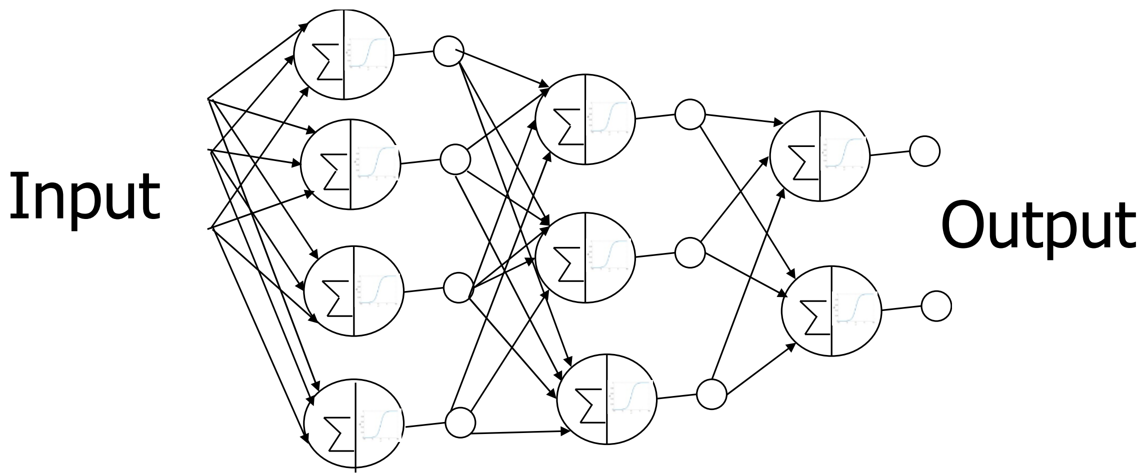
Deep Visual Learning

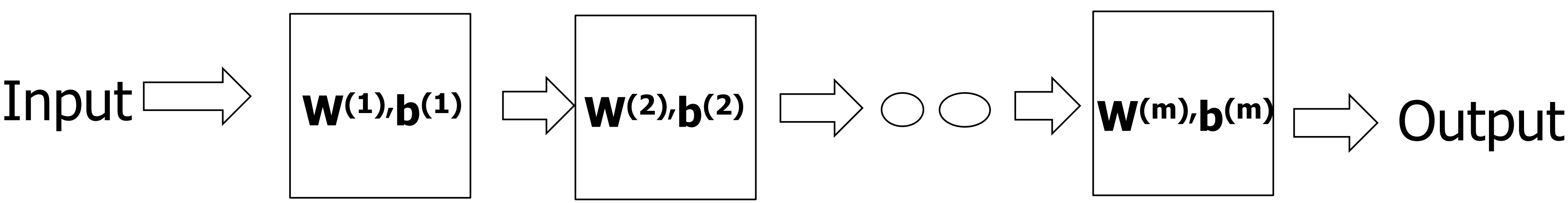
Jayanta Mukhopadhyay
Dept. of Computer Science and Engg.

Deep learning

- Learning using a “deep” neural network



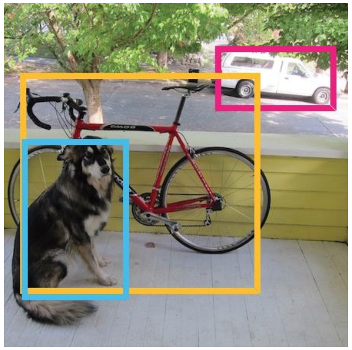
Classical ANN:
Only a few
hidden layers.

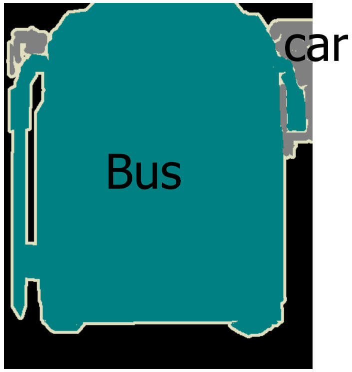


Deep architecture: Many hidden layers.

Deep learning for solving vision problems

- Object recognition
- Object Localization
- Semantic Segmentation
- Video summarization
- Tracking objects
- …





Deep architecture: Why so late in application?

- Concepts introduced in 80’s.
- Basic principles remain the same.
- Two major reasons.
  - Availability of large scale annotated data.
    - Penetration of internet and smart phones.
    - Wide spread of social networking.
    - Online shopping, etc.
  - Advancement of computing power.
    - High throughput GPU computing.

Classical Image Classification

hand-crafted feature extractor → Classifier Algorithm → output

Tiger?

Cat?

Lion?

- Edges
- SIFT/SURF key Point
- HOG Regional Features
- Motion Features, etc.

- Bayesian
- LDA
- SVM
- KNN

Classification Challenges

- Very tedious and costly to develop hand-crafted features to handle various challenges.

View Point variation

Deformation

Occlusion

Intraclass Variation

Illumination

Clutter

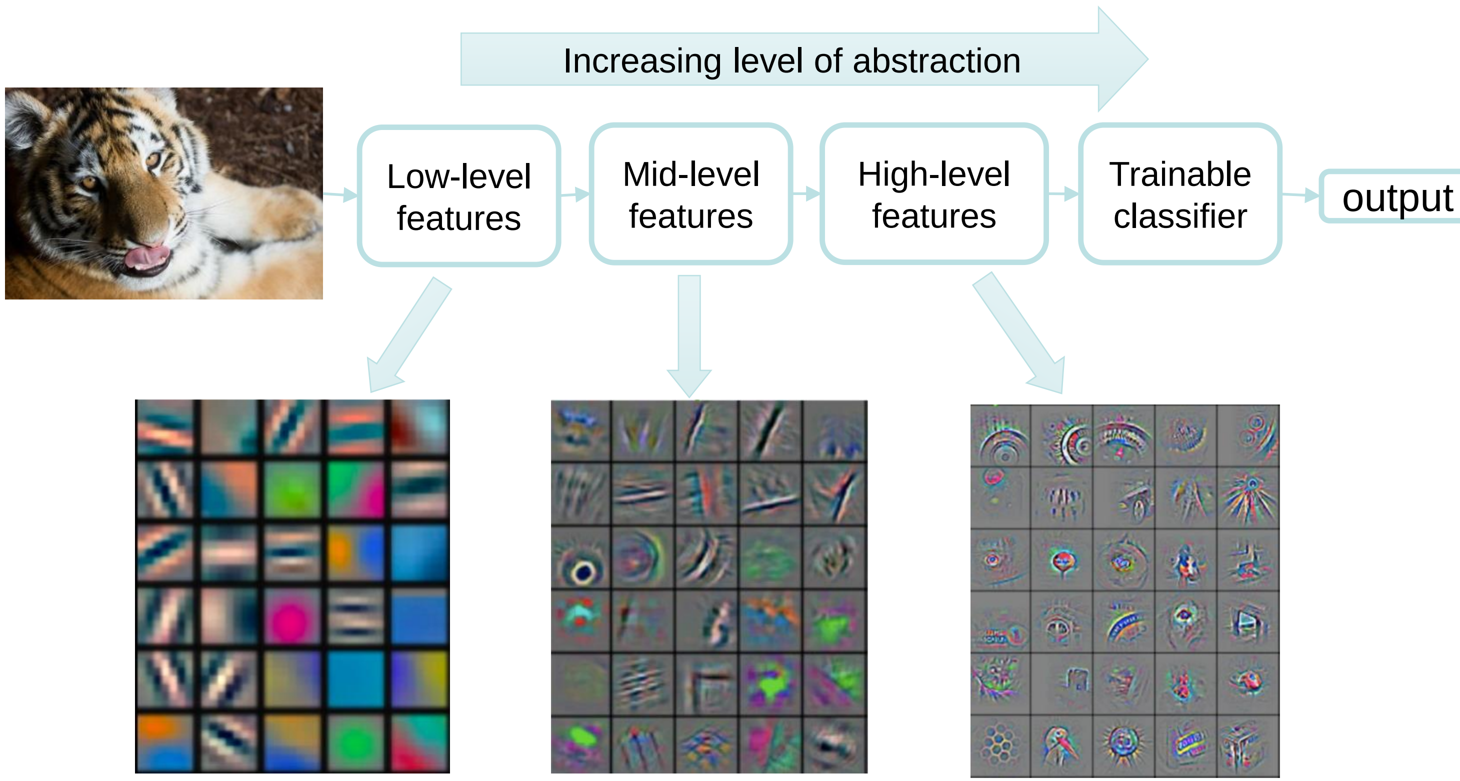
Instances

Scale

Highly dependent on one application, and not transferable easily to other applications.

Classification through deep learning

Learning filters of feature extraction and also classifier.



*Feature visualization of convolutional net trained on ImageNet (Zeiler and Fergus, 2013)

Supervised Learning

Data: (x, y) where x is data, y is label

Goal: Learn a function f to map $x \rightarrow y$

Examples: Classification, Regression,
Object detection, Semantic
Segmentation, Image Captioning, etc.

Supervised Learning

Data Driven Approach to learn the model in three steps:

Step 1: Define Model

$$\hat{y} = f(x, w)$$

Predicted output (image label)

Model structure

Input data (Image pixels)

Model weights

Learn a parametric function f composed by weight parameters w to classify Image x as class label y.

Supervised Learning

Step 2: Collect data.

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Training input

True output

Supervised Learning

Step 3: Learn the model.

Total Loss = Data Loss + Regularization Loss

Predicted output

$$w^* = \arg\min_w \overbrace{\frac{1}{N}\sum_{i=1}^{N} \ell(\overbrace{f(x_i, w)}, y_i) + R(w)}$$

Learned weights

Minimize average loss over training set

Loss function: Measures “badness” of prediction

Regularizer: Penalizes complex models

Loss

- A **loss function** tells how good our current classifier is.
- **Data loss**: Model predictions should match training data
  - **Softmax Loss** (Multinomial Logistic Regression):

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

$$L_i = -\log(P(Y = y_i | X = x_i))$$

Cross-entropy Loss

- Another form of softmax loss.
  - **2-class entropy:**
    - $-(y\log(p)+(1-y)\log(1-p))$; p: *Prob.* $(y=1|o)$
  - **Multiclass:**

$$-\sum_{c=1}^{M} y_{o,c}\, log(p_{o,c})$$

Estimated Prob. of o belonging to c

Binary indicator (1 if o belongs to c, else 0).

True Prob. of o belonging to c

More general:

$$-\sum_{c=1}^{M} q_{o,c}\, log(p_{o,c})$$

Regularization Loss

i

- **Regularization Loss**: Model should be "simple", so it works on test data as "W" is not unique with just data loss.
  - L_2 Regularization (Weight Decay) $R(W) = \sum_k \sum_l W_{k,l}^2$
  - L_1 Regularization $R(W) = \sum_k \sum_l |W_{k,l}|$
  - Elastic net (L_1 + L_2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

How to find best weights w^* ?

$$w^* = \arg\min_w \overbrace{\frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i, w), y_i) + R(w)}^{g(w)}$$

$$= \arg\min_w g(w)$$

- Gradient descent
  - Back propagation algorithm

Gradient Descent

How to update weights?

Initialize w randomly
While true:
  Compute gradient $\nabla g(w)$ at current point

  Move downhill a little bit: $w = w - \alpha \nabla g(w)$

updating the weights at each iteration

Learning rate: How big each step should be

Back Propagation

- **Forward pass**:
  - Run graph "forward" to compute loss
- **Backward pass**:
  - Run graph "backward" to compute gradients with respect to loss
- Efficient to compute gradients for big, complex models.

Learning filters for feature extraction

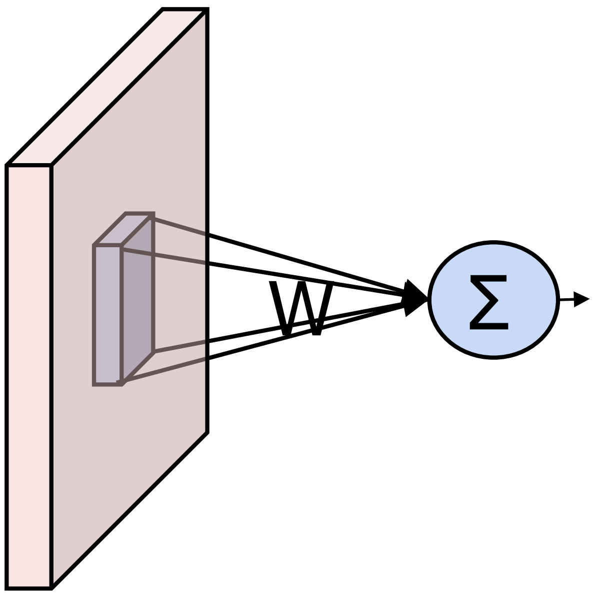
- Correlation with a mask or kernel

w_1	w_2	w_3
w_4	w_c	w_5
w_6	w_7	w_8

$$g(x,y) = w_1 f(x-1,y+1) + w_2 f(x,y+1) + w_3 f(x+1,y+1) + w_4 f(x-1,y) +$$
$$w_c f(x,y) + w_5 f(x+1,y) + w_6 f(x-1,y-1) + w_7 f(x,y+1) + w_8 f(x+1,y+1)$$

Convolution in neural architecture

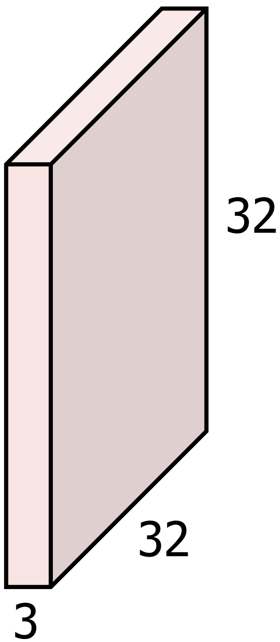
- Output of a neuron: weighted sum of inputs
  - weights defined by a kernel
  - Sparse connectivity
- Shared weights for every node
  - Sufficient to describe the model by a kernel



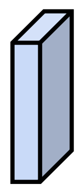
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter (kernel)

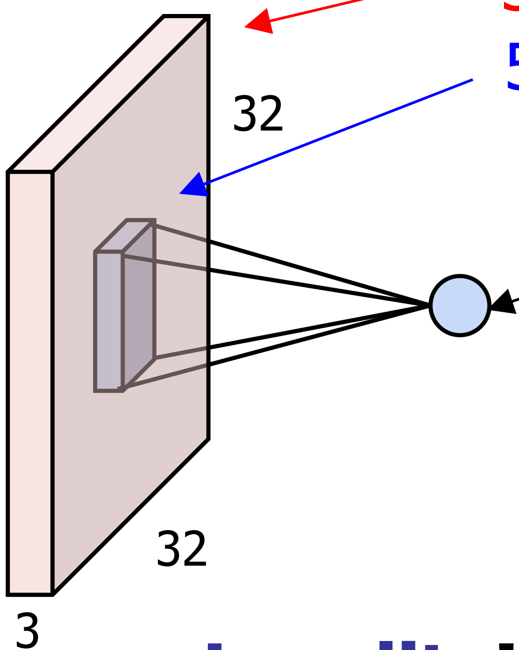


Convolve the filter with the image i.e. "slide over the image spatially, computing dot products".

Convolution Layer

32x32x3 image *x*

5x5x3 filter w



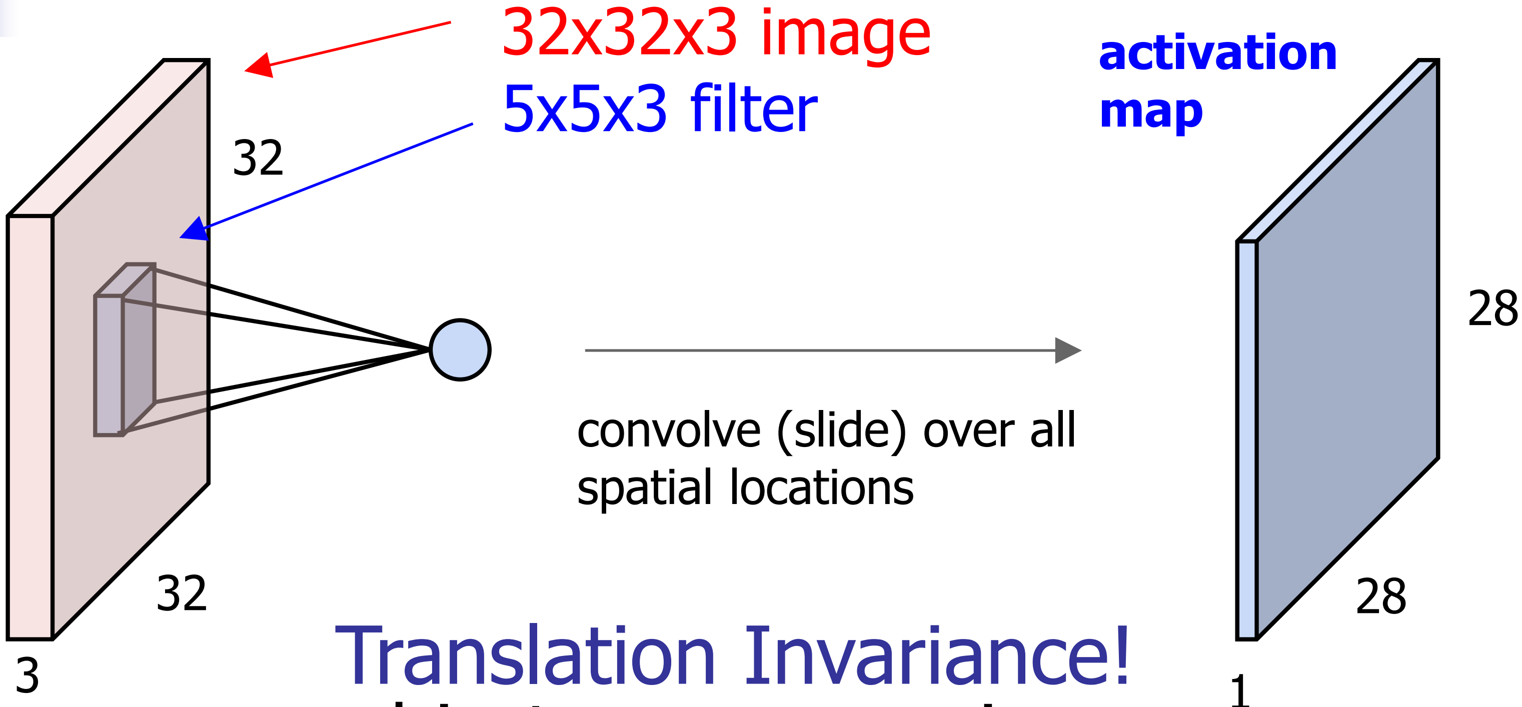
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Locality!

Objects tend to have a local spatial support.

Convolution Layer

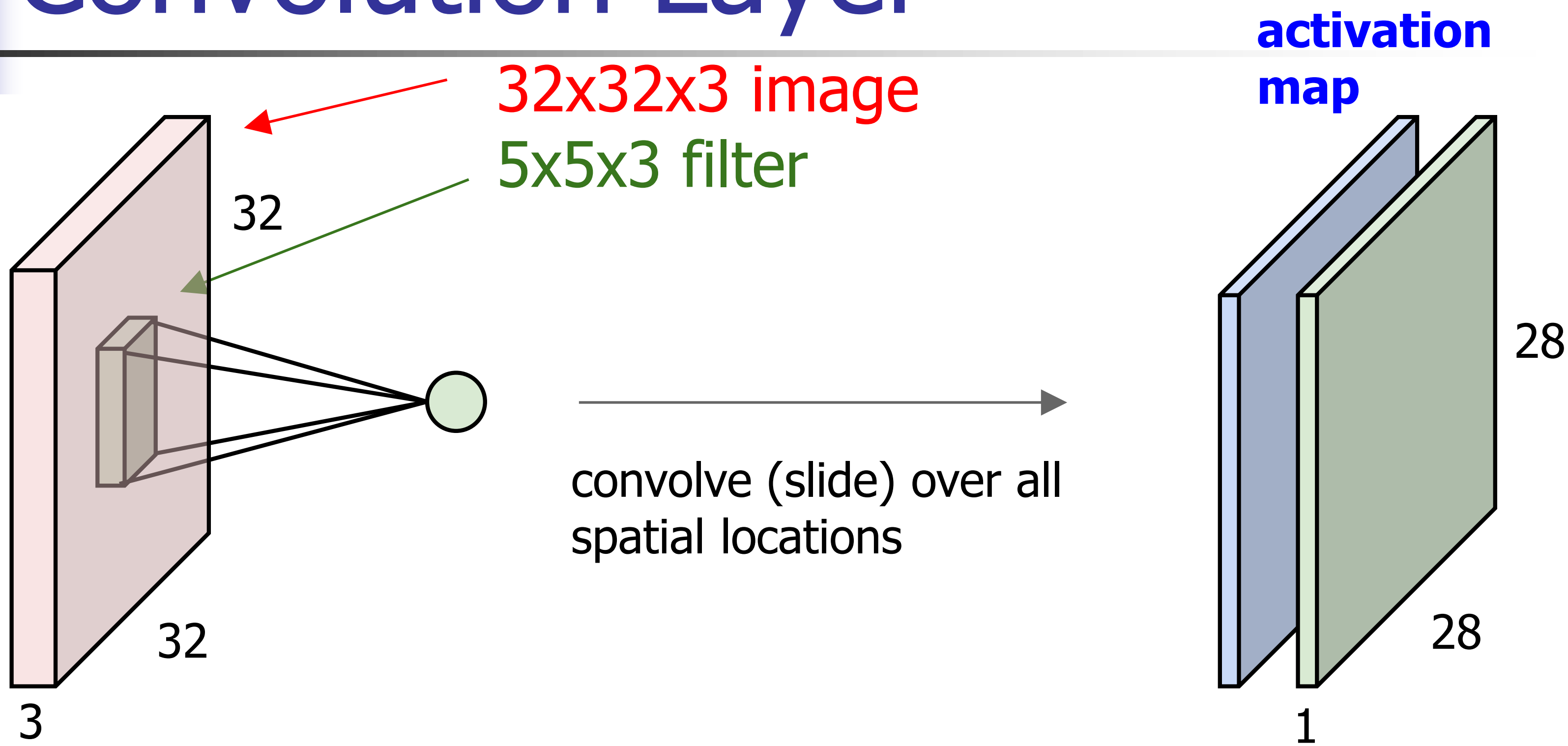


Translation Invariance!

object appearance is independent of location

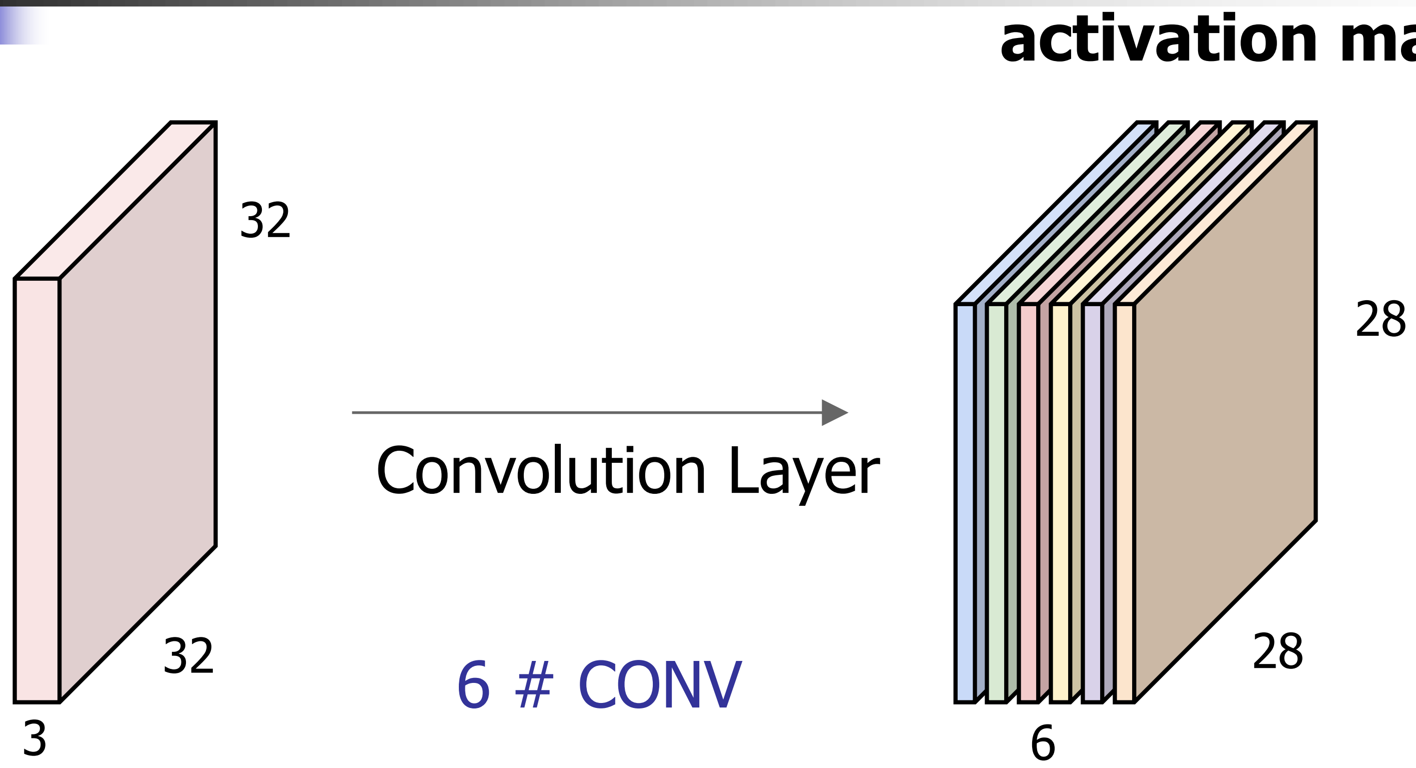
Weight sharing!

Convolution Layer



Consider a second, green filter.

Convolution Layer (CONV)



For example, if we had 6 5x5x3 filters, we'll get 6 separate activation maps:

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

Features of CONV

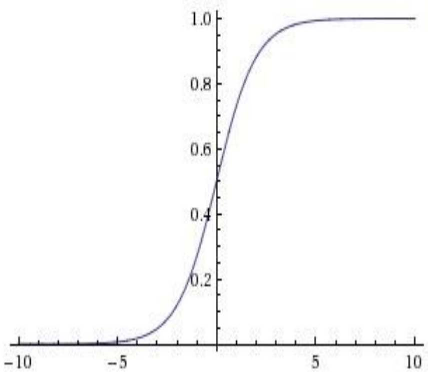
- **Locality**:
  - objects tend to have a local spatial support
- **Translation invariance**:
  - object appearance is independent of location
- **Weight sharing**
  - units connected to different locations have the same weights
  - equivalently, each unit is applied to all locations
  - weights of filters are invariant.
- Each unit output of filter is connected to a local rectangular area in the input.
  - – Receptive Field

Non-Linear Layer

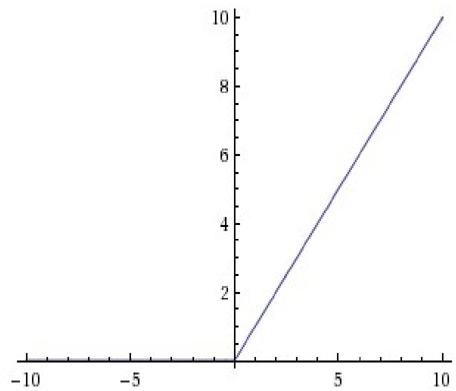
- Increase the nonlinearity of the entire architecture without affecting the receptive fields of the convolution layer.
  - Commonly used in CNN is ReLU.

Non Linearity: Activation Functions - A few examples

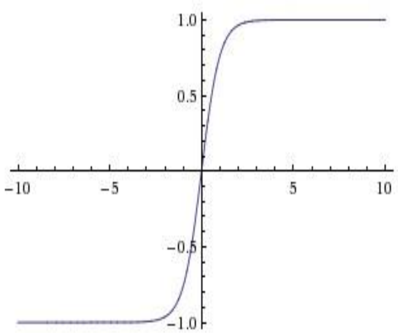
Sigmoid $\sigma(x) = 1/(1+e^{-x})$



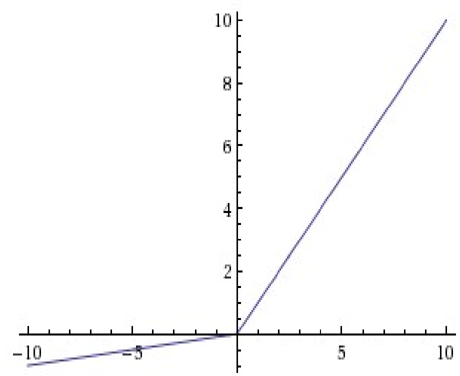
ReLU (Rectified Linear Unit)



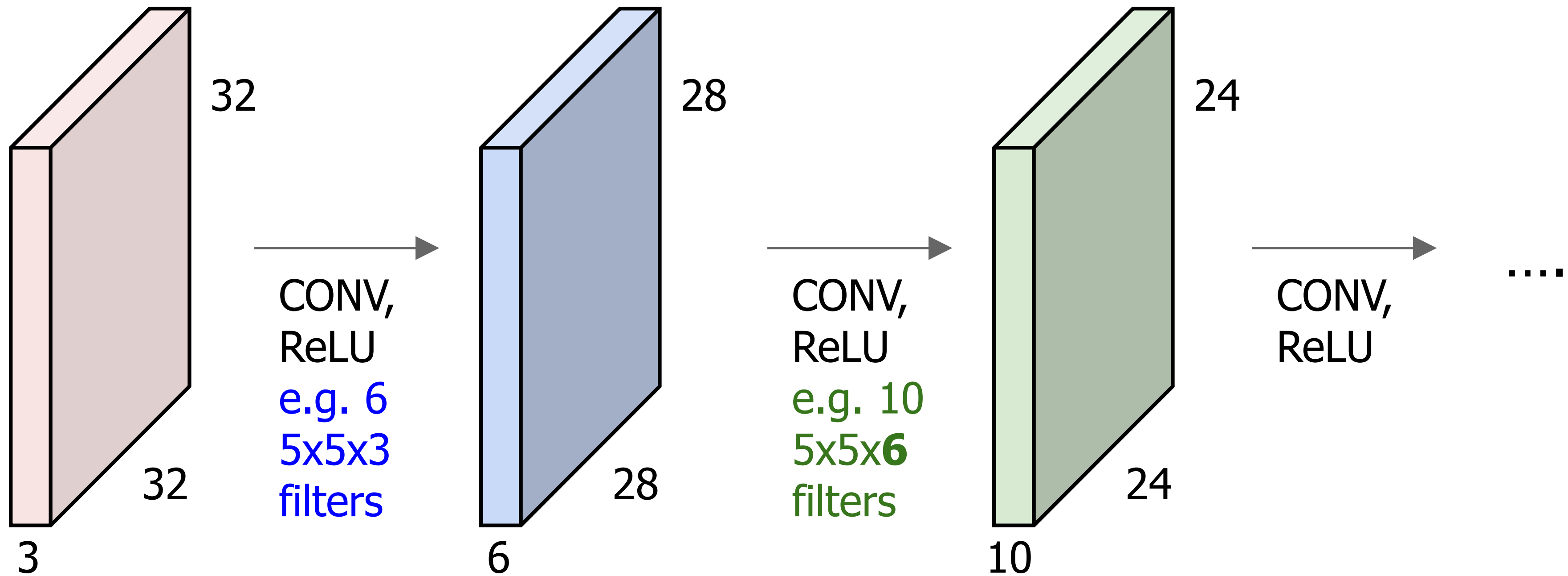
tanh(x)



Leaky ReLU



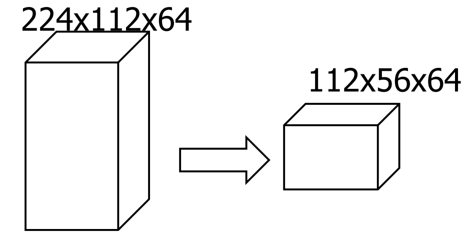
Convolutional Neural Networks (CNN)



A CNN is a sequence of convolution layers and nonlinearities.

Parameters involved in convolution layer

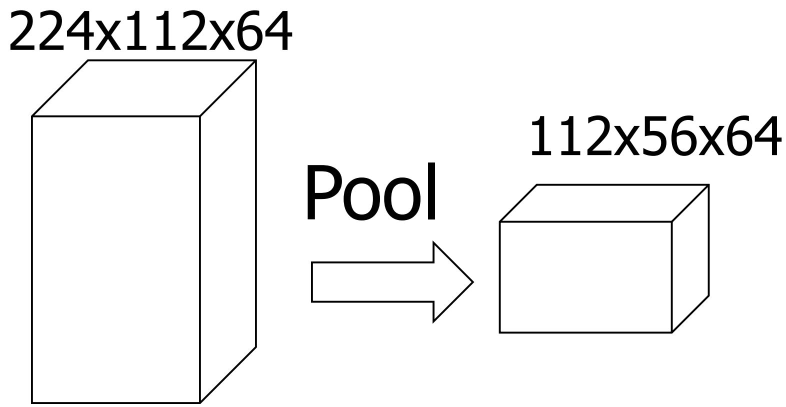
- Input Volume size W_1 x H_1 x D_1
- No. of filters K with size F_w x F_h x D_1 convolved with stride (S_w, S_h).
- Input zero padded by (P_w, P_h) on both sides.
- Output volume size W_2 x H_2 x D_2?
  - $W_2 = (W_1 - F_w + 2P_w)/S_w + 1$
  - $H_2 = (H_1 - F_h + 2P_h)/S_h + 1$
  - $D_2 = K$
- Parameters ?
  - $(F_w * F_h * D_1) * K$ weights + K biases
- **d-th depth slice of output is the result of convolution of d-th filter over the padded input volume with a stride, then offset by d-th bias**



Pooling Layer (POOL)

- To progressively reduce the spatial size of the representation.
  - to reduce the amount of parameters and computation in the network.
  - to control overfitting.
- Pooling partitions the input image into a set of non-overlapping rectangles.
- For each such sub-region, outputs an aggregated value of the features in that region.
  - Maximum value (Max pooling)
  - Average value (Average pooling)
- Operates over each activation map independently

Pooling Layer (POOL)



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

Maxpool with 2x2 filter and stride 2

6	8
3	4

Single depth slice

Parameters involved in pooling

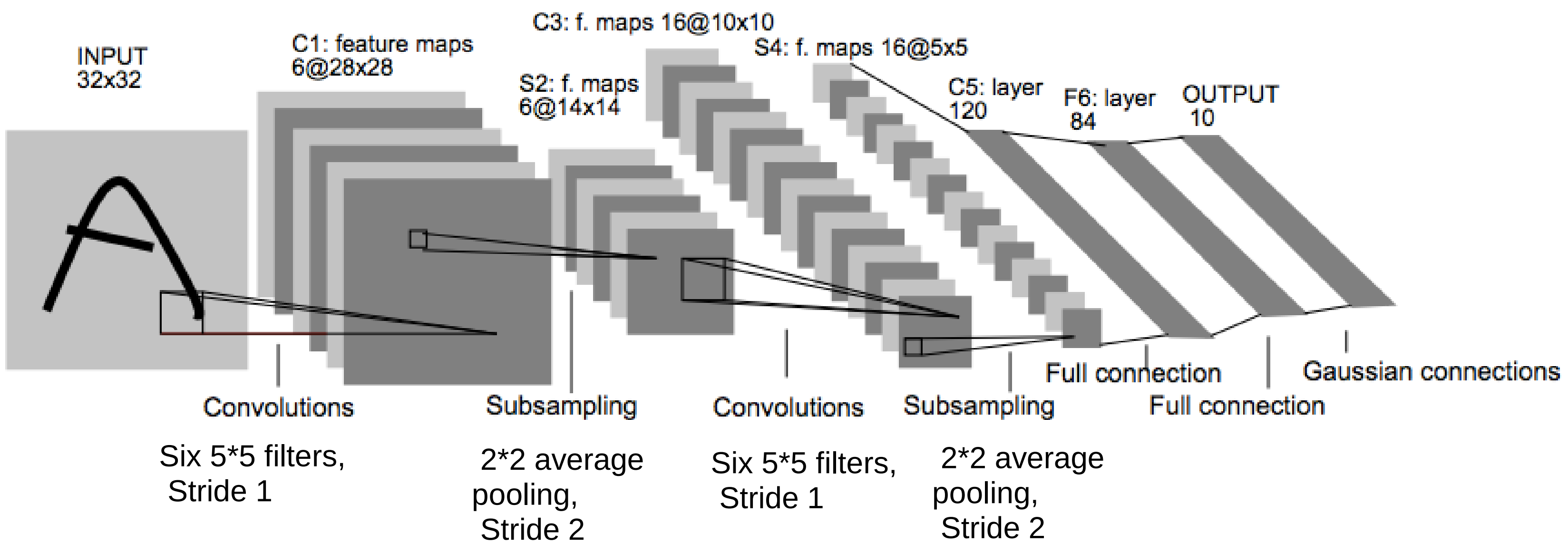
- Input Volume size W_1 x H_1 x D_1
- Pool size F_w x F_h with stride (S_w, S_h).
- Output volume size W_2 x H_2 x D_2?
  - $W_2 = (W_1 - F_w)/S + 1$
  - $H_2 = (H_1 - F_h)/S + 1$
  - $D_2 = D_1$
- Parameters ?
  - 0!
- **Uncommon to use zero-padding in Pooling layers.**

Fully Connected Layer (FC)

- Contains neurons that connect to the entire input volume
  - as in ordinary Neural Networks.
- Input volume to FC layer can also be treated as Deep Features.
- If the FC layer is a classifier, the input to FC can also be treated as feature vector representation for the sample.

LeNet: A typical example

- I/P→CONV→POOL→CONV→POOL→FC→FC→O/P
  - Number of parameters: 60k
  - Number of floating point operations per inference: 341k
  - Sigmoid used for non-linearity.



*Y. Lecun et al, Proceedings of the IEEE, 1998

Efficient computation with smaller kernels

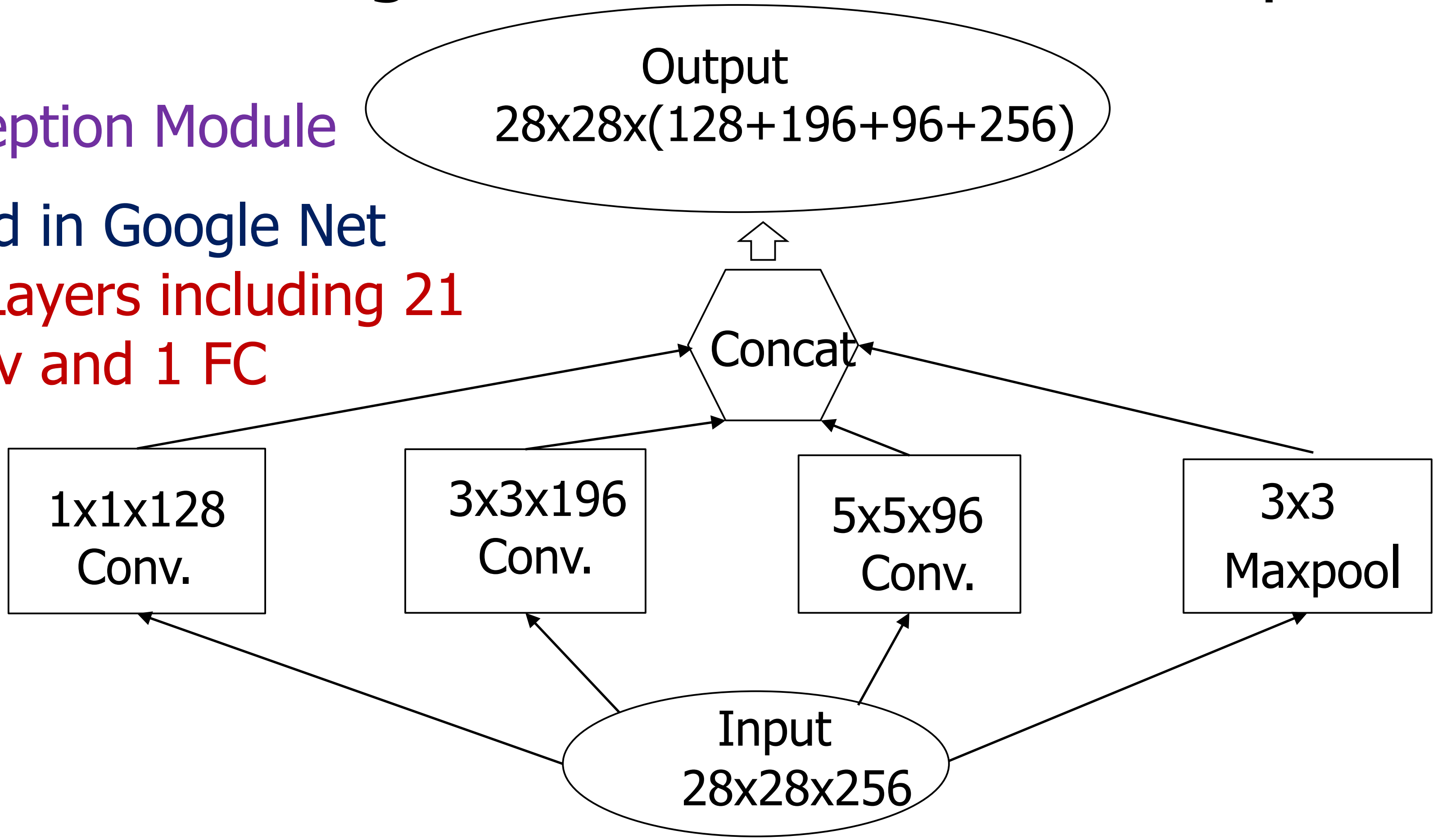
- Successive filtering with smaller sized kernels covers equivalent receptive field area of a larger size.
  - Stack of three 3x3 conv (stride 1) layers has **same effective field** as one 7x7 conv layer
  - Deeper with more non-linearity
    - Usually deeper the model better the accuracy
    - Till it overfits!
  - Fewer parameters:
    - $3*(3^2\ C^2)$ vs. $(7^2\ C^2)$ for C channels
- Used in VGG network
  - no. of layers: 13 / 16 / 19 with only 3x3 kernels

Handling scale in feature representation

- Concatenating multiscale feature descriptor

Inception Module

Used in Google Net
57 Layers including 21 Conv and 1 FC

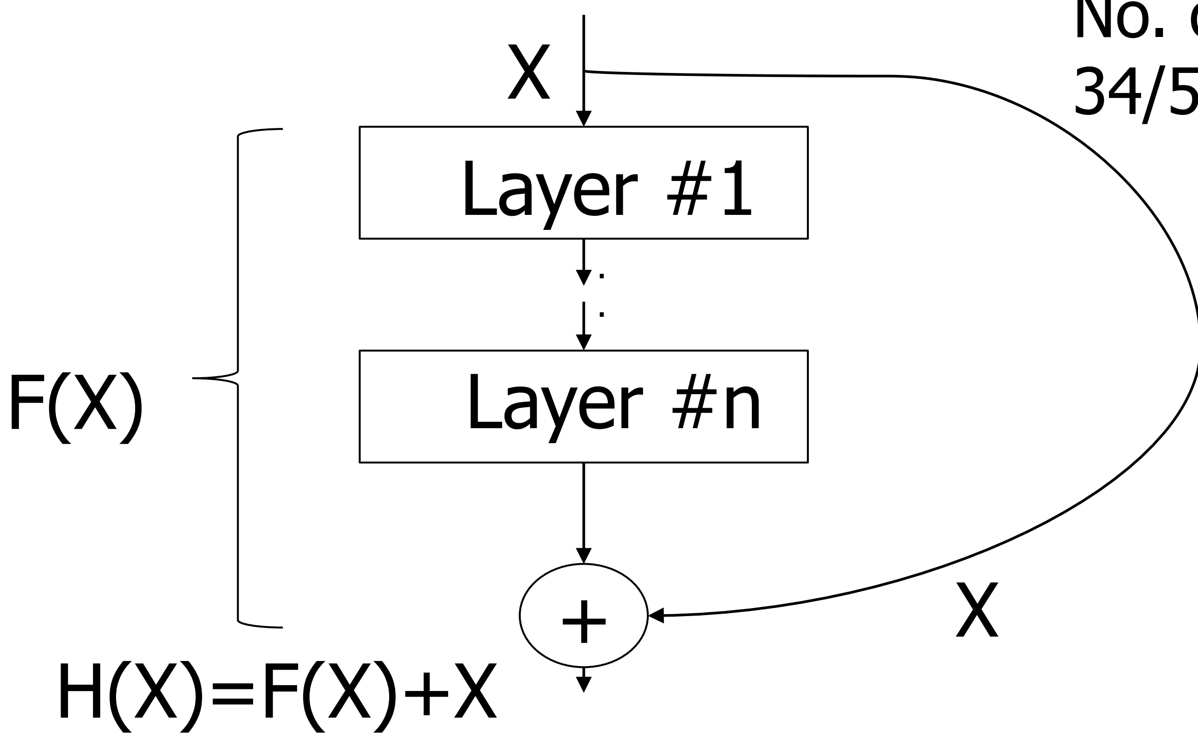


Vanishing gradient problem

- Gradient becomes zero (vanishes) at deeper layers!
- Learn residual mapping!

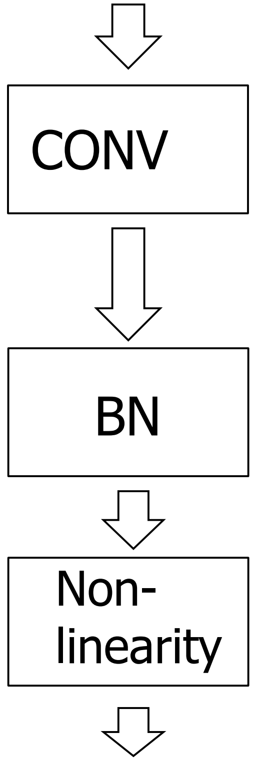
Used in ResNet

No. of layers: 34/50/101/152



Batch Normalization

- Normalizes input activation map to a layer by considering its distribution over a batch of training samples.
  - Each dimension of the input feature map individually normalized
  - To make Gaussian activation maps.
- Advantages
  - Improves gradient flow through the network.
  - Allows higher learning rates.
  - Reduces the strong dependence on initialization.
  - Acts as a form of regularization.
- Usually inserted after FC / CONV layers, and before non-linearity.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Drop out

- Randomly dropping out nodes of network (at hidden / visible layers) during training.
  - Temporarily removing it from the network, along with all its incoming and outgoing connections.
  - To regulate overfitting, more effective for smaller dataset.
  - Simulates learning sparse representation in hidden layers.
- Implementation
  - Retain output of a node with a probability p.
    - Typically within [0.5,1] at hidden layers and [0.8,1] in visible layers.

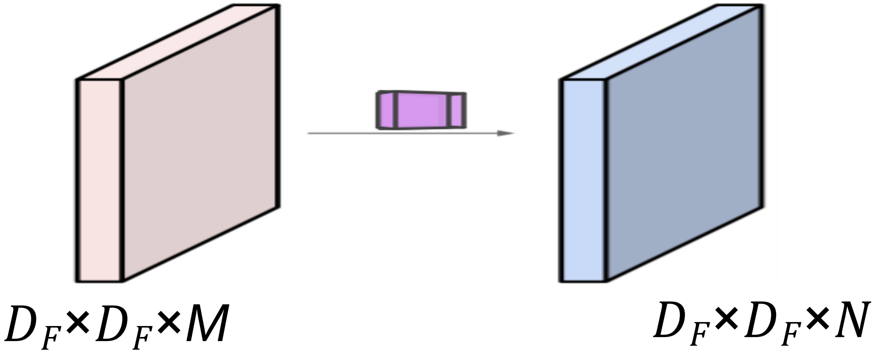
Learning weights with drop out

- Weights become larger due to drop out.
  - Needs to be scaled at the end training.
  - A simple heuristic.
    - Outgoing weights of a unit retained with probability p during training, multiplied by p at test time.
  - Scaling may be carried out during training time at each weight update.
    - No need to rescale weight for the test network.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting ,JMLR, 15(Jun):1929−1958, 2014.

Depthwise Separable Convolutions

- Suppose, we have $D_F \times D_F \times M$ input feature map, $D_F \times D_F \times N$ output feature map and $D_k \times D_k$ spatial sized conventional convolution filters.

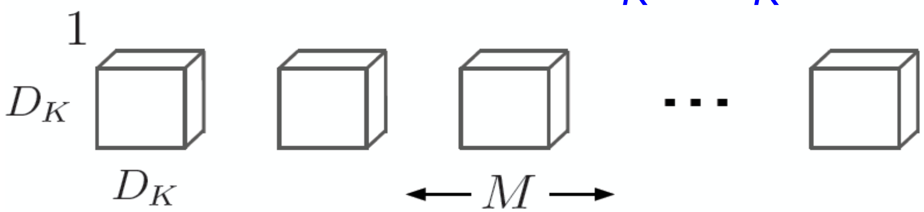


- What is the computational cost for such a convolution operation?
  - $D_k \cdot D_k \cdot M \cdot D_F \cdot D_F \cdot N$
- What is the number of parameters?
  - $D_k \cdot D_k \cdot M \cdot N$

Courtesy: Ankita Chatterjee

Depthwise Separable Convolutions

- Now, think of M filters which are $D_K \times D_K$ (not $D_K \times D_K \times M$) and think each M of these filters are operated separately on M channels of input of spatial size $D_F \times D_F$
- Number of parameters? $D_K \cdot D_K \cdot M$



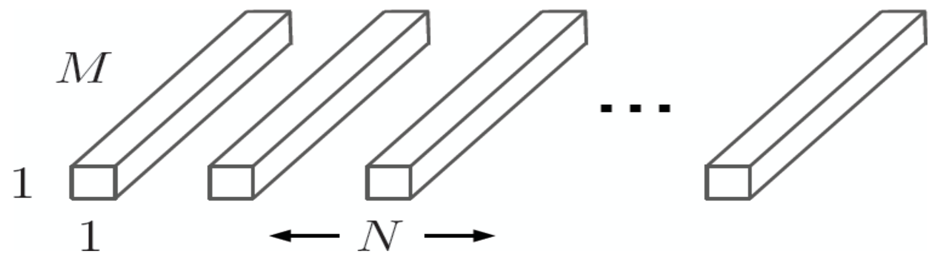
- What is the computational cost for such a convolution operation? $D_K \cdot D_K \cdot D_F \cdot D_F \cdot M$

- This operation is known as Depthwise Convolution operation.

Courtesy: Ankita Chatterjee

Depthwise Separable Convolutions

- What is the output shape now? $D_F \times D_F \times M$
- Where did the N (output channels) go?
  - not there as depthwise convolution operates only on input channels.
- Now think about 1×1 traditional convolution on $D_F \times D_F \times M$ featuremap to get $D_F \times D_F \times N$ output.
- What is the computation cost? $1 \cdot 1 \cdot M \cdot D_F \cdot D_F \cdot N = D_F \cdot D_F \cdot M \cdot N$



Used in MobileNet-V1

Alternate layers of Conv and D-S Conv

What is the number of parameters? $1 \cdot 1 \cdot M \cdot N$

- This operation is called 1×1 pointwise convolution

Courtesy: Ankita Chatterjee

MobileNet-V1

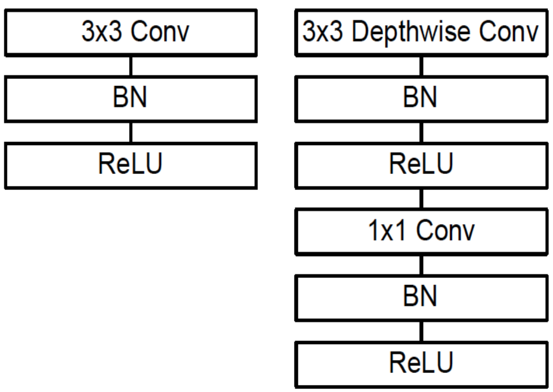


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Image taken from: *MobileNet Paper*

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Image taken from: *MobileNet Paper*

Courtesy: Ankita Chatterjee

Width and Resolution Multiplier

- Used as parameters for scaling the model architectures.
- Width multiplier $\alpha \in (0, 1]$ to thin a network uniformly at each layer
  - The number of input channels (from M): αM
  - The number of output channels (from N): αN
  - Computational cost? $D_k \cdot D_k \cdot \alpha M \cdot D_F \cdot D_F + D_F \cdot D_F \cdot \alpha M \cdot \alpha N$
  - Reduces roughly by α^2
- Resolution multiplier $\rho \in (0, 1]$ to reduce the image resolution and the internal representation of every layer by by this factor
- With width multiplier α and resolution multiplier ρ, the computational cost? $D_k \cdot D_k \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \rho D_F \cdot \rho D_F \cdot \alpha M \cdot \alpha N$
  - Another reduction by ρ^2

Courtesy: Ankita Chatterjee

MobileNet-V1

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Image taken from: *MobileNet Paper*

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

Image taken from: *MobileNet Paper*

EfficientNets: Compound Model Scaling

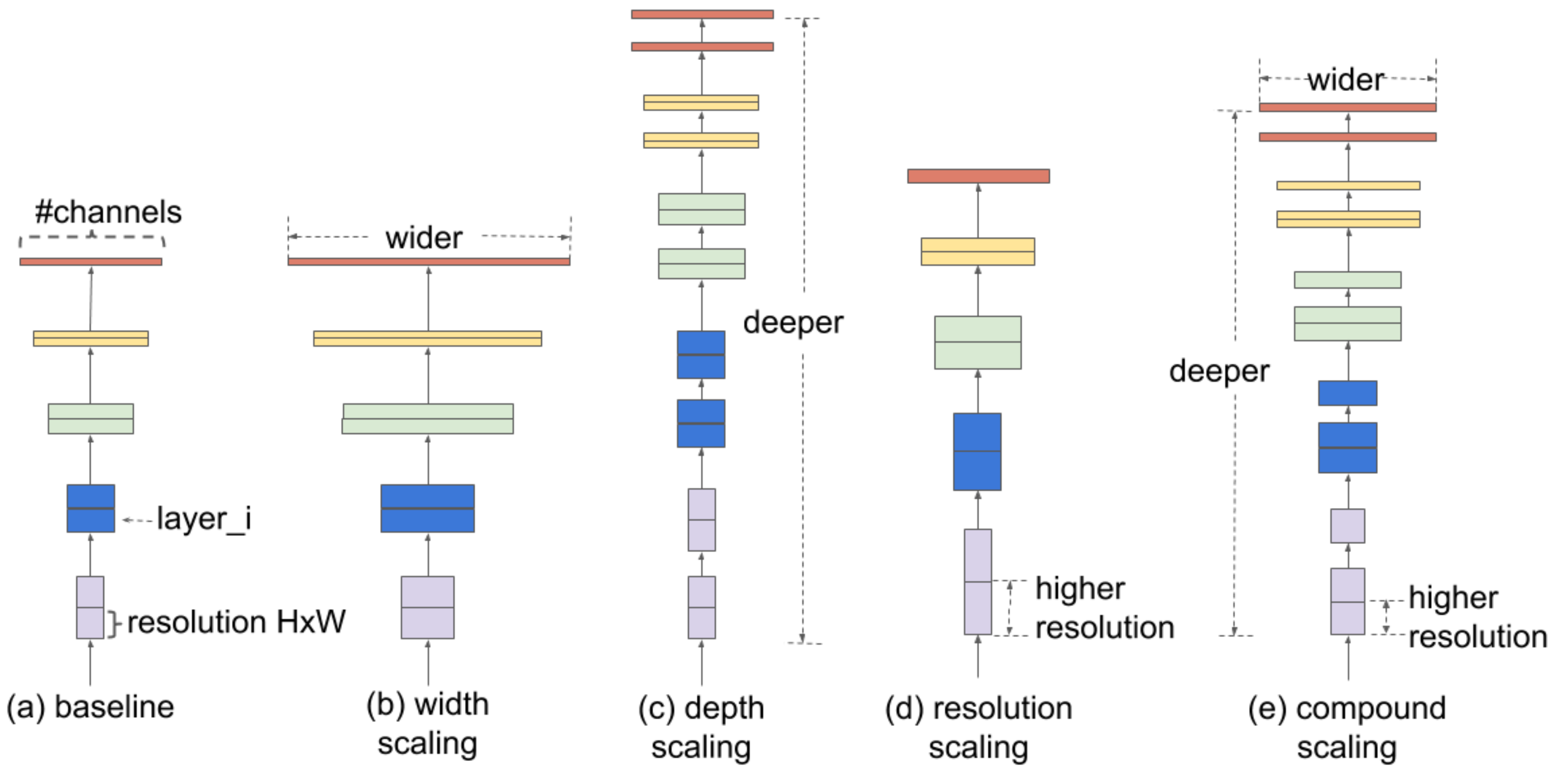
- By balancing all dimensions of the network—width, depth, and image resolution—against the available resources to get the best overall performance.

- To perform a grid search to find the relationship between different scaling dimensions of the baseline network under a fixed resource constraint

- Apply those coefficients to scale up the baseline network to the desired target model size or computational budget

Courtesy: Ankita Chatterjee

Compound Model Scaling



Comparison of different scaling methods. Unlike conventional scaling methods (b)-(d) that arbitrary scale a single dimension of the network, compound scaling method uniformly scales up all dimensions in a principled way.
Image taken from: *EfficientNet* Paper

Courtesy: Ankita Chatterjee

EfficientNet Architecture

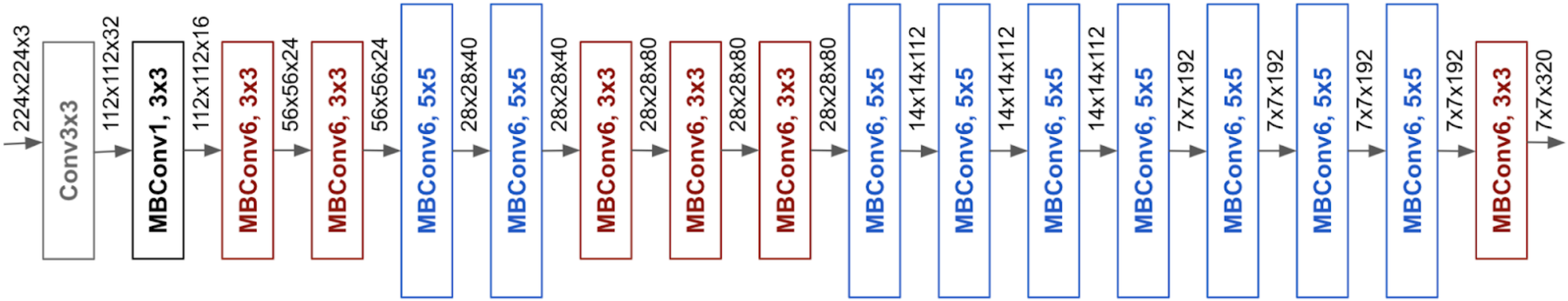
- The effectiveness of model scaling also relies heavily on the baseline network

- A new baseline network is developed by performing a neural architecture search using the AutoML MNAS framework, which optimizes both accuracy and efficiency (FLOPS).

- The resulting architecture uses mobile inverted bottleneck convolution (MBConv), similar to MobileNetV2 and MnasNet, but is slightly larger

Courtesy: Ankita Chatterjee

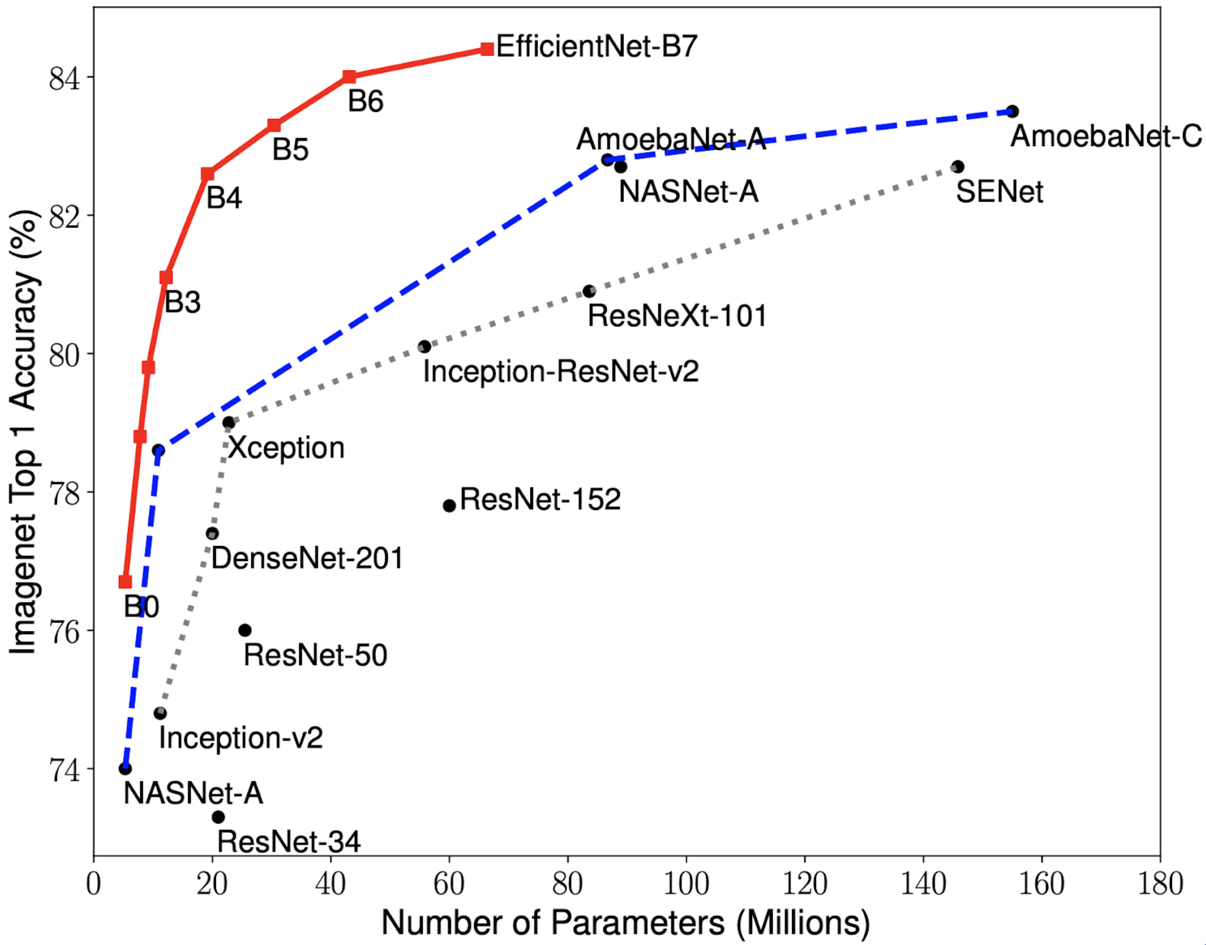
EfficientNet Architecture



The architecture for baseline network EfficientNet-B0 is simple and clean, making it easier to scale and generalize.
Image taken from: *EfficientNet* Paper

Courtesy: Ankita Chatterjee

EfficientNet Architecture



Model Size vs. Accuracy

EfficientNet-B0:
the baseline network developed by AutoML MNAS,

Efficient-B1 to B7:
obtained by scaling up the baseline network.

In particular, EfficientNet-B7 achieves new state-of-the-art 84.4% top-1 / 97.1% top-5 accuracy, while being 8.4x smaller than the best existing CNN.

Courtesy: Ankita Chatterjee

Image taken from: EfficientNet Paper

EfficientNet performance on other baseline architectures

Model	FLOPS	Top-1 Acc.
Baseline MobileNetV1 (Howard et al., 2017)	0.6B	70.6%
Scale MobileNetV1 by width (w=2)	2.2B	74.2%
Scale MobileNetV1 by resolution (r=2)	2.2B	72.7%
compound scale (d=1.4, w=1.2, r=1.3)	**2.3B**	**75.6%**
Baseline MobileNetV2 (Sandler et al., 2018)	0.3B	72.0%
Scale MobileNetV2 by depth (d=4)	1.2B	76.8%
Scale MobileNetV2 by width (w=2)	1.1B	76.4%
Scale MobileNetV2 by resolution (r=2)	1.2B	74.8%
MobileNetV2 compound scale	**1.3B**	**77.4%**
Baseline ResNet-50 (He et al., 2016)	4.1B	76.0%
Scale ResNet-50 by depth (d=4)	16.2B	78.1%
Scale ResNet-50 by width (w=2)	14.7B	77.7%
Scale ResNet-50 by resolution (r=2)	16.4B	77.5%
ResNet-50 compound scale	**16.7B**	**78.8%**

Scaling Up MobileNets and ResNet.

Image taken from: EfficientNet Paper

Courtesy: Ankita Chatterjee

EfficientNet performance vs. other models

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.1%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.1%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.6%**	**95.7%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.9%**	**96.4%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.6%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.3%**	**97.0%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

Image taken from: EfficientNet Paper

Courtesy: Ankita Chatterjee

EfficientNet performance on other datasets using transfer learning

	Comparison to best public-available results						Comparison to best reported results					
	Model	Acc.	#Param	Our Model	Acc.	#Param(ratio)	Model	Acc.	#Param	Our Model	Acc.	#Param(ratio)
CIFAR-10	NASNet-A	98.0%	85M	EfficientNet-B0	98.1%	4M (21x)	†Gpipe	**99.0%**	556M	EfficientNet-B7	98.9%	64M (8.7x)
CIFAR-100	NASNet-A	87.5%	85M	EfficientNet-B0	88.1%	4M (21x)	Gpipe	91.3%	556M	EfficientNet-B7	**91.7%**	64M (8.7x)
Birdsnap	Inception-v4	81.8%	41M	EfficientNet-B5	82.0%	28M (1.5x)	GPipe	83.6%	556M	EfficientNet-B7	**84.3%**	64M (8.7x)
Stanford Cars	Inception-v4	93.4%	41M	EfficientNet-B3	93.6%	10M (4.1x)	‡DAT	**94.8%**	-	EfficientNet-B7	94.7%	-
Flowers	Inception-v4	98.5%	41M	EfficientNet-B5	98.5%	28M (1.5x)	DAT	97.7%	-	EfficientNet-B7	**98.8%**	-
FGVC Aircraft	Inception-v4	90.9%	41M	EfficientNet-B3	90.7%	10M (4.1x)	DAT	92.9%	-	EfficientNet-B7	**92.9%**	-
Oxford-IIIT Pets	ResNet-152	94.5%	58M	EfficientNet-B4	94.8%	17M (5.6x)	GPipe	**95.9%**	556M	EfficientNet-B6	95.4%	41M (14x)
Food-101	Inception-v4	90.8%	41M	EfficientNet-B4	91.5%	17M (2.4x)	GPipe	93.0%	556M	EfficientNet-B7	**93.0%**	64M (8.7x)
Geo-Mean						**(4.7x)**						**(9.6x)**

Image taken from: EfficientNet Paper

Courtesy: Ankita Chatterjee

Training steps:

- Preprocessing of training dataset.
  - Normalize data.
  - Decorrelate data (Diagonal Covariance Matrix).
  - Whitening of data (Identity Covariance Matrix).
  - Subtract Mean.

Training steps:

- Data augmentation.
  - Horizontal Flips
  - Random Crops on scaled input
  - Color jitter
  - Distortions
  - Transformations
- Weight initialization
- Train the network by update of the weight parameters.

Few Training Tips

- Start with small regularization and find learning rate that makes the loss go down.
- Can overfit very small portion of the training data.
- Train first few epochs with few samples to initiate the hyper-parameters.
- If big gap between training accuracy and validation accuracy, then it is overfitting.
  - Try increase regularization.
- If no gap, then may increase model capacity.

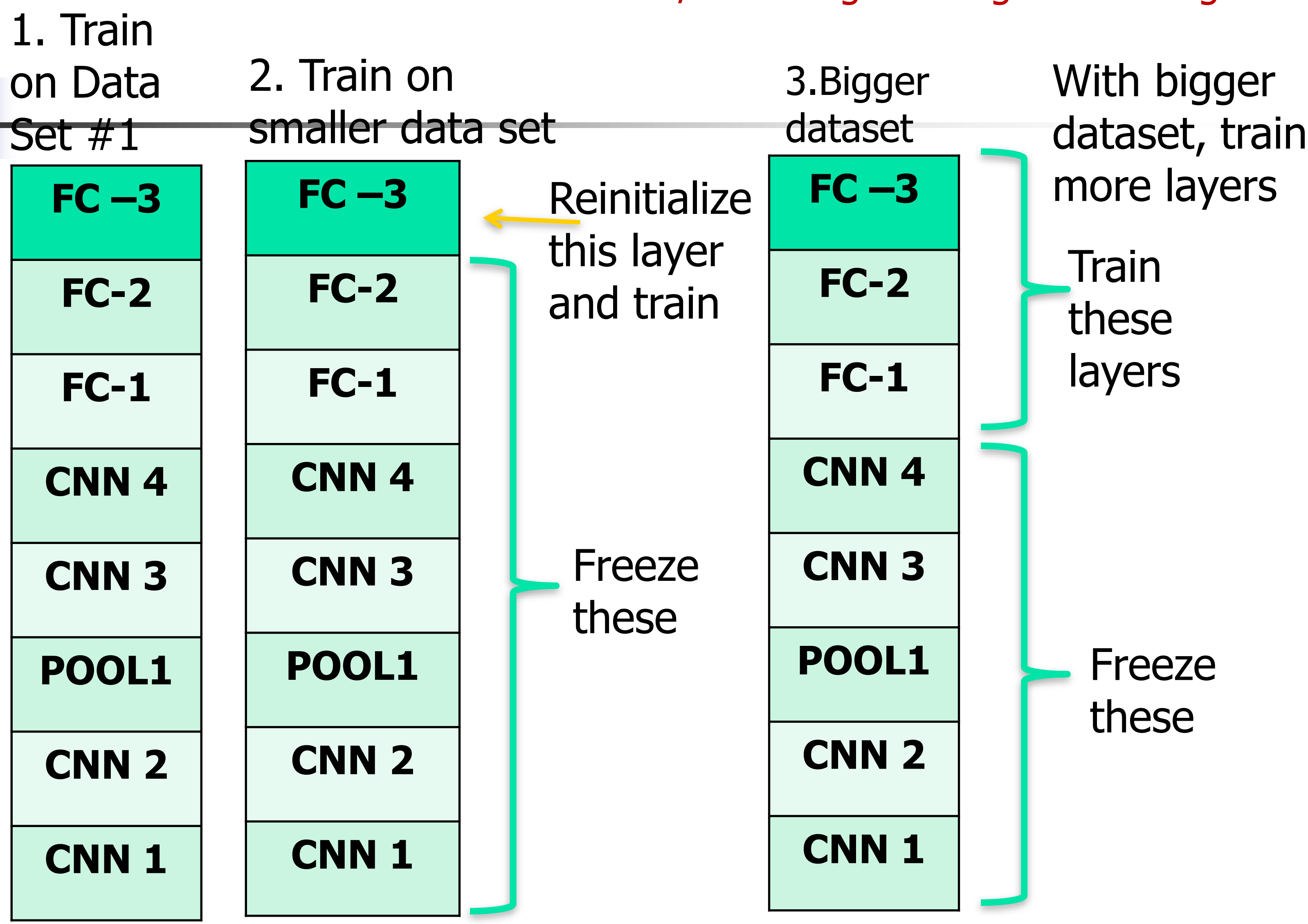
Transfer Learning

- No need of a lot of a data to train a CNN.
- Pre-trained models can be initialized for CNNs at the early stage of training.

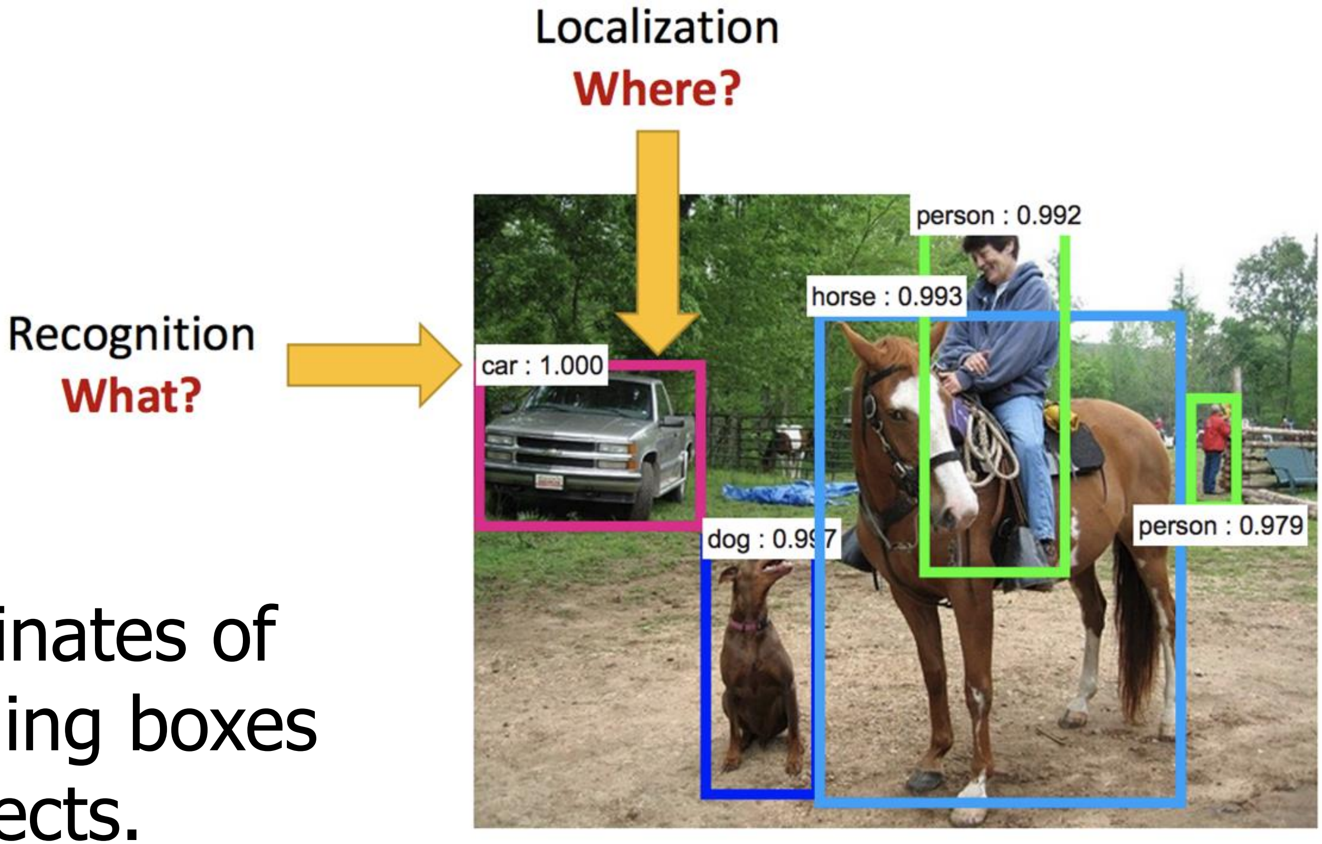
*Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Transfer Learning

Lower learning rate when fine-tuning; 1/10 of original LR good starting Point.



Object Recognition and Localization

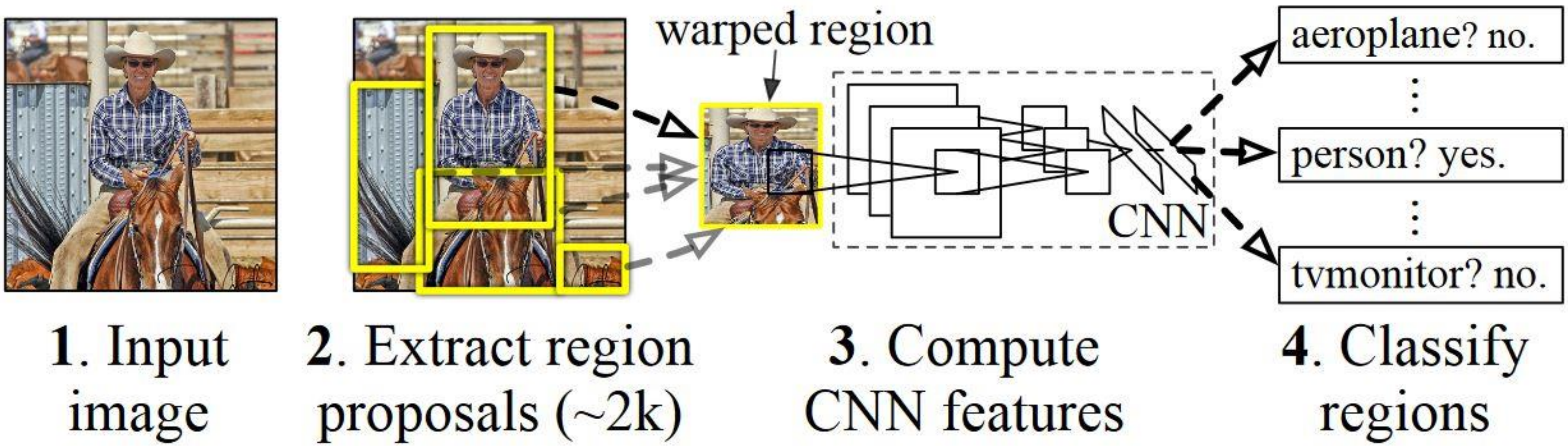


Predicts

(i) Coordinates of bounding boxes of objects.

(ii) Class probs.

Two stage processing: Localization and recognition



*Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

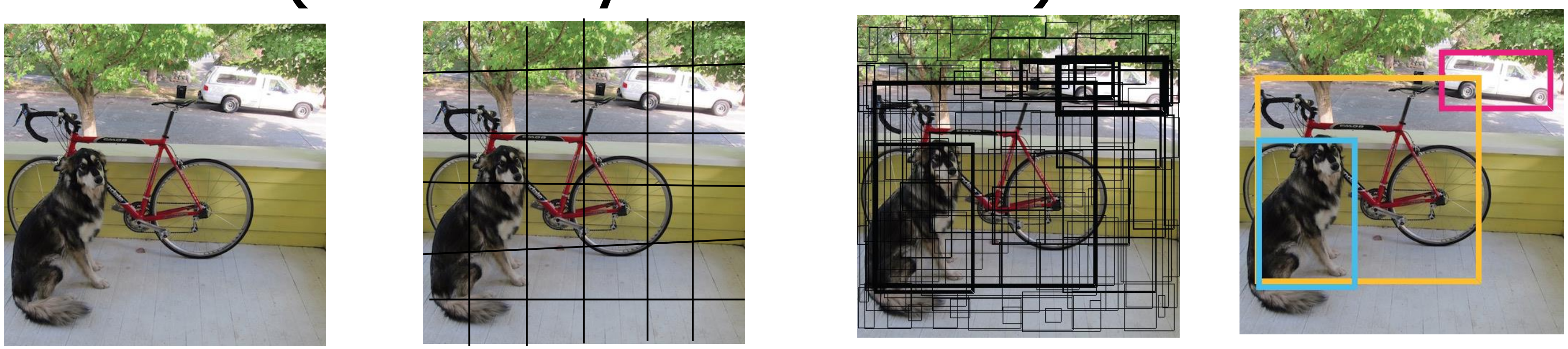
Two stage processing: Object Recognition and Localization

	Region Proposal	Feature Extraction	Classification
Pre-CNN	Exhaustive	Hand Crafted	Linear
RCNN	Region Proposal	CNN	Linear SVM
Fast RCNN	Region Proposal	Deep	
Faster RCNN	Deep		

J. Redmon, et al, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 779-788.

Single stage processing:

- YOLO (You Only Look Once)



1. Divide into SxS non-overlapping grid.
2. For each grid, predict B boxes containing objects (x,y,w,h) with confidence score.
3. Get softmax probs. of object classes using a pre-trained network for the box with max. confidence with non-maximal suppression.

End to end training for both predicting bounding boxes and object classes.

Final output: SxSX(5B+C) tensor.

YOLO architecture and loss function

- A series of convolution layers (coupled with max pooling) followed by a few fully connected layers.
  - Typically: For image of size 448x448
  - S=7, B=2, C (no. of classes): 20
- Loss function:
  - Localization loss
  - Confidence loss
  - Classification loss
    - All of them use MSE.

Semantic Segmentation

- Label each pixel in the image with a category label.
- Don’t differentiate instances, only care about pixels.

Instance Level Semantic Segmentation

- Even differentiate instances

Semantic Segmentation

Building Blocks of CNNs:

- Convolution
- Down-Sampling
  - MaxPool, AvgPool, Strided Convolution (S>1)
- Up-Sampling
  - UnPooling, Upconvolution

Up- Sampling: Max Unpooling

1	2	**6**	3
3	**5**	2	1
1	2	2	1
7	3	4	**8**

MaxPool →

5	6
7	8

After few layers
in Network

a	b
c	d

← Max Unpool

0	0	**b**	0
0	**a**	0	0
0	0	0	0
c	0	0	**d**

0	0	**1**	0
0	**1**	0	0
0	0	0	0
1	0	0	**1**

Pooling Indices

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

Upsampled convolution

5	6
7	8

2x2 Upsample

5	0	6	0
0	0	0	0
7	0	8	0
0	0	0	0

Input

2	1.1	2.4	.6
1.2	1.3	1.4	.7
.28	1.5	3.2	.8
.7	.75	.8	.4

Filter

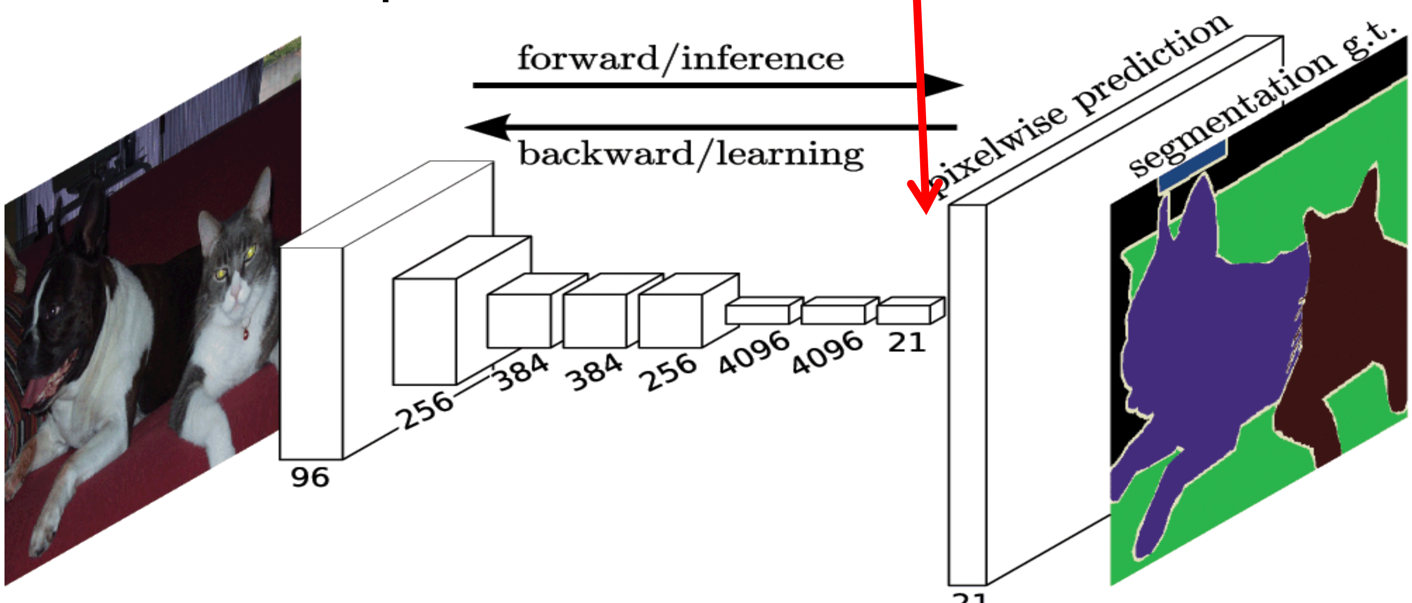
.05	.1	.05
.1	.4	.1
.05	.1	.05

Fully Convolutional Network (FCN)

- No FC layer, only CNNs.
- Dense prediction
  - Downsampled output upsampled followed by 1x1 convolution for providing softmax score at every pixel.
  - No. of channel at the output layer=No. of classes.
  - Cross-entropy loss function.

Semantic Segmentation

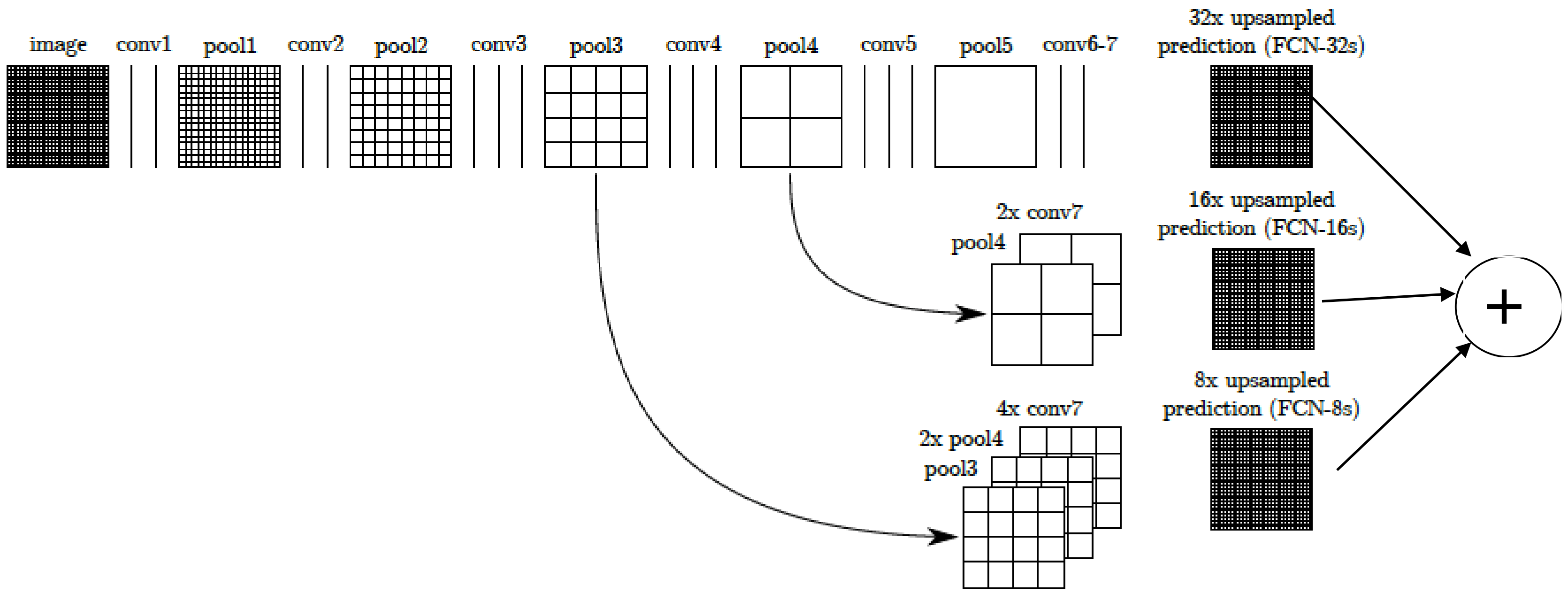
- The upsampling of learned low resolution semantic feature maps is done using upconvolutions which are initialized w
- ith billinear interpolation filters.



Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015

Fusion of prediction

- Combines prediction of higher layer with lower layer (summing them).
- Performs at 3 stages.



Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015

Segnet: Encoder-Decoder Network

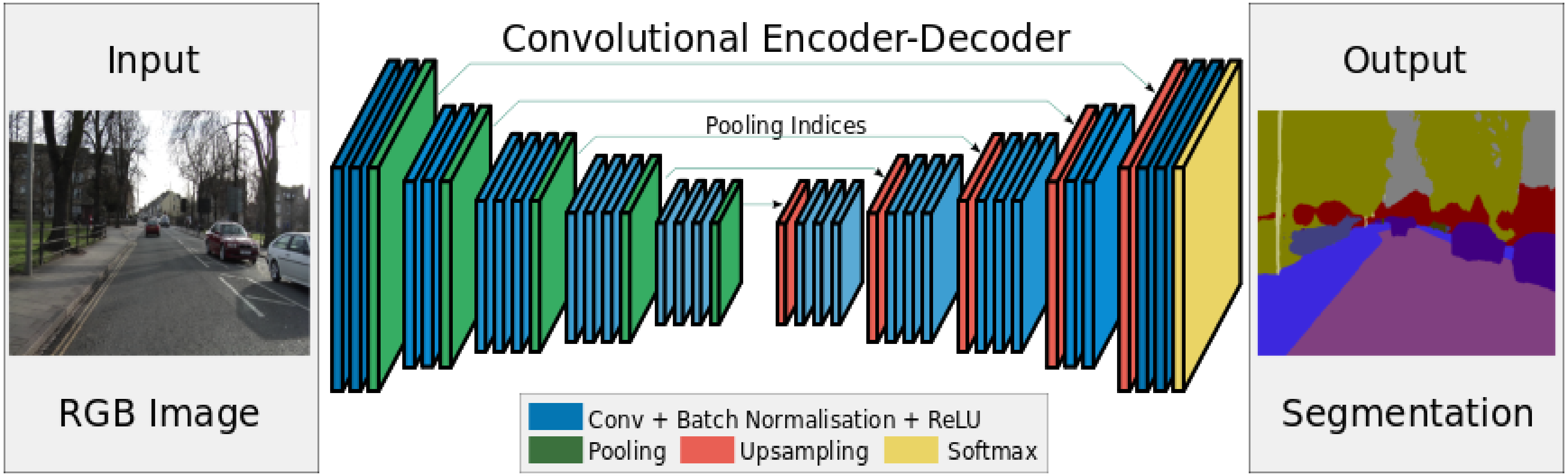
Encoder

- Takes an input image and generates a high-dimensional feature vector
- Aggregate features at multiple levels

Decoder

- Takes a high-dimensional feature vector and generates a semantic segmentation mask
- Decode features aggregated by encoder at multiple levels.
- Semantically project the discriminative features (lower resolution) learnt by the encoder onto the pixel space (higher resolution) to get a dense classification.

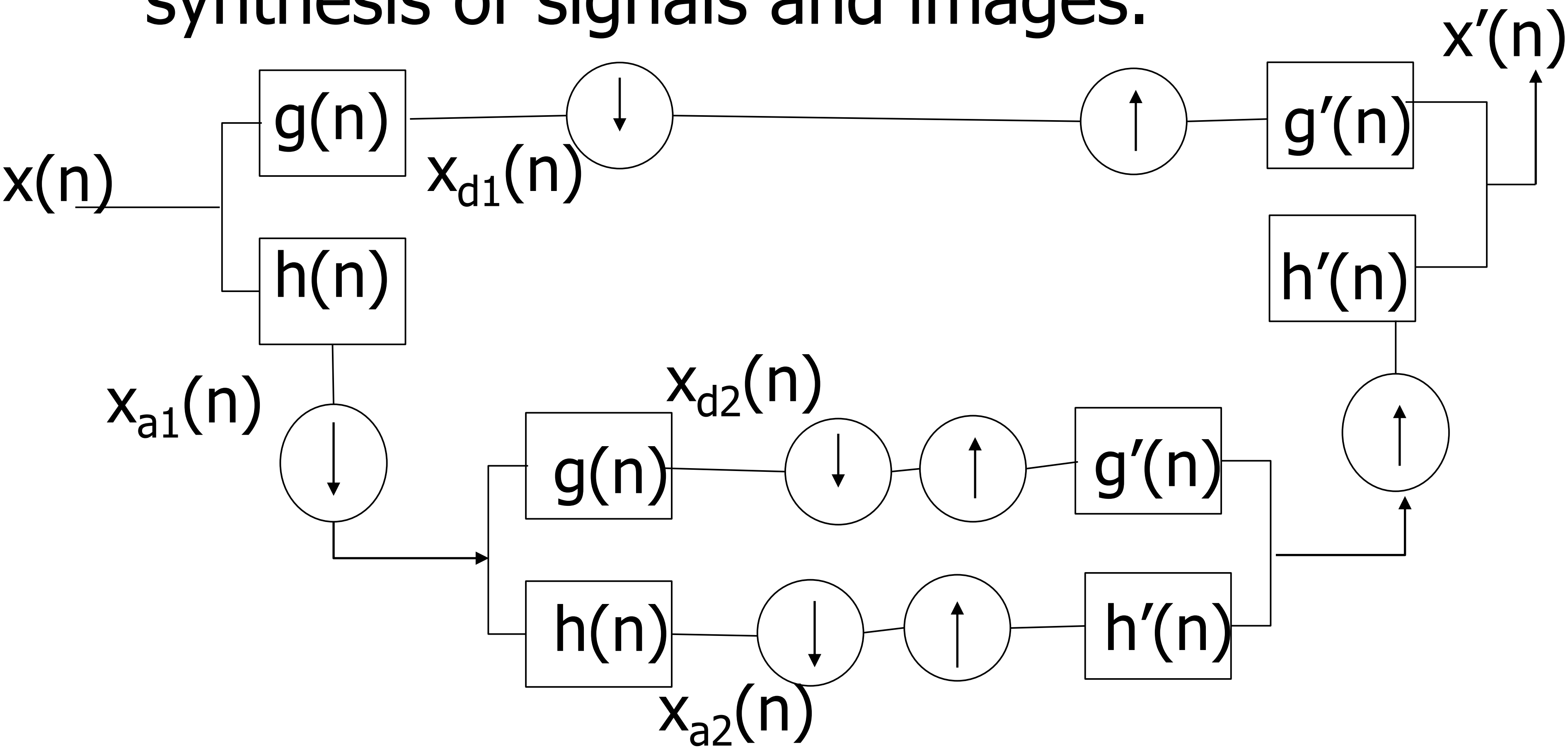
Semantic Segmentation (Encoder-Decoder)



SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation, Vijay Badrinarayanan, et al. PAMI, 2017

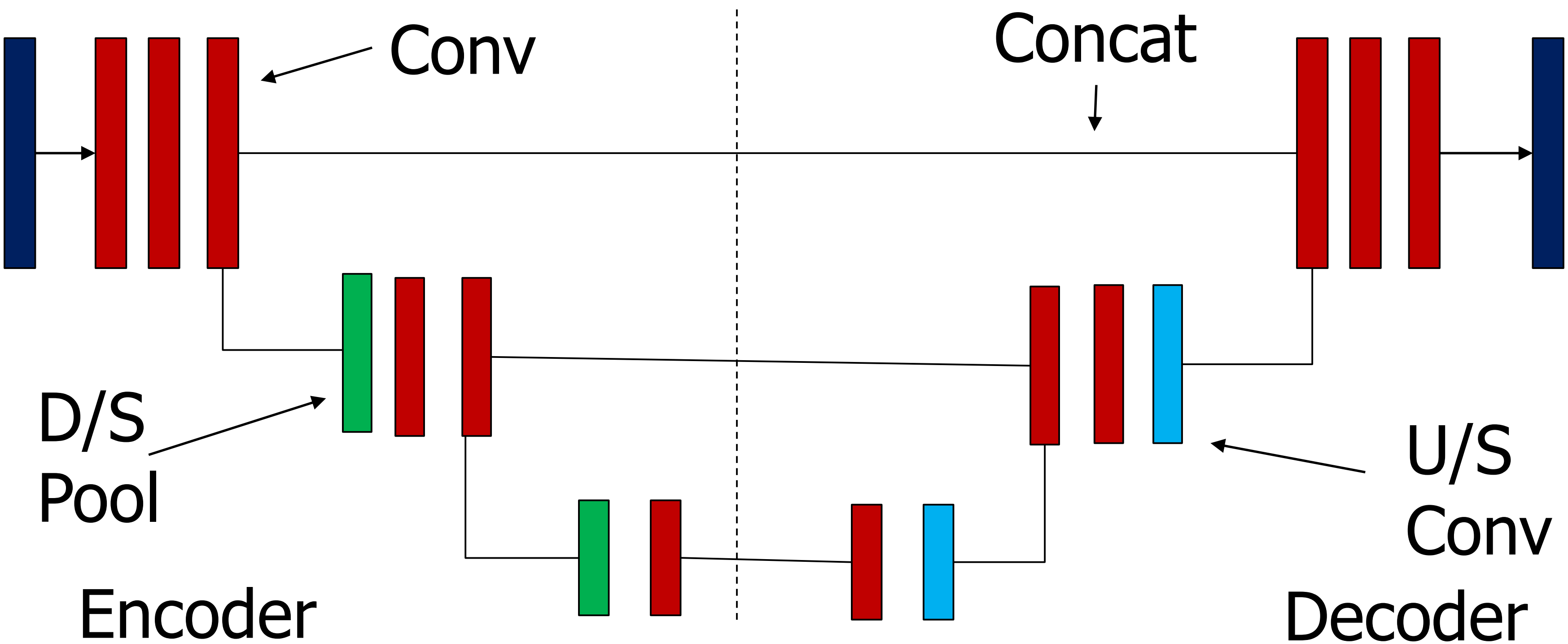
U-Net

- Inspiration from wavelet analysis and synthesis of signals and images.



U-Net architecture

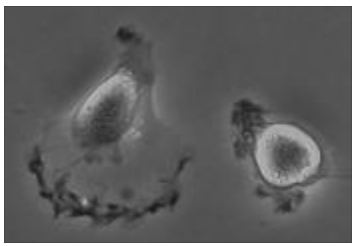
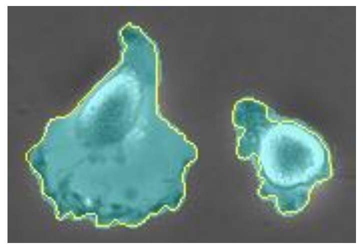
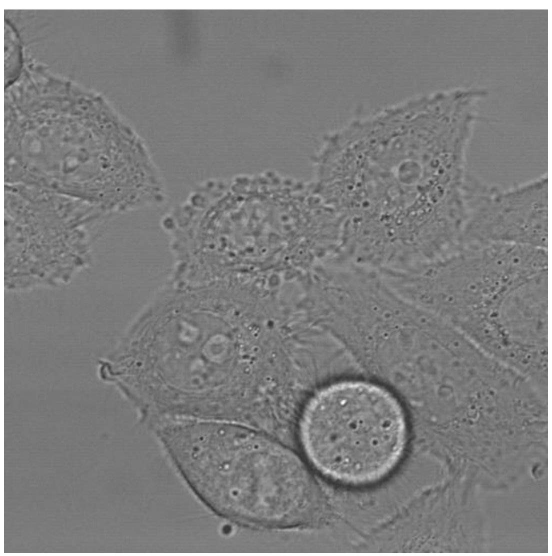
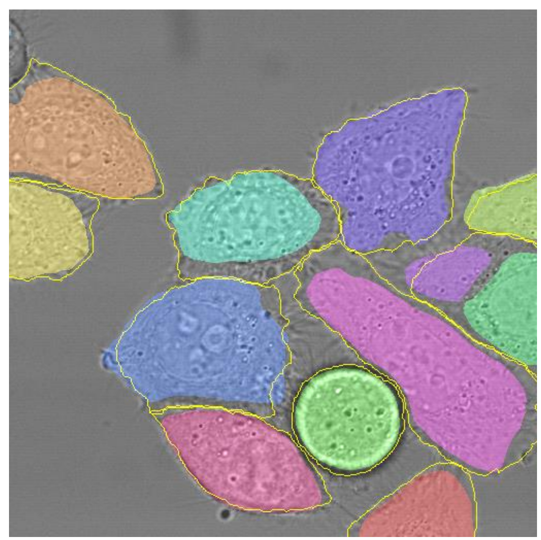
- At the final layer a 1x1 convolution used to map each feature vector to the desired number of classes.



U-Net: Convolutional Networks for Biomedical Image Segmentationm, Olaf Ronneberger, Philipp Fischer, Thomas Brox, https://arxiv.org/abs/1505.04597

Results

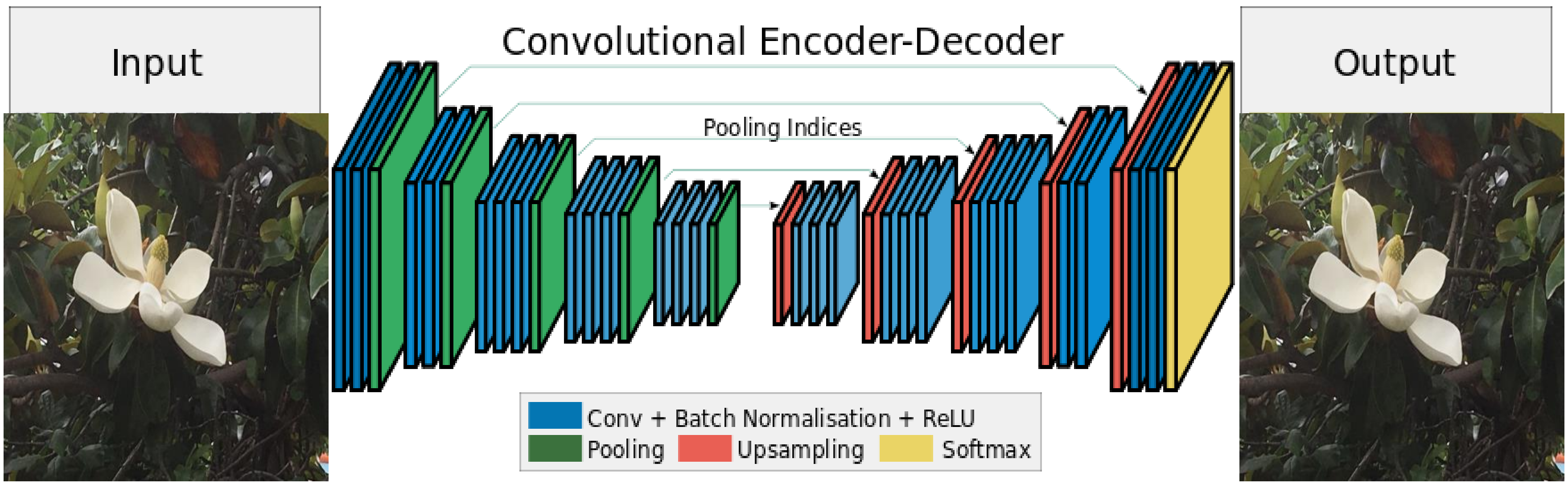
Ground truth (Manual)
Yellow border
Color parts:
Segmented results

Convolutional Encoder Decoder

Self supervised Learning

Typical architecture



Loss function: MSE

Auto encoder: Decoding the same image.

Encoder - Decoder: Applications

- Denoising
- Segmentation
- Colorization
- Super-resolution
- …

Summary

- Deep architecture works in the same principle of artificial neural network.
  - A large number of hidden layers.
  - A large number of weights.
- Convolution Neural Network (CNN)
  - Learns filter weights.
    - Sharing of weights.
  - Two types of layers
    - Convolutional and Pooling Layers
  - Two stages
    - Feature extraction
    - Classification

Summary

- Region proposal networks for simultaneous object localization and classification.
  - YOLO – end to end single staged network.
- Fully Convolutional Networks
  - Semantic Segmentation
  - U-Net
  - Encoder Decoder Network

Thank you!