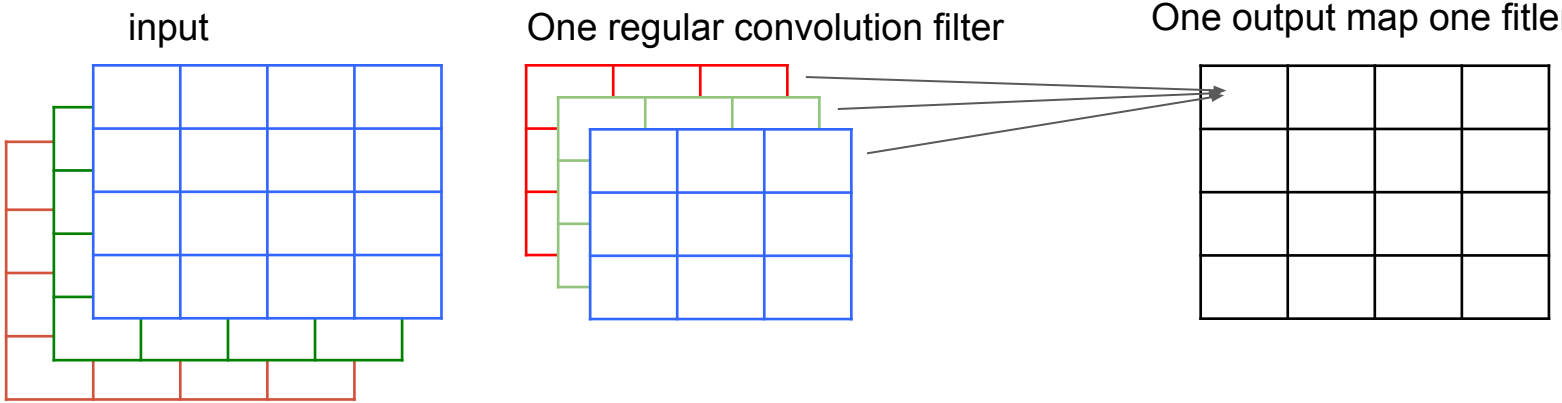
Xception

Depthwise separable convolution: two-step convolution

1. Depthwise convolution
2. 1x1 convolution

Typical convolution 3x3 filter is 3x3xinput channel

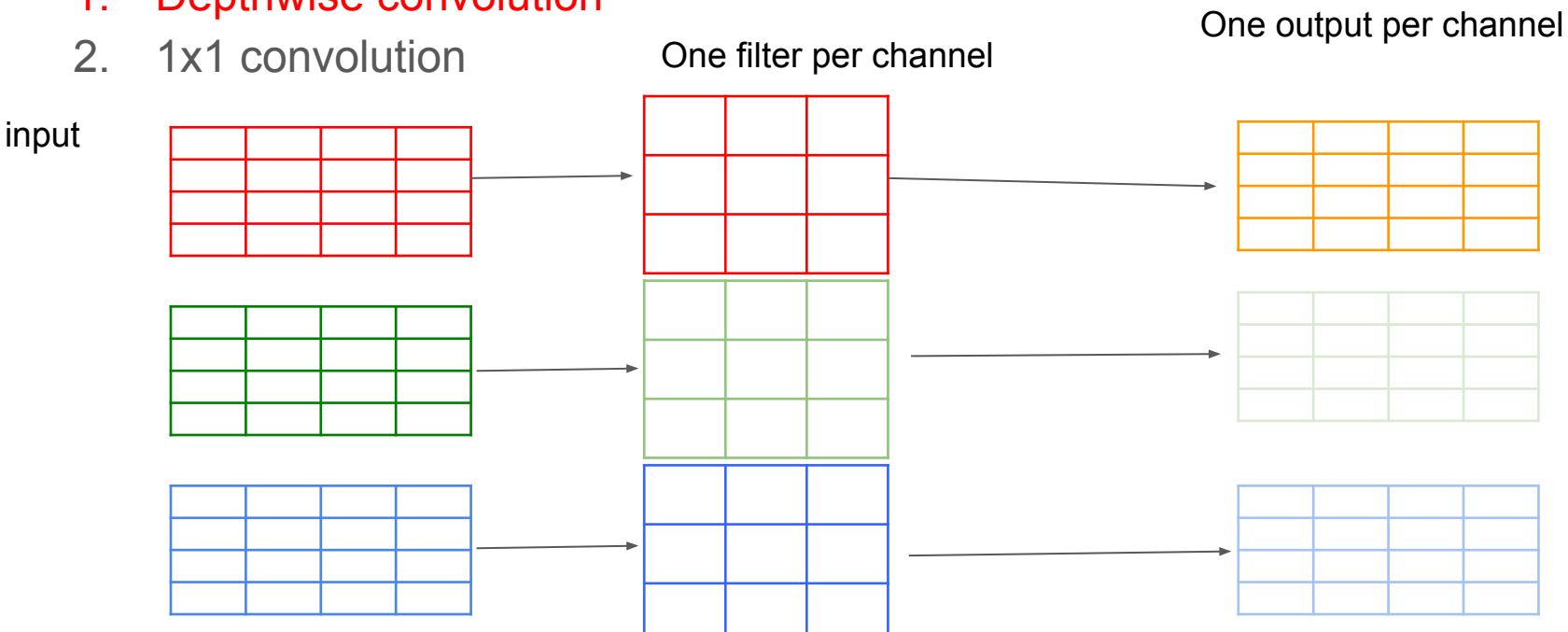
Depthwise convolution 3x3 filter is 3x3x1
1 filter per input channel



Xception

Depthwise separable convolution: two-step convolution

1. Depthwise convolution
2. 1x1 convolution

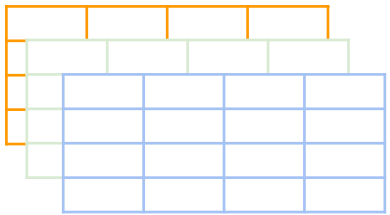


Xception

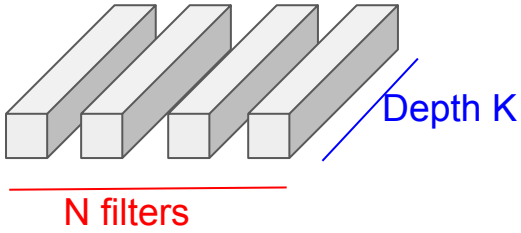
Depthwise separable convolution: two-step convolution

1. Depthwise convolution
2. 1x1 convolution

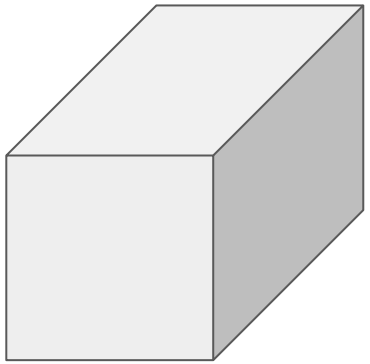
Output from depthwise convolution



1x1 filters



Final output



Xception

Replace convolutions in inception with depthwise separable convolutions

Smaller model

Faster compute

Comparable with Inception v3 while much faster

Used in MobileNet and other models

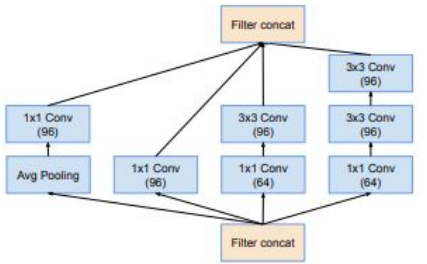
François Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions” 2016.
Andrew G. Howard, et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications” 2017.

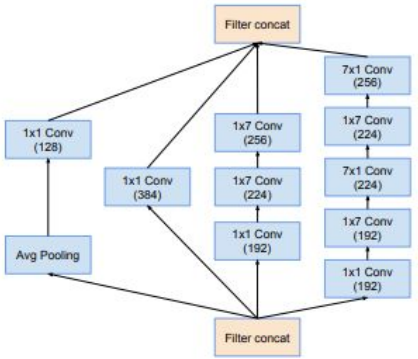
Inception v4

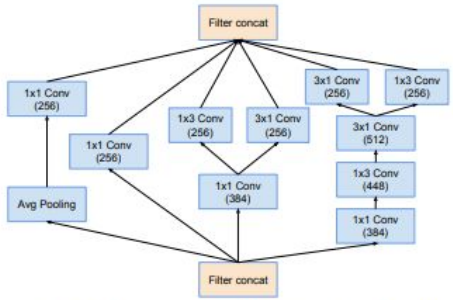
Same three types of inception blocks from v3

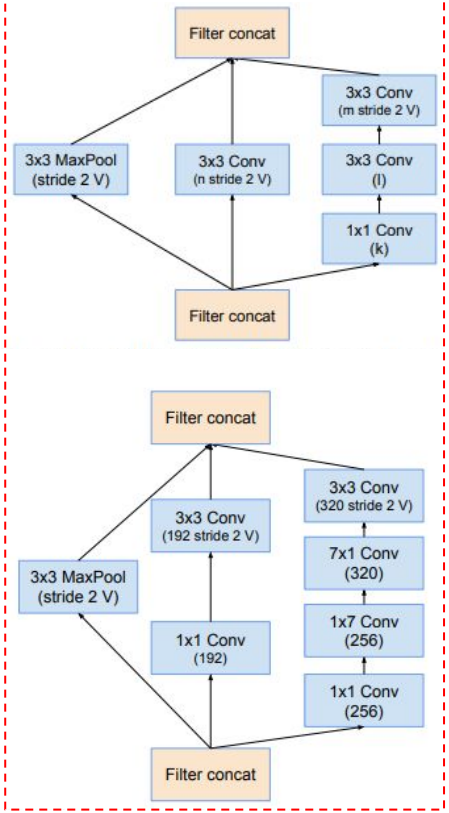
Add two types of reduction blocks for reducing the size of the grid (super pooling blocks)

Inception blocks









Reduction blocks

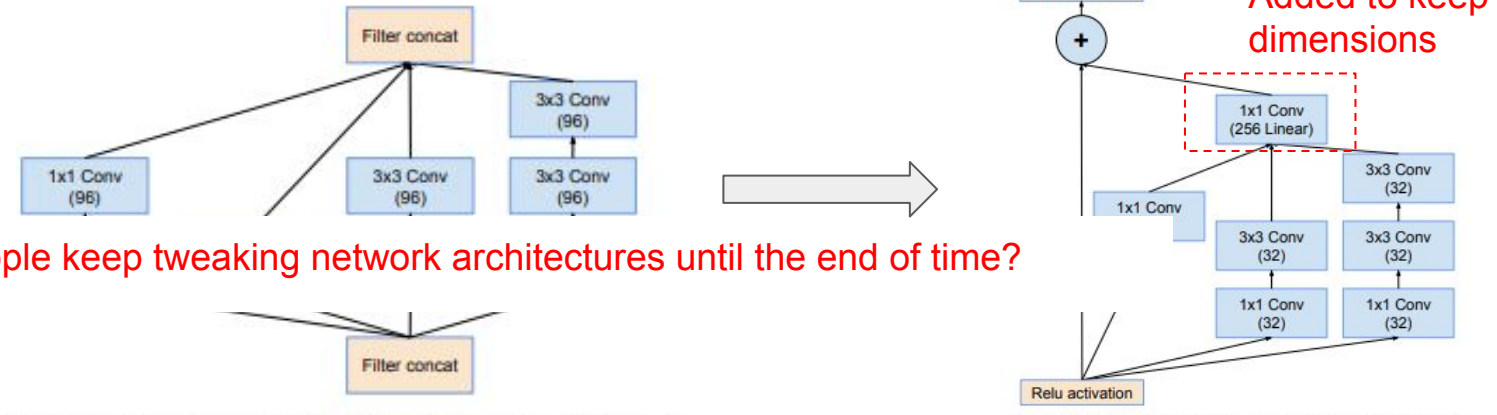
Christian Szegedy, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning” 2016.

Inception-ResNet v1-v2

Introduce residual connections into inception blocks

Poolings changed to additions, 1x1 added to keep dimensions for the residual

Similar idea to ResNeXt



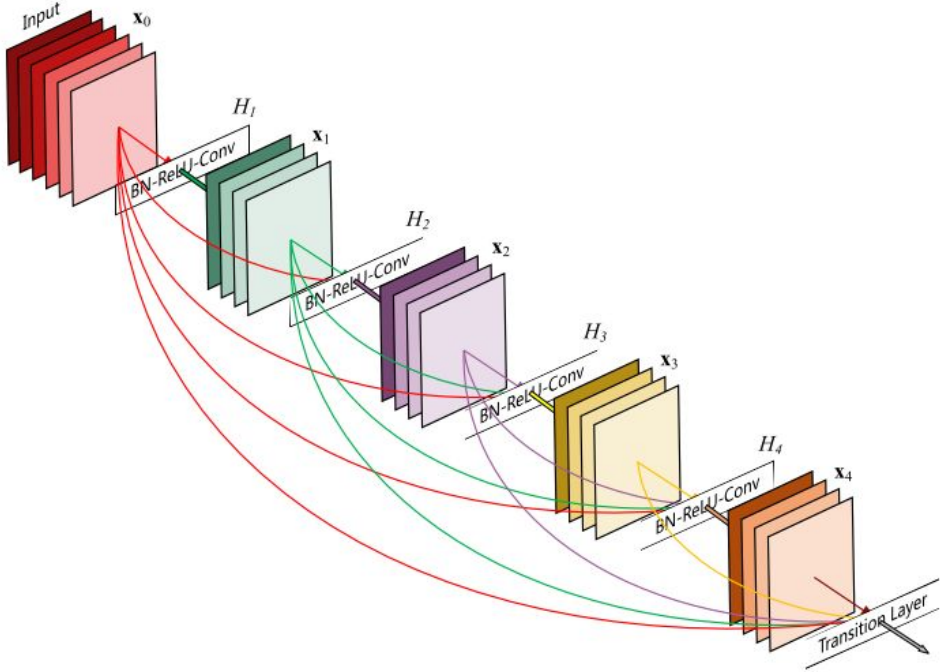
Do people keep tweaking network architectures until the end of time?

Christian Szegedy, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning” 2016.

DenseNet

Instead of adding the residual concatenate the feature maps from previous layers

Densely connect multiple layers

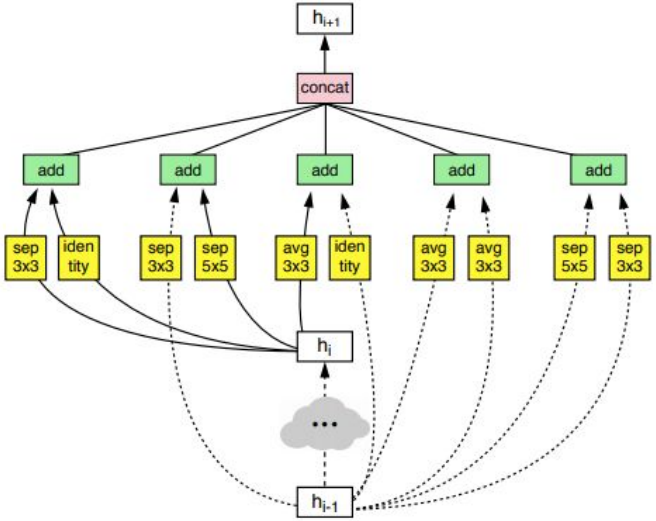


NasNet

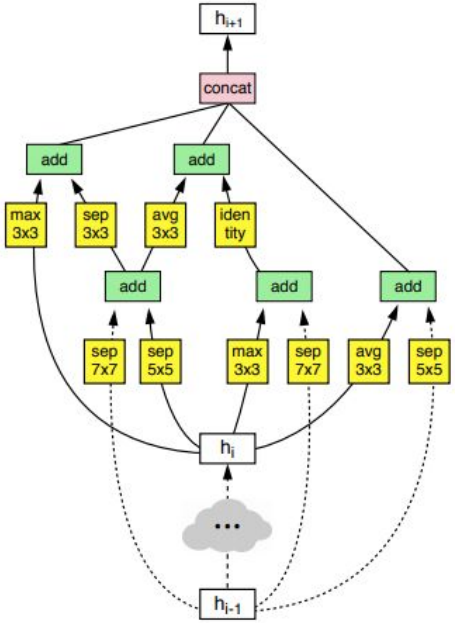
Neural Architecture Search Network

Use reinforcement learning to search for the best network configuration

Lowers compute while maintaining high accuracy



Inception block



Reduction block

Barret Zoph, et al. “Learning Transferable Architectures for Scalable Image Recognition,” 2017.

Swish activation function

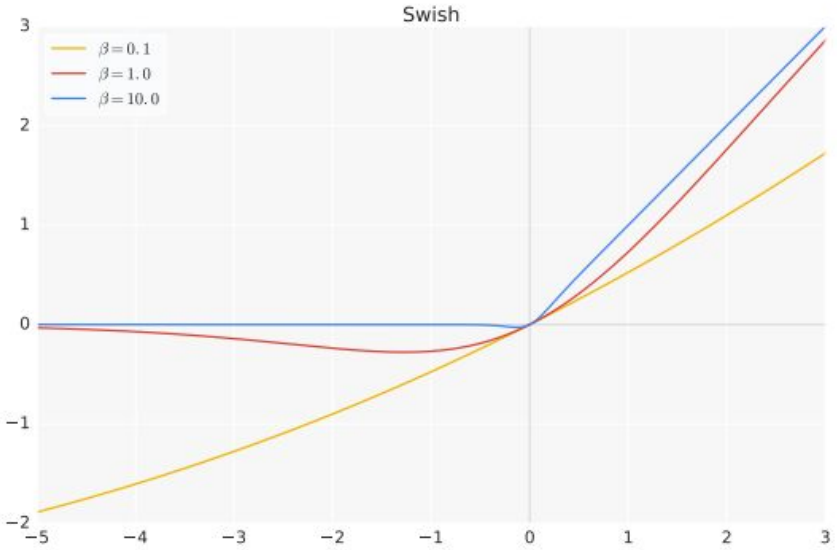
Can use RL to search for activation function

Best function: y = x sigmoid (βx)

Proven theoretically to be the best general non-linearity

Works well for many tasks

Just replace ReLU with Swish



Prajit Ramachandran, et al., “Searching for Activation Functions,” 2017.
Mirco Milletarí, et al., “Expectation propagation: a probabilistic view of Deep Feed Forward Networks,” 2018.

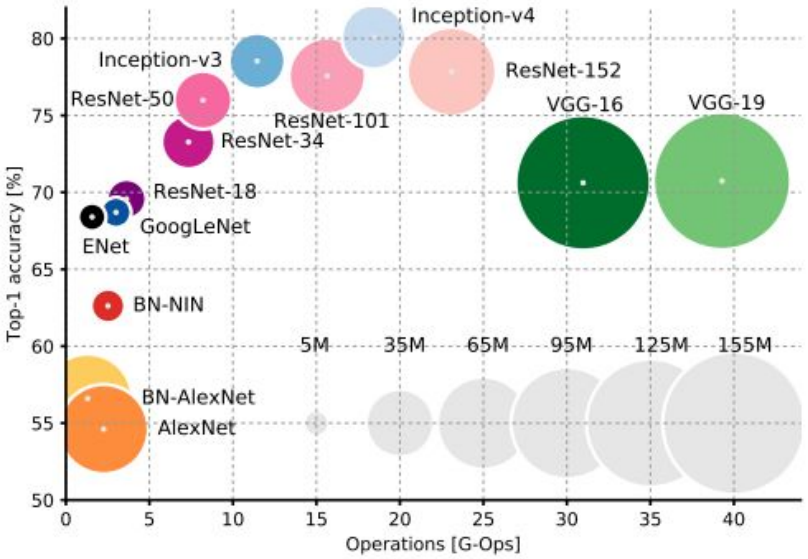
Object classifiers summary

We've gone a long way from LeNet. Most successful tricks:

Dropout, residual, batch norms, multi-path, data augmentation

These object classifiers are often called backbones and used by other models

As developers, your main concerns would be the accuracy, model size, transferability and compute.



Alfredo Canziani, et al. "An analysis of deep neural network models for practical applications," 2017

Agenda

Building blocks

Object classifiers

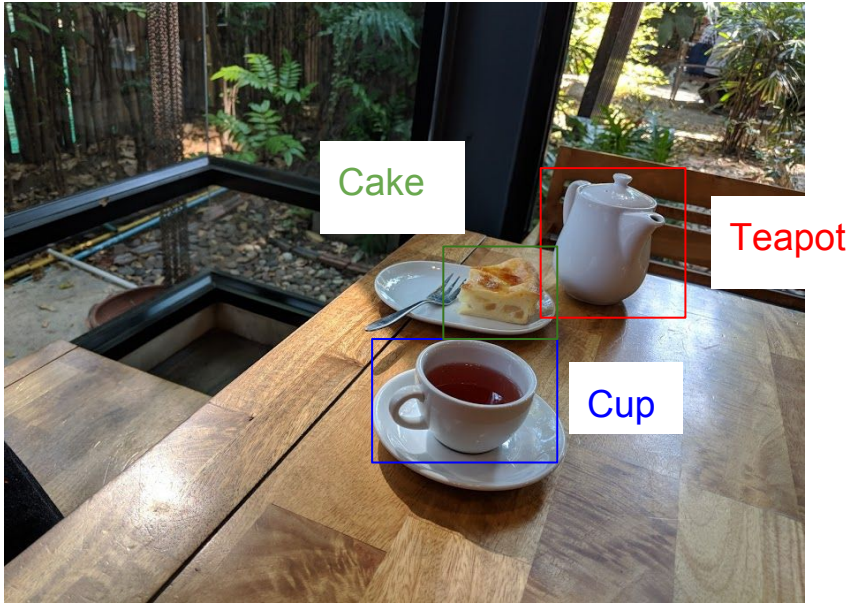
- From LeNet to NASNet

Object detectors

- Region proposal based detectors
- Single stage detectors

Object detection

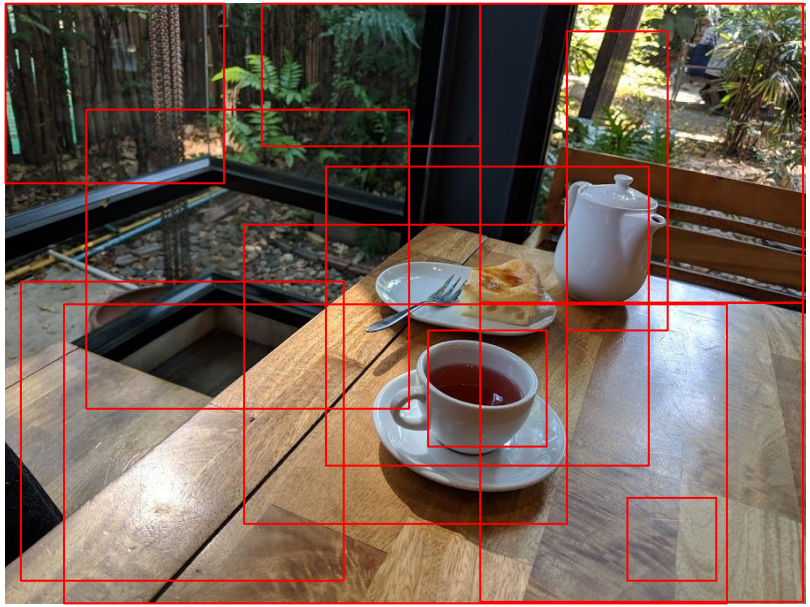
Locate every object in an image and identify them



Simplest solution: sliding window

Slide a object classifier over every possible region and scale

Expensive. Can we do better?



Object classifier (backbone)

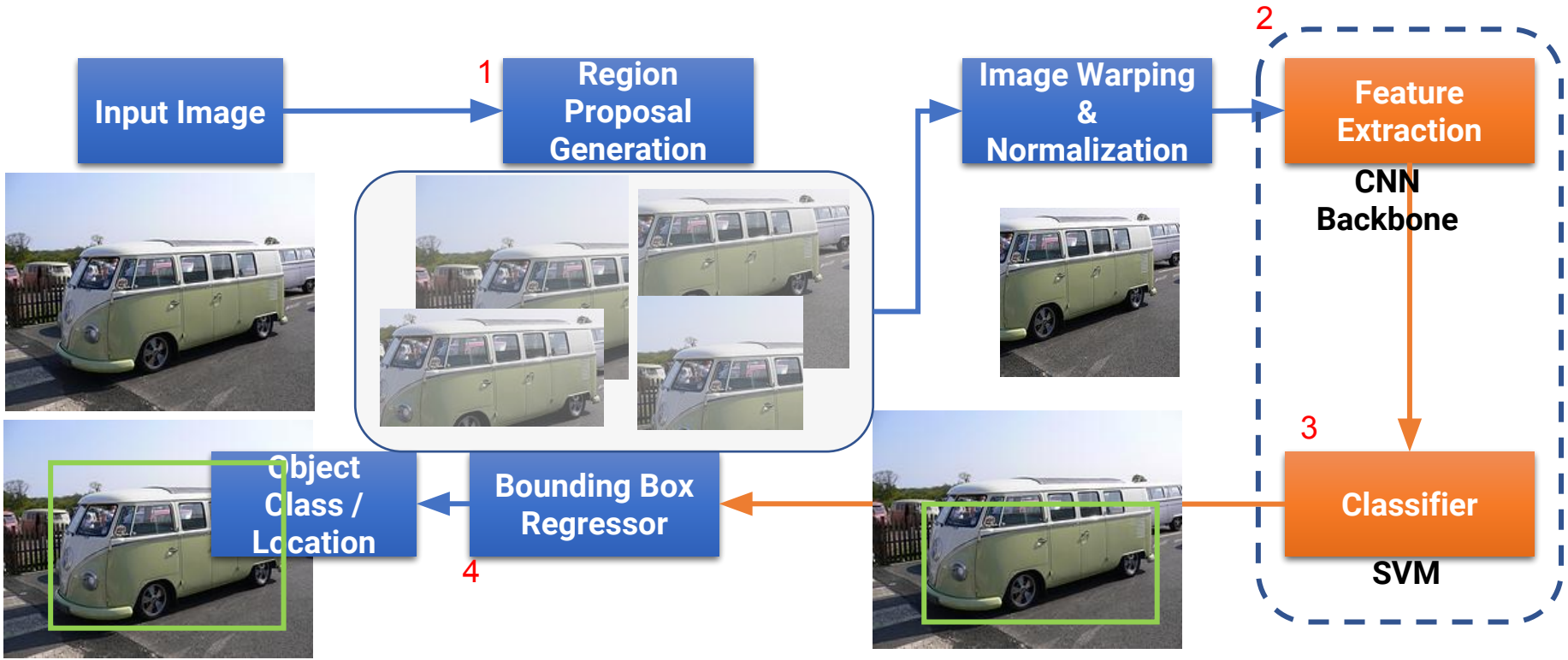
Candidate lists with probabilities

Object Detection Comprehensive Review

1. Proposal-based detector
   - R-CNN
   - Fast-RCNN
   - Faster-RCNN

2. Single stage detector
   - YOLO
   - SSD

Regions with Convolutional Neural Network Feature (R-CNN)



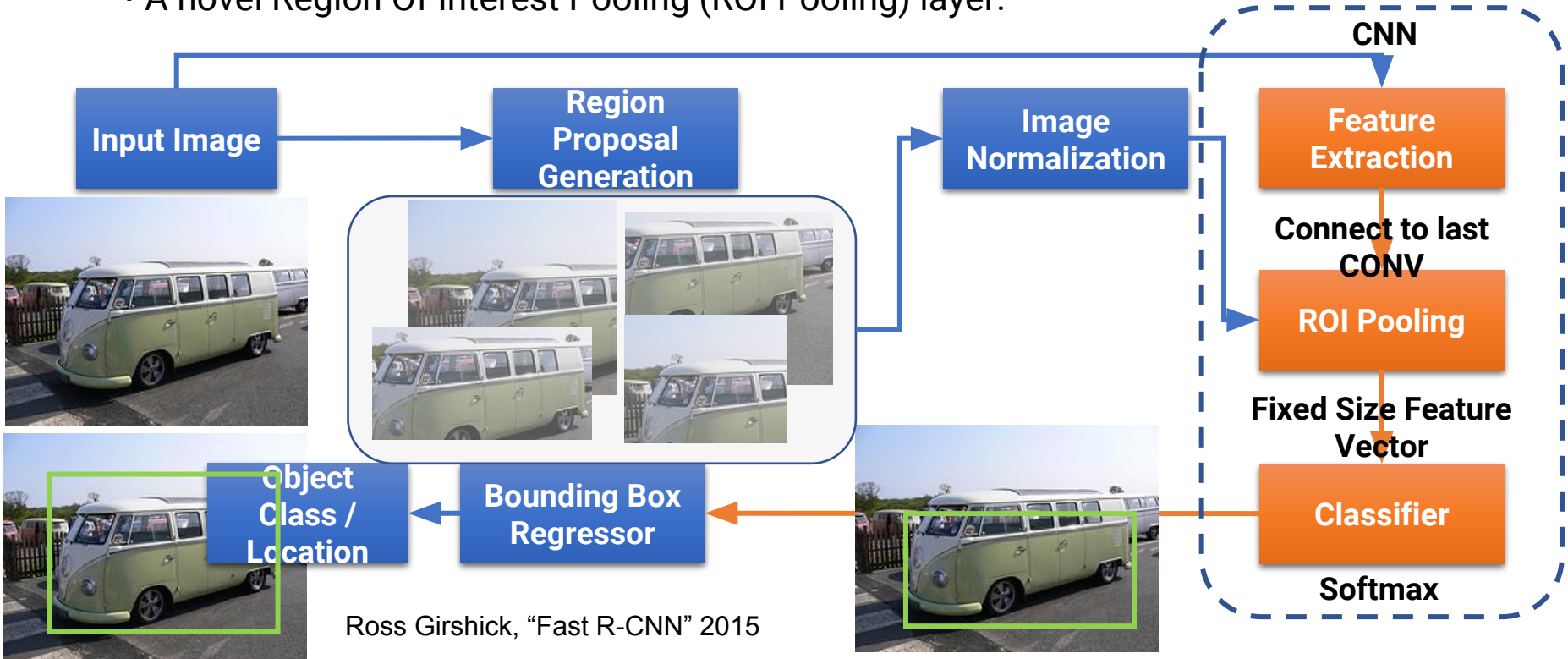
Ross Girshick, “Rich feature hierarchies for accurate object detection and semantic segmentation” 2013.

Regions with Convolutional Neural Network Feature (R-CNN)

- Region proposal generation (selective search) is computationally expensive.
- No computation sharing between regions.
- In the training process, R-CNN suffers from the following complex multi-stages training.
    - The backbone needs to be adapted to the new task (detection and regression)
    - The SVM classifier
    - The bounding box regressor

Fast-RCNN

- Computation sharing for each region proposal by computing the feature extraction once then extract features for each region.
- A novel Region Of Interest Pooling (ROI Pooling) layer.

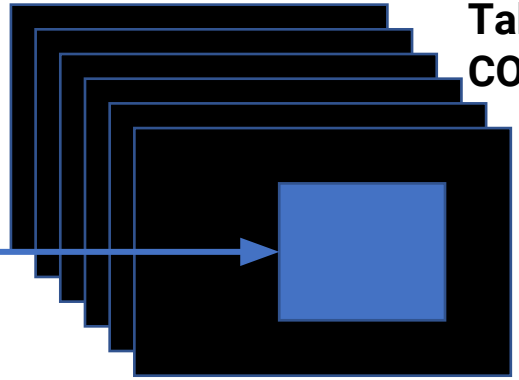


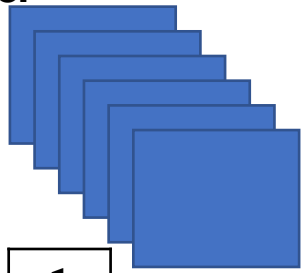
Ross Girshick, “Fast R-CNN” 2015

Fast-RCNN

Feature Extraction

Take features directly from CONV layer

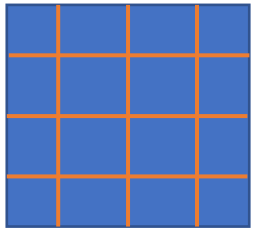




Corresponding location respect to region proposal location

Last CONV output feature maps

Selective Search



1
2
3
5
78
90
15
..

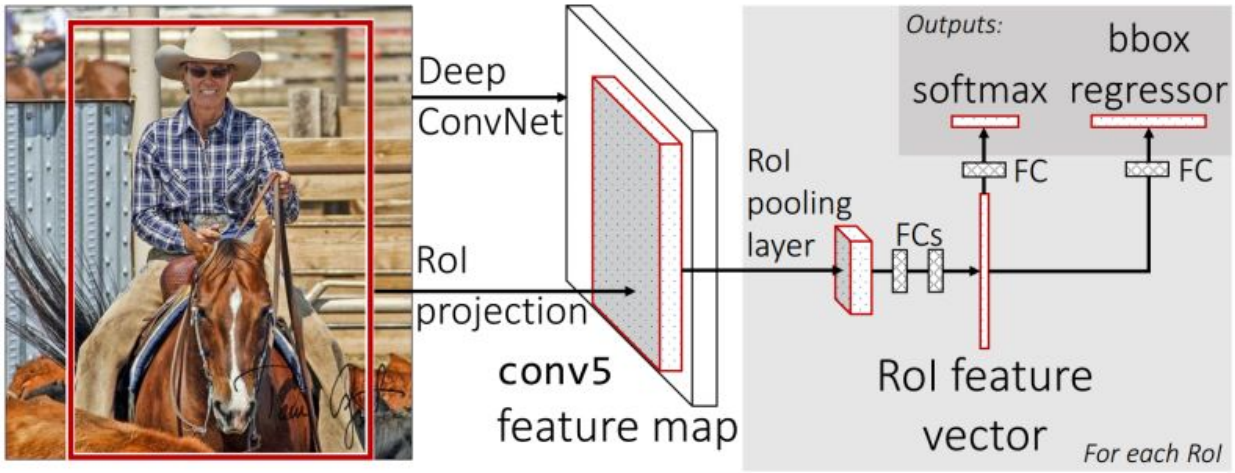
Fixed size feature vector

Divided into fixed partition and take max value

Fast-RCNN

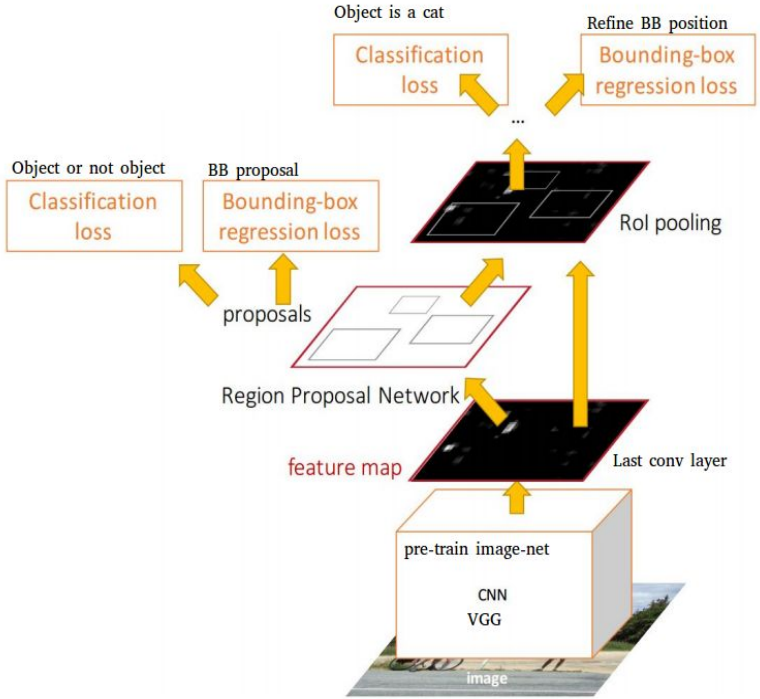
- Fast R-CNN still relay on external object proposal algorithm (Selective Search)

Selective Search



Faster-RCNN

- Combine the region proposal and classification into a single network
- End-to-End learning from region proposal to classification stage.



Shaoqing Ren, et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks” 2015

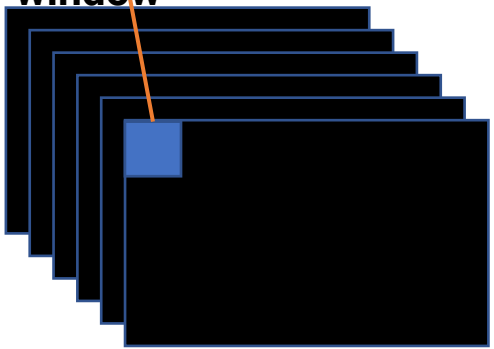
Faster-RCNN

Input Image
Feature Extraction
Region Proposal Generation
Connect to last CONV
CNN
ROI Pooling
Fixed Size Feature Vector
Bounding Box Regressor
Classifier
Softmax
Object Class / Location

Region proposal

Feature Extraction

3x3 sliding window



1
2
3
5
78
90
15
..

Fixed Size feature vector

1
2
3
5
78
90
15
..

Region Proposal Regressor

Relative location (x,y) to image size

Last CONV output feature maps

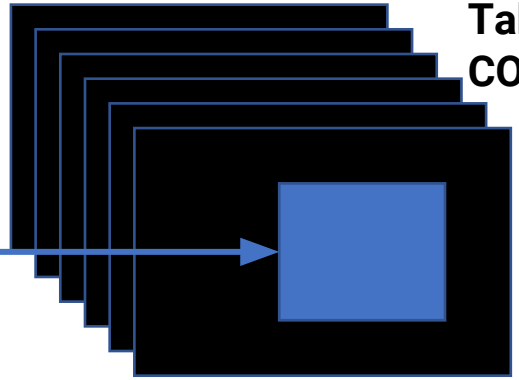
Objection Proposal Classifier

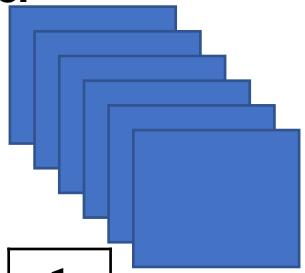
Object / Not Object

Final output

Feature Extraction

Take feature directly from CONV layer

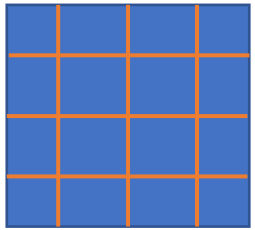




Corresponding location respect to region proposal location

Last CONV output feature maps

Region Proposal Generator



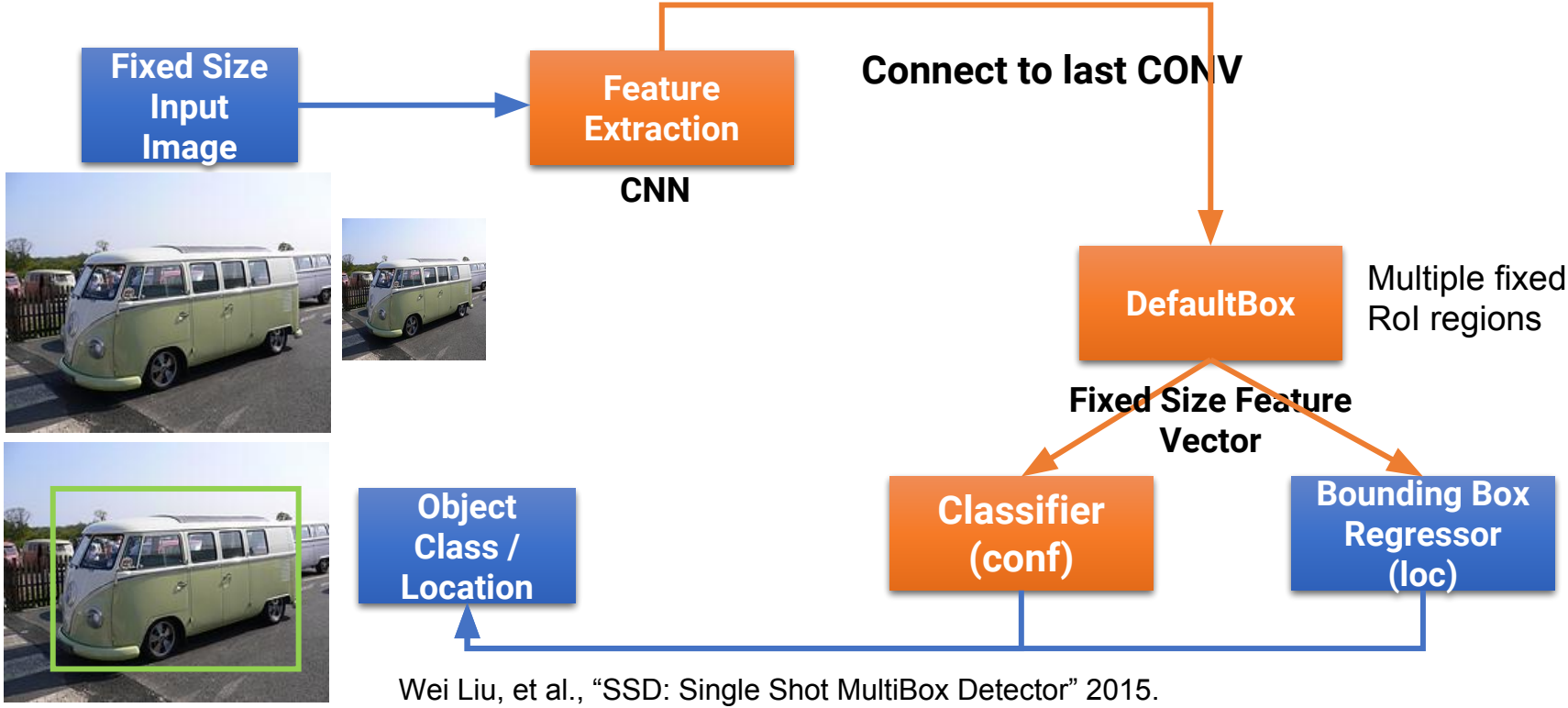
1
2
3
5
78
90
15
..

Fixed size feature vector

ROI Pooling: Divided into fixed partition and take max value

SSD : Single Shot Multibox Detector

- Trade off accuracy for computational speed => Realtime detector

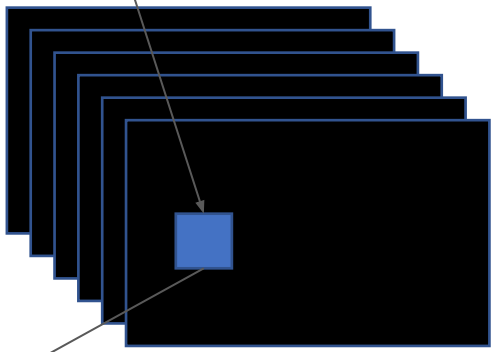


Wei Liu, et al., “SSD: Single Shot MultiBox Detector” 2015.

SSD : Single Shot Multibox Detector : Default Box Idea

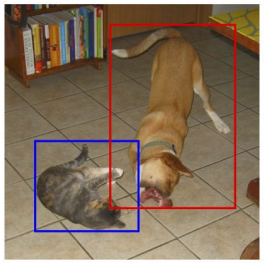
Feature Extraction

3x3 conv

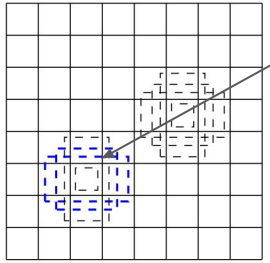


In training process, SSD will optimize loc,conf which corresponds to the object's real position based on IOU criterion.

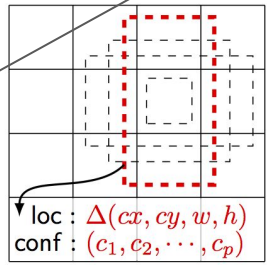
Last CONV output feature maps



(a) Image with GT boxes



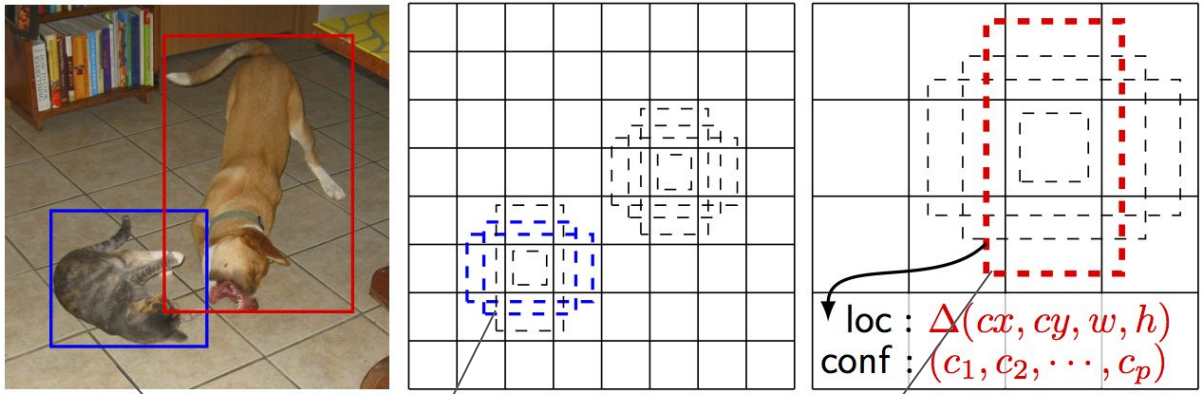
(b) 8×8 feature map



(c) 4×4 feature map

Every feature map cell is associated with a set of default bounding boxes of different dimensions and aspect ratios. Represents "default box" in xywh coordinate system (loc) and class output (conf)

SSD : Single Shot Multibox Detector



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

$$MultiBoxLoss(x, c, l, g) = L_{conf}(CrossEntropyLoss(x, c)) + L_{loc}(SmoothL_1Loss(x, l, g))$$

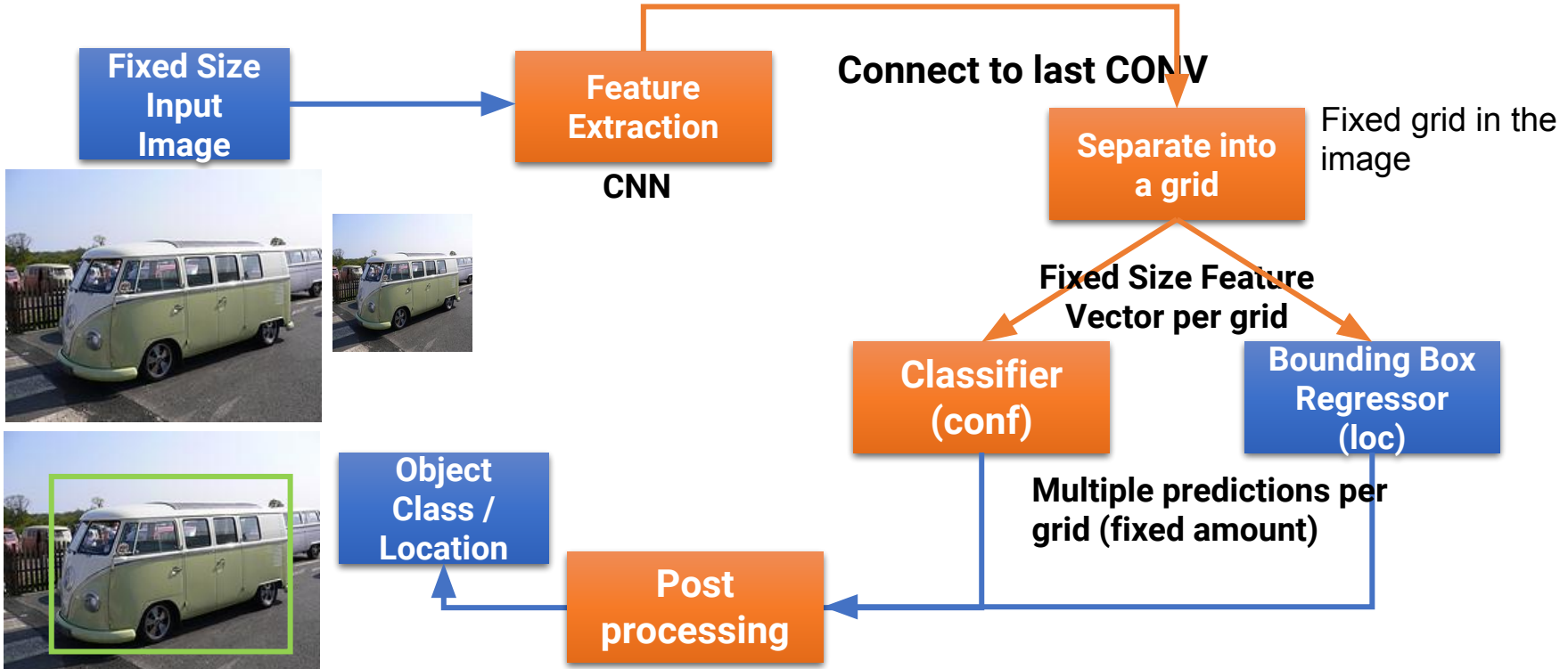
IOU > 0.5 (Match with O1)

IOU > 0.5 (Match with O2)

SSDLoss = Classification loss + Localization Loss

YOLO: You Only Look Once

- Similar concept as SSD, fixed locations to look for things



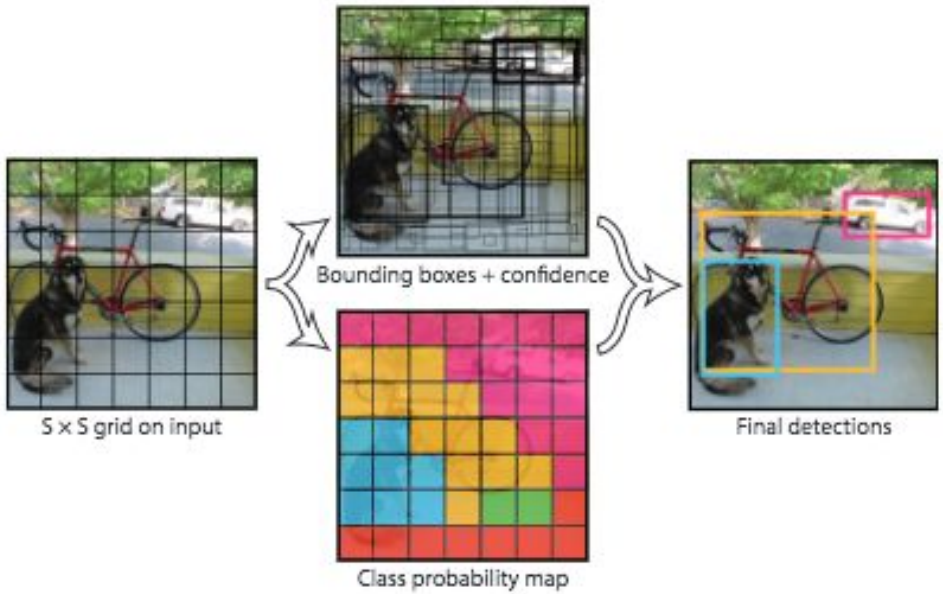
Joseph Redmon, “You Only Look Once: Unified, Real-Time Object Detection,” 2015

YOLO (You Only Look Once)

SSD have multiple boxes per location (multi-resolution), YOLO has one box per location

YOLO makes multiple predictions per box at once.

Limitation of YOLO: cannot predicts things that are close together.



YOLO v2 (YOLO9000)-v3

Batch norm

Anchor boxes: fixed set of bounding boxes per grid

Handles multi-resolution by concatenating features from multiple parts of the backbone

Joseph Redmon, et al. “YOLO9000: Better, Faster, Stronger” 2016
Joseph Redmon, et al. “YOLOv3: An Incremental Improvement” 2018

Summary

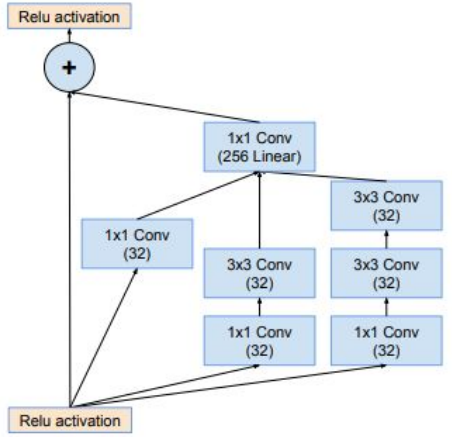
Building blocks

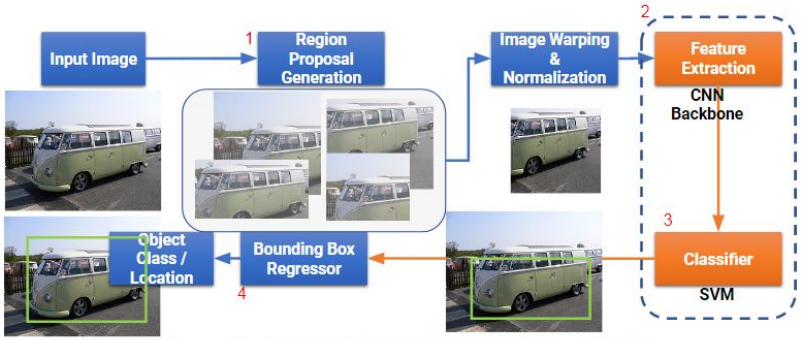
Object classifiers

- From LeNet to NASNet

Object detectors

- Region proposal based detectors
- Single stage detectors





Other networks to look for

Backbones

Feature Pyramid Networks (multi-scale backbone) https://arxiv.org/abs/1612.03144

SqueezeNet (small network) https://arxiv.org/abs/1602.07360

DARTS (improved NASNet) https://arxiv.org/abs/1806.09055

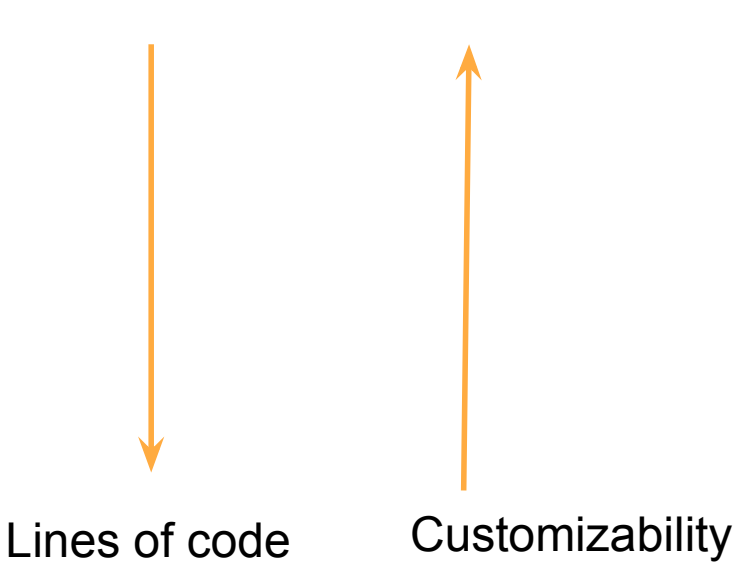
Object detectors

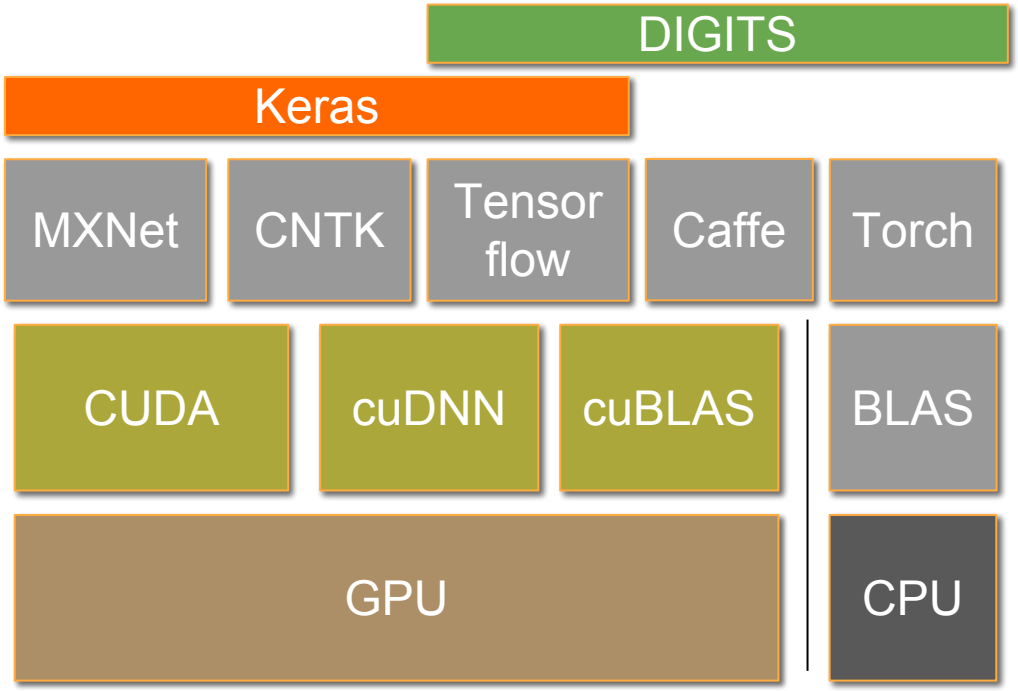
Mask R-CNN (good object detector that also does segmentation) https://arxiv.org/abs/1703.06870

RetinaNet (fast detector that have good accuracy) https://arxiv.org/abs/1708.02002

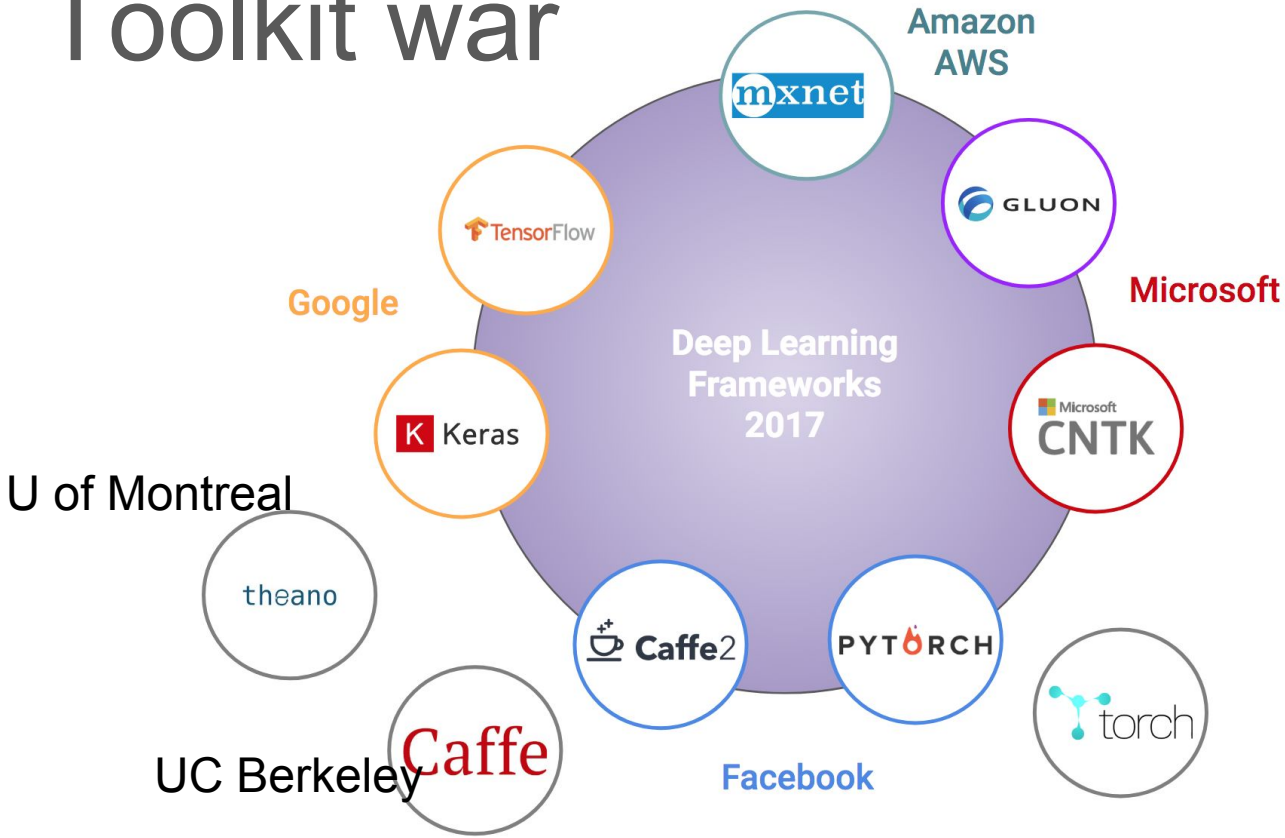
What toolkit

Tradeoff between customizability and ease of use





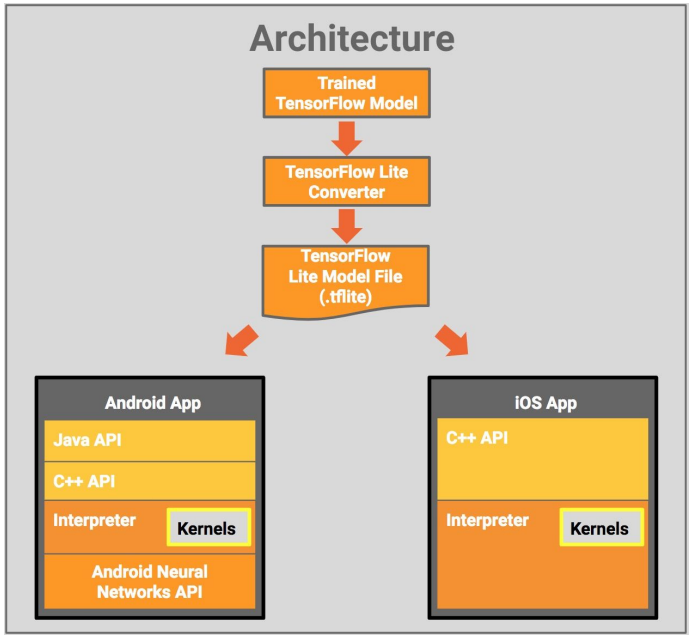
Toolkit war



https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

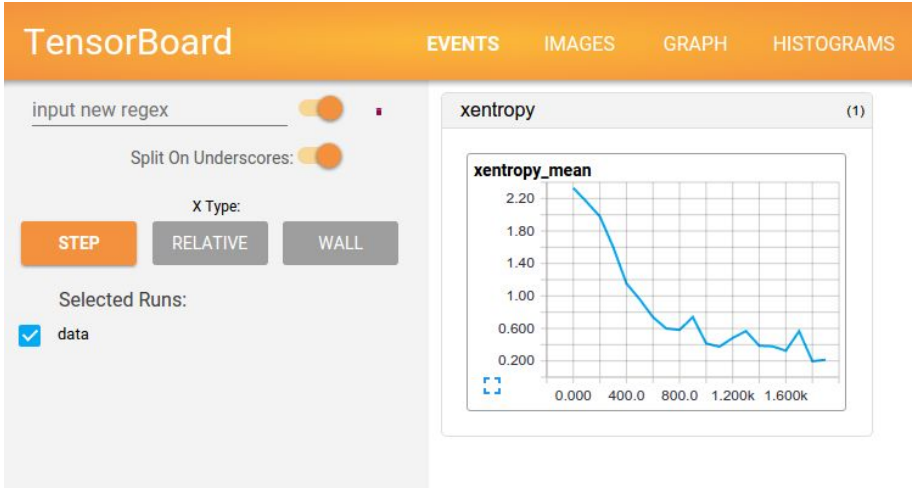
Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile
  - TensorRT support
- Best tools: TensorFlow
  - Tensorboard



Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile
  - TensorRT support
- Best tools: TensorFlow
  - Tensorboard
- Community: TensorFlow



Keras steps

- Define the network
- Compile the network
- Fit the network

```
def get_feedforward_nn():
  input1 = Input(shape=(21,))
  x = Dense(100, activation='relu')(input1)
  x = Dense(100, activation='relu')(x)
  x = Dense(100, activation='relu')(x)
  out = Dense(1, activation='sigmoid')(x)

  model = Model(inputs=input1, outputs=out)
  model.compile(optimizer=Adam(),
                loss='binary_crossentropy',
                metrics=['acc'])
  return model
```

```
  model_feedforward_nn.fit(x_train_char, y_train, epochs=epochs, batch_size=batch_size, verbose=ve
rbose,
                           callbacks=callbacks_list_feedforward_nn,
                           validation_data=(x_val_char, y_val))
```

Keras is easy!

Dense [source]

```
keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializ
```

Just your regular densely-connected NN layer.

`Dense` implements the operation: `output = activation(dot(input, kernel) + bias)` where `activation` is the element-wise activation function passed as the `activation` argument, `kernel` is a weights matrix created by the layer, and `bias` is a bias vector created by the layer (only applicable if `use_bias` is `True`).

- **Note**: if the input to the layer has a rank greater than 2, then it is flattened prior to the initial dot product with `kernel`.

Example

```
# as first layer in a sequential model:
model = Sequential()
model.add(Dense(32, input_shape=(16,)))
# now the model will take as input arrays of shape (*, 16)
# and output arrays of shape (*, 32)

# after the first layer, you don't need to specify
# the size of the input anymore:
model.add(Dense(32))
```

Dropout [source]

```
keras.layers.Dropout(rate, noise_shape=None, seed=None)
```

Applies Dropout to the input.

Dropout consists in randomly setting a fraction `rate` of input units to 0 at each update during training time, which helps prevent overfitting.

Arguments

- **rate**: float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape `(batch_size, timesteps, features)` and you want the dropout mask to be the same for all timesteps, you can use `noise_shape=(batch_size, 1, features)`.
- **seed**: A Python integer to use as random seed.

Conv1D

[source]

Number of filters

```
keras.layers.Conv1D(filters, kernel_size, strides=1, padding='valid', dilation_rate=1, activation=None, use_b
```

Size of filter

1D convolution layer (e.g. temporal convolution).

This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide an `input_shape` argument (tuple of integers or `None`, e.g. `(10, 128)` for sequences of 10 vectors of 128-dimensional vectors, or `(None, 128)` for variable-length sequences of 128-dimensional vectors.

Arguments

- **filters**: Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).
- **kernel_size**: An integer or tuple/list of a single integer, specifying the length of the 1D convolution window.
- **strides**: An integer or tuple/list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: One of `"valid"`, `"causal"` or `"same"` (case-insensitive). `"valid"` means "no padding". `"same"` results in padding the input such that the output has the same length as the original input. `"causal"` results in causal (dilated) convolutions, e.g. output[t] does not depend on input[t+1:]. Useful when modeling temporal data where the model should not violate the temporal order. See WaveNet: A Generative Model for Raw Audio, section 2.1.