Deep Learning for Visual Recognition
- Week 2

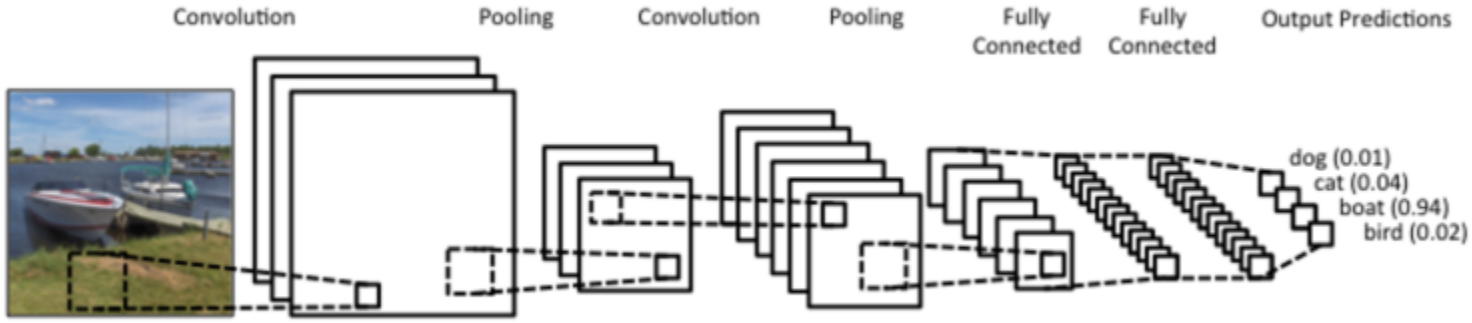
Kyu-Hwan Jung, Ph.D
kyuhwanjung@gmail.com

Class 6
Overview of CNN

Convolutional Neural Network (CNN)

What We See

What Computers See



Elements of Convolutional Neural Network

- Local Connectivity
- Parameter Sharing

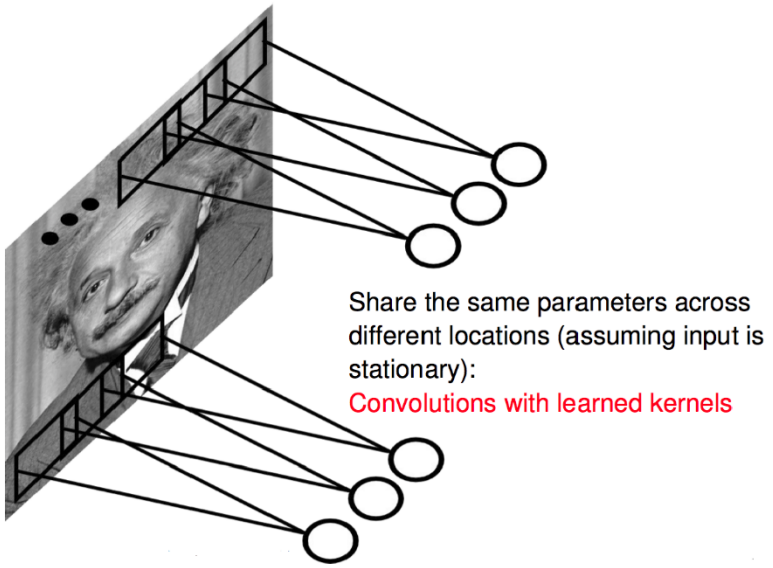
→ Convolution Layer

- Pooling/Subsampling

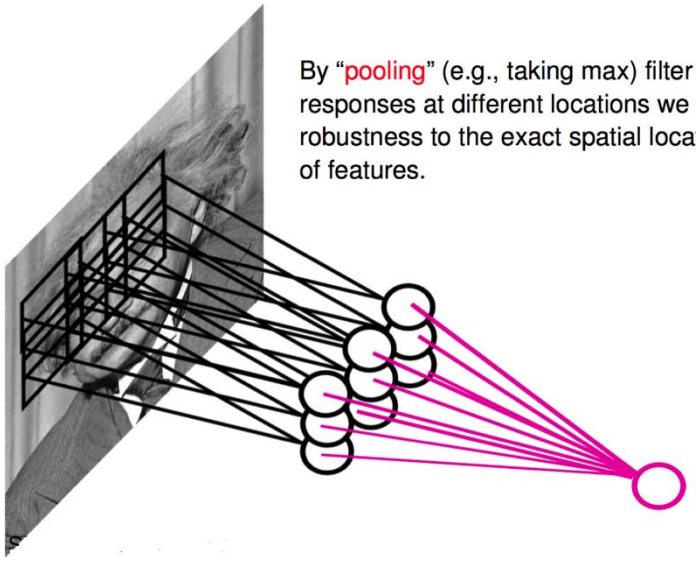
→ Pooling Layer

- Nonlinearity

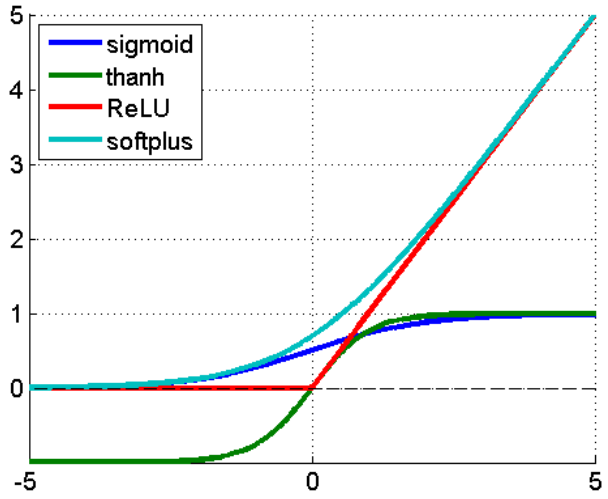
→ Activation Function



Convolution



Pooling



Activation Function

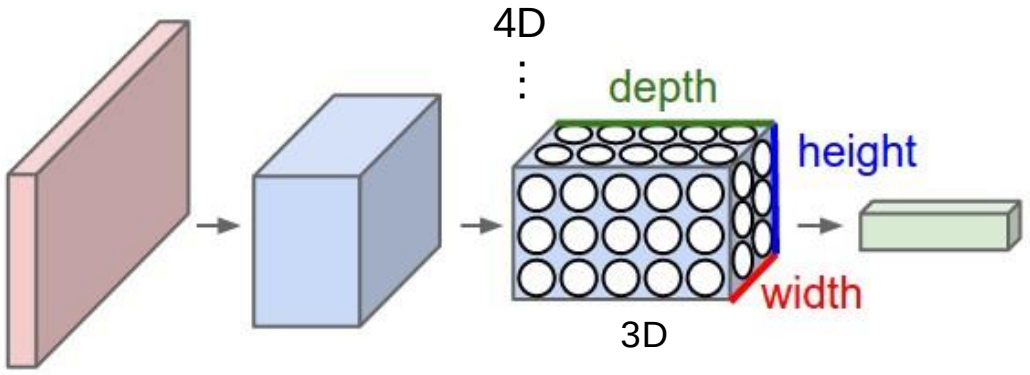
Data Structure

- 4D Tensor
  - (Sample, Channel, Height, Width)
  - Height and width for feature map size
  - Channel for number of feature map(or number of filter)
  - Sample is mini-batch size

cuDNN Example :
cudnnSetTensor4dDescriptor(outputDesc,CUDNN_TENSOR_NCHW,
CUDNN_FLOAT,sampleCnt,channels,height,width)

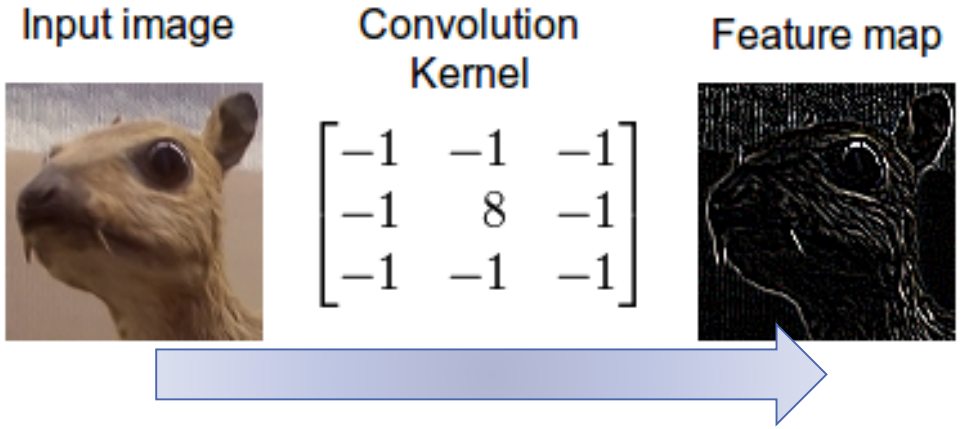
4D Tensor (2, 2, 3, 3)

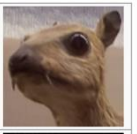
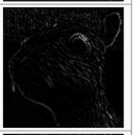
Sample 1			Sample 2			
1	2	3	19	20	21	Channel 1
4	5	6	22	23	24	
7	8	9	25	26	27	
10	11	12	28	29	30	Channel 2
13	14	15	31	32	33	
16	17	18	34	35	36	



Convolution

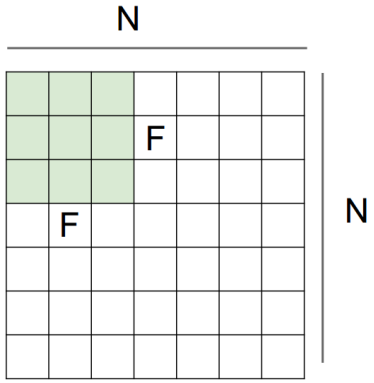
- Basic Usage of Convolution
  - To remove unimportant information from the image : Filter
  - To capture necessary information for given task : Feature
  - To map input image patch of same size into single number : Kernel
  - Choosing right convolution kernel is crucial for the success of analysis => How can we choose the best kernel?



Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

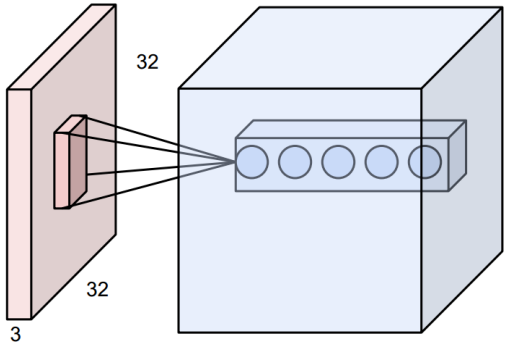
Convolution

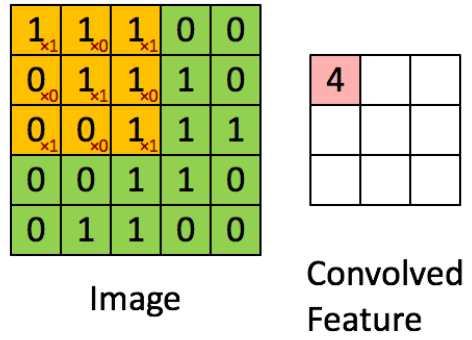
- Local Connectivity
  - Multiple neurons all looking at the same region(receptive field) of the input volume
    stacked along the depth
  - The size of receptive field for each dimension, stride and padding size and method are all hyper-parameters.



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = … :\



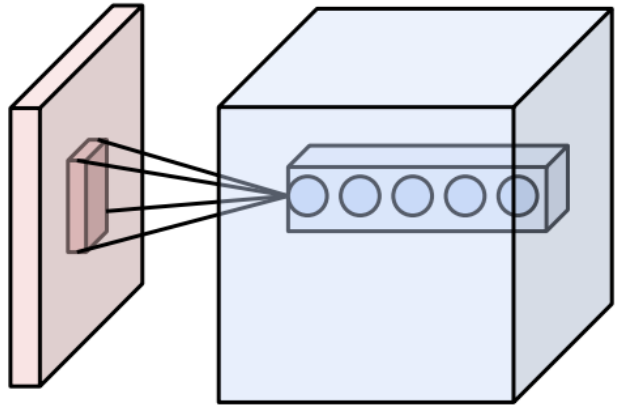


Convolution

- Need for Weight Sharing

Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 1**
Number of neurons: **30**

Output volume: (32 - 5) / 1 + 1 = 28, so: **28x28x 30**
How many weights for each of the 28x28x 30 neurons? **5x5x3 = 75**



Before:
#weights in such layer: (32*32*30) * 75 = **3 million :**

Now: (paramater sharing)
#weights in the layer: 30 * 75 = **2250**.

Example trained weights

IDEA: lets not learn the same thing across all spatial locations

Convolution

- Zero-padding
  - Adding zero-valued(most common) pixels surrounding the input feature map.
  - Provide some useful characteristics as below :

“Same convolution” (preserves size)

Input [9x9]

3x3 neurons, stride 1, pad **1** => [9x9]
3x3 neurons, stride 1, pad **1** => [9x9]

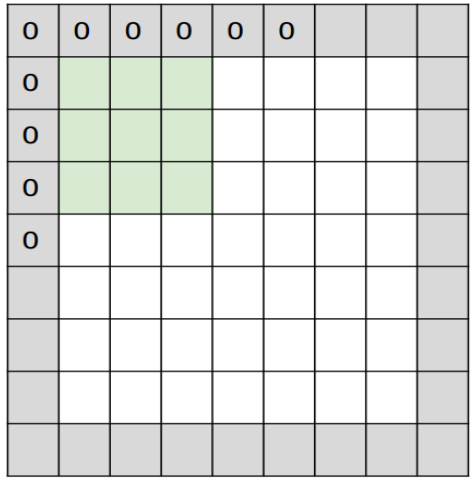
- No headaches when sizing architectures
- Works well

“Valid convolution” (shrinks size)

Input [9x9]

3x3 neurons, stride 1, pad **0** => [7x7]
3x3 neurons, stride 1, pad **0** => [5x5]

- **Headaches** with sizing the full architecture
- **Works Worse!** Border information will “wash away”, since those values are only used once in the forward function



Convolution vs Cross-correlation

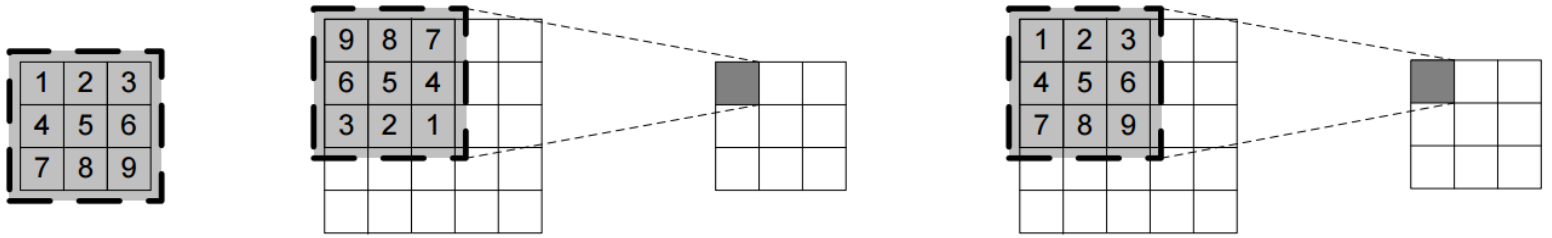
- In practice, we use cross-correlation instead of 2d convolution operation.
  - Because the filter weights are randomly initialized and learned, both convolution and cross-correlation results in similar result.
  - To perform convolution operation, we need to flip the weights in both dimension or rotate the matrix 180 degree.
  - Using cross-correlation is easy to debug and faster

cudnnConvolutionMode_t

Value	Meaning
CUDNN_CONVOLUTION	In this mode, a convolution operation will be done when applying the filter to the images.
CUDNN_CROSS_CORRELATION	In this mode, a cross-correlation operation will be done when applying the filter to the images.

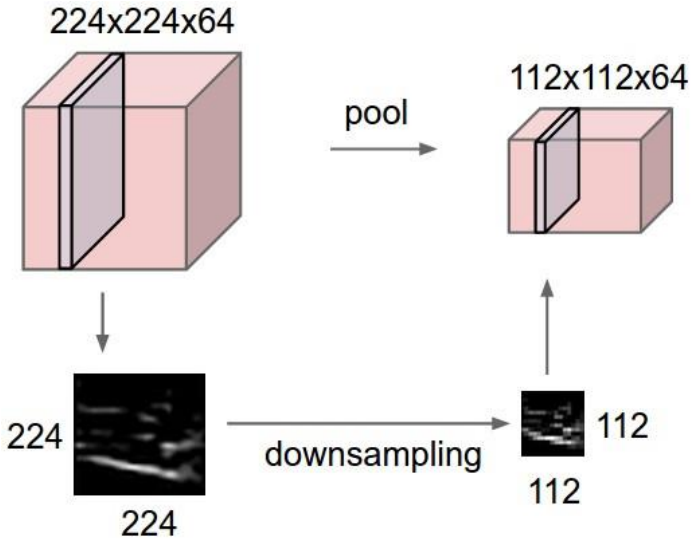
$$Y(x,y) = \sum_{u=0}^{K_x} \sum_{v=0}^{K_y} X(x-u, y-v) w(u,v)$$

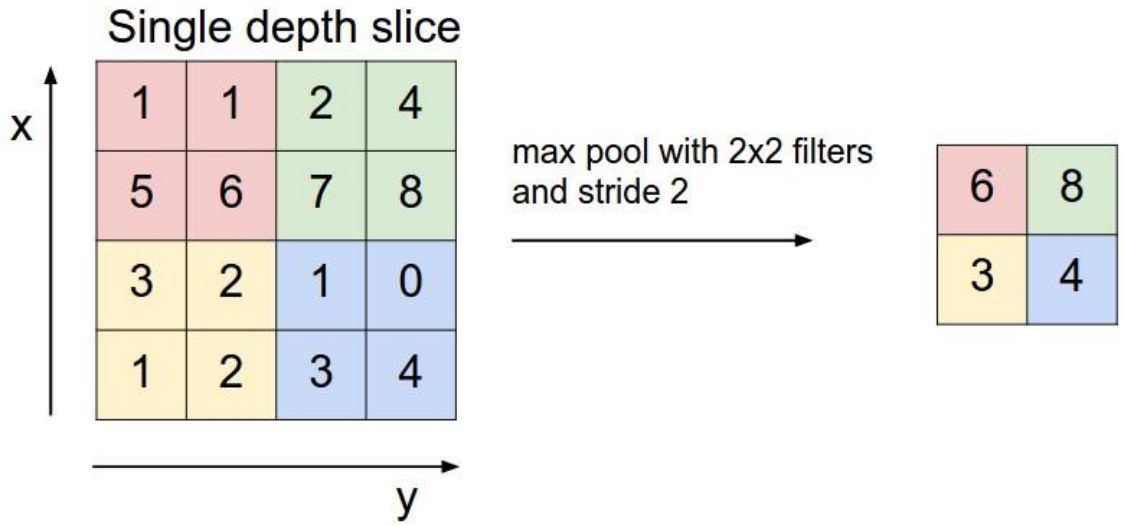
$$Y(x,y) = \sum_{u=0}^{K_x} \sum_{v=0}^{K_y} X(x+u, y+v) w(u,v)$$



Pooling

- Pooling or Subsampling Feature Map
  - Introduces invariance to local translations
  - Reduces the number of hidden units and enlarge the receptive field
  - Max(most popular) or average pooling
  - Need to save the 'switch' index for backpropagation

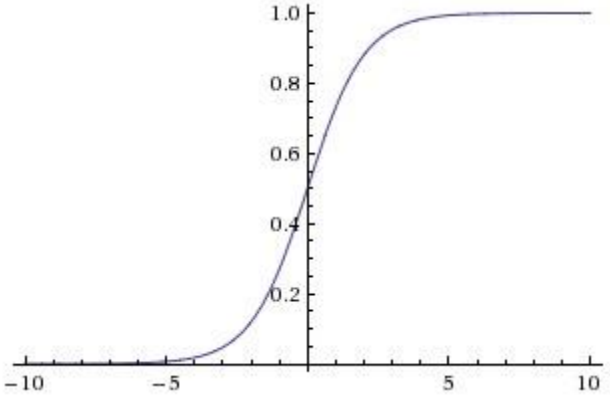




Basic Activation Functions

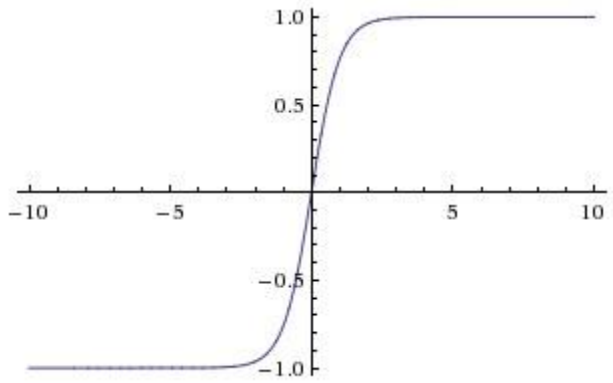
- Activation Function
  - Differentiable nonlinear function which controls the firing rate of neurons.
  - Squashes real-value input to the output of specific range.
  - The output range and gradient of activation function have significant impact to training neural network

Sigmoid



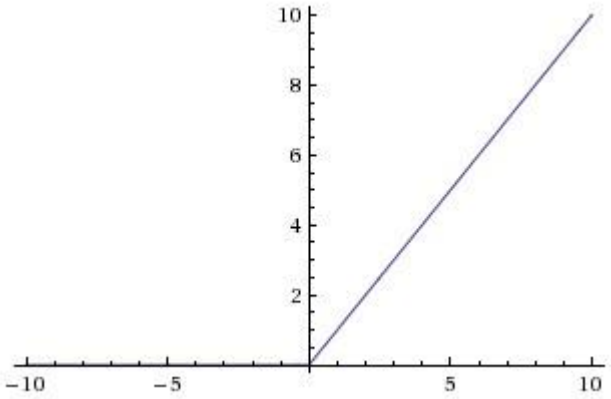
$$g(z) = \frac{1}{1 + e^{-z}}$$

Tanh



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

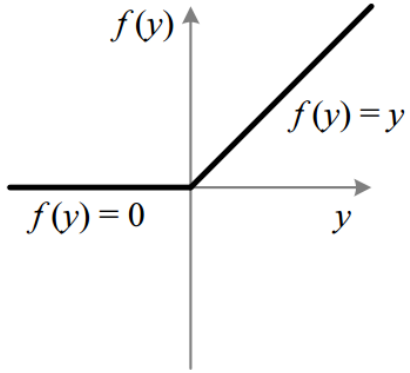
Rectified Linear Unit

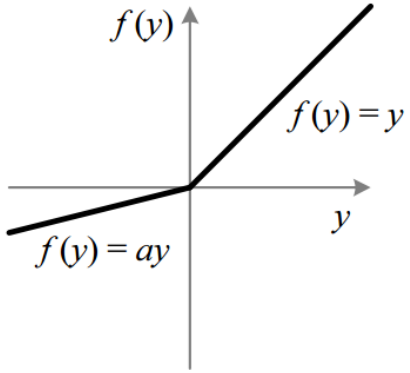


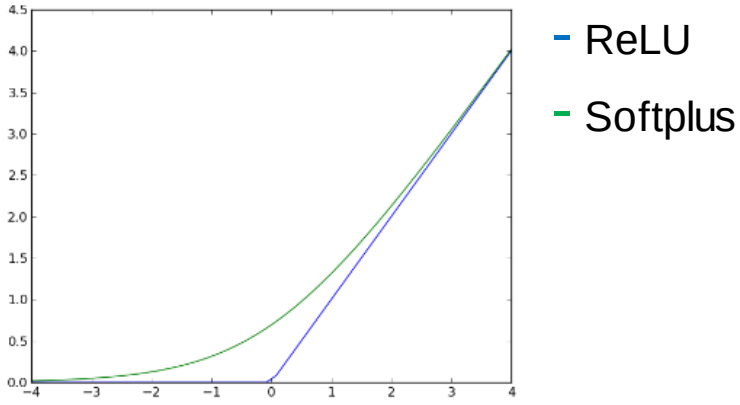
$$g(z) = \max(0, z)$$

Other Activation Functions

- To reduce 'dying' neurons
  - Leaky ReLU : Allow fixed small slope in the negative input range
  - Parametric ReLU : Allow 'parameterized' slope in the negative input range
  - Softplus : Smooth function which is approximately linear function in the positive input range and allow some gradient in the negative input range







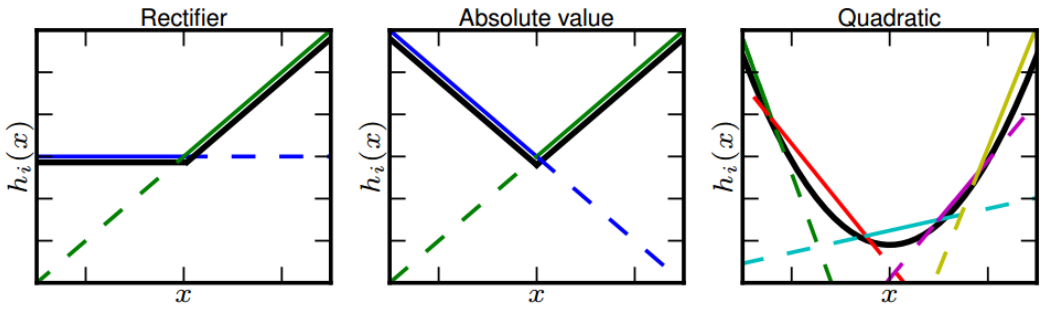
$$g(z) = \ln(1 + e^z)$$

More Complex Activation Function

- Maxout Network(I. Goodfellow, 2013)
  - Generalization of ReLU
  - Take maximum of set of piecewise linear function
  - Can approximate any convex function

$$h_i(x) = \max_{j \in [1,k]} z_{ij}$$

$$z_{ij} = x^T W_{\ldots ij} + b_{ij}$$

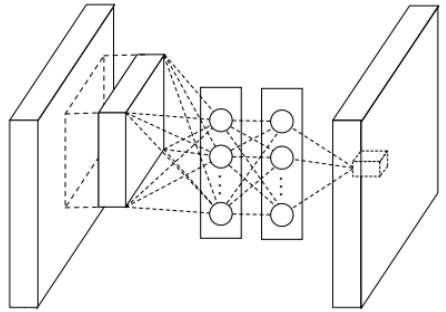


- Network-in-Network(M. Lin, 2013)
  - Universal approximator(MLP) as an activation function

$$f^1_{i,j,k_1} = \max({w^1_{k_1}}^T x_{i,j} + b_{k_1}, 0).$$

$$\vdots$$

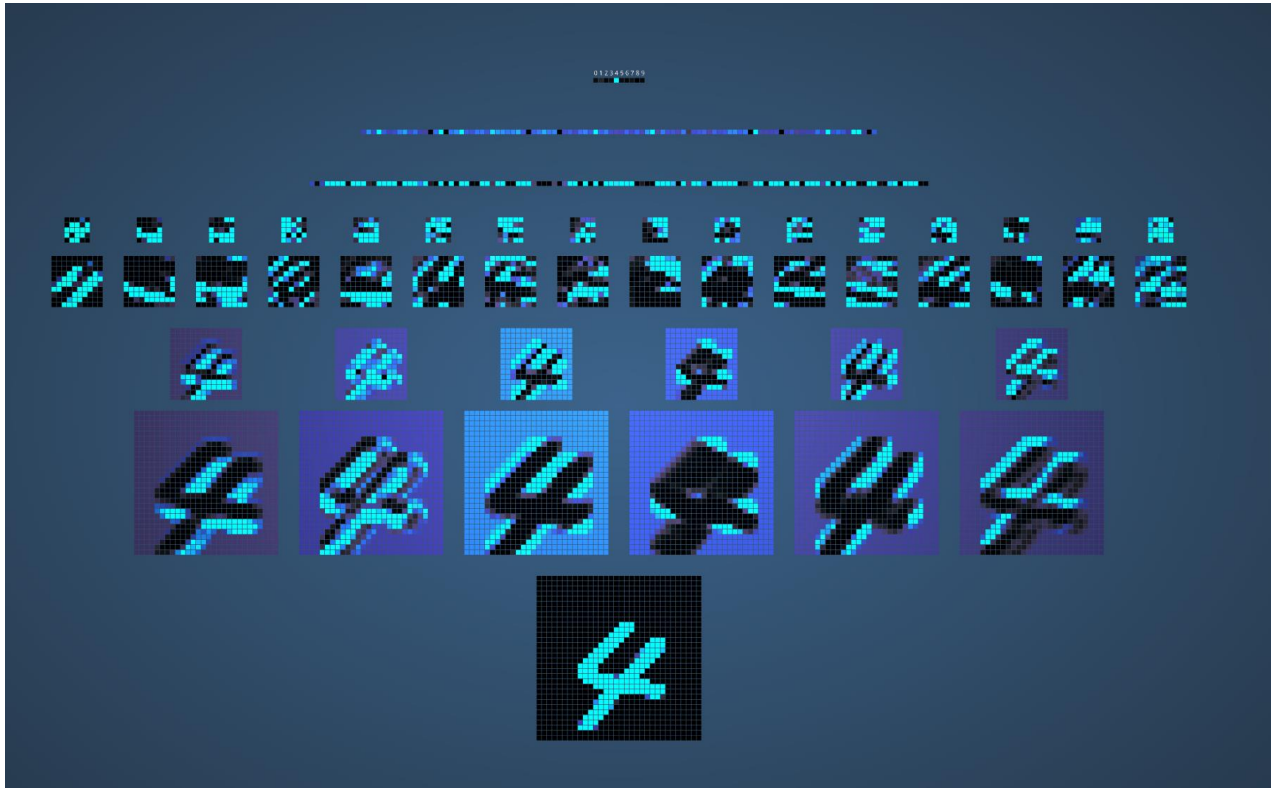
$$f^n_{i,j,k_n} = \max({w^n_{k_n}}^T f^{n-1}_{i,j} + b_{k_n}, 0).$$



mlpconv

Visualizing CNN

- Visualizing Number Classifying CNN
  - http://scs.ryerson.ca/~aharley/vis/conv/flat.html



Lab

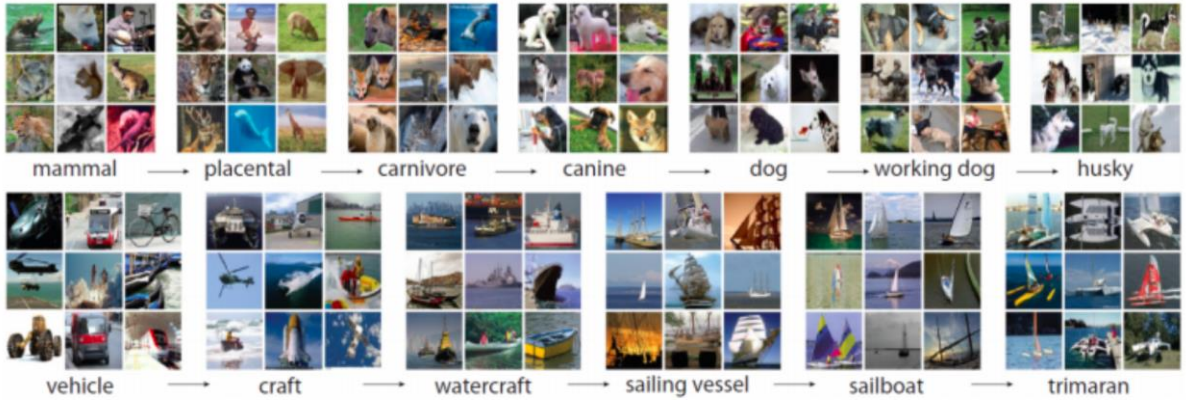
Lab

- Convolution and Pooling
  - https://www.tensorflow.org/api_docs/python/nn/convolution
  - https://www.tensorflow.org/api_docs/python/nn/pooling
  - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part2/notebook
- Simple CNN for MNIST
  - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part2/notebook
- Simple CNN for CIFAR-10
  - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part2/notebook

Class 7
Modern CNNs

ImageNet Large Scale Visual Recognition Challenge

- ImageNet Dataset
  - 22K Categories
  - 14M annotated images



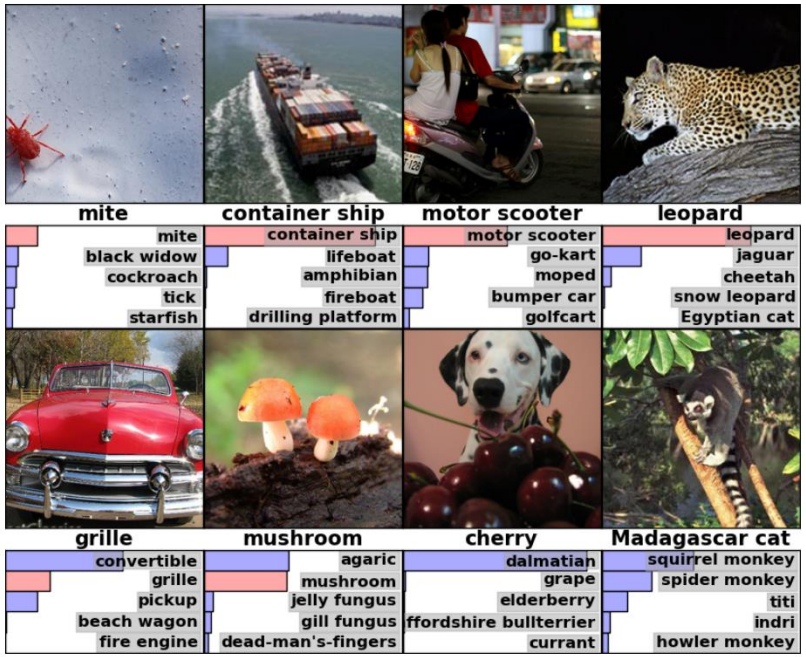
→ Fungus: 134 classes, 90K images

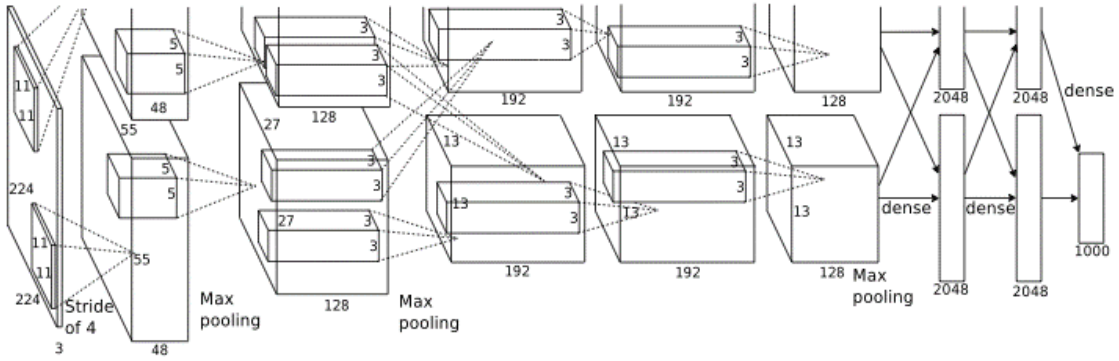
→ Vehicle: 262 classes, 226K images

Must Watch : https://www.youtube.com/watch?v=40riCqvRoMs

Convolutional Neural Network in ILSVRC (2012)

IMAGENET





ILSVRC 2012

SuperVision	15.3%	Deep CNN
ISI	26.1%	FV + PA
OXFORD_VGG	26.7%	FV + SVM
XRCE/INRIA	27.1%	FV + SVM
Univ. of Amsterdam	29.6%	FV + SVM
LEAR-XRCE	34.5%	FV + NCM

ILSVRC 2013

Clarifai	11.7%	Deep CNN
NUS	13.0%	SVM based + Deep CNN
ZF	13.5%	Deep CNN
Andrew Howard	13.6%	Deep CNN
OverFeat-NYU	14.1%	Deep CNN
UvA-Euvison	14.2%	Deep CNN

Convolutional Neural Network in ILSVRC (2012~)

- Performance

Team	Year	Place	Top-5 test error
SuperVision	2012	1	16.42%
ISI	2012	2	26.17%
VGG	2012	3	26.98%
Clarifai	2013	1	11.74%
NUS	2013	2	12.95%
ZF	2013	3	13.51%
GoogLeNet	2014	1	6.66%
VGG	2014	2	7.32%
MSRA	2014	3	8.06%
Andrew Howard	2014	4	8.11%
DeeperVision	2014	5	9.51%

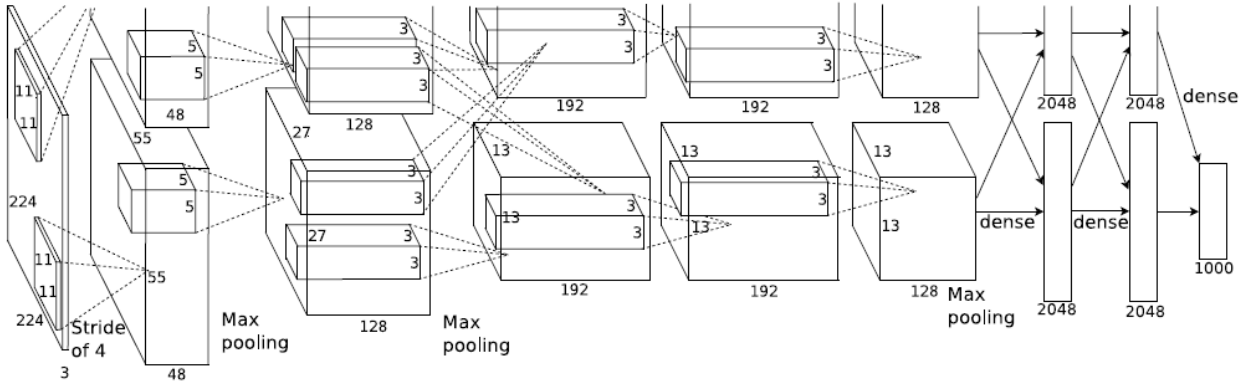
Team	Date	Top-5 test error
GoogLeNet	2014	6.66%
Deep Image	01/12/2015	5.98%
Deep Image	02/05/2015	5.33%
Microsoft	02/05/2015	4.94%
Google	03/02/2015	**4.82%**

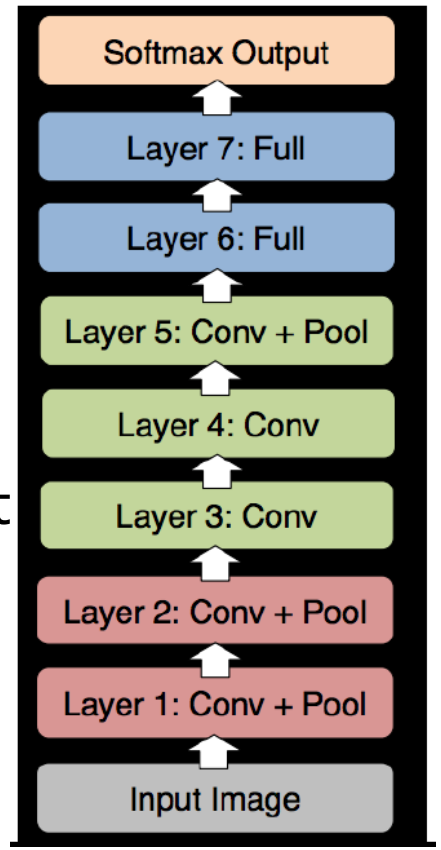
Alex Net

- Winner of ILSVRC 2012(16.4%)
  - Multi-GPU Implementation
  - Local Response Normalization(+1.2%)

$$b_{x,y}^{i} = a_{x,y}^{i} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^{j})^{2} \right)^{\beta}$$

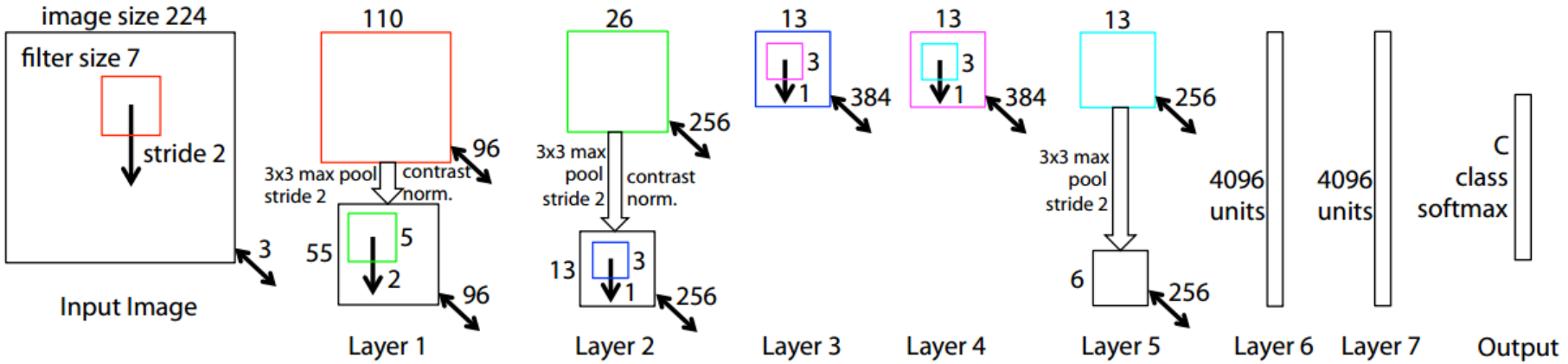
  - Overlapping pooling(+0.3%)
  - Dropout and ReLU
  - Data augmentation(+1%) : Random Crop and Color Shift



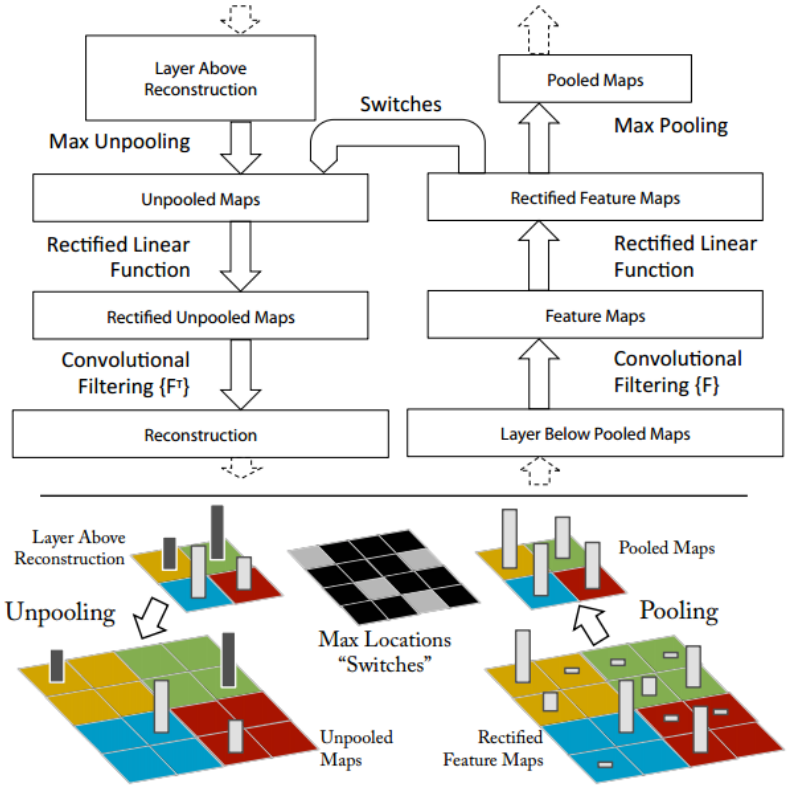


ZF Net

- Winner of ILSVRC 2013(11.74%)
  - Smaller filter in the input layer(11->7)
  - Smaller stride in the input layer(4->2)
  - All other details are similar to Alex Net



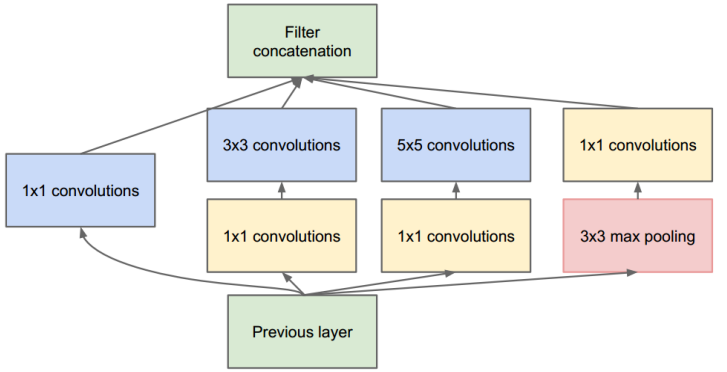
Visualizing CNNs

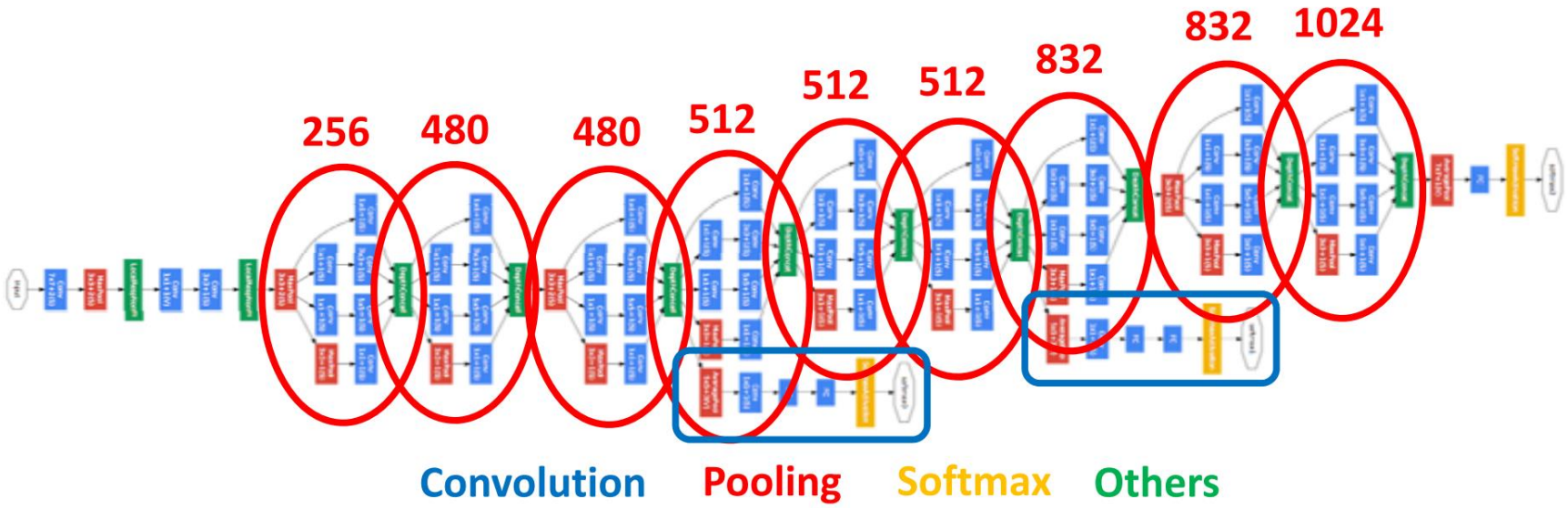


DeConvnet (Zeiler and Fergus, 2013)

GoogLeNet

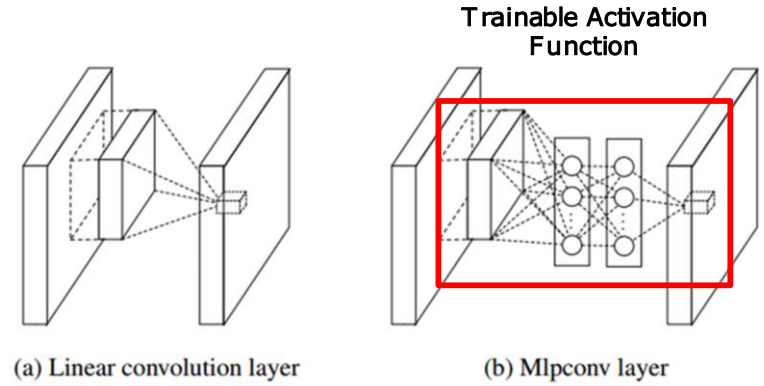
- Winner of ILSVRC 2014(6.67%)
  - 9 Inception module : 22 conv layers
  - Auxiliary classifier to facilitate training
  - 1x1 convolutions for dimension reduction
  - Far less parameters than previous nets (Alex Net : 60M, GoogLeNet : 5M)

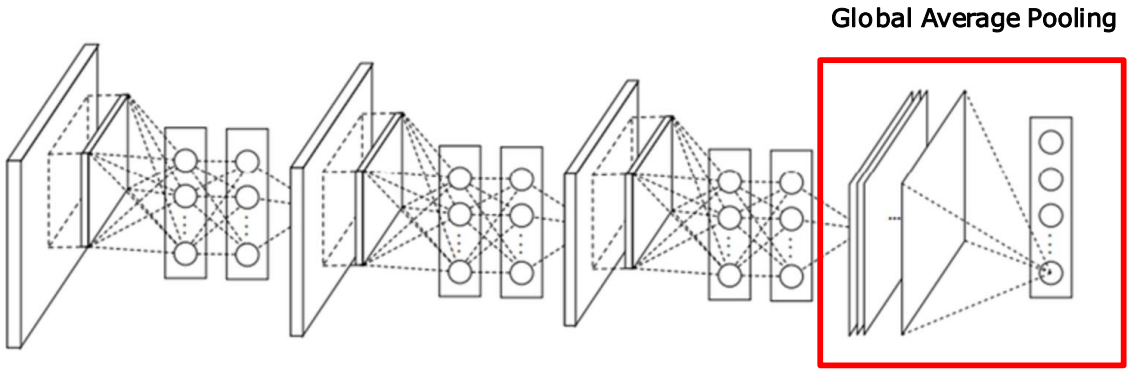




Network in Network

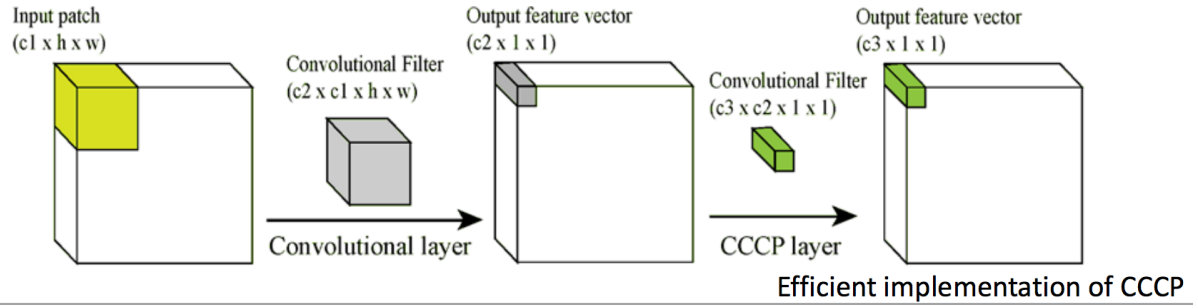
- Extracting Nonlinear Features using MLPconv Layer
  - Instead of using linear convolution layer for extracting linear features, NIN uses MLPconv which has more than 1 layers for nonlinear feature extraction.
  - We can view MLPconv as a combination of ordinary convolution layer and trainable nonlinear activation function.
  - NIN also uses global average pooling instead of fully-connected layer in classifier part of the entire CNN.
  - This reduces significant number of parameters in the fully-connected layer, which prevents overfitting.



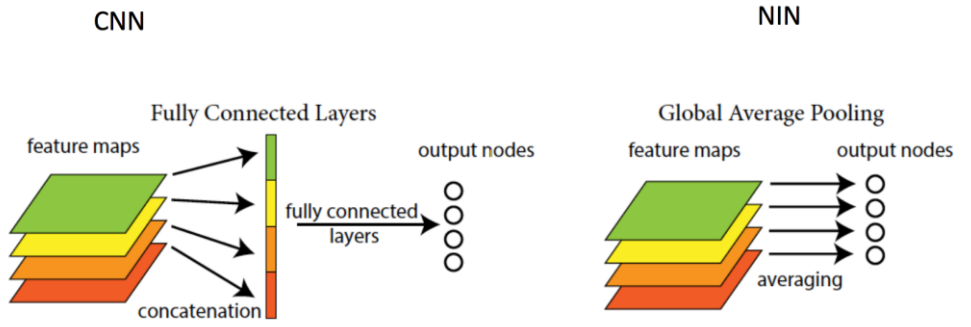


Network in Network

- Cascaded 1x1 Convolution
  - Using 1x1 convolution for reducing dimensionality of feature map.
  - We can see this as a cascade of cross-channel parametric(learnable) pooling.



Efficient implementation of CCCP

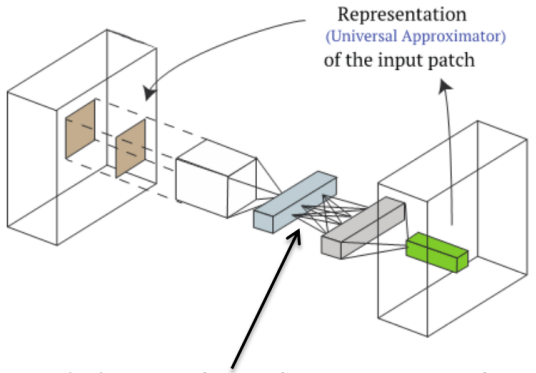


Explicitly confidence map of each category

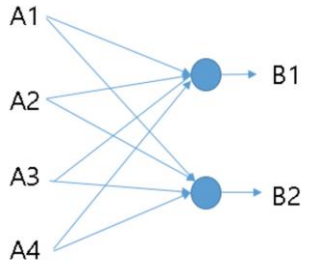
Local patch is projected to its value in a feature map using **a small network**

$$y_i = \phi(w_i^T y_{i-1} + b_i)$$

$$y_0 = x$$



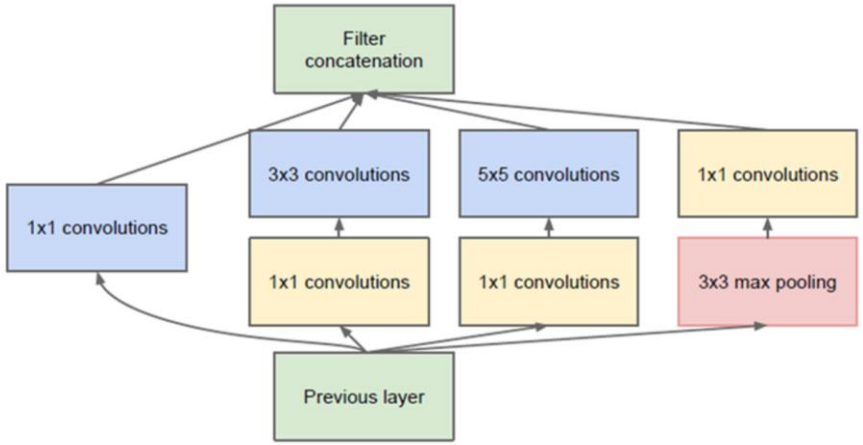
Cascaded Cross Channel Parametric Pooling (CCCP)



Fully-connected Layer
= 1x1 convolution

GoogLeNet - revisited

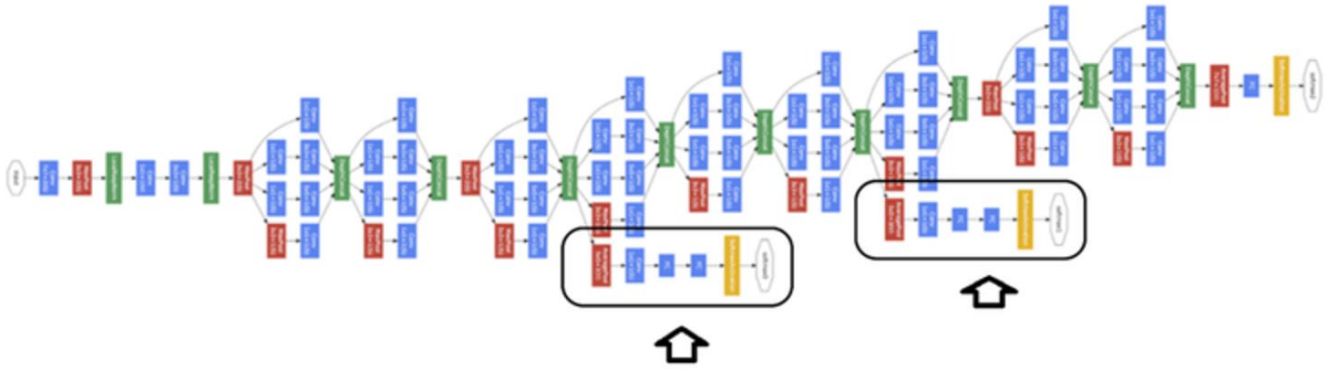
- Model Architecture
  - Stack of 'Inception' Modules



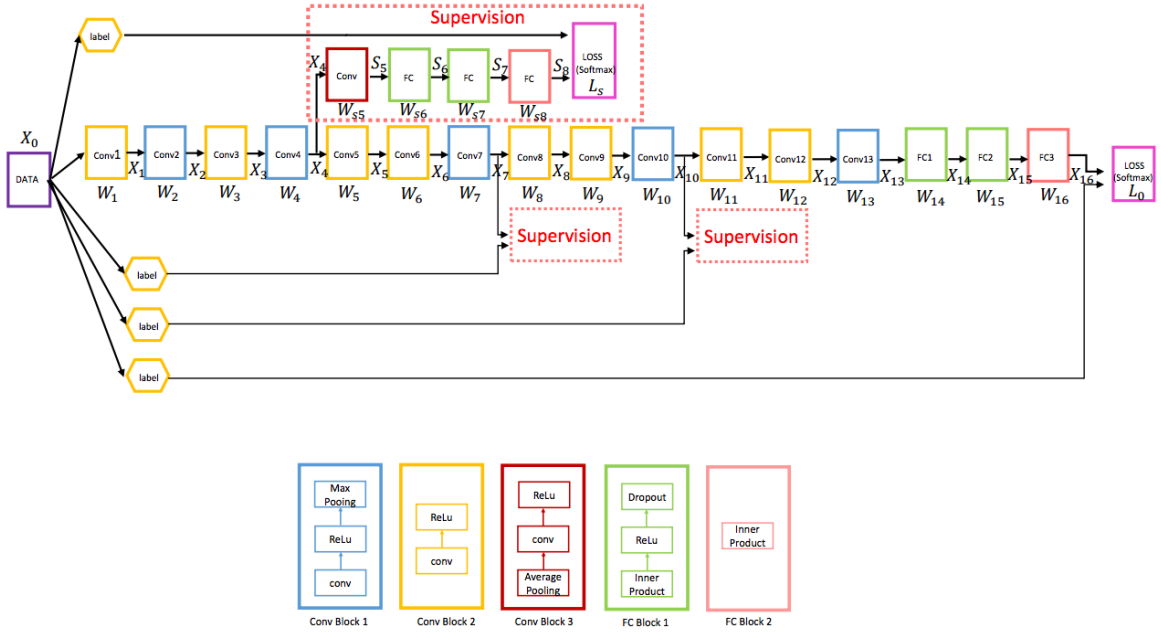
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

GoogLeNet - revisited

- Model Architecture
  - Auxiliary Classifier
    - Solving vanishing gradient problem in very deep CNNs
    - Also works as a regularizer for the intermediate layers



GoogLeNet 2014



Wang et. al. 2015

Inception-v1

- Factorized GoogLeNet
  - Almost same with GoogLeNet
  - 5x5 convolution filters -> Two consecutive 3x3 filters
  - By combining with batch normalization(to be explained soon), new state-of-the-art in ILSVRC2012 dataset is achieved.

Team	Date	Top-5 test error
GoogLeNet	2014	6.66%
Deep Image	01/12/2015	5.98%
Deep Image	02/05/2015	5.33%
Microsoft	02/05/2015	4.94%
Google	03/02/2015	**4.82%**

Inception +
Batch Normalization

VGG Net

- The Runner-up of ILSVRC 2014(7.32%)
  - Very deep structure with small filters
  - Memory intensive(140M parameters)
  - Useful for transfer learning

Suppose input has depth C & we want output depth C as well

1x CONV with 7x7 filters	3x CONV with 3x3 filters
Number of weights:	Number of weights:
C*(7*7*C) = **49 C^2**	C*(3*3*C) + C*(3*3*C) + C*(3*3*C) = 3 * 9 * C^2 = **27 C^2**

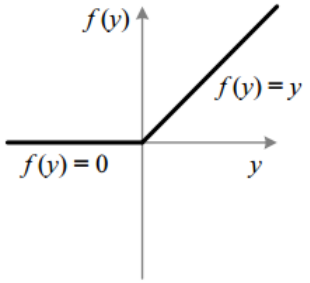
Fewer parameters and more nonlinearities = GOOD.

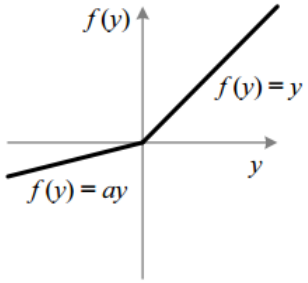
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

MS Net

- First CNN Surpassing Human Performance(4.94%)
  - Parametric ReLU
  - Smart weight initialization
  - Train deeper network





$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}$$

$$\frac{1}{2} n_l Var[w_l] = 1, \quad \forall l. \quad \Rightarrow \quad \sqrt{2/n_l} \qquad n_l = k_l^2 c_l$$

input size	VGG-19 [25]	model A	model B	model C
224	3×3, 64 3×3, 64 2×2 maxpool, /2	7×7, 96, /2	7×7, 96, /2	7×7, 96, /2
112	3×3, 128 3×3, 128 2×2 maxpool, /2	2×2 maxpool, /2	2×2 maxpool, /2	2×2 maxpool, /2
56	3×3, 256 3×3, 256 3×3, 256 3×3, 256 2×2 maxpool, /2	3×3, 256 3×3, 256 3×3, 256 3×3, 256 3×3, 256 2×2 maxpool, /2	3×3, 256 3×3, 256 3×3, 256 3×3, 256 3×3, 256 3×3, 256 2×2 maxpool, /2	3×3, 384 3×3, 384 3×3, 384 3×3, 384 3×3, 384 3×3, 384 2×2 maxpool, /2
28	3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 768 3×3, 768 3×3, 768 3×3, 768 3×3, 768 3×3, 768 2×2 maxpool, /2
14	3×3, 512 3×3, 512 3×3, 512 3×3, 512 2×2 maxpool, /2	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 spp, {7, 3, 2, 1}	3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 3×3, 512 spp, {7, 3, 2, 1}	3×3, 896 3×3, 896 3×3, 896 3×3, 896 3×3, 896 3×3, 896 spp, {7, 3, 2, 1}
fc_1 fc_2 fc_3	4096 4096 1000			
depth (conv+fc)	19	19	22	22
complexity (ops., $\times 10^{10}$)	1.96	1.90	2.32	5.30

Baidu Net

- Brute-force Approach with State-of-the-Art Performance(4.58%?)
  - Intensive data augmentation with high resolution models
  - High performance computing

- Scale up, up to 100 nodes
  - High bandwidth low latency ⟹ **Infiniband**
- 36 nodes, 144 GPUs, 6.9TB Host, 1.7TB Device
  - **0.6 PFLOPS**
- **Highly Optimized software stack**
  - RDMA/GPU Direct
  - New data partition and communication strategies

Possible variations

Augmentation	The number of possible changes
Color casting	68920
Vignetting	1960
Lens distortion	260
Rotation	20
Flipping	2
Cropping	82944(crop size is 224x224, input image size is 512x512)

The Deep Image system learned from **~2 billion** examples, out of **90 billion** possible candidates.

Lab

Lab

- AlexNet for CIFAR-10
  - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part2/notebook
  - http://deepcognition.ai/
- VGGnet for CIFAR-10
  - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part2/notebook
- GoogLeNet for CIFAR-10
  - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part2/notebook

Class 8

State of the Art CNNs for Classification

New ILSVRC Classification Record

- Google Again.

Jeff Dean
공개적으로 공유함 - 2015. 12. 4.

Rethinking the Inception Architecture for Computer Vision

An Arxiv paper posted yesterday at http://arxiv.org/abs/1512.00567 by my colleagues **+Christian Szegedy**, **+Vincent Vanhoucke**, **+Sergey Ioffe**, **+Jon Shlens**, and **+Zbigniew Wojna** details a bunch of improvements to the Inception image classification model that they've been working on. An ensemble of four of these models achieves **3.46% top-5 error** on the validation set of the Imagenet whole image ILSVRC2012 classification task, compared with an ensemble of the initial version of Inception that won last year's 2014 Imagenet classification challenge with a 6.66% top-5 error rate (a 49% reduction in top-5 error rate).

For comparison, **+Andrej Karpathy** estimates that a well-trained human (him) can achieve about 5.1%, as detailed in his delightfully written "_What I learned from competing against a ConvNet on ImageNet_" blog post at http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

The latest Inception models were trained using TensorFlow, our newly open-sourced machine learning system (see http://tensorflow.org).

Team	Date	Top-5 test error
GoogLeNet	2014	6.66%
Deep Image	01/12/2015	5.98%
Deep Image	02/05/2015	5.33%
Microsoft	02/05/2015	4.94%
Google	03/02/2015	**4.82%**

3.46%

Inception-v2

- Scalability is the Key
  - But naïve scaling up kills efficiency which is of growing importance.
- Observations and Lessons Learned
  - **Principle 1** : Avoid representational bottlenecks especially early in the network.
    - The representation size should be gently decrease from the input to outputs
  - **Principle 2** : Higher dimensional representation are easier to process.
    - Increasing activation will allow disentangled features thus result in faster training.
  - **Principle 3** : Spatial aggregation after reducing the dimension of input is beneficial.
    - Strong correlation between adjacent unit leads to much less information loss during dimension reduction and even it promotes faster learning.
  - **Principle 4** : Balance the width and depth of the network
    - Optimal improvement of the network can be achieved when both are increased in parallel.

    => We Need a Wider and Deeper Network !! But How??

Inception-v2

- Factorization into Smaller Convolutions

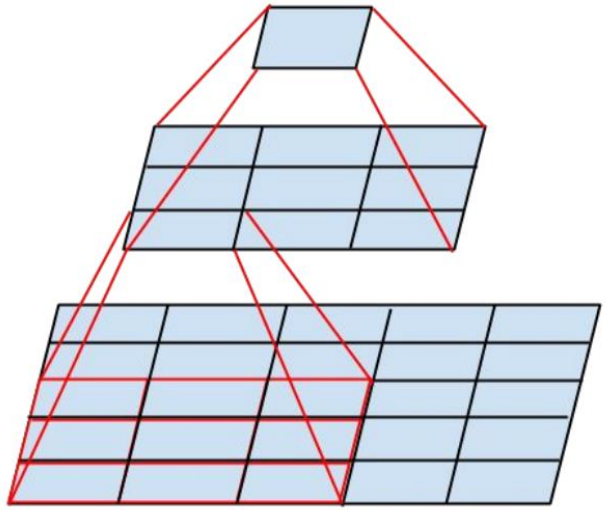


Fig.4

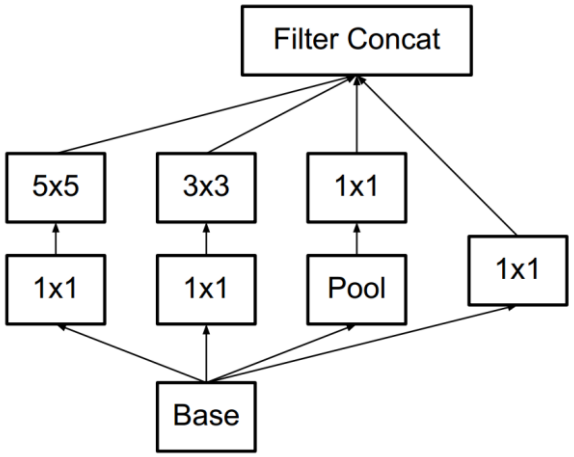
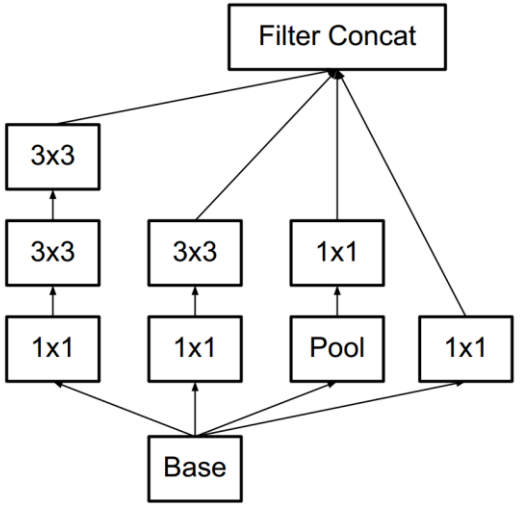


Fig.5



Replacing 5x5 convolutions with two-layered 3x3 convolutions

[Num of parameters]
5x5 convolution = 5x5 = 25
Two layered 3x3 convolution = 3x3 + 3x3 = 18

Inception-v2

- Spatial Factorization into Asymmetric Convolutions

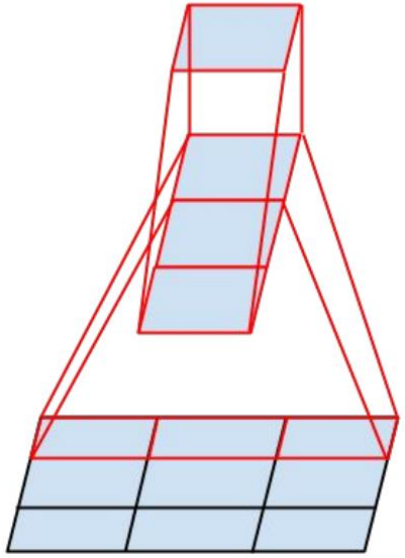


Fig.6

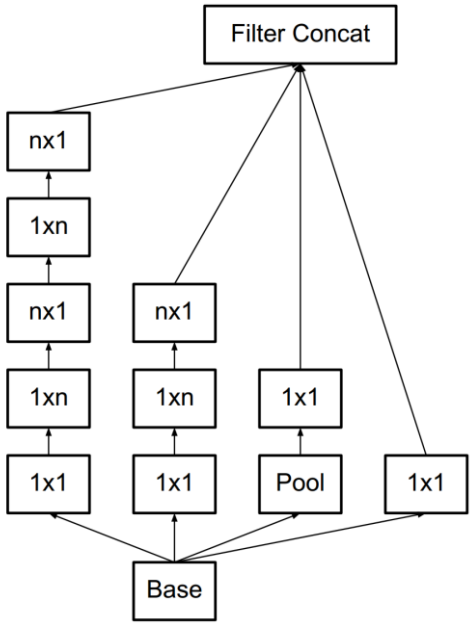
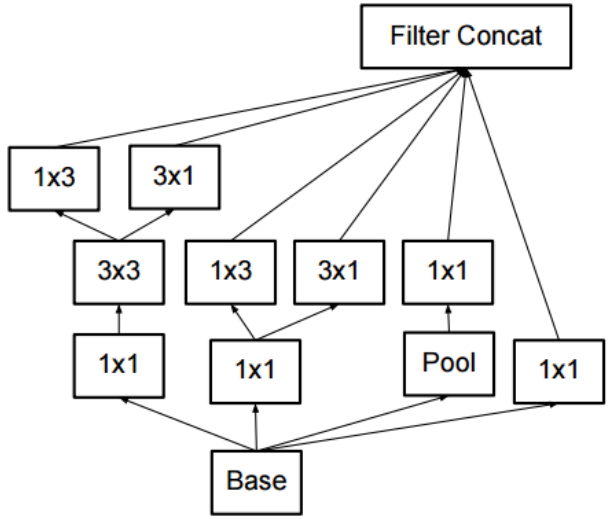


Fig.7



Replacing 3x3 convolutions with
3x1 and 1x3 convolutions

[Num of parameters]
3x3 convolution = 3x3 = 9
3x1 and 1x3 convolution = 3x1 + 1x3 = 6

Inception-v2

- Efficient Grid Size Reduction

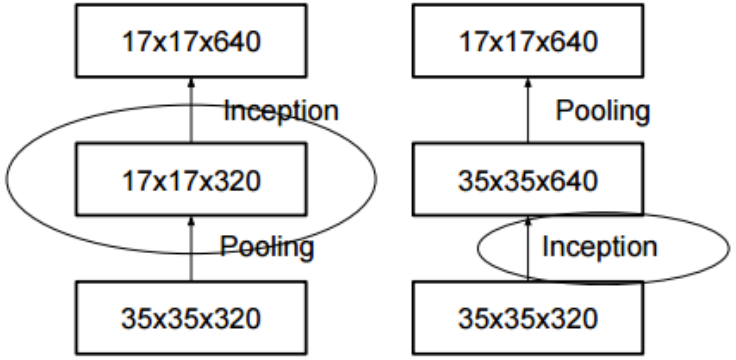


Figure 9. Two alternative ways of reducing the grid size. The solution on the left violates the principle 1 of not introducing an representational bottleneck from Section 2. The version on the right is 3 times more expensive computationally.

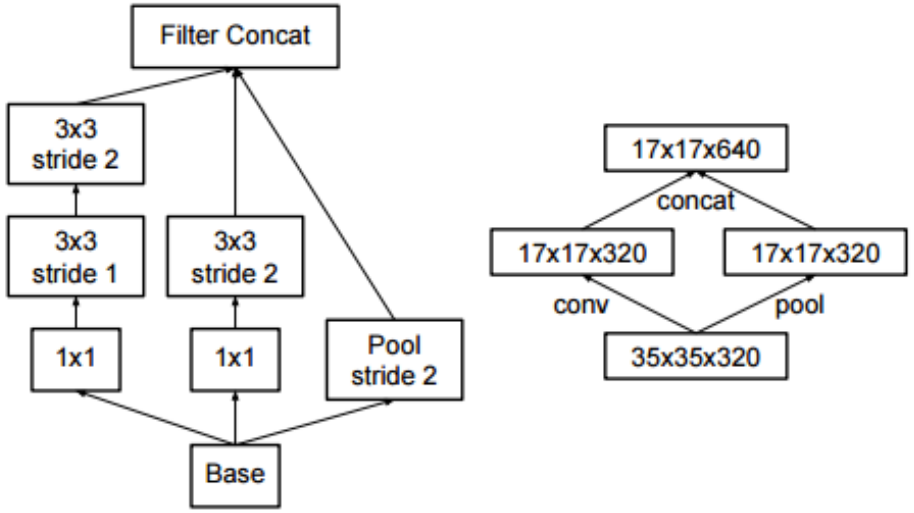


Figure 10. Inception module that reduces the grid-size while expands the filter banks. It is both cheap and avoids the representational bottleneck as is suggested by principle 1. The diagram on the right represents the same solution but from the perspective of grid sizes rather than the operations.

Inception-v2

- Combining All Modules based on the Principles
  - Three 3x3 convolutions instead of 7x7 convolution layer
  - Three traditional Inception module followed by grid reduction in Fig. 10
  - Five factorized Inception in Fig. 5 followed by grid reduction in Fig. 10
  - Two Inception module in Fig. 6
  - Pooling + FC + Softmax
  - 42 Layer deep, 2.5 times computational cost than original GoogLeNet(22 layers)
  - Label smoothing regularization
    - Replace $q(k|x) = \delta_{k,y}$ with $q'(k|x) = (1-\epsilon)\delta_{k,y} + \epsilon u(k)$
  - Modifications to auxiliary classifier
    - No AC at the earlier steps, but AC helps for the lower plateau

type	patch size/stride or remarks	input size	
conv	3×3/2	299×299×3	
conv	3×3/1	149×149×32	
conv padded	3×3/1	147×147×32	
pool	3×3/2	147×147×64	
conv	3×3/1	73×73×64	
conv	3×3/2	71×71×80	
conv	3×3/1	35×35×192	
3×Inception	As in figure 5	35×35×288	+Fig.10
5×Inception	As in figure 6	17×17×768	+Fig.10
2×Inception	As in figure 7	8×8×1280	
pool	8 × 8	8 × 8 × 2048	
linear	logits	1 × 1 × 2048	
softmax	classifier	1 × 1 × 1000	

Inception-v2 and v3

- Performance
  - New state-of-the art in ILSVRC2012 validation set

Network	Top-1 Error	Top-5 Error	Cost Bn Ops
GoogLeNet [20]	29%	9.2%	1.5
BN-GoogLeNet	26.8%	-	**1.5**
BN-Inception [7]	25.2%	7.8	2.0
Inception-v2	23.4%	-	3.8
Inception-v2 RMSProp	23.1%	6.3	3.8
Inception-v2 Label Smoothing	22.8%	6.1	3.8
Inception-v2 Factorized 7 × 7	21.6%	5.8	4.8
Inception-v2 BN-auxiliary	**21.2%**	**5.6%**	4.8

Inception v3

Single Crop

Network	Crops Evaluated	Top-5 Error	Top-1 Error
GoogLeNet [20]	10	-	9.15%
GoogLeNet [20]	144	-	7.89%
VGG [18]	-	24.4%	6.8%
BN-Inception [7]	144	22%	5.82%
PReLU [6]	10	24.27%	7.38%
PReLU [6]	-	21.59%	5.71%
Inception-v3	12	19.47%	4.48%
Inception-v3	144	**18.77%**	**4.2%**

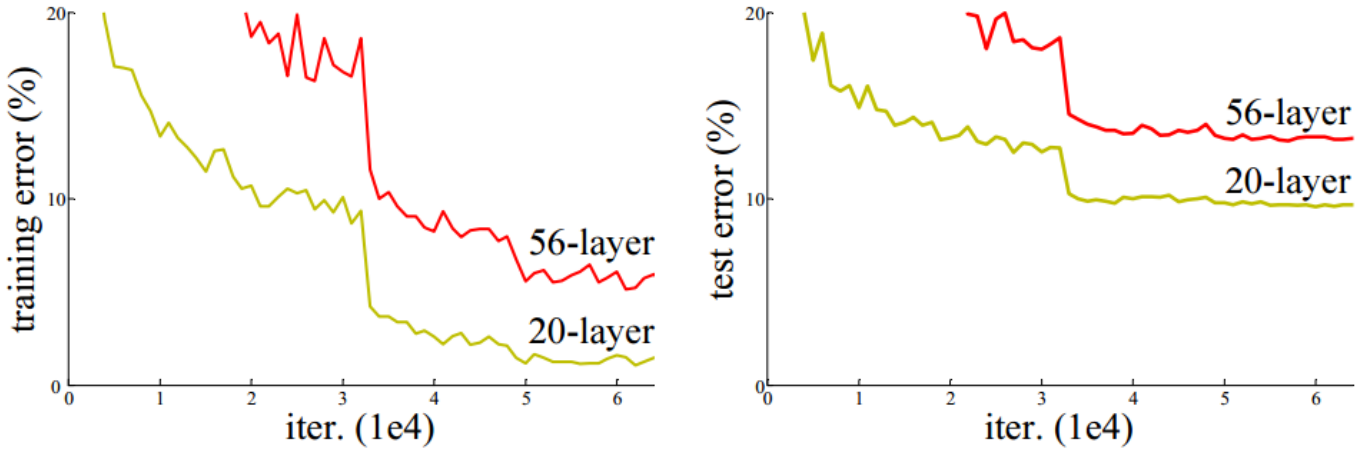
Single Model Multi-crop

Network	Models Evaluated	Crops Evaluated	Top-1 Error	Top-5 Error
VGGNet [18]	2	-	23.7%	6.8%
GoogLeNet [20]	7	144	-	6.67%
PReLU [6]	-	-	-	4.94%
BN-Inception [7]	6	144	20.1%	4.9%
Inception-v3	4	144	**17.2%**	**3.58%***

Ensemble

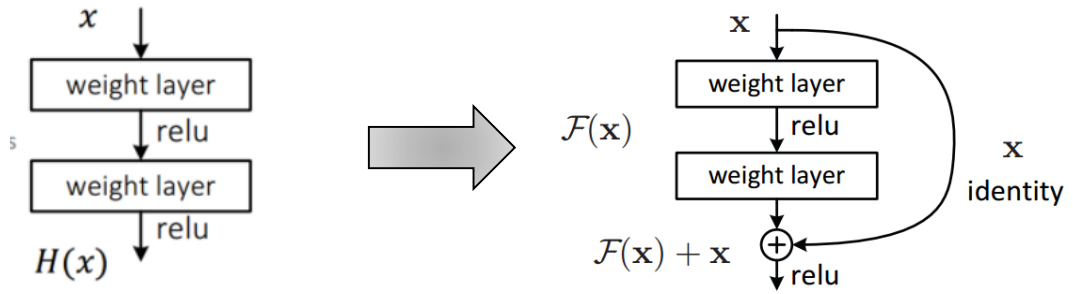
ResNet

- Simply Stacking More Layers is Not the Answer
  - Training and testing error on CIFAR-10 with 20-layer and 56-layer networks.
  - This is not overfitting : both training and test error increases with the depth.
  - By construction, there exists a deeper model with exactly same performance by adding layers with identity mapping.
  - But we cannot find deep network the performance which is comparable to this constructed network



ResNet

- Explicitly Making Layers to Learn Residual Mapping
  - Denoting underlying desired mapping as $\mathcal{H}(\mathbf{x})$, we let the stacked nonlinear layers fit another mapping $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$
  - If the optimal mapping is identity, it would be easier to push residual to zero than learning identity with stack of nonlinear layers.
  - The original formulation $\mathcal{H}(\mathbf{x}) := \mathcal{F}(\mathbf{x}) + \mathbf{x}$ can be achieved with 'shortcut connection' which is just an identity mapping.
  - This identity shortcut connection do not add any parameters or complexity.
  - The identity connection plays role of 'preconditioning' which make the layers easy to approximate the underlying mapping.

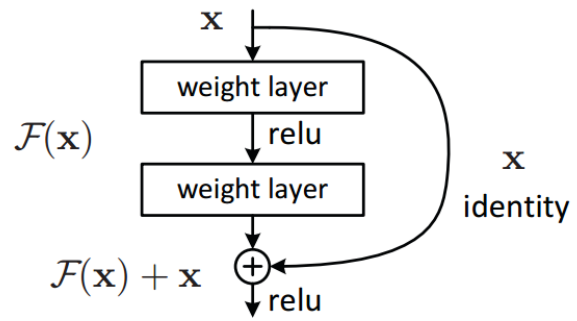


ResNet

- Identity Mapping by Shortcuts
  - Building block is defined by $\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$. where $\mathcal{F}(\mathbf{x}, \{W_i\})$ represent the residual mapping to be learned.
  - For the building block below that has two layers, $\mathcal{F} = W_2\sigma(W_1\mathbf{x})$ where σ is ReLU.
  - We adopt second nonlinearity after element-wise addition $\mathcal{F} + \mathbf{x}$
  - When input-output dimension is not same, we perform linear projection W_s to make the shortcut connections to match the dimensions as

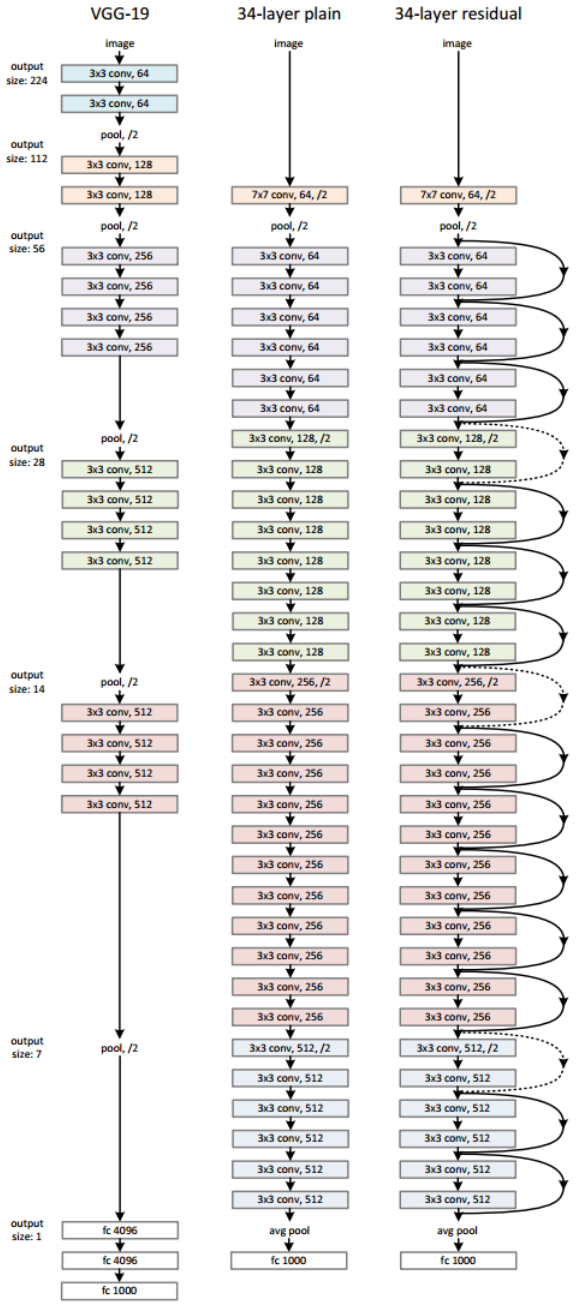
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

  - When the function $\mathcal{F}(\mathbf{x}, \{W_i\})$ is multiple convolutional layers, the element-wise addition is performed channel by channel. Conv layers with depth 2 or 3 are used here.
  - The form of $\mathcal{F}(\mathbf{x}, \{W_i\})$ is flexible but single layer is not beneficial since it leads to $\mathbf{y} = W_1\mathbf{x} + \mathbf{x}$

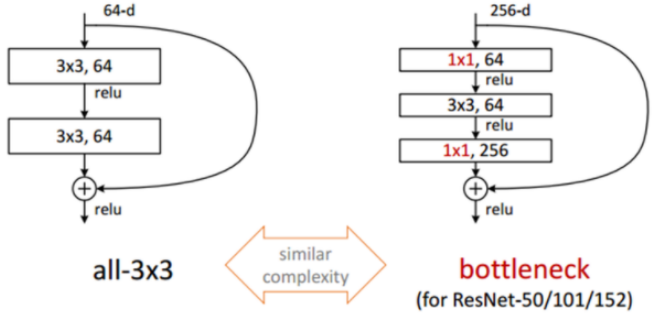


ResNet

- Residual Network
  - Add shortcut connections
    - When input-output dimension matches, add identity connection.
    - When input-output dimension dismatches,
      a)zero-padding, b)add linear projection(1x1 conv.)
- Implementation Detail
  - Training
    - Scale jittering in [256, 480] for shorter side and crop 224x224.
    - Random crop and random flip, perf-pixel mean substraction.
    - Color augmentation and batch normalization, ms-init
    - SGD with mini-batch 256, learning rate starts from 0.1 and decayed by 10 when plateau. Weight decay 0.0001 with momentum 0.9
  - Testing
    - 10-crop testing with fully-convolutional from.
    - Averaging scores from multiple scales(224,256,384,480,640)



ResNet



- ImageNet Performance
  - Comparison with other models
    - Time complexity of 50-layer ResNet : 3.8 billion FLOPs,
      152-layer ResNet : 11.3 billion, VGG-16/19 : 15.3/19.6 billion FLOPs

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

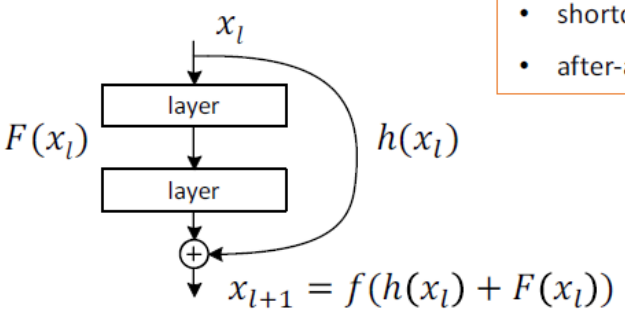
Single model on validation

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

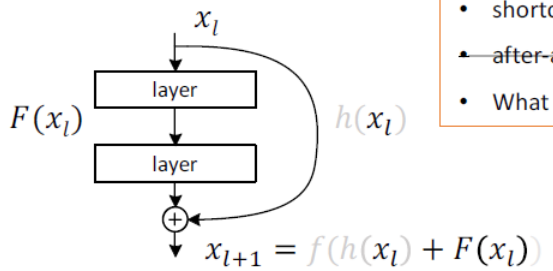
Ensemble on test

ResNet v2

- Identity Mappings in Residual Networks
  - Identity short cut simplifies the whole network



- shortcut mapping: h = identity
- after-add mapping: f = ReLU



- shortcut mapping: h = identity
- ~~after-add mapping: f = ReLU~~
- What if f = identity?

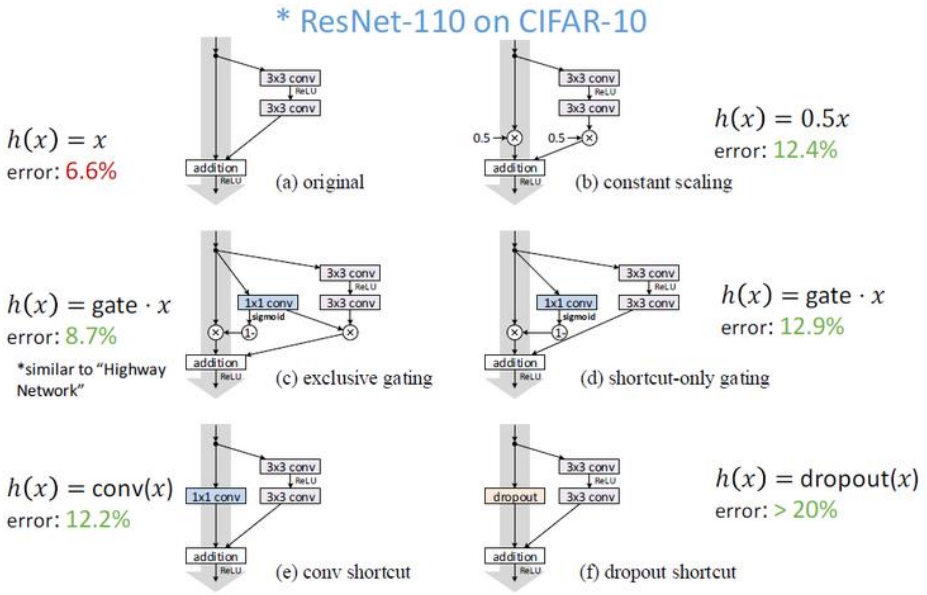
$$x_{l+1} = x_l + F(x_l) \Rightarrow x_{l+2} = x_{l+1} + F(x_{l+1})$$

$$x_{l+2} = x_l + F(x_l) + F(x_{l+1})$$

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$

ResNet v2

- Identity Mappings in Residual Networks
  - Identity shortcut shows best performance

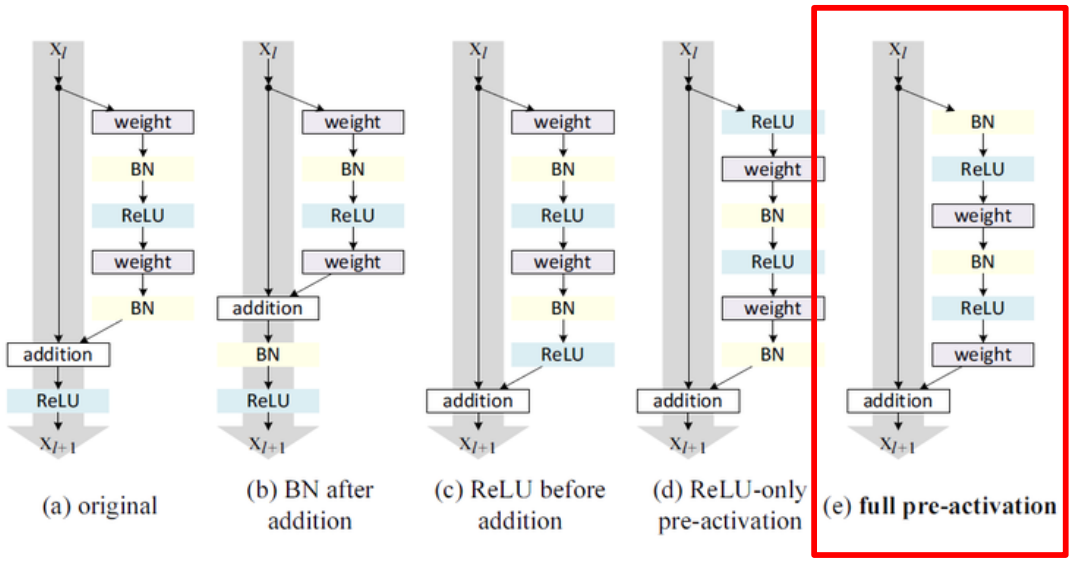


Modifications in shortcut

case	Fig.	on shortcut	on $\mathcal{F}$	error (%)	remark
original [1]	Fig. 2(a)	1	1	**6.61**	
constant scaling	Fig. 2(b)	0	1	fail	This is a plain net
		0.5	1	fail	
		0.5	0.5	12.35	frozen gating
exclusive gating	Fig. 2(c)	$1-g(\mathbf{x})$	$g(\mathbf{x})$	fail	init b_g=0 to −5
		$1-g(\mathbf{x})$	$g(\mathbf{x})$	8.70	init b_g=-6
		$1-g(\mathbf{x})$	$g(\mathbf{x})$	9.81	init b_g=-7
shortcut-only gating	Fig. 2(d)	$1-g(\mathbf{x})$	1	12.86	init b_g=0
		$1-g(\mathbf{x})$	1	6.91	init b_g=-6
1×1 conv shortcut	Fig. 2(e)	1×1 conv	1	12.22	
dropout shortcut	Fig. 2(f)	dropout 0.5	1	fail	

ResNet v2

- Identity Mappings in Residual Networks
  - BN + Pre-activation shows best performance

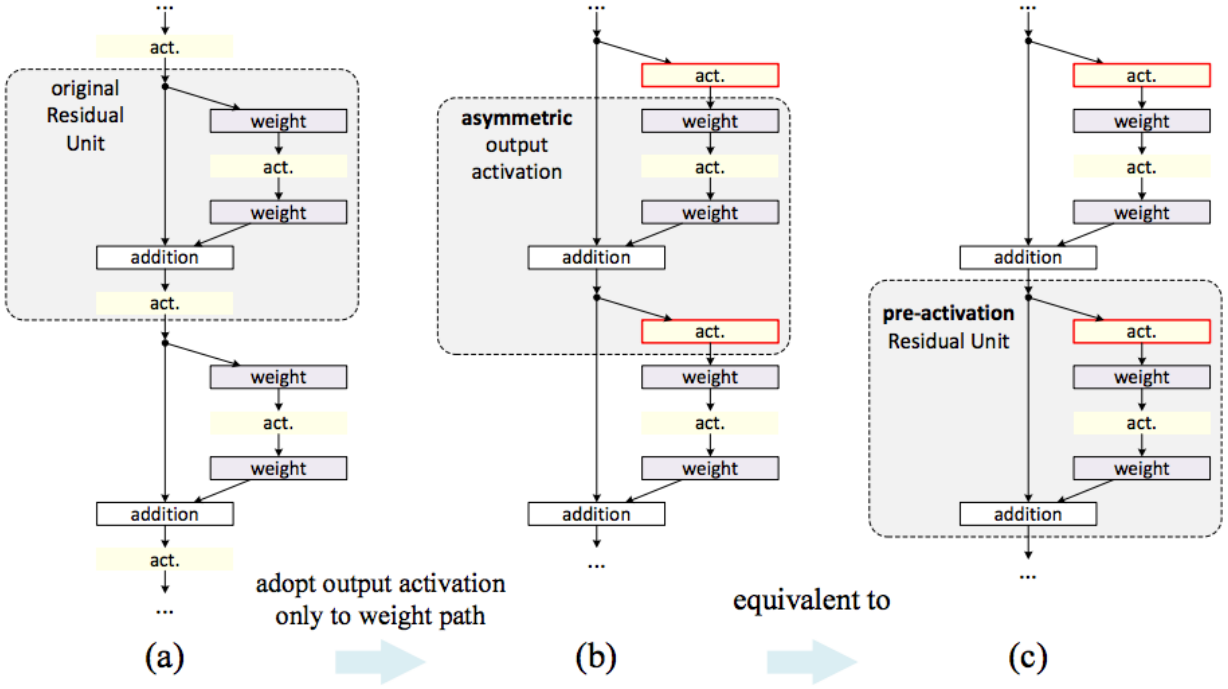


case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**

Modifications in the position of activation function

ResNet v2

- Identity Mappings in Residual Networks
  - Asymmetric after-addition activation = pre-activation residual unit



CIFAR-10	error (%)
NIN [15]	8.81
DSN [16]	8.22
FitNet [17]	8.39
Highway [7]	7.72
All-CNN [14]	7.25
ELU [12]	6.55
FitResNet, LSUV [18]	5.84
ResNet-110 [1] (1.7M)	6.61
ResNet-1202 [1] (19.4M)	7.93
ResNet-164 [ours] (1.7M)	5.46
ResNet-1001 [ours] (10.2M)	4.92 (4.89±0.14)
ResNet-1001 [ours] (10.2M)†	**4.62** (4.69±0.20)

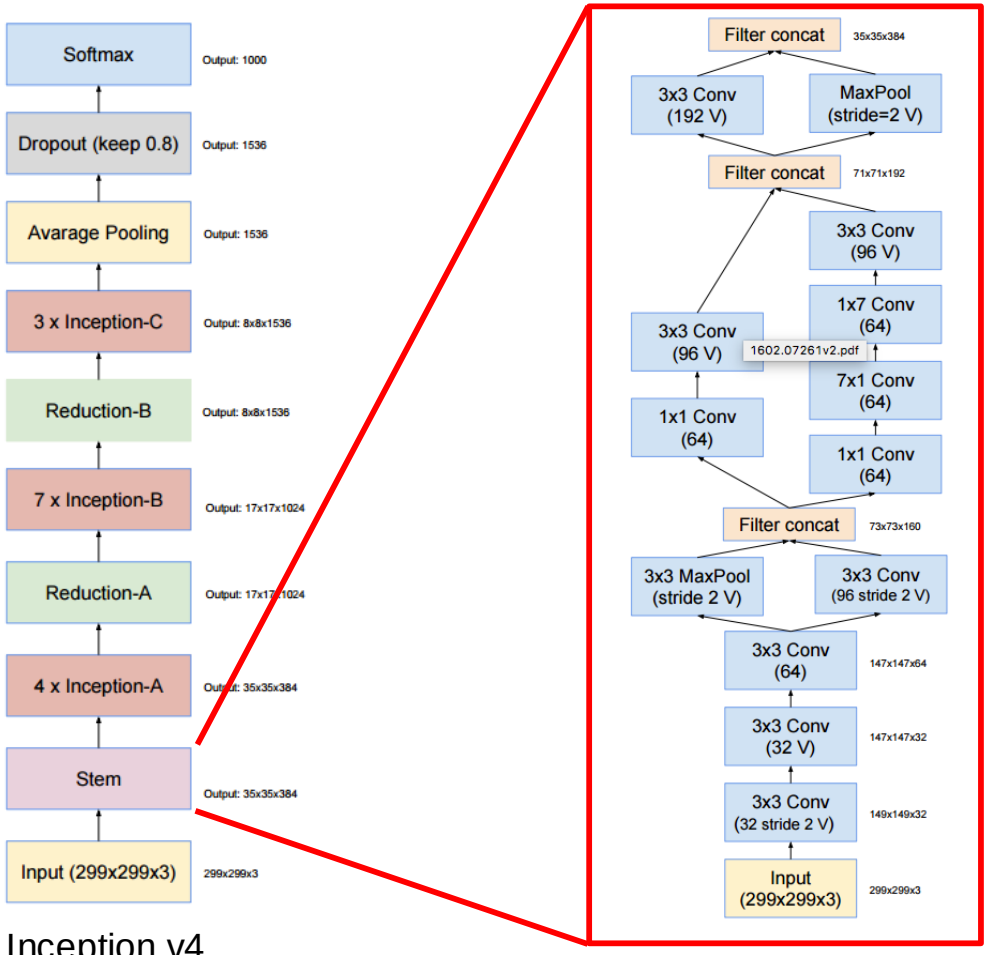
CIFAR-100	error (%)
NIN [15]	35.68
DSN [16]	34.57
FitNet [17]	35.04
Highway [7]	32.39
All-CNN [14]	33.71
ELU [12]	24.28
FitNet, LSUV [18]	27.66
ResNet-164 [1] (1.7M)	25.16
ResNet-1001 [1] (10.2M)	27.82
ResNet-164 [ours] (1.7M)	24.33
ResNet-1001 [ours] (10.2M)	**22.71** (22.68±0.22)

ILSVRC 2012

method	augmentation	train crop	test crop	top-1	top-5
ResNet-152, original Residual Unit [1]	scale	224×224	224×224	23.0	6.7
ResNet-152, original Residual Unit [1]	scale	224×224	320×320	21.3	5.5
ResNet-152, **pre-act** Residual Unit	scale	224×224	320×320	21.1	5.5
ResNet-200, original Residual Unit [1]	scale	224×224	320×320	21.8	6.0
ResNet-200, **pre-act** Residual Unit	scale	224×224	320×320	**20.7**	**5.3**
ResNet-200, **pre-act** Residual Unit	scale+asp ratio	224×224	320×320	**20.1**†	**4.8**†
Inception v3 [19]	scale+asp ratio	299×299	299×299	21.2	5.6

Inception v4 and Inception-ResNet-v2

- Modifying Stem for Better Feature Map Extraction



Inception v4

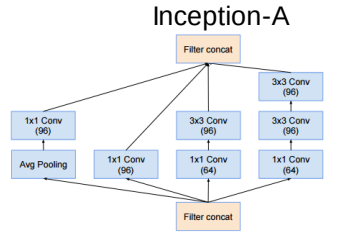


Figure 4. The schema for 35 × 35 grid modules of the pure Inception-v4 network. This is the Inception-A block of Figure 9.

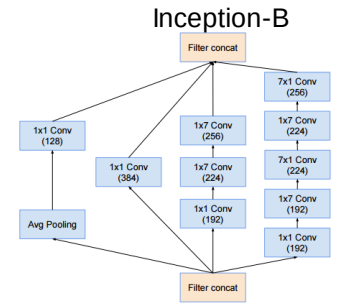


Figure 5. The schema for 17 × 17 grid modules of the pure Inception-v4 network. This is the Inception-B block of Figure 9.

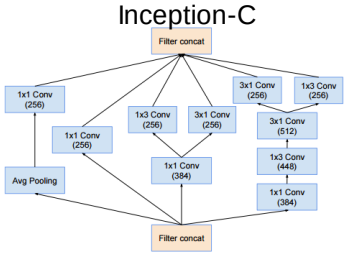


Figure 6. The schema for 8 × 8 grid modules of the pure Inception-v4 network. This is the Inception-C block of Figure 9.

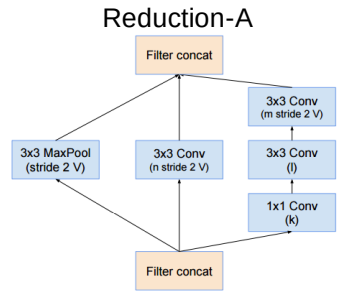


Figure 7. The schema for 35 × 35 to 17 × 17 reduction module. Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The k, l, m, n numbers represent filter bank sizes which can be looked up in Table 1.

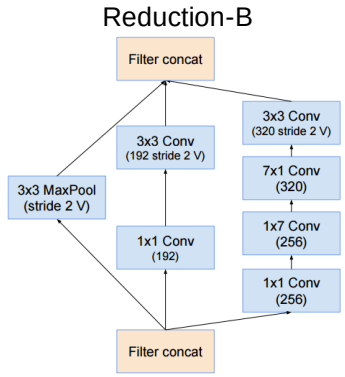
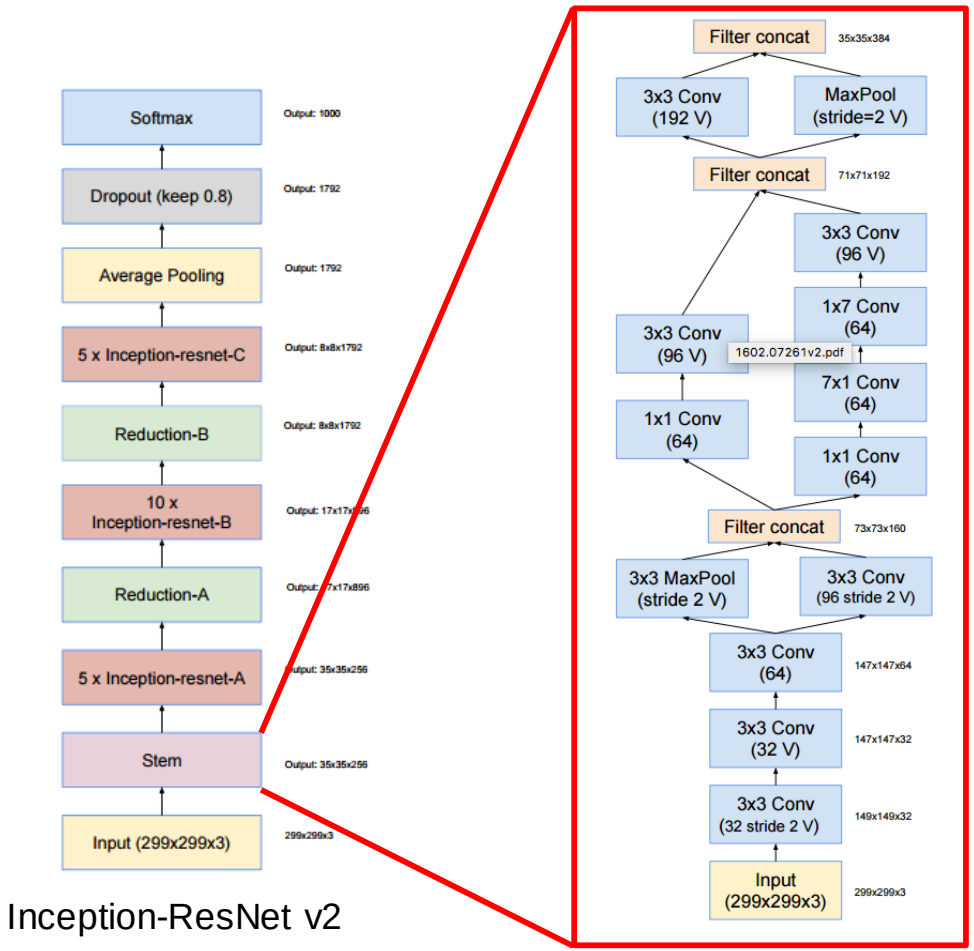


Figure 8. The schema for 17 × 17 to 8 × 8 grid-reduction module. This is the reduction module used by the pure Inception-v4 network in Figure 9.

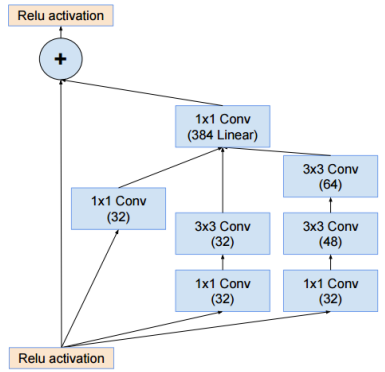
Inception v4 and Inception-ResNet-v2

- Adopting Residual Connection to Inception

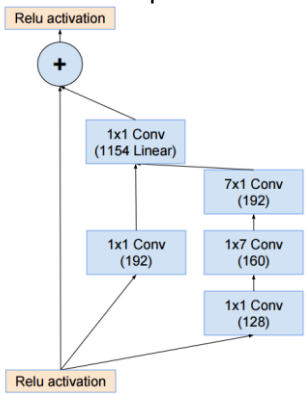


Inception-ResNet v2

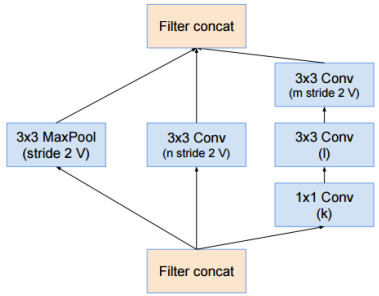
Inception-ResNet-A



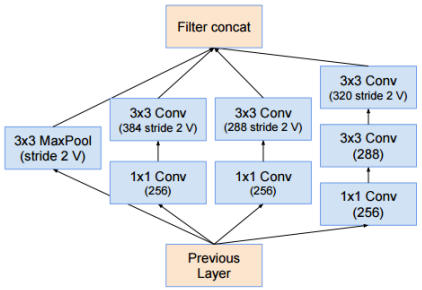
Inception-ResNet-B



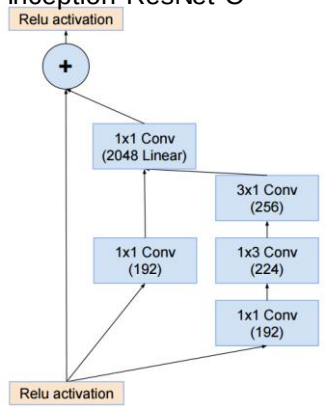
Reduction-A



Reduction-B



Inception-ResNet-C



Inception v4 and Inception-ResNet-v2

- Performance on ILSVRC2012

Network	Top-1 Error	Top-5 Error
BN-Inception [6]	25.2%	7.8%
Inception-v3 [15]	21.2%	5.6%
Inception-ResNet-v1	21.3%	5.5%
Inception-v4	20.0%	5.0%
Inception-ResNet-v2	19.9%	4.9%

Table 2. Single crop - single model experimental results. Reported on the non-blacklisted subset of the validation set of ILSVRC 2012.

Network	Crops	Top-1 Error	Top-5 Error
ResNet-151 [5]	10	21.4%	5.7%
Inception-v3 [15]	12	19.8%	4.6%
Inception-ResNet-v1	12	19.8%	4.6%
Inception-v4	12	18.7%	4.2%
Inception-ResNet-v2	12	18.7%	4.1%

Table 3. 10/12 crops evaluations - single model experimental results. Reported on the all 50000 images of the validation set of ILSVRC 2012.

Network	Crops	Top-1 Error	Top-5 Error
ResNet-151 [5]	dense	19.4%	4.5%
Inception-v3 [15]	144	18.9%	4.3%
Inception-ResNet-v1	144	18.8%	4.3%
Inception-v4	144	17.7%	3.8%
Inception-ResNet-v2	144	17.8%	3.7%

Table 4. 144 crops evaluations - single model experimental results. Reported on the all 50000 images of the validation set of ILSVRC 2012.

Network	Models	Top-1 Error	Top-5 Error
ResNet-151 [5]	6	–	3.6%
Inception-v3 [15]	4	17.3%	3.6%
Inception-v4 + 3× Inception-ResNet-v2	4	16.5%	3.1%

Inception v4 and Inception-ResNet-v2

- ImageNet 2016

Ordered by classification error

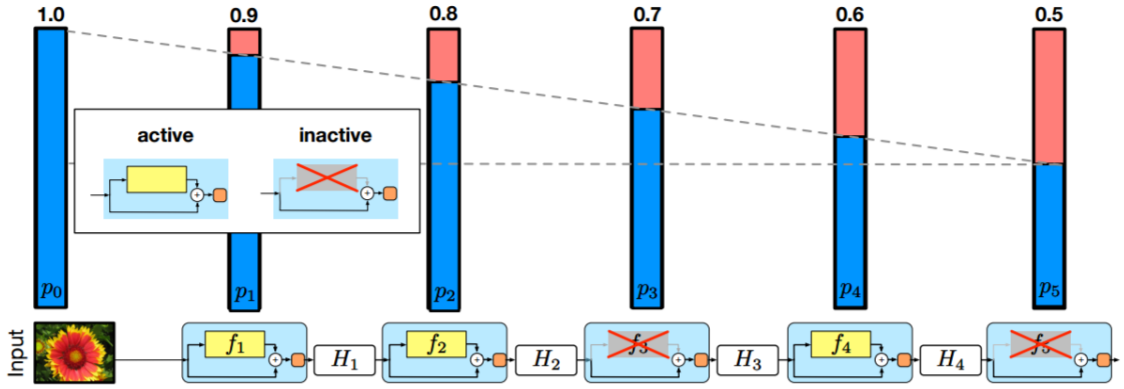
Team name	Entry description	**Classification error**	Localization error
Trimps-Soushen	Ensemble 2	0.02991	0.077668
Trimps-Soushen	Ensemble 3	0.02991	0.077087
Trimps-Soushen	Ensemble 4	0.02991	0.077429
ResNeXt	Ensemble C, weighted average, tuned on val. [No bounding box results]	0.03031	0.737308
CU-DeepLink	GrandUnion + Fused-scale EnsembleNet	0.03042	0.098892
CU-DeepLink	GrandUnion + Multi-scale EnsembleNet	0.03046	0.099006
CU-DeepLink	GrandUnion + Basic Ensemble	0.03049	0.098954
ResNeXt	Ensemble B, weighted average, tuned on val. [No bounding box results]	0.03092	0.737484
CU-DeepLink	GrandUnion + Class-reweighted Ensemble	0.03096	0.099369
CU-DeepLink	GrandUnion + Class-reweighted Ensemble with Per-instance Normalization	0.03103	0.099349
ResNeXt	Ensemble C, weighted average. [No bounding box results]	0.03124	0.737526
Trimps-Soushen	Ensemble 1	0.03144	0.079068
ResNeXt	Ensemble A, simple average. [No bounding box results]	0.0315	0.737505
SamExynos	3 model only for classification	0.03171	0.236561
ResNeXt	Ensemble B, weighted average. [No bounding box results]	0.03203	0.737681
KAISTNIA_ETRI	Ensembles A	0.03256	0.102015
KAISTNIA_ETRI	Ensembles C	0.03256	0.102056
KAISTNIA_ETRI	Ensembles B	0.03256	0.100676

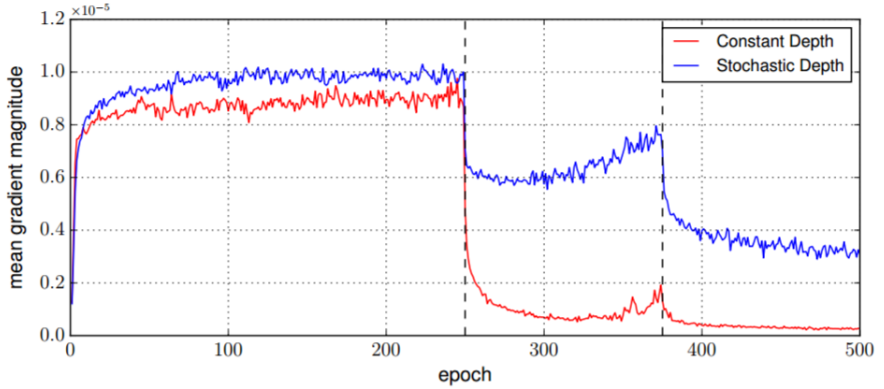
Inception v0-v4

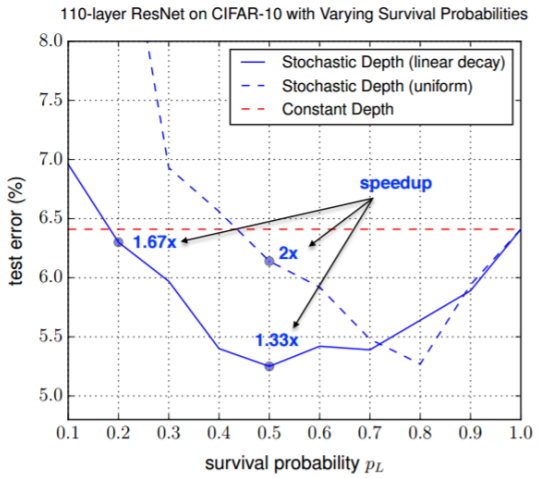
- [v0] Going Deeper with Convolutions, 6.67% test error, http://arxiv.org/abs/1409.4842
- [v1] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 4.8% test error, http://arxiv.org/abs/1502.03167
- [v2, v3] Rethinking the Inception Architecture for Computer Vision, 3.5% test error, http://arxiv.org/abs/1512.00567
- [v4] Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, 3.08% test error, http://arxiv.org/abs/1602.07261

Stochastic Depth

- Training with effectively shallower network, testing with ensemble of variable depth network







Train : $H_\ell = \texttt{ReLU}(b_\ell f_\ell(H_{\ell-1}) + \mathrm{id}(H_{\ell-1})) \quad b_\ell \in \{0, 1\}$

Test : $H_\ell^{\text{Test}} = \texttt{ReLU}(p_\ell f_\ell(H_{\ell-1}^{\text{Test}}; W_\ell) + H_{\ell-1}^{\text{Test}}) \quad p_\ell = 1 - \frac{\ell}{L}(1 - p_L)$

Lab

Lab

- Inception for CIFAR-10
  - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part2/notebook
- ResNet for CIFAR-10
  - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part2/notebook

Class 9

Common Techniques for Training Very Deep CNNs

Weight Initialization

- All zero Initialization
  - Cannot learn : No source of asymmetry(especially for tanh)
- Small Random Number
  - Practically efficient for symmetry breaking.
  - The source of randomness has relatively little impact on the performance
  - Too small initial weight leads to slow learning.
- Controlling the variance of internal data(for linear units)
  - Xavier Initializatior

$$\mathrm{Var}(W) = \frac{1}{n_{\mathrm{in}}}$$

  - Glorot & Bengio

$$\mathrm{Var}(W) = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}}$$

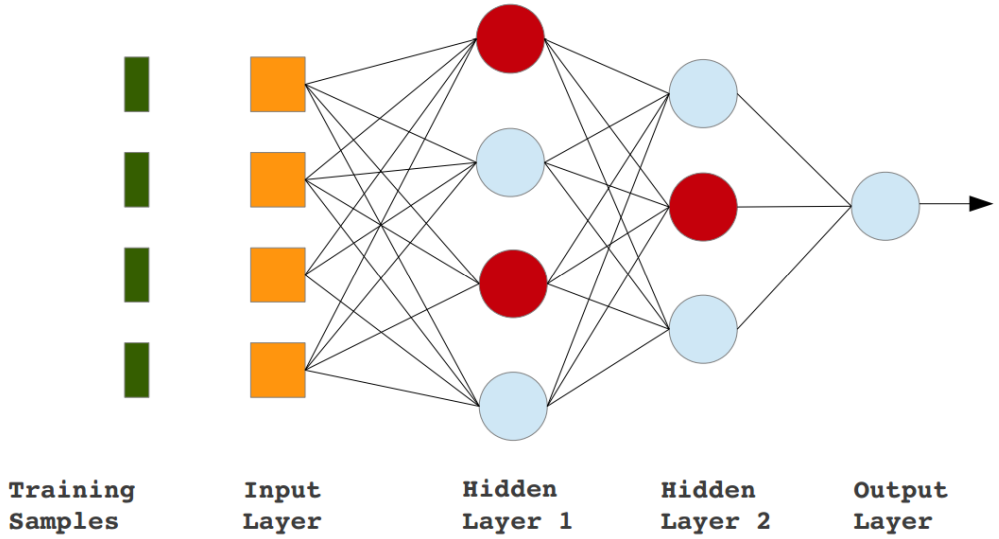
  - ReLU Networks

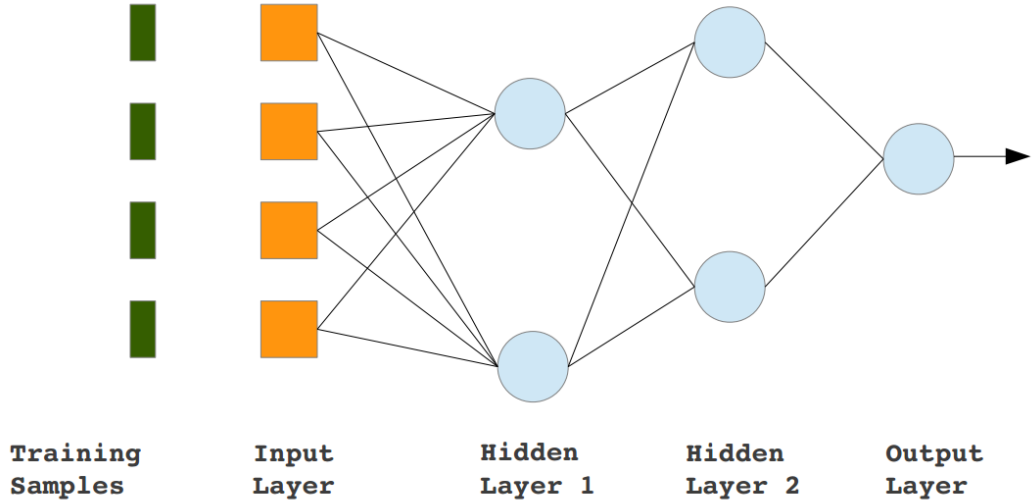
$$\mathrm{Var}(W) = \frac{2}{n_{\mathrm{in}}}$$

- Practical Advice
  - Use ReLU with Initialization of Weight 2/n_in and bias to 0

Dropout

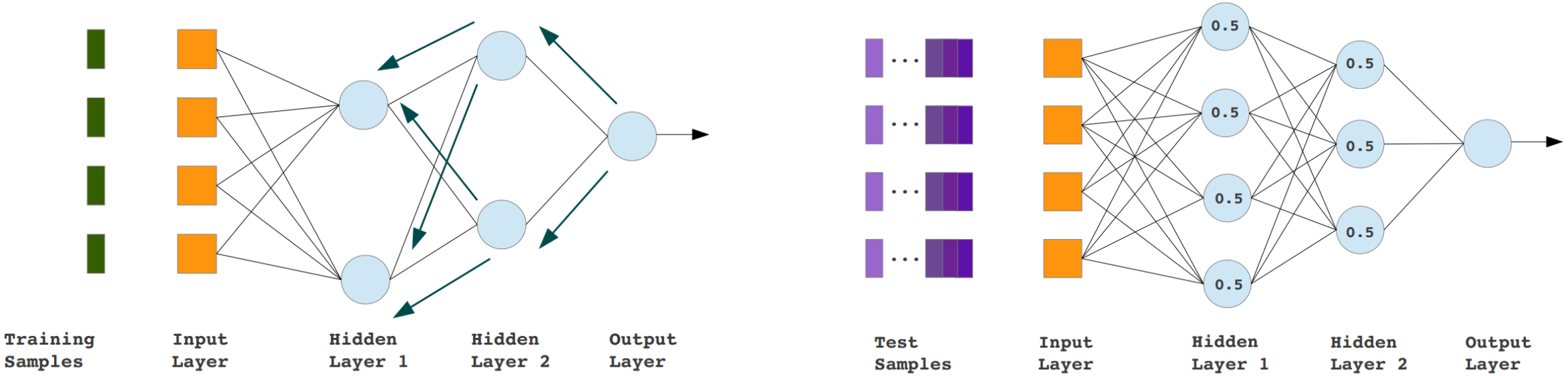
- Training Dropout Network
  - Forward Pass





Dropout

- Training Dropout Network
  - Backward Pass and Test



Dropout

- Implementation

Python Forward propagation

```python
def dropout_forward(x, dropout_param):
  """
  Performs the forward pass for (inverted) dropout.
  Inputs:
  - x: Input data, of any shape
  - dropout_param: A dictionary with the following keys: (p,test/train,seed)
  Outputs: (out, cache)
  """
  # Get the current dropout mode, p, and seed
  p, mode = dropout_param['p'], dropout_param['mode']
  if 'seed' in dropout_param:
    np.random.seed(dropout_param['seed'])

  # Inititalization of outputs and mask
  mask = None
  out = None

  if mode == 'train':
    # Create an apply mask (normally p=0.5 for half of neurons), we scale all
    # by p to avoid having to multiply by p on backpropagation, this is called
    # inverted dropout
    mask = (np.random.rand(*x.shape) < p) / p
    # Apply mask
    out = x * mask
  elif mode == 'test':
    # During prediction no mask is used
    mask = None
    out = x

  # Save mask and dropout parameters for backpropagation
  cache = (dropout_param, mask)

  # Convert "out" type and return output and cache
  out = out.astype(x.dtype, copy=False)
  return out, cache
```

Python Backward propagation

```python
def dropout_backward(dout, cache):
  """
  Perform the backward pass for (inverted) dropout.
  Inputs:
  - dout: Upstream derivatives, of any shape
  - cache: (dropout_param, mask) from dropout_forward.
  """
  # Recover dropout parameters (p, mask , mode) from cache
  dropout_param, mask = cache
  mode = dropout_param['mode']

  dx = None
  # Back propagate (Dropout layer has no parameters just input X)
  if mode == 'train':
    # Just back propagate dout from the neurons that were used during dropout
    dx = dout * mask
  elif mode == 'test':
    # Disable dropout during prediction/test
    dx = dout

  # Return dx
  return dx
```

Batch Normalization

- Batch Normalization
  - Removing 'internal covariate shift' which means the change of distribution of input to each layer due to the change of parameters during training.
  - Facilitates faster and more stable learning

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$
Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

1: $N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma^2_{\mathcal{B}}]$$

11: In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
12: **end for**

Batch Normalization

- Batch Normalization
  - Accelerating Deep Network Training by Reducing Internal Covariate Shift (https://arxiv.org/abs/1502.03167)
  - By using BN
    - We can remove dropout : BN provides similar regularization effect as dropout since the activations observed for a training example are affected by the random selection of examples in the same mini-batch. We need to shuffle training data more thoroughly.
    - Reduce L2 weight regularization
    - Removed local response normalization
    - Increase learning rate and accelerate learning rate decay
    - Reduce the photometric distortion : Focus more on 'real' data set.
  - Inception v1
    - Same with GoogLeNet except that 5x5 convolution layers are replaced with two consecutive 3x3 layers.

Batch Normalization

- Performance

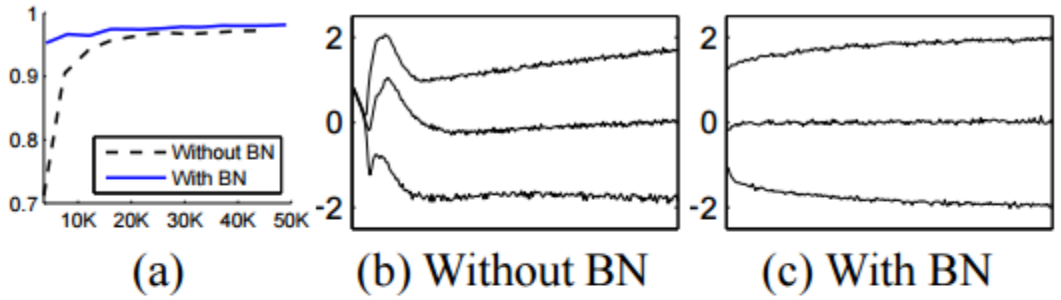
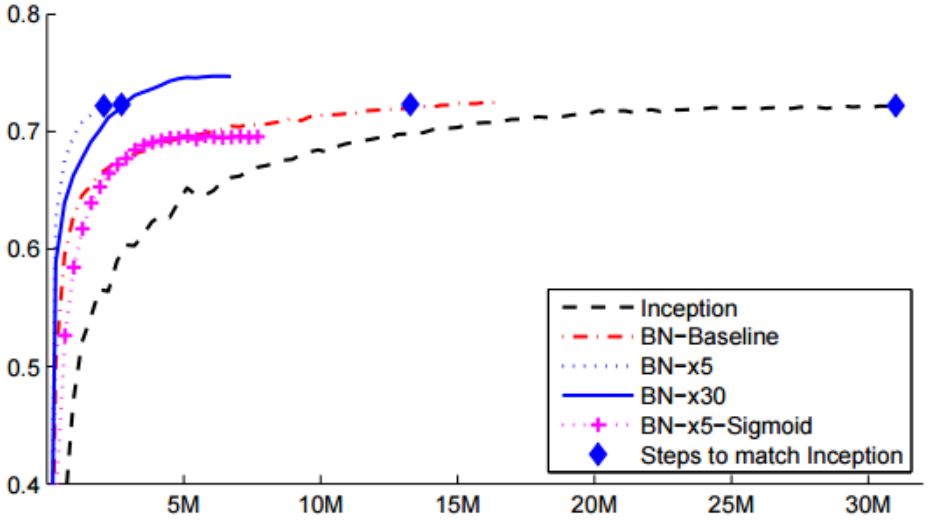


Figure 1: (a) *The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy.* (b, c) *The evolution of input distributions to a typical sigmoid, over the course of training, shown as* $\{15, 50, 85\}$*th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.*



Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Batch Normalization

- Implementation

Python Forward Propagation

```python
def batchnorm_forward(x, gamma, beta, bn_param):
  """
  Forward pass for batch normalization (Use on FC layers).
  Input:
  - x: Data of shape (N, D)
  - gamma,beta: Scale/Shift parameter of shape (D,)
  - bn_param: Dictionary with the following keys: (mode,eps,momentum,r_mean/var)
  Returns a tuple of: (out, cache)
  """
  mode = bn_param['mode']
  eps = bn_param.get('eps', 1e-5)
  momentum = bn_param.get('momentum', 0.9)

  N, D = x.shape
  running_mean = bn_param.get('running_mean', np.zeros(D, dtype=x.dtype))
  running_var = bn_param.get('running_var', np.zeros(D, dtype=x.dtype))

  out, cache = None, None
  if mode == 'train':
    # Forward pass
    # Step 1: Calculate mean
    mu = 1 / float(N) * np.sum(x, axis=0)
    # Step 2: Subtract the mean of every training sample
    xmu = x - mu
    # Step 3 - Calculate denominator
    carre = xmu**2
    # Step 4 - Calculate variance
    var = 1 / float(N) * np.sum(carre, axis=0)
    # Step 5 - Add eps for numerical stability then get square root
    sqrtvar = np.sqrt(var + eps)
    # Step 6 - Invert square root
    invvar = 1. / sqrtvar
    # Step 7 - Calculate normalization
    va2 = xmu * invvar
    # Step 8 - Calculate
    va3 = gamma * va2
    # Step 9 - Shape out (N,D)
    out = va3 + beta

    # Calculate running mean and variance to be used on prediction
    running_mean = momentum * running_mean + (1.0 - momentum) * mu
    running_var = momentum * running_var + (1.0 - momentum) * var
    # Store values
    cache = (mu, xmu, carre, var, sqrtvar, invvar,
             va2, va3, gamma, beta, x, bn_param)
  elif mode == 'test':
    # On prediction get the running mean/variance
    running_mean =  bn_param['running_mean']
    running_var  =  bn_param['running_var']
    xbar = (x - running_mean)/np.sqrt(running_var+eps)
    out = gamma*xbar + beta
    cache = (x, xbar, gamma, beta, eps)
  else:
    raise ValueError('Invalid forward batchnorm mode "%s"' % mode)
  # Save updated running mean/variance
  bn_param['running_mean'] = running_mean
  bn_param['running_var'] = running_var
  # Return outputs
  return out, cache
```

Python Backward Propagation

```python
def batchnorm_backward(dout, cache):
  """
  Backward pass for batch normalization (Use on FC layers).
  Use computation graph to guide the backward propagation!
  Inputs:
  - dout: Upstream derivatives, of shape (N, D)
  - cache: Variable of intermediates from batchnorm_forward.

  Returns a tuple of: (dx(N,D), dgamma(D), dbeta(D))
  """
  dx, dgamma, dbeta = None, None, None

  # http://cthorey.github.io./backpropagation/
  mu, xmu, carre, var, sqrtvar, invvar, va2, va3, gamma, beta, x, bn_param = cache
  eps = bn_param.get('eps', 1e-5)
  N, D = dout.shape

  # Backprop Step 9
  dva3 = dout
  dbeta = np.sum(dout, axis=0)
  # Backprop step 8
  dva2 = gamma * dva3
  dgamma = np.sum(va2 * dva3, axis=0)
  # Backprop step 7
  dxmu = invvar * dva2
  dinvvar = np.sum(xmu * dva2, axis=0)
  # Backprop step 6
  dsqrtvar = -1. / (sqrtvar**2) * dinvvar
  # Backprop step 5
  dvar = 0.5 * (var + eps)**(-0.5) * dsqrtvar
  # Backprop step 4
  dcarre = 1 / float(N) * np.ones((carre.shape)) * dvar
  # Backprop step 3
  dxmu += 2 * xmu * dcarre
  # Backprop step 2
  dx = dxmu
  dmu = - np.sum(dxmu, axis=0)
  # Basckprop step 1
  dx += 1 / float(N) * np.ones((dxmu.shape)) * dmu

  return dx, dgamma, dbeta
```

Lab

Lab

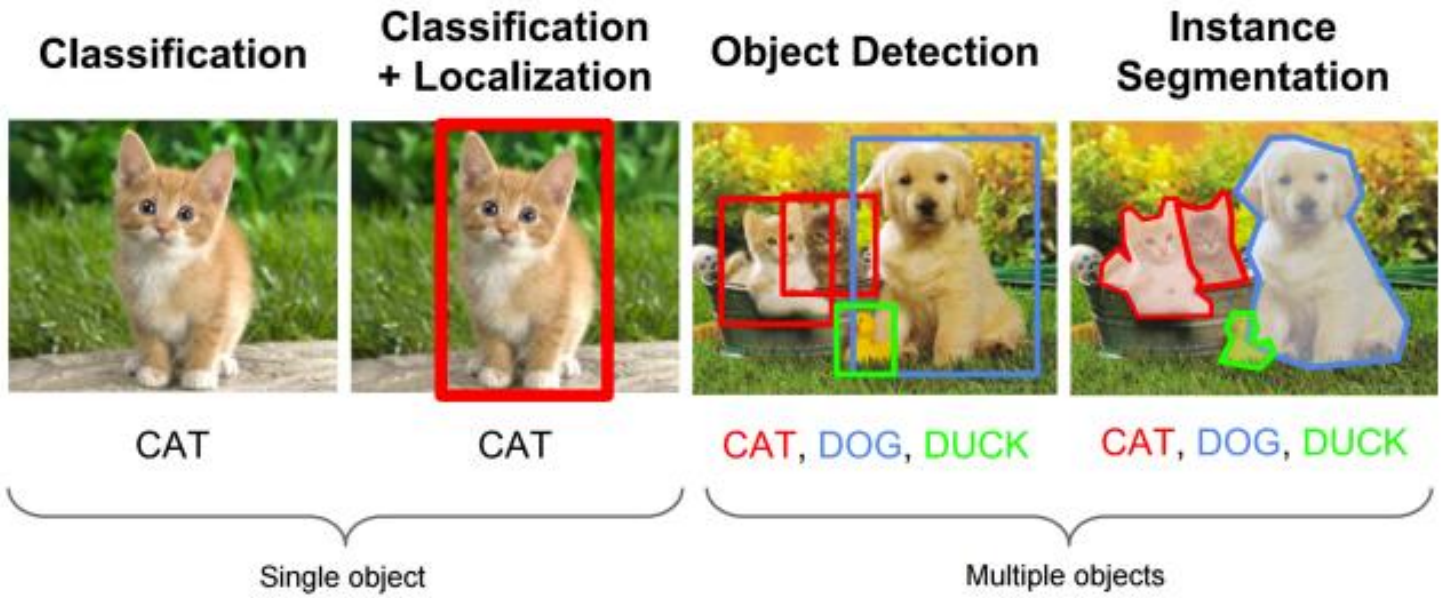
- Weight Initialization Example
  - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part2/notebook
- Dropout Example
  - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part2/notebook
- Batch Normalization Exmaple
  - https://github.com/KyuhwanJung/tensorflow_tutorial/tree/master/part2/notebook

Class 10

State of the Art CNNs for Object Localization and Detection

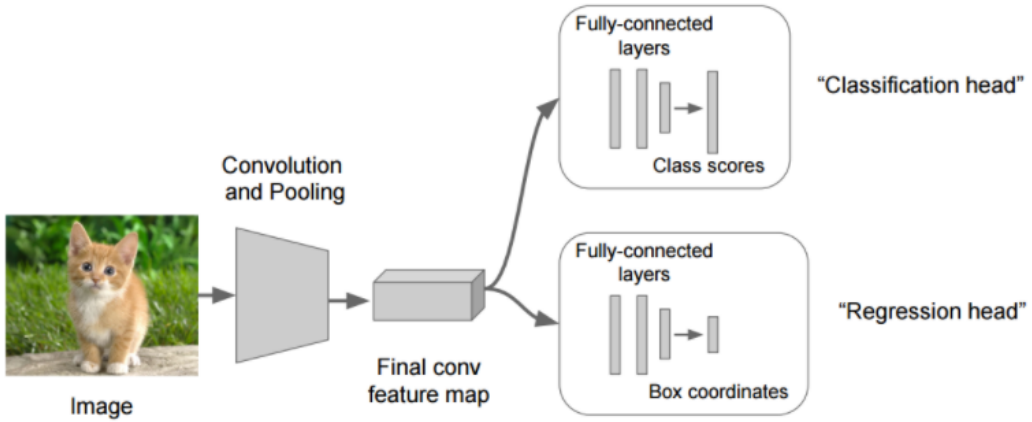
Overview of Localization and Detection

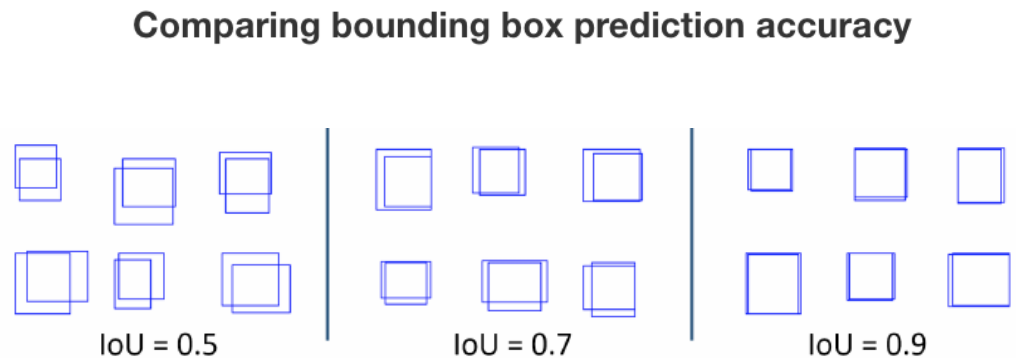
- Various Tasks in Visual Recognition



Overview of Localization and Detection

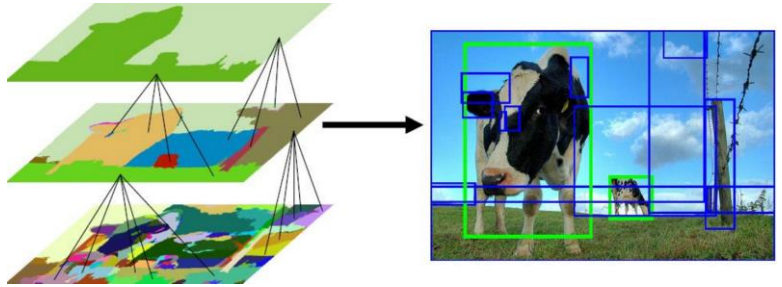
- Object Localization

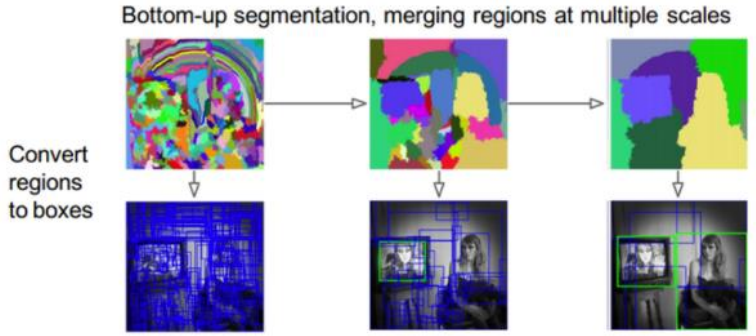




R-CNN

- Multi-phase Architecture for Object Detection
  - 1)Generate category-independent region proposals using selective search
  - 2)Extract a fixed length feature vector from CNN
  - 3)Classify region using class-specific linear SVM





Selective Search

Algorithm 1: Hierarchical Grouping Algorithm

Input: (colour) image
Output: Set of object location hypotheses L

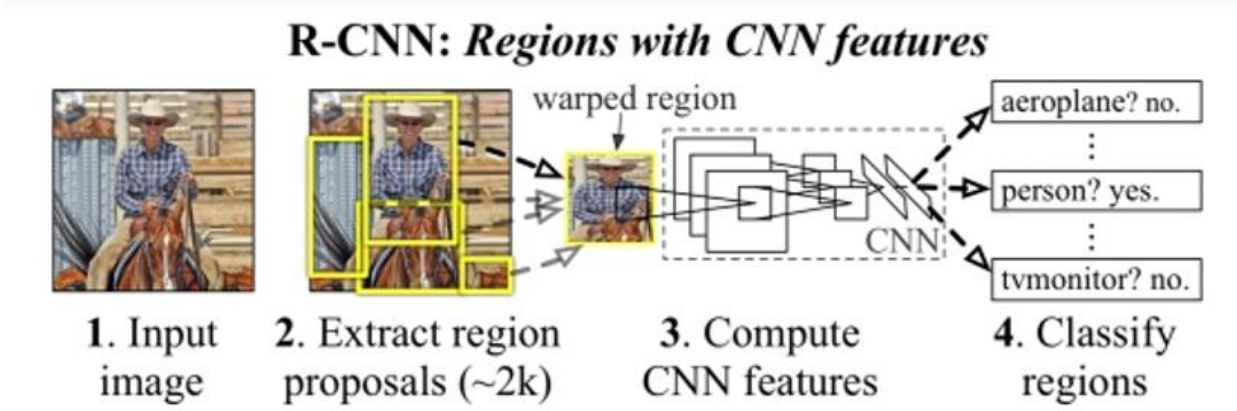
Obtain initial regions $R = \{r_1, \cdots, r_n\}$ using [13]
Initialise similarity set $S = \emptyset$
foreach *Neighbouring region pair* (r_i, r_j) **do**
  Calculate similarity $s(r_i, r_j)$
  $S = S \cup s(r_i, r_j)$

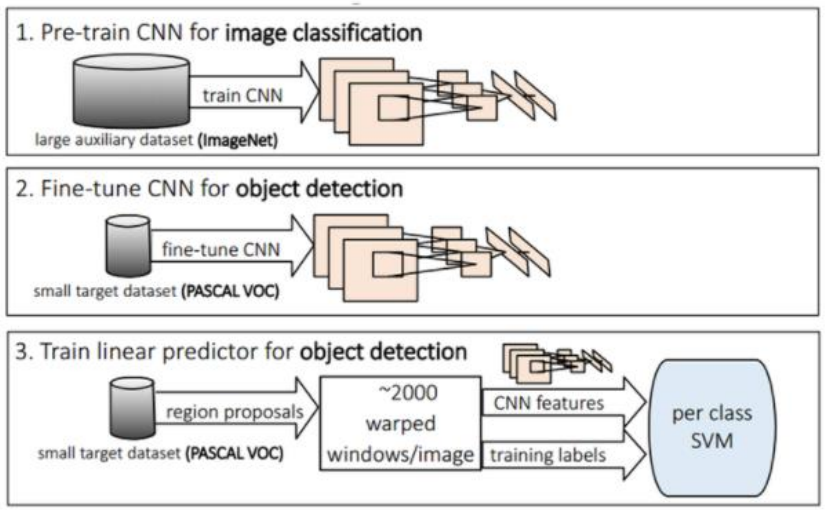
while $S \neq \emptyset$ **do**
  Get highest similarity $s(r_i, r_j) = \max(S)$
  Merge corresponding regions $r_t = r_i \cup r_j$
  Remove similarities regarding $r_i : S = S \setminus s(r_i, r_*)$
  Remove similarities regarding $r_j : S = S \setminus s(r_*, r_j)$
  Calculate similarity set S_t between r_t and its neighbours
  $S = S \cup S_t$
  $R = R \cup r_t$

Extract object location boxes L from all regions in R

R-CNN

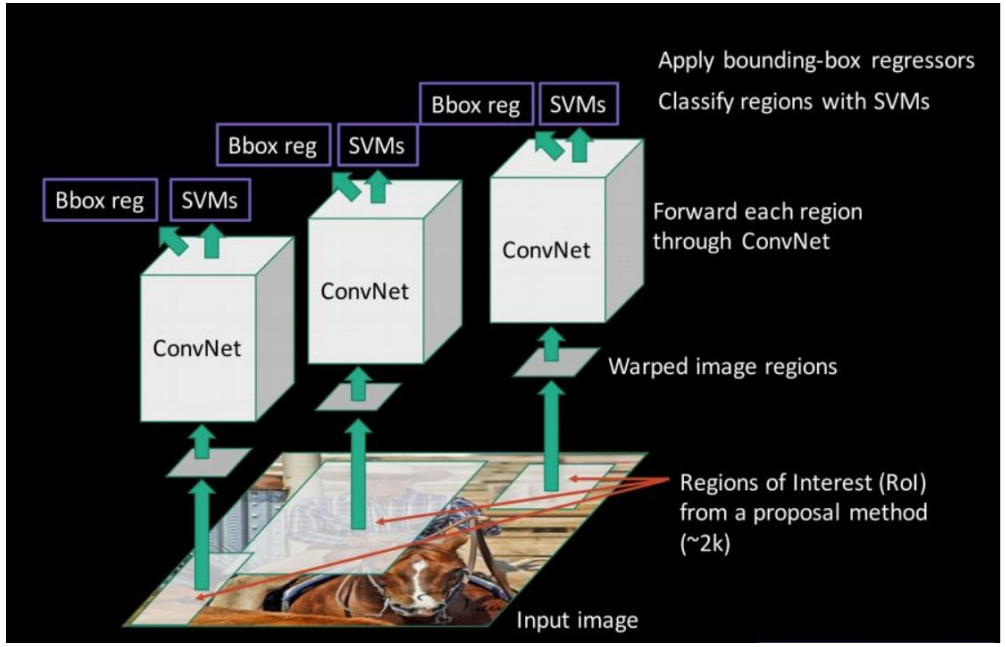
- Multi-phase Architecture for Object Detection
  - 1)Generate category-independent region proposals using selective search
  - 2)Extract a fixed length feature vector from CNN
  - 3)Classify region using class-specific linear SVM





R-CNN

- Multi-phase Architecture for Object Detection
  - Multi-phase
  - Too Slow(47 seconds for 1 image)
  - Need large space to save features for ROI



Bounding Box Regression

$$\mathbf{w}_\star = \underset{\hat{\mathbf{w}}_\star}{\operatorname{argmin}} \sum_i^N (t_\star^i - \hat{\mathbf{w}}_\star^{\mathrm{T}} \boldsymbol{\phi}_5(P^i))^2 + \lambda \|\hat{\mathbf{w}}_\star\|^2 .$$

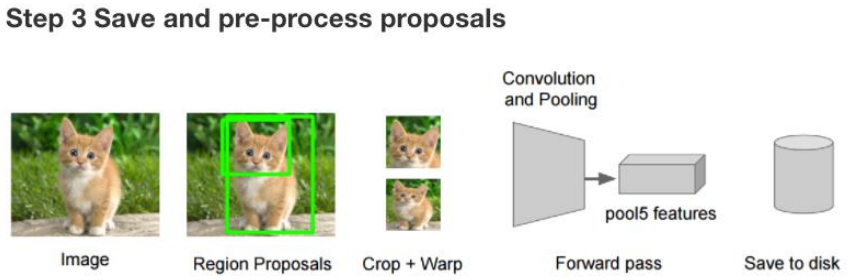
The regression targets $t_\star$ for the training pair (P, G) are defined as

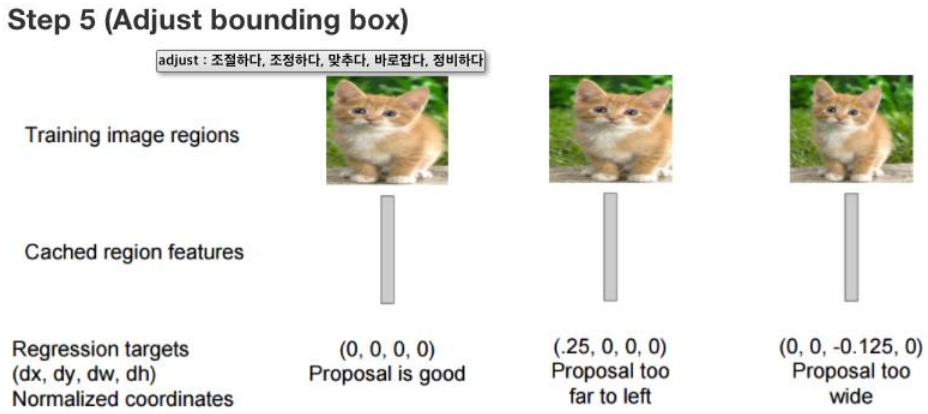
$$\begin{aligned} t_x &= (G_x - P_x)/P_w \\ t_y &= (G_y - P_y)/P_h \\ t_w &= \log(G_w/P_w) \\ t_h &= \log(G_h/P_h). \end{aligned}$$

Learn the transformation of proposal box P to the ground truth G

R-CNN

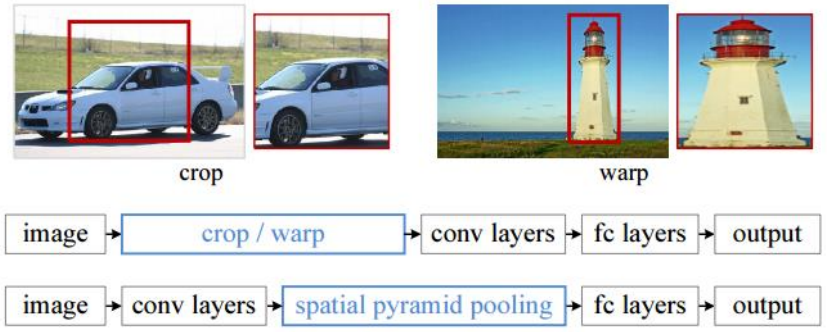
- Summary
  1) Take a pre-trained imagenet cnn (ex Alexnet)
  2) Re-train the last fully connected layer with the objects that need to be detected + "no-object" class
  3) Get all proposals(=~2000 p/image), resize them to match the cnn input, then save to disk.
  4) Train SVM to classify between object and background (One binary SVM for each class)
  5) BB Regression: Train a linear regression classifier that will output some correction factor

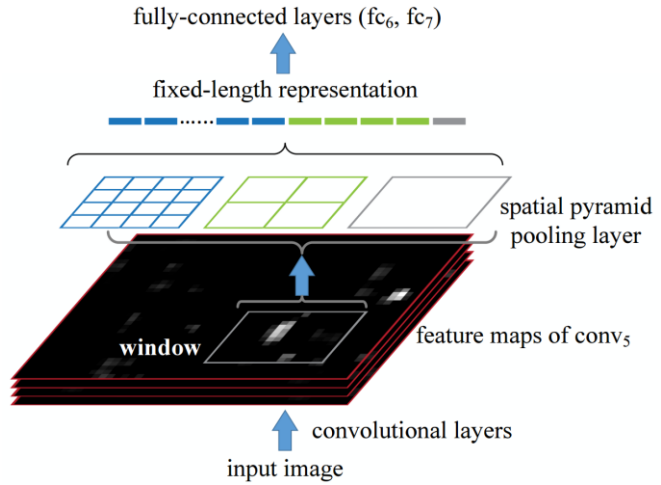




SPP-net

- Extraction of Features of From Candidate Window
  - R-CNN warps or crops object to fit in 224x224 input size.
  - R-CNN should apply convent for every candidate window(x2000) for feature extraction.
  - Apply convent once and extract fixed representation from the feature map using spatial pyramid pooling.
  - Still multi-phase architecture (Input-Feature Extractor – Linear SVM + BB Regressor)



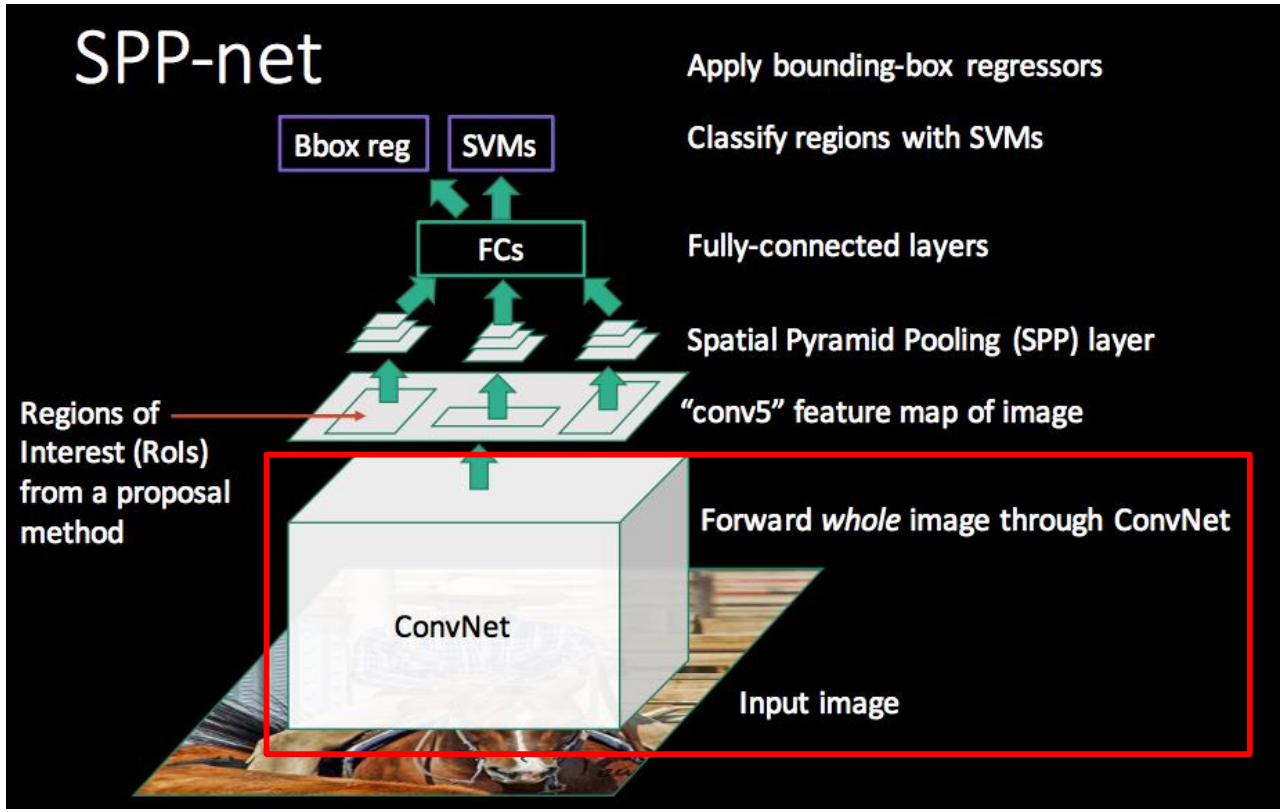


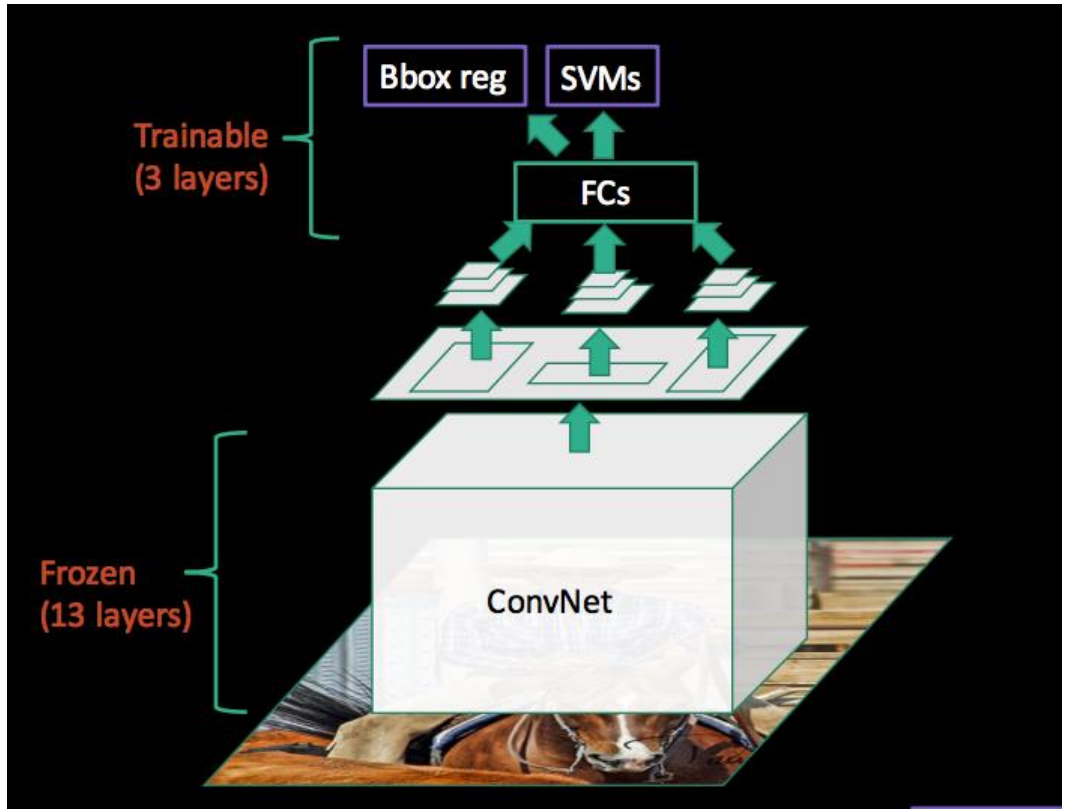
	SPP (1-sc) (ZF-5)	SPP (5-sc) (ZF-5)	R-CNN (ZF-5)
$ftfc_7$	54.5	55.2	55.1
$ftfc_7$ bb	58.0	**59.2**	**59.2**
conv time (GPU)	0.053s	0.293s	14.37s
fc time (GPU)	0.089s	0.089s	0.089s
total time (GPU)	0.142s	0.382s	14.46s
speedup (*vs.* RCNN)	**102**×	**38**×	-

Table 10: Detection results (mAP) on Pascal VOC 2007, **using the same pre-trained model** of SPP (ZF-5).

SPP-net

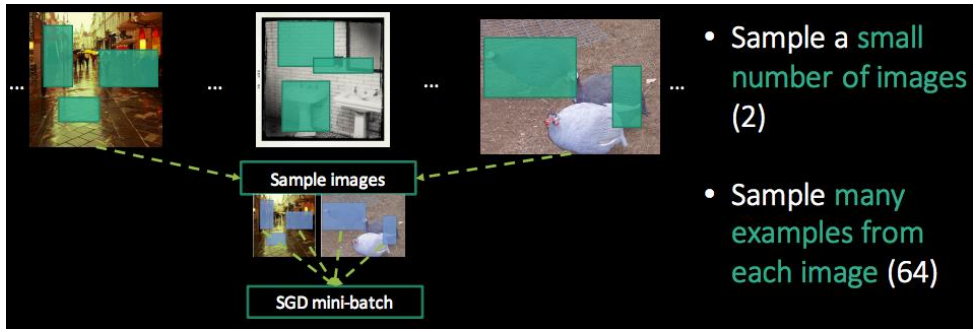
- The Good and Bad of SPP-net
  - Faster than R-CNN by sharing feature map extraction
  - Ad hoc training, slow training and do not update parameters below SPP layer.

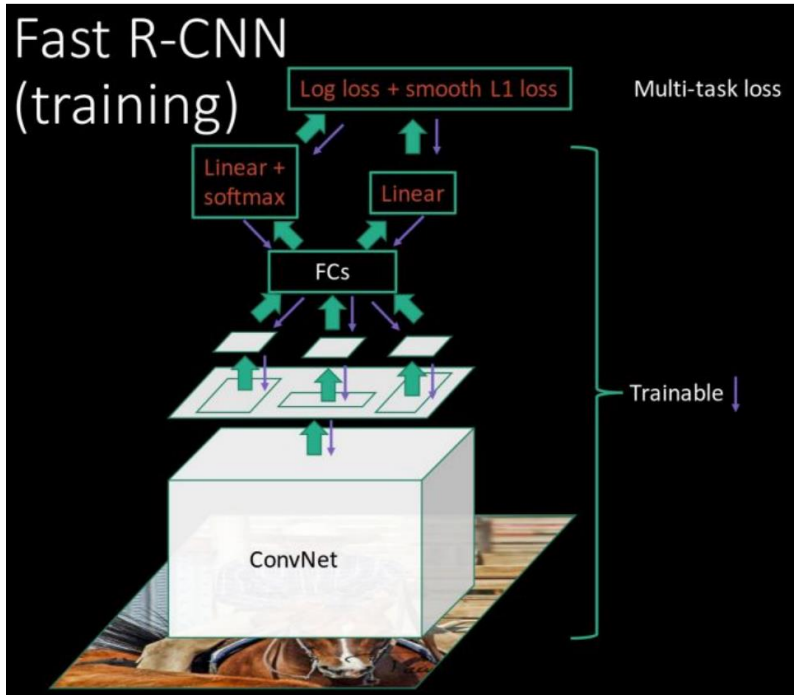


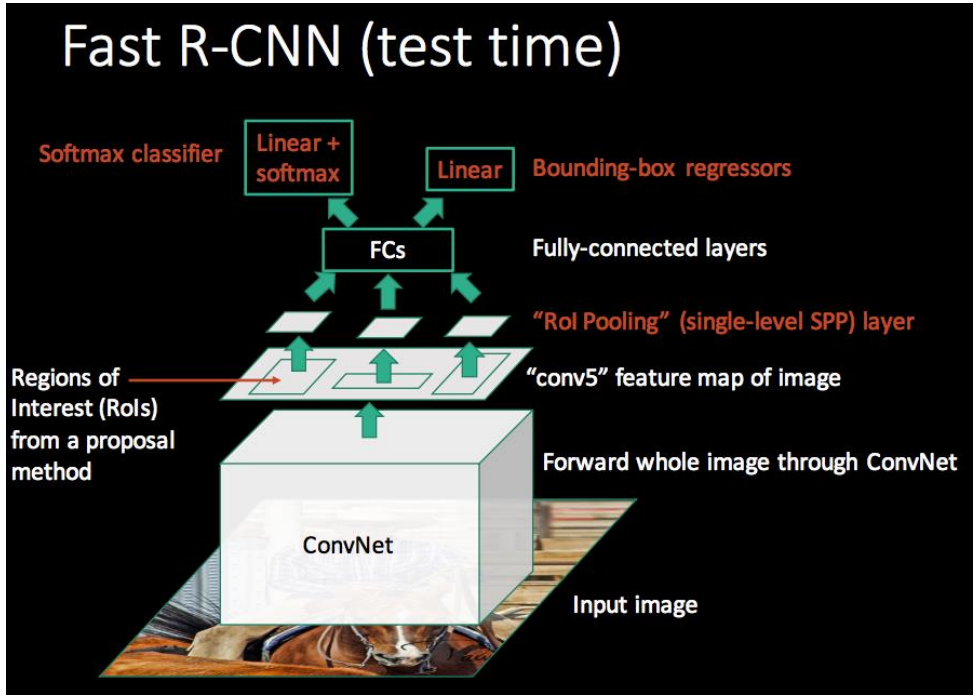


Fast R-CNN

- Using Shared Feature Maps
  - One network, trained in one stage.
    - Softmax vs SVM
    - Fast training by hierarchical sampling.

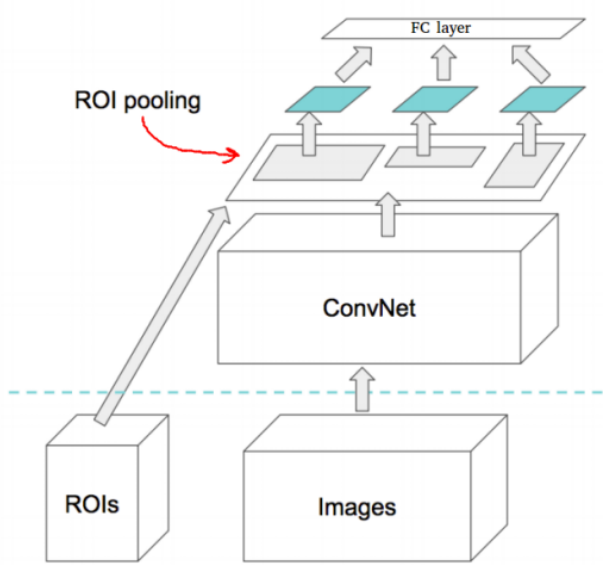


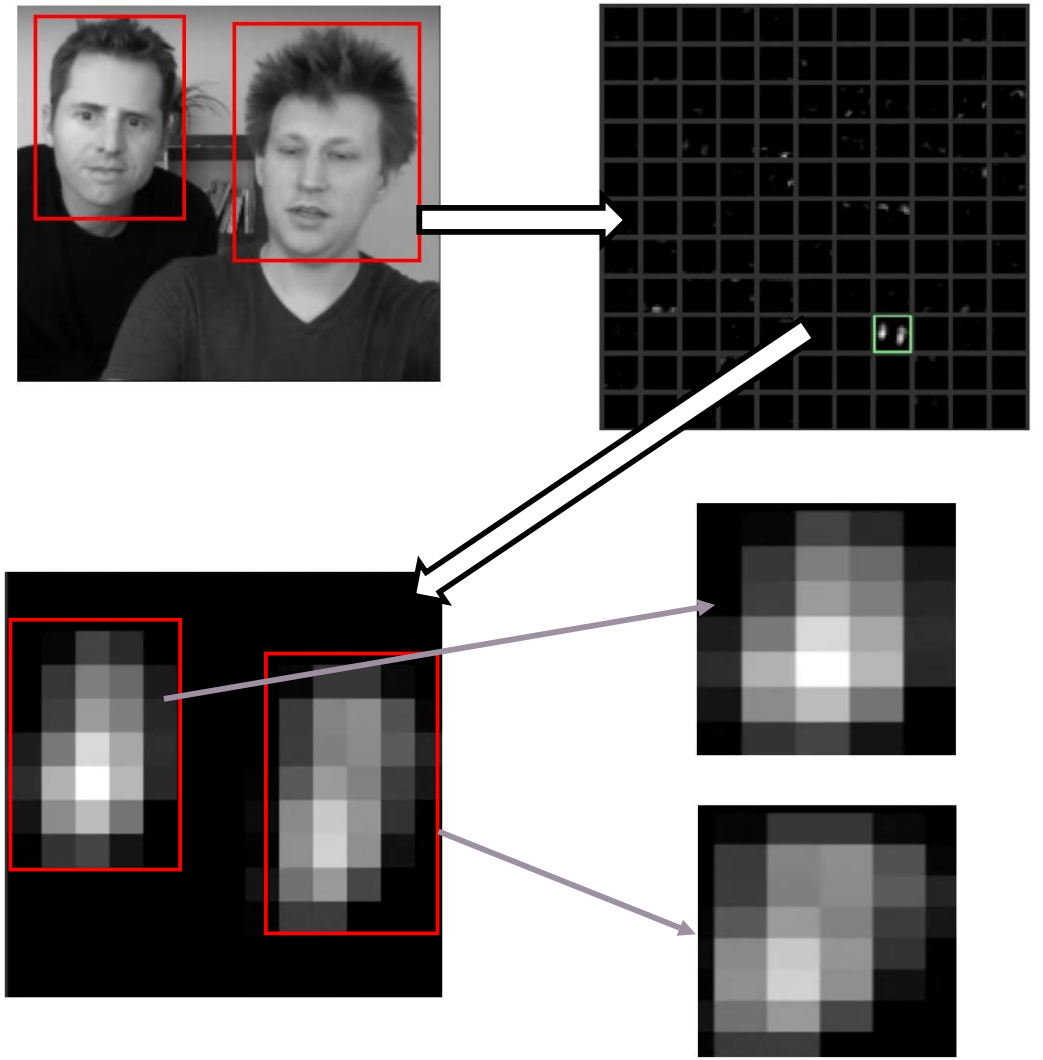




Fast R-CNN

- ROI Pooling
  - Max pooling layer

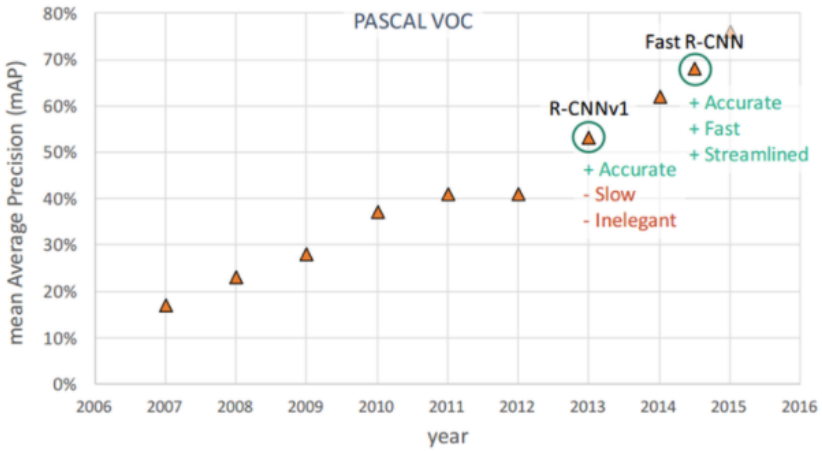




Fast R-CNN

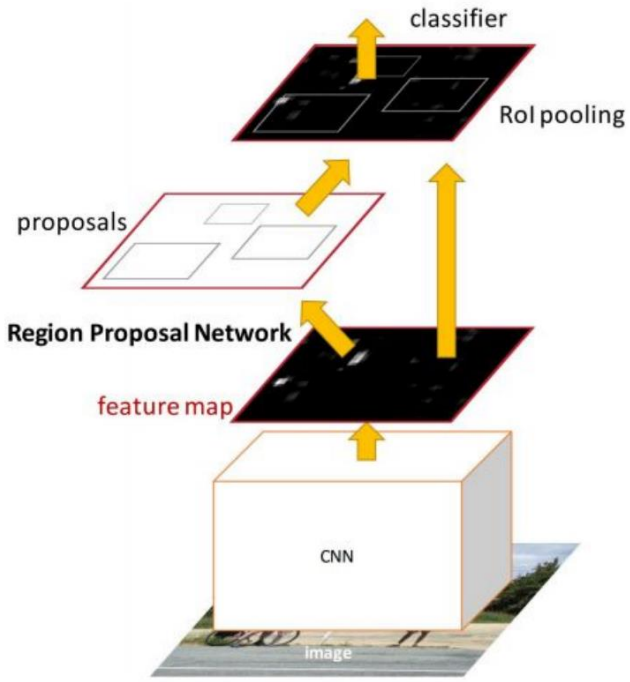
- Performance
  - Fast, more accurate

	Fast R-CNN	R-CNN [1]	SPP-net [2]
Train time (h)	**9.5**	84	25
- Speedup	**8.8x**	1x	3.4x
Test time / image	**0.32s**	47.0s	2.3s
Test speedup	**146x**	1x	20x
mAP	**66.9%**	66.0%	63.1%



Faster R-CNN

- Toward Real-time Object Detection
  - No dependency on external region proposal modules
  - Fast R-CNN + Region Proposal Network




```
# Let M0 be an ImageNet pre-trained network
1. train_rpn(M0) → M1              # Train an RPN initialized from M0, get M1
2. generate_proposals(M1) → P1     # Generate training proposals P1 using RPN M1
3. train_fast_rcnn(M0, P1) → M2    # Train Fast R-CNN M2 on P1 initialized from M0
4. train_rpn_frozen_conv(M2) → M3  # Train RPN M3 from M2 without changing conv layers
5. generate_proposals(M3) → P2
6. train_fast_rcnn_frozen_conv(M3, P2) → M4  # Conv layers are shared with RPN M3
7. return add_rpn_layers(M4, M3.RPN)         # Add M3's RPN layers to Fast R-CNN M4
```

Faster R-CNN

- Region Proposal Network
  - Fully convolutional neural network which shares computation with Fast R-CNN object detection network
  - For *n* x *n* sliding windows on the last shared convolutional feature map (n=3 in the paper), apply *n* x *n* convolution to get lower dimensional(256 for ZF, 512 for VGG) feature.
  - We extract k 'anchor' boxes with variable aspect ratio and scale.
  - We estimate 2k(object or not) and 4k(bounding box coordinate relative to anchors) outputs for k(typically 9) anchors.
  - This is transition invariant.

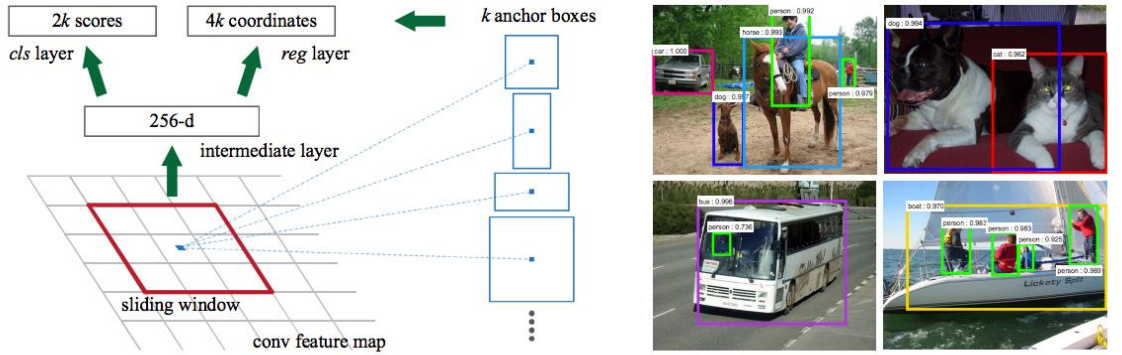
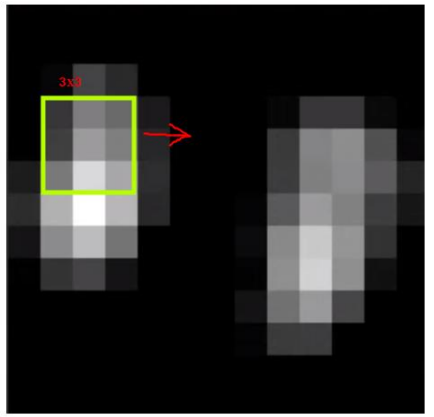


Figure 3: **Left**: Region Proposal Network (RPN). **Right**: Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.

Faster R-CNN

- Summary
  1) Get a trained (ie imagenet) convolution neural network
  2) Get feature maps from the last (or deep) convolution layer
  3) Train a region proposal network that will decide if there is an object or not on the image, and also propose a box location
  4) Give results to a custom (python) layer
  5) Give proposals to a ROI pooling layer (like Fast RCNN)
  6) After all proposals get reshaped to a fix size, send to a fully connected layer to continue the classification
- Training
  1) RPN Classification (Object or not object)
  2) RPN Bounding box proposal
  3) Fast RCNN Classification (Normal object classification)
  4) Fast RCNN Bounding-box regression (Improve previous BB proposal)



Faster R-CNN

- Performance

Detection results on **PASCAL VOC 2007 test set.**

method	# proposals	data	mAP (%)
SS	2000	07	66.9†
SS	2000	07+12	70.0
RPN+VGG, unshared	300	07	68.5
RPN+VGG, shared	300	07	69.9
RPN+VGG, shared	300	07+12	**73.2**
RPN+VGG, shared	300	COCO+07+12	**78.8**

Timing (ms) on a K40 GPU, except SS proposal is evaluated in a CPU.

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	**10**	47	**198**	**5 fps**
ZF	RPN + Fast R-CNN	31	**3**	25	**59**	**17 fps**

More Object Detection Models

- http://www.erogol.com/object-detection-literature/