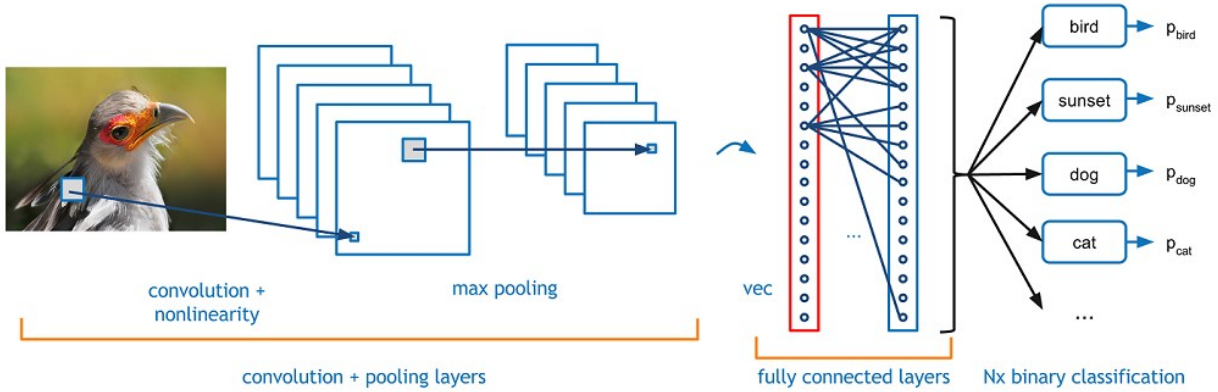
Chapter 3
Convolutional Neural Networks

(Artificial Neural Networks which use a required depth of convolutional and pooling layers to remove disturbances and non-relevant information and extract the feature space in supervised classification or regression problems)



Two main Limitations of Fully Connected NNs

(in classification or regression for real-world patterns)

1. Since the former introduced NNs use fully connected layers and the dimension of the real-world patterns (images-Speech-time series,…) is really high following challenges arise:

   (2-1) the required wirings are massive and complex and every "call" takes considerable time.

   (2-2) the optimization (call and recall iterations) will take a long time.

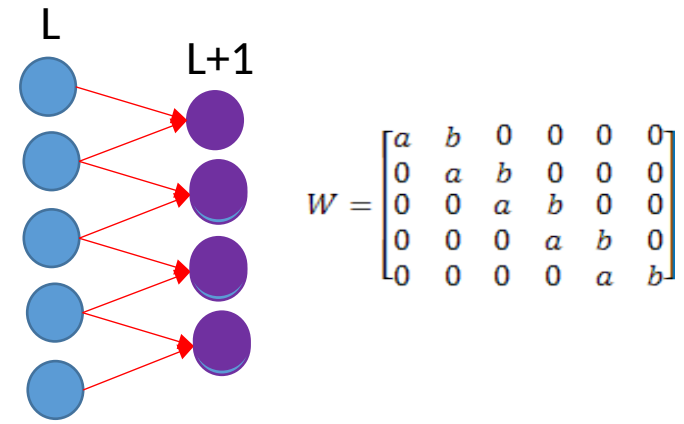
   (2-3) due to large redundancies, over-fitness may be occurred.

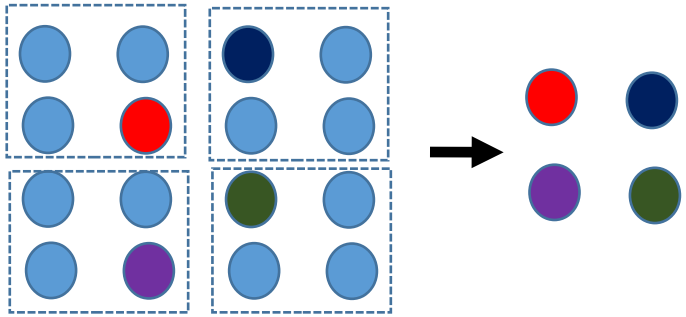
2. **There are various sources of Disturbances(non-relevant information) and uncertainties (distortions) which exist on the input patterns in different forms, and the required filtering and scaling procedures may not be done by the former introduced NNs.**

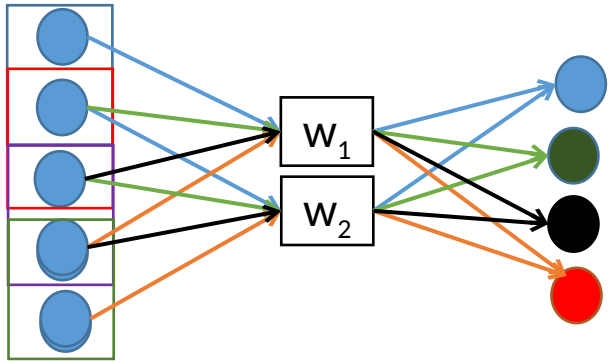
Solutions for the First Limitation:

1. locally and sparse connected layers
2. Sub-sampling
3. Shared weights

Illustrative Examples







About the Second Limitation

Next slides (through an illustrative example) will explain that:

1. Why former classic NNs can not filter disturbances and nullities and why they are not invariant against uncertainties?

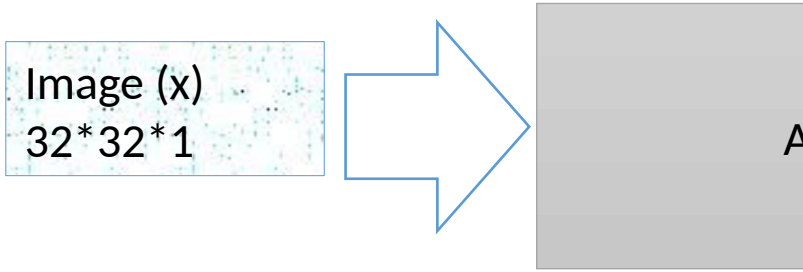
2. How our brain (as a natural NN) can remove disturbances, nullities and uncertainties?

3. What are Convolutional NNs (CNNs) ? and How do they successfully act like our brain?

A simple Classification Problem

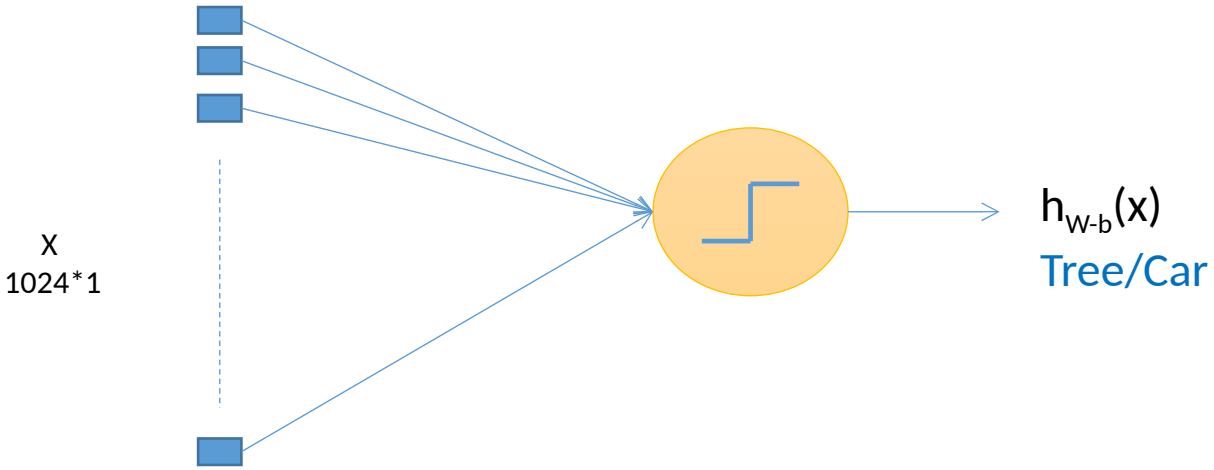
Class 1: Tree (image: 32 *32)

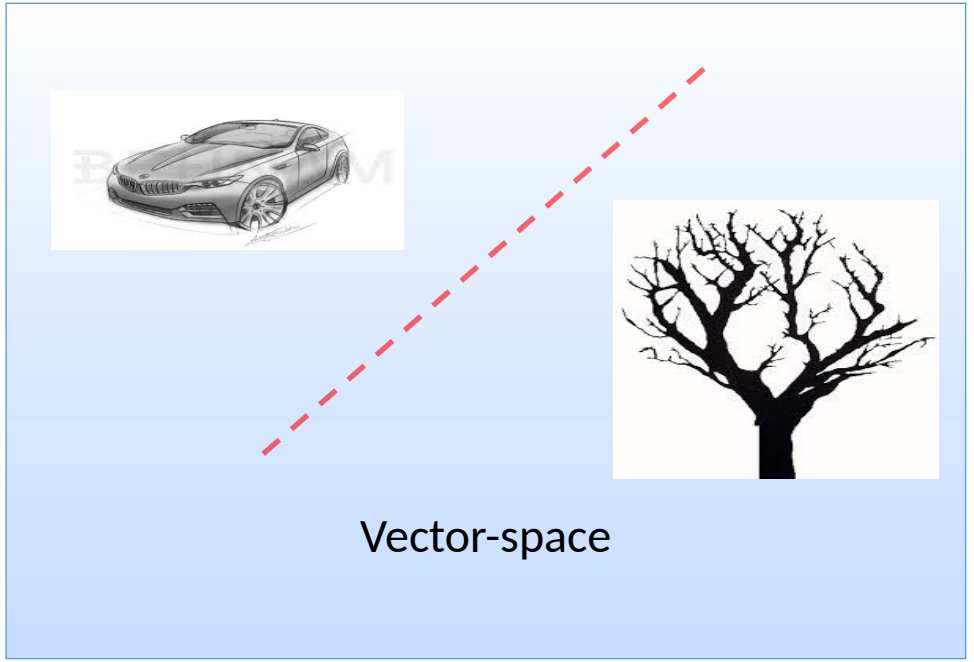
Class 2: Car(image: 32 *32)



Perceptron-Adaline (One-layer)

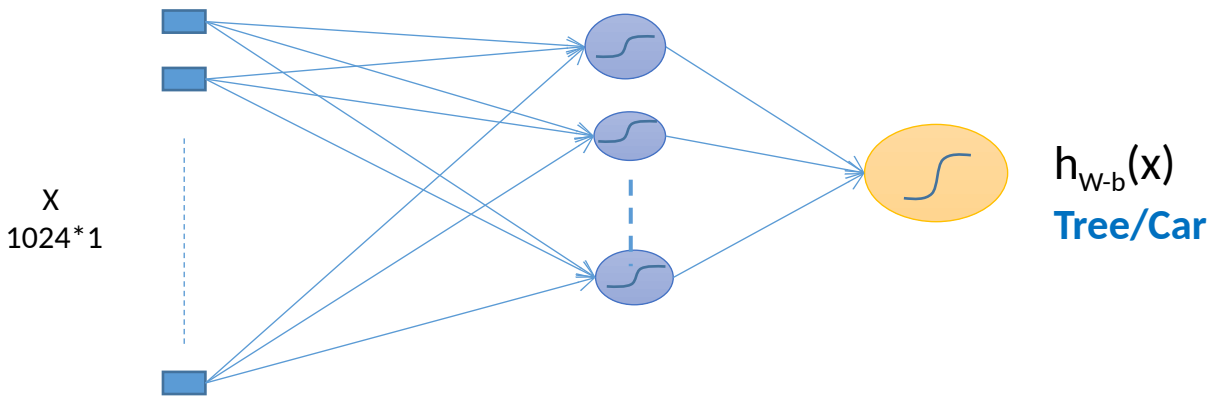
- All input images belong to a unique type of car or tree.
- At each class, objects have a little differences in size, color, position, light and perspective (almost no uncertainties).
- There is not any non-relevant object in images (no disturbance).

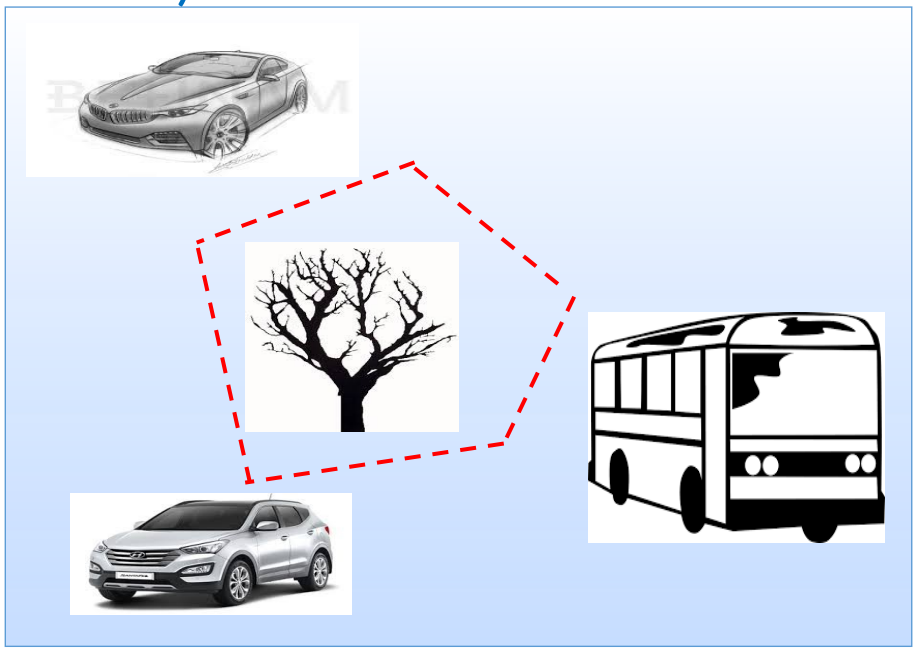




MADALIN (two layers- Fully Connected)

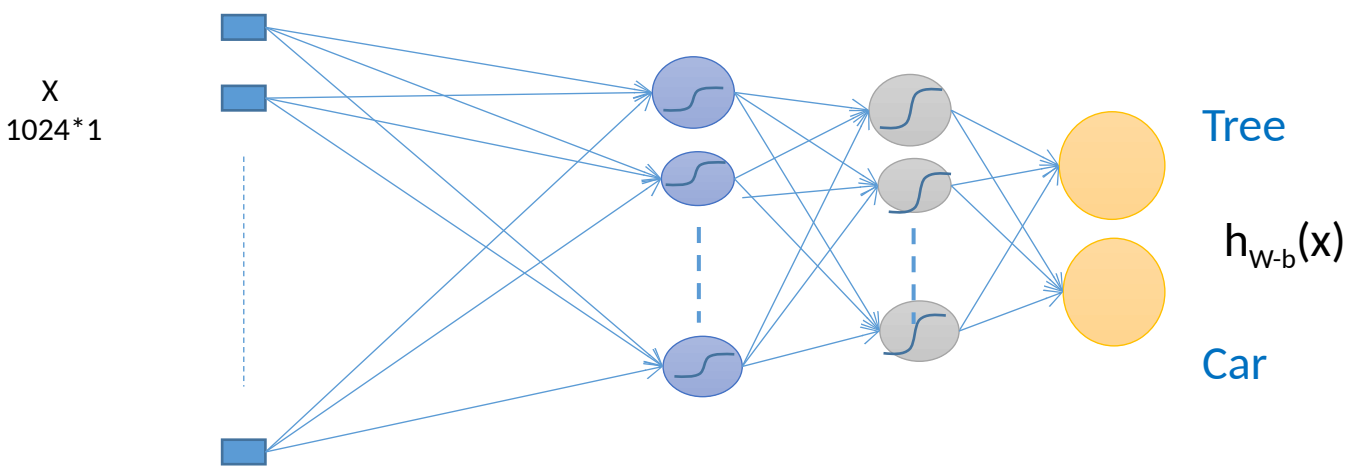
- images of car belong to more than one type of car.
- At each class and each type of objects, objects have a little differences in size, position, light, color and perspective.
- There is not any non-relevant object in images (no noise).

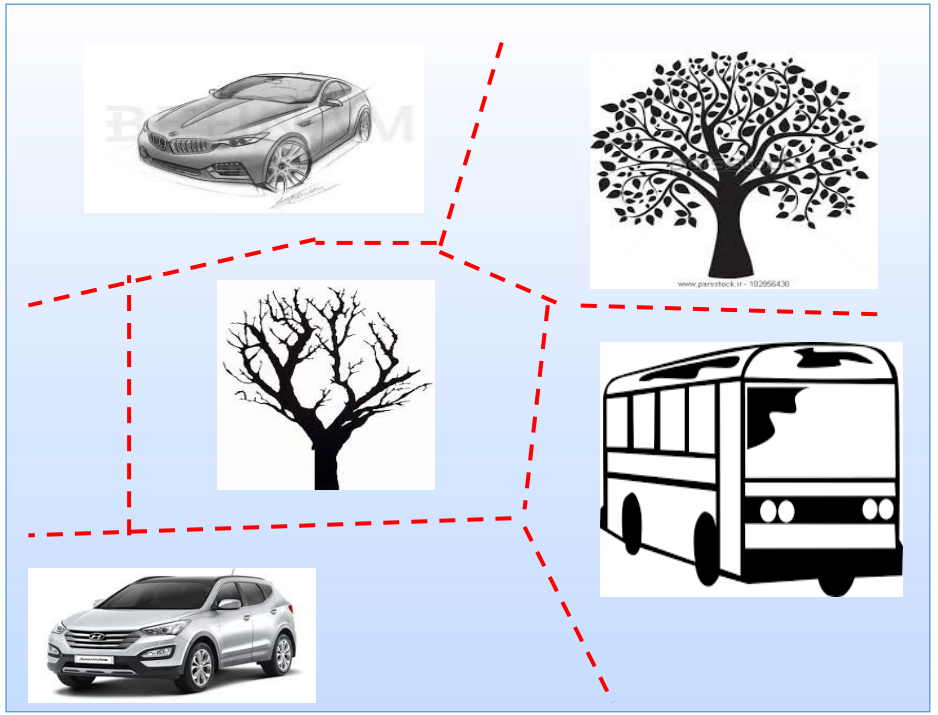




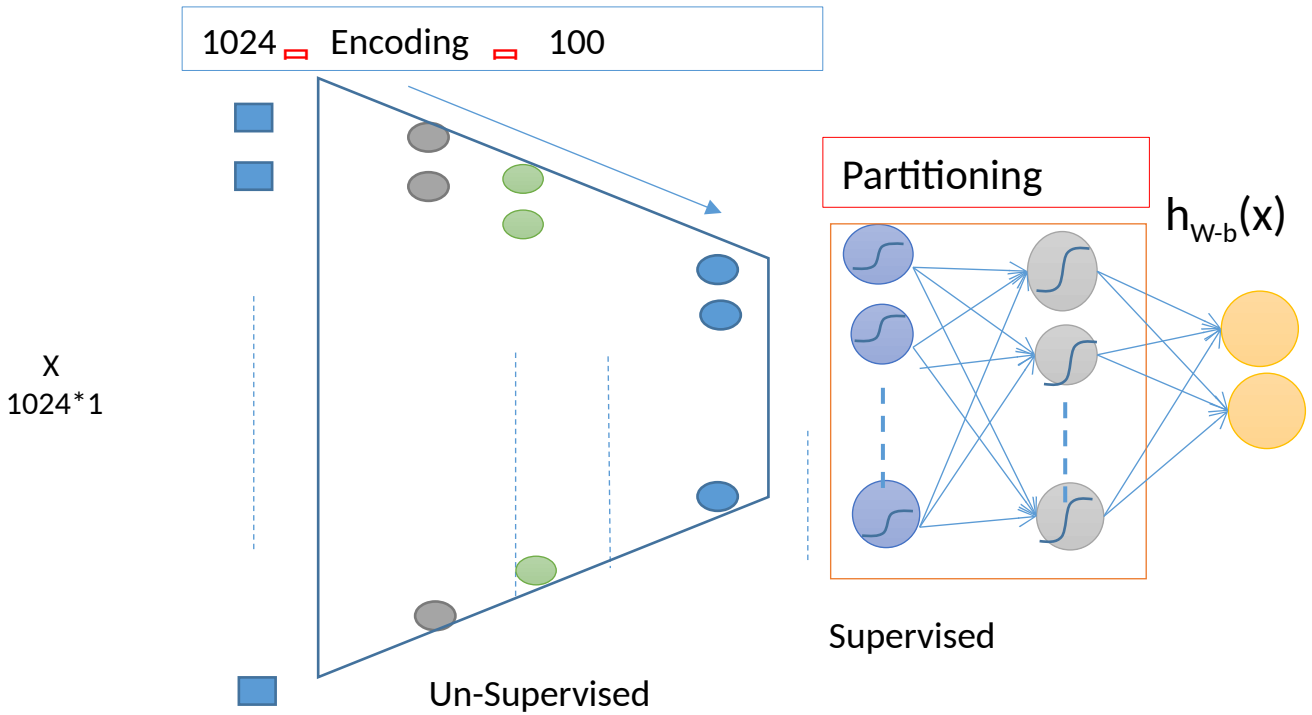
Multi-Layer-Perceptron (MLP-Fully Connected)

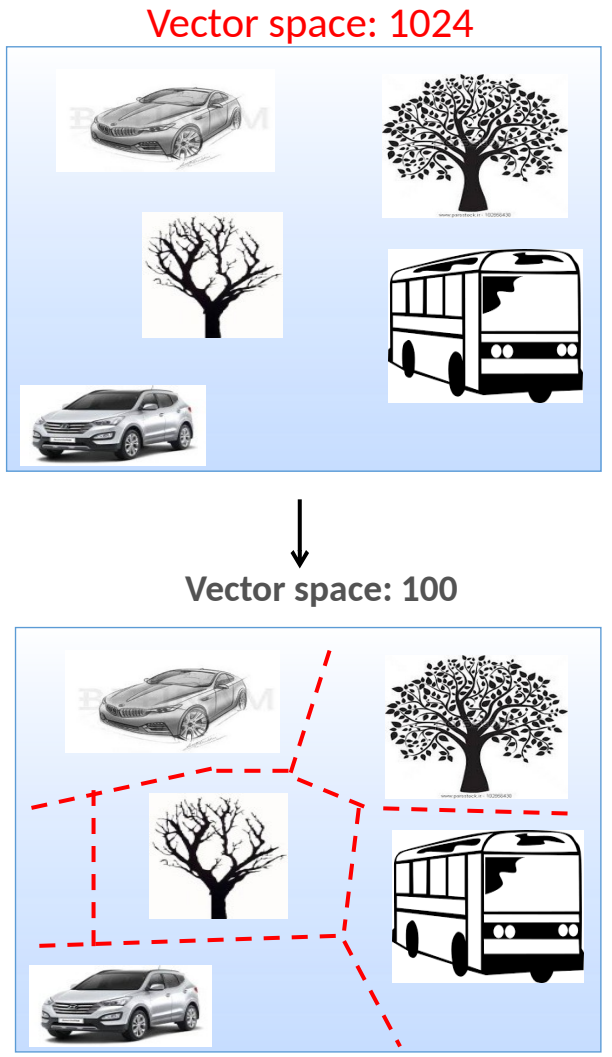
- Types of cars and trees are more than one.
- Their scales, colors, positions, rotations, projections change a little.
- There is not any non-relevant object in image (no noise).





MLNN: MLP + AutoEncoder/Stacked RBMs





For Reliable Partitioning and Generalization

Real-world Images

- There are plenty types of cars and trees.
- Their scales, colors, positions, rotations, projections and perspectives are different.
- There are non-relevant objects in images (noise).





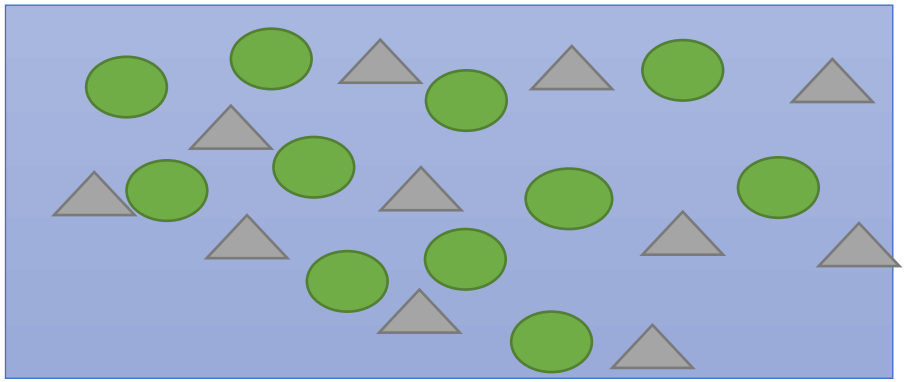




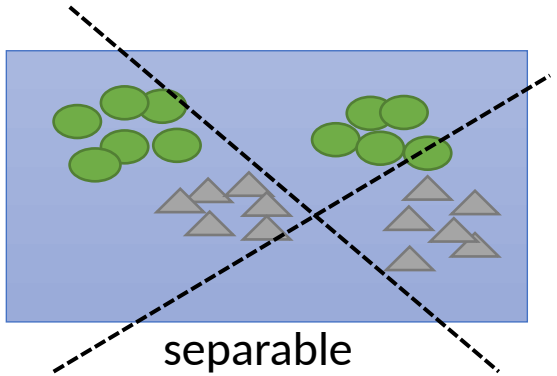



Non-Separable Vector Space

- Cars and Trees are not Separable in vector space!
- Nearest Neighbor of a car may be a tree !!!.



Non - separable



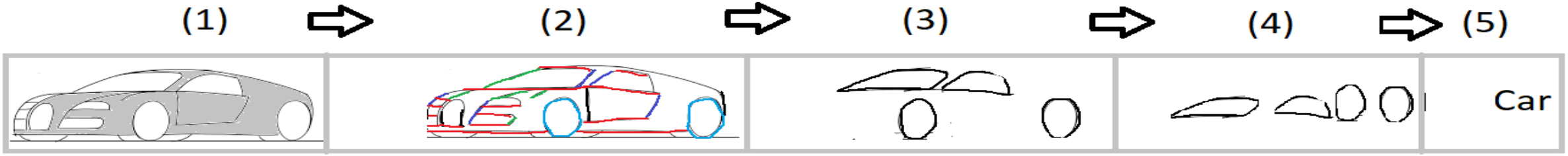
- Perceptron/MADALINE/MLP/MLNN will fail to classify such objects.
- These methods can only make boundaries between clusters and they can not make easily a new separable space.
- Now, what should we do?

Chapter 2 – part 1

How do our Brain classify visual objects ?

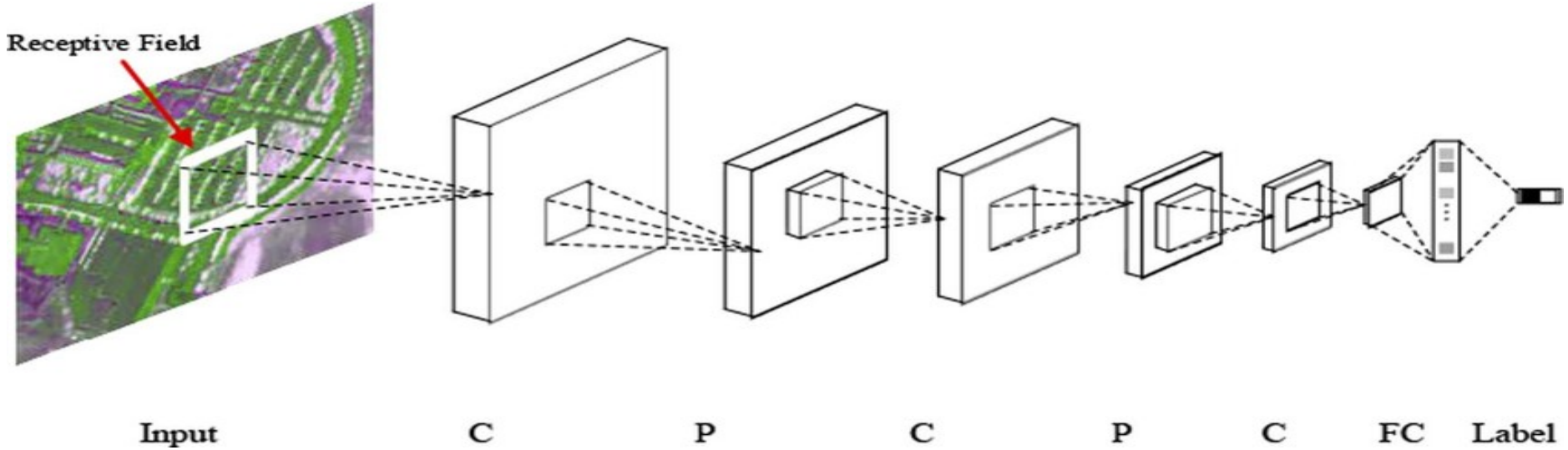
Any Idea…?

How do our Brain classify such objects ?



Input ▭ Edges / Corners/Curvatures ▭ Sub skeletons ▭ whole object ▭ Label

Local Semi-Local Semi-Global Global



Key Operations

1. Filtering

   Non-Relevant data (infrequent sub-patterns) are removed, gradually, through Filtering and frequent sub-patterns are extracted as effective features.

2. Scaling

   Extracted effective features gradually become invariant to different sizes, positions, perspectives, color and light.

   "At each Filtering/scaling the dimensionality reduction is occurred"

3. **Final Classification**

   All extracted and scaled features make a separable space and allow us to classify different classes

Image ▭ {Edges / Corners /curvatures ⊟ Sub skeletons ⊟ whole object } ▭ Label

(Input Patterns) ▭ { Filtering/scaling .... Filtering/scaling ... } ▭ (Classifying)

Important Notes:

1. the scales of the frequent sub-patterns are different and they have propagated in different ways over spatial dimensions.
2. Non-Relevant Data are distributed randomly among effective features in different sizes.
3. In many cases we can not remove disturbances, nullities and uncertainties by using only one Filtering/Scaling Operation and a sequenced Filtering/scaling operations are required.
4. Our brain Actually makes cascaded Filtering/scaling operations

Block Diagram

Chapter 2 – part 1

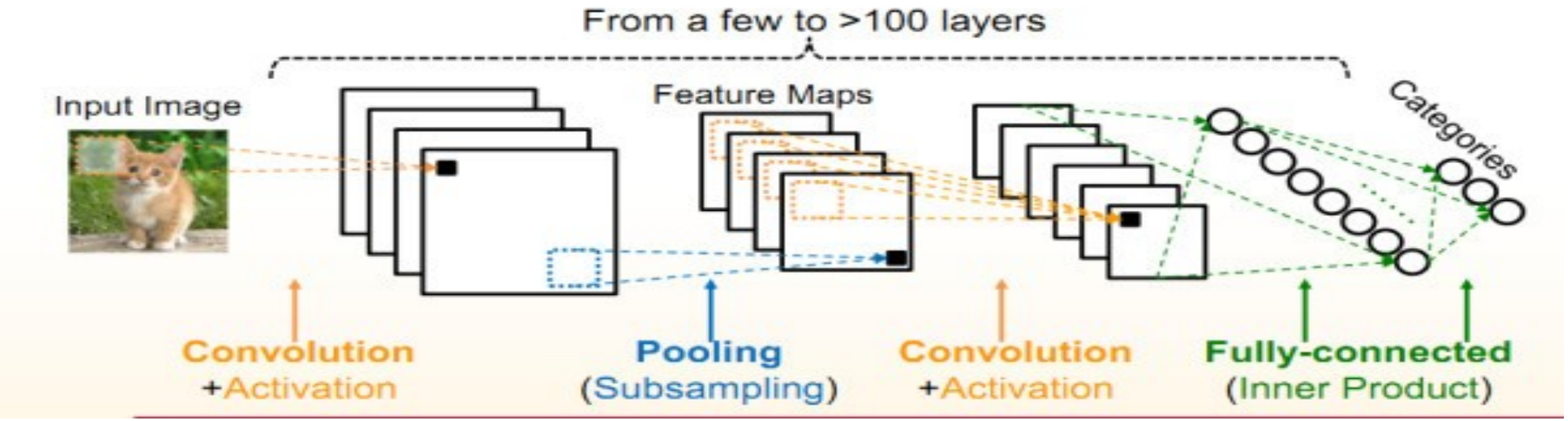
Convolutional Neural Networks (CNNs)

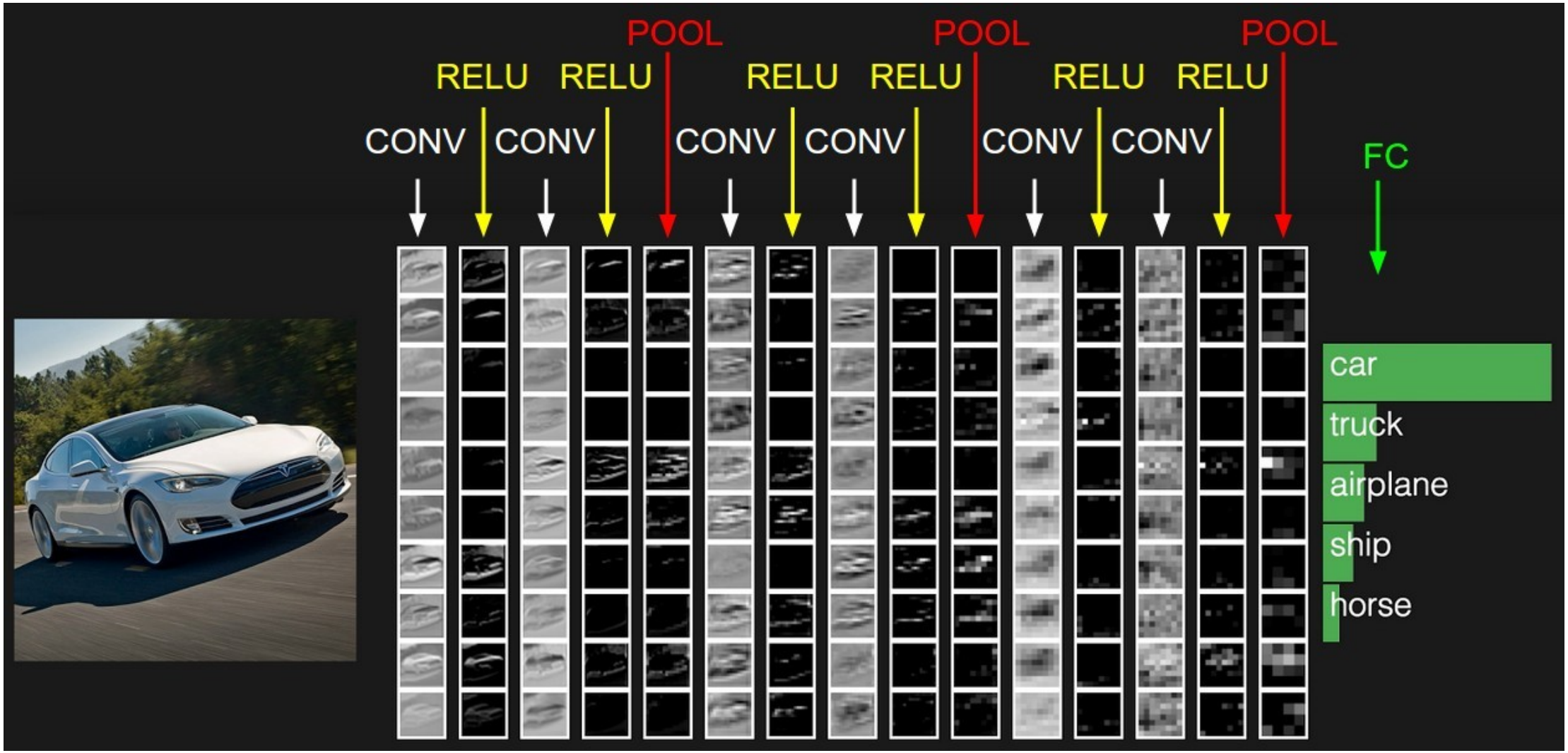
Convolutional Neural Networks (CNNs)-Yann LeCun 1994

Filtering is done by Convolution

Scaling/Dimensionality reduction is done by Max/Mean Pooling

Final Classification is done by Fully connected layers





http://cs231n.github.io/convolutional-networks/

Convolution

- Convolution in CNNs is a variant of convolution in LTI systems.
- Here, convolution is defined as follows:

z=Relu (x.w+bi), x: A Patch of input signal, w : Filter, bi: bias

Differences with signal & systems convolution:

- Bias is added
- Flipping is Ignored
- The result is passed from an activation function (Relu)

x (Chosen Patch)

			−1	0	1
			0	−2	1
			1	1	0

a*b=(6*6)

w (Filter Window)

9−	3	2
0	5	0
0	6	0

0.5 bias

c*l=(3*3)

Activation Function Relu/ tanh

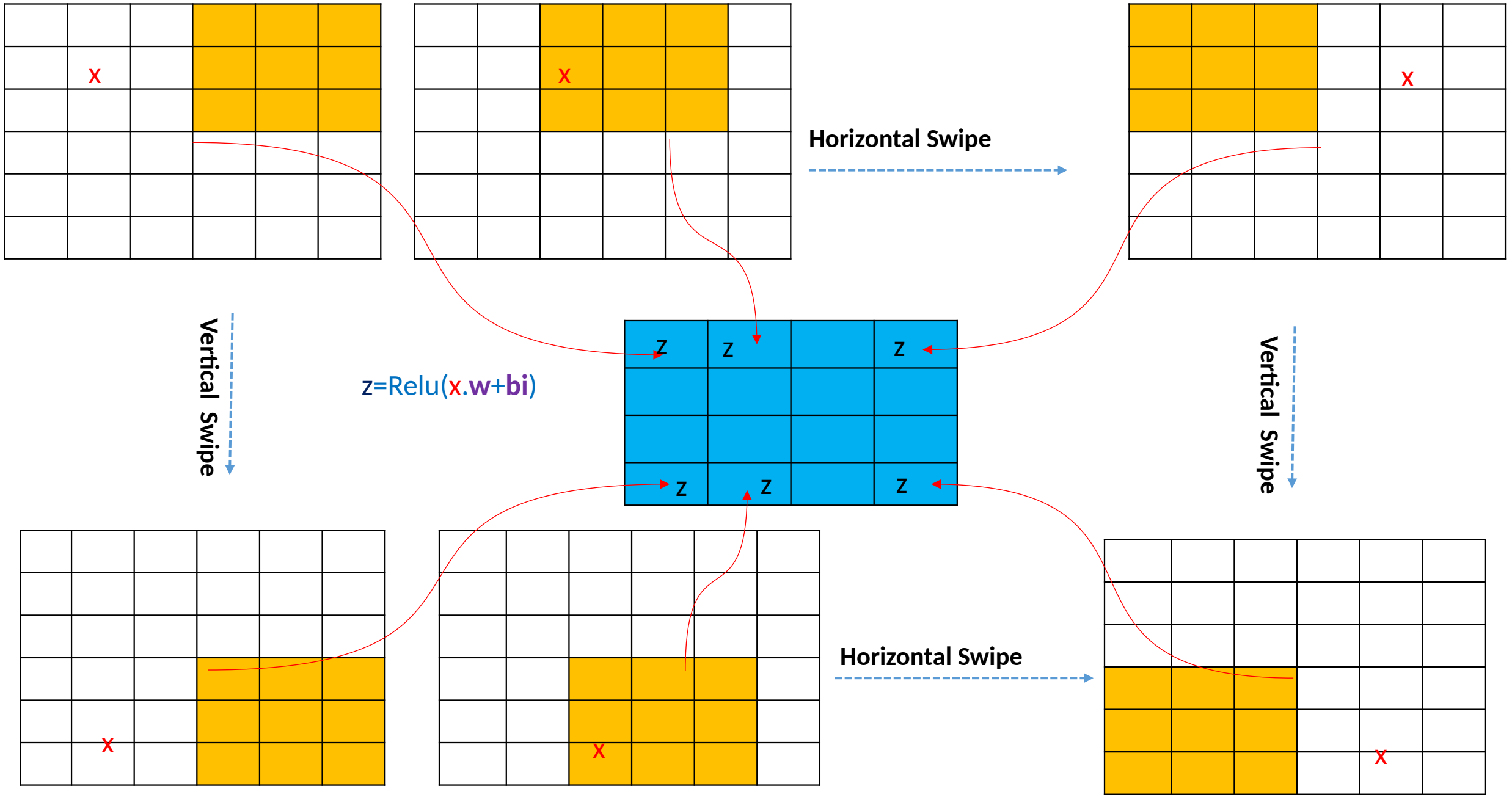
Z (unit in Feature map)

			7.5

Feature Map

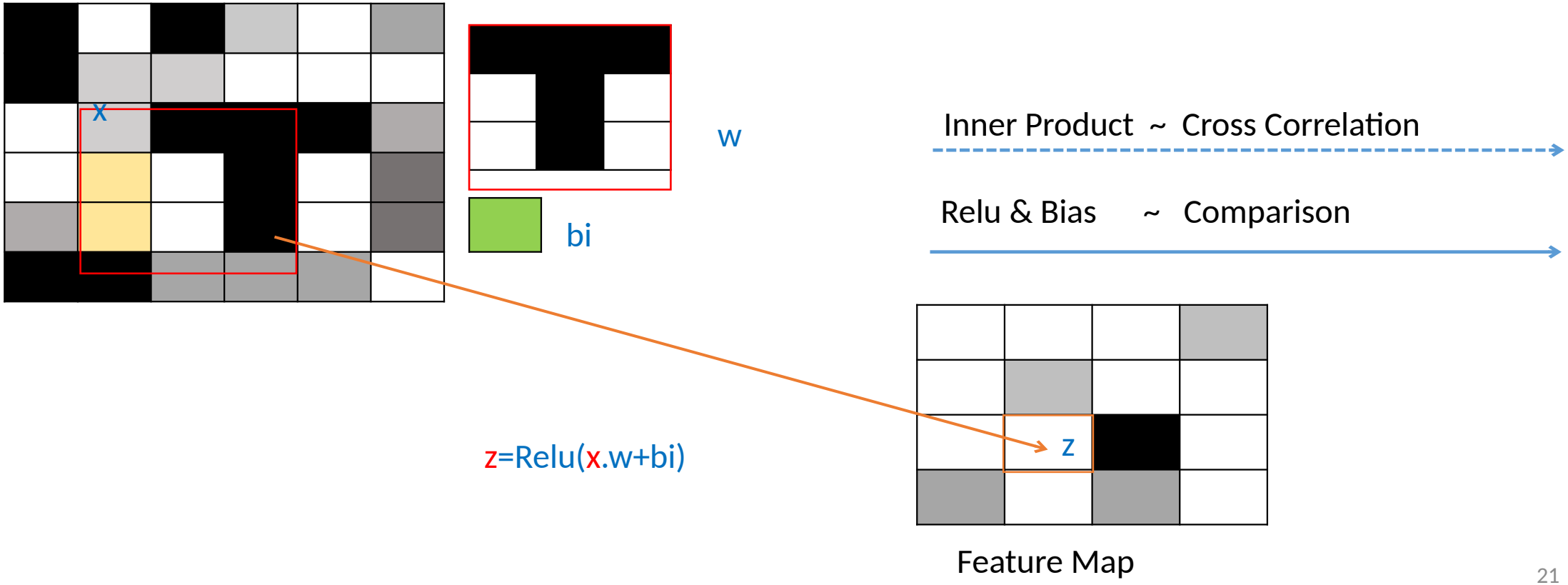
(a-c+1)*(b-l+1)=4*4

z=Relu(1*2+0*3+(-1)*4+...+1*6+1*0+0.5)

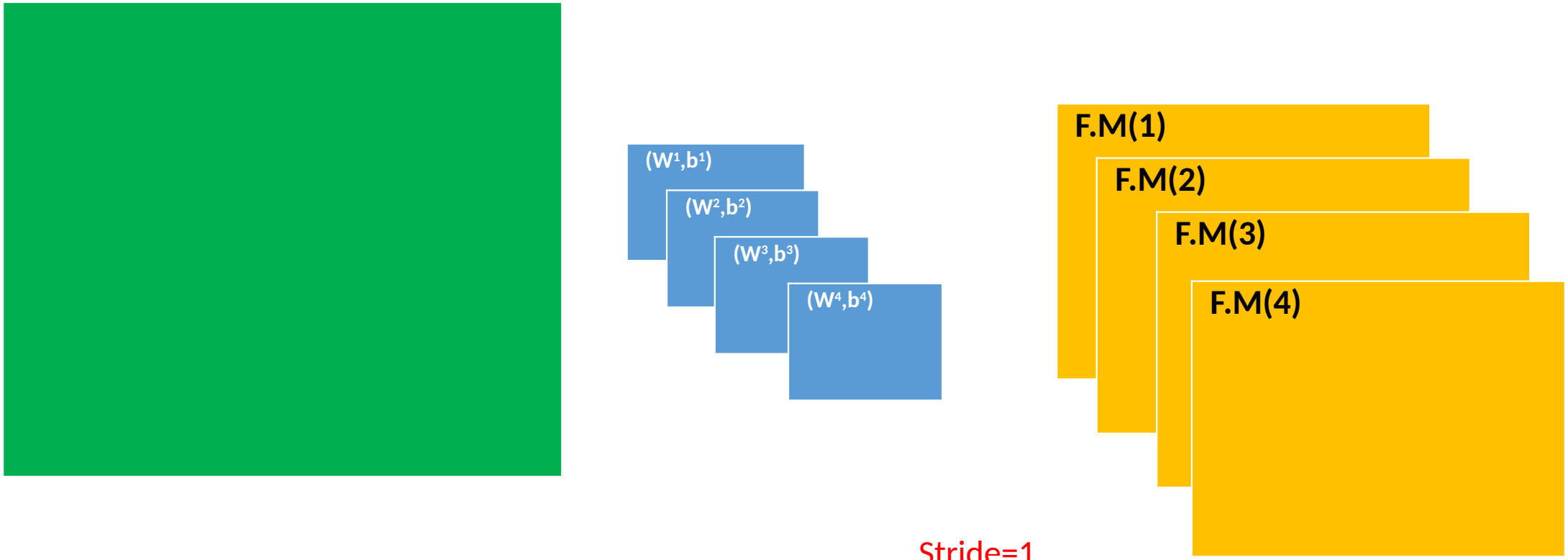

X
X
X
Horizontal Swipe
Vertical Swipe
z=Relu(x.w+bi)
Z
Z
Z
Z
Z
Z
Vertical Swipe
X
X
X
Horizontal Swipe

How does Convolution Extract Information (Filtering)?

1. **Each Filter becomes similar to a frequent sub-pattern in the considered kernel size**
   Each Filter can amplify the effect of a frequent sub-pattern in the feature map
   (due to its bigger correlation coefficient)
2. **All sub-patterns which has low correlation coefficient are removed.**
   (Using "Relu" and "Bias" causes that all convolutions in which correlation is below a certain threshold is removed)



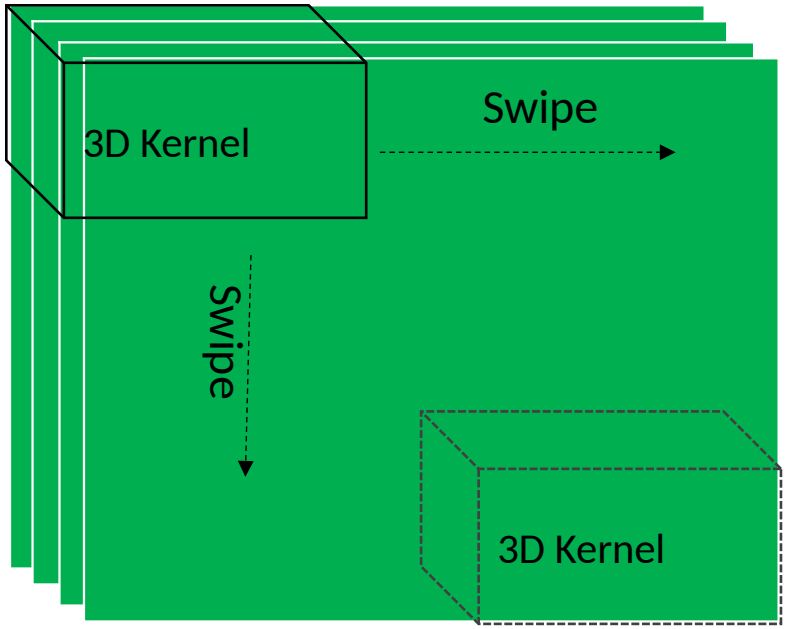
More than one Feature Map

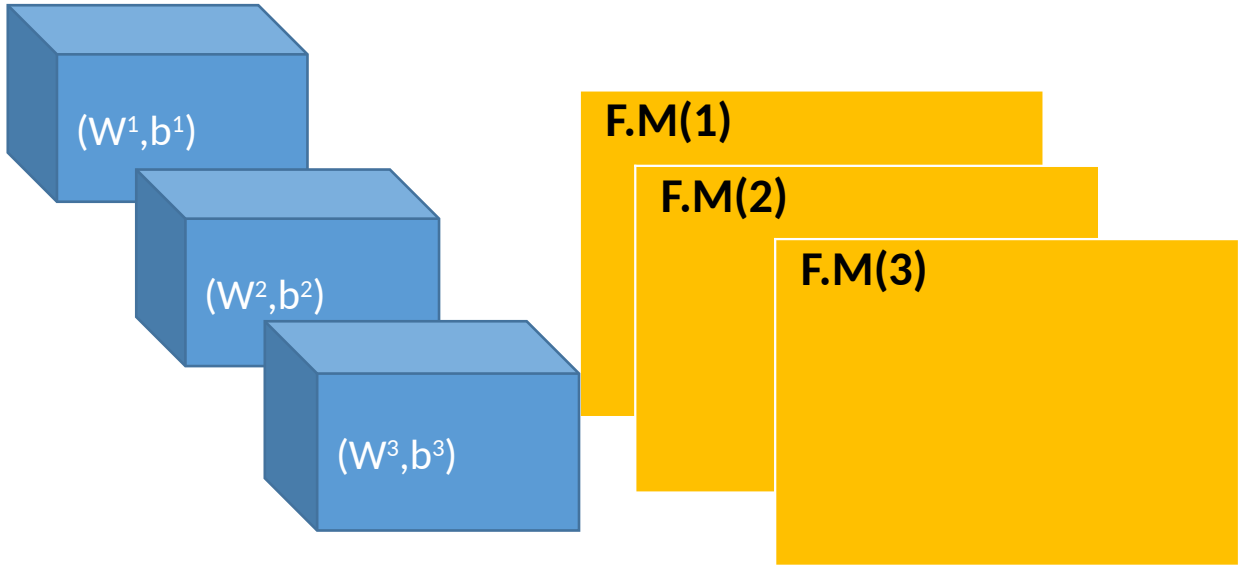


•Every Feature map contains an independent extracted frequented sub-pattern.

3D Convolution

1. For Colored Images
2. For 2nd, 3rd, …. Convolution layers





Image(a*b*4)

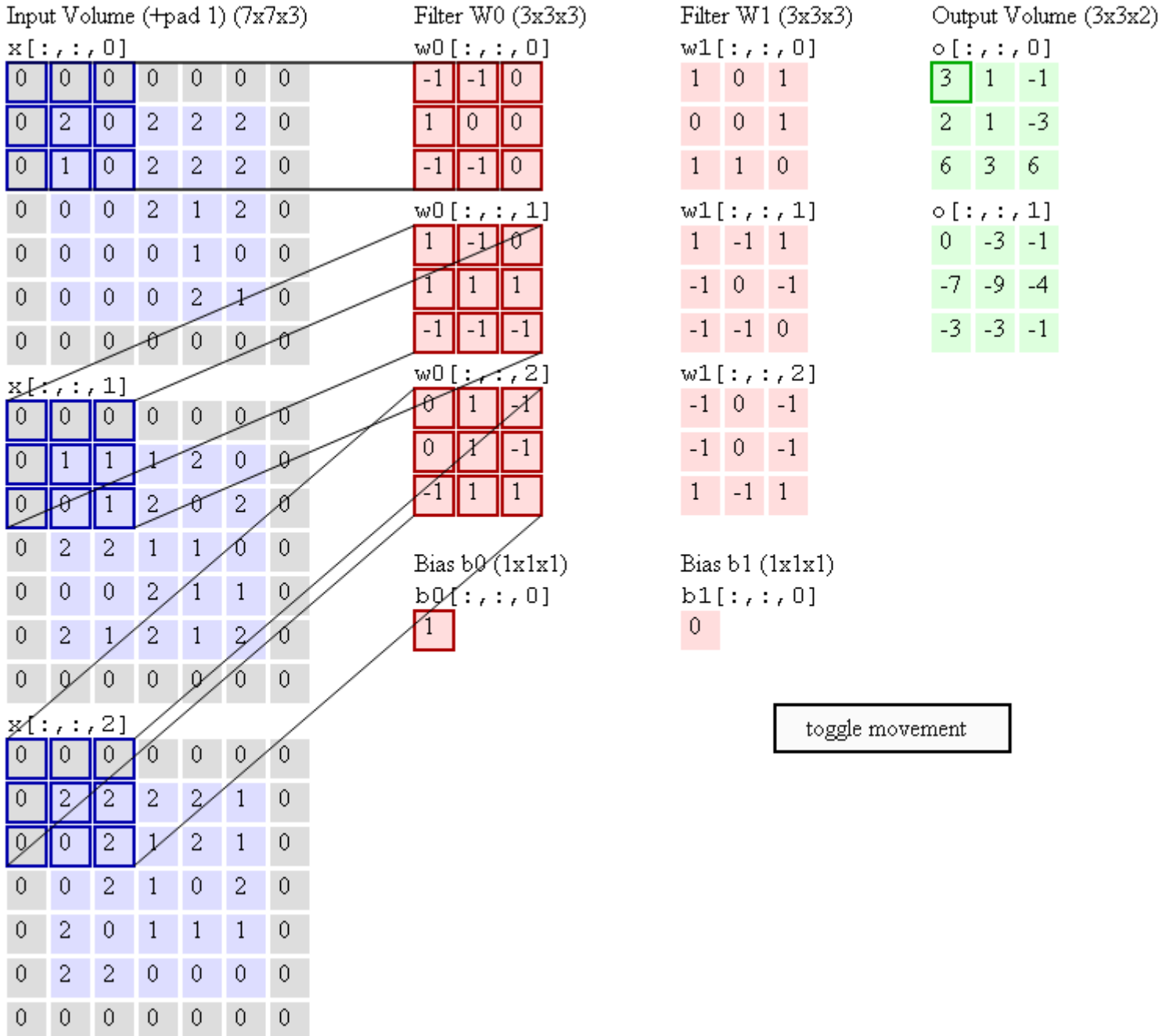
3 3D-Filters (c*l*4)+ 3 biases

Stride=1

3 Feature Maps (a-c+1*b-l+1)

An illustrative example of 3D convolution swiping patches are done with Stride=2

http://cs231n.github.io/convolutional-networks/



Max-pooling , Mean-pooling

(Dimension-Reduction(subsampling) & scaling)

1	2	6	0	0	2
2	0	3	2	1	0
0	4	3	3	5	1
1	2	3	0	1.2	2
5	6	4	1	0	4.1
9	2	0	2	7.8	4

Max-pooling

Size 3*3

6	5
9	7.8

1	2	6	0	0	2
2	0	3	2	1	0
0	4	3	3	5	1
1	2	3	0	1.2	2
5	6	4	1	0	4.1
9	2	0	2	7.8	4

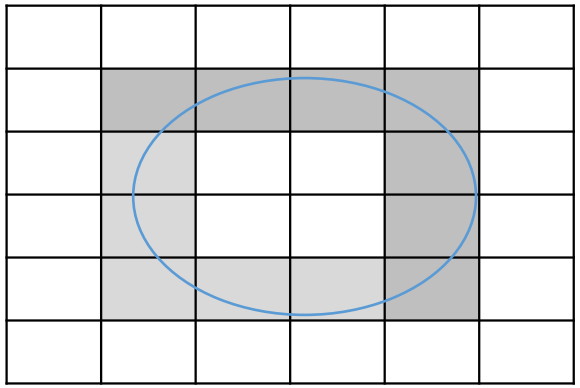
Mean-pooling

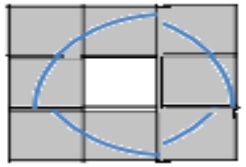
Size 3*3

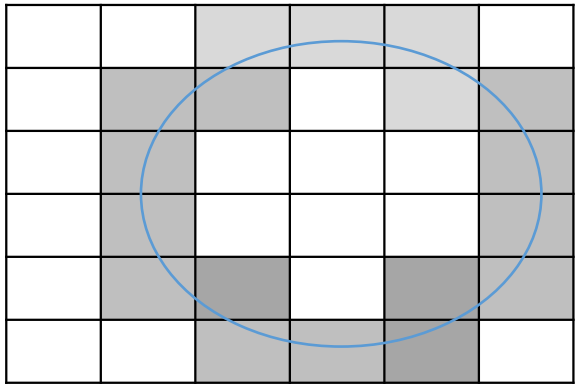
2.33	1.3
3.13	2.32

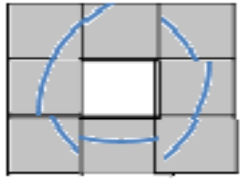
Scaling by Max-Pooling

Pooling Size 2*2

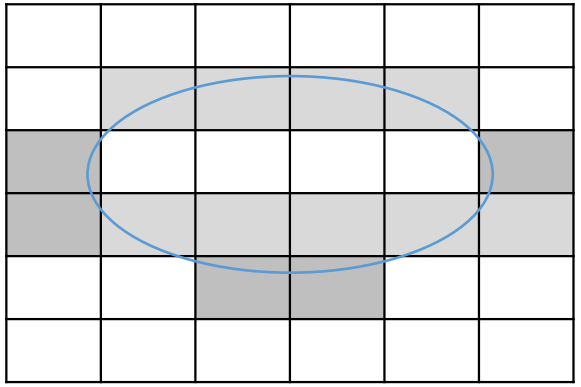


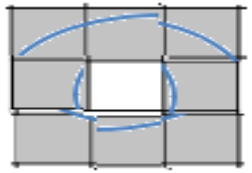


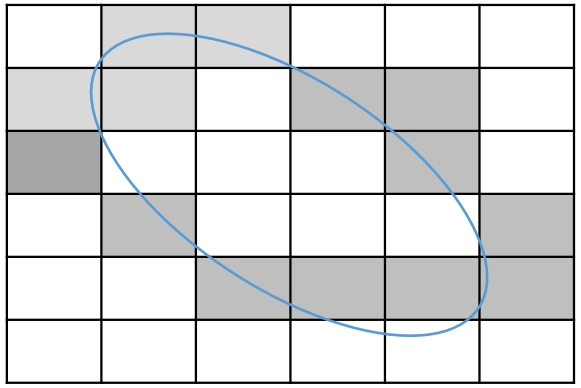


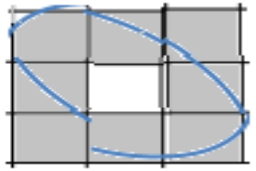


Pooling Size 2*2









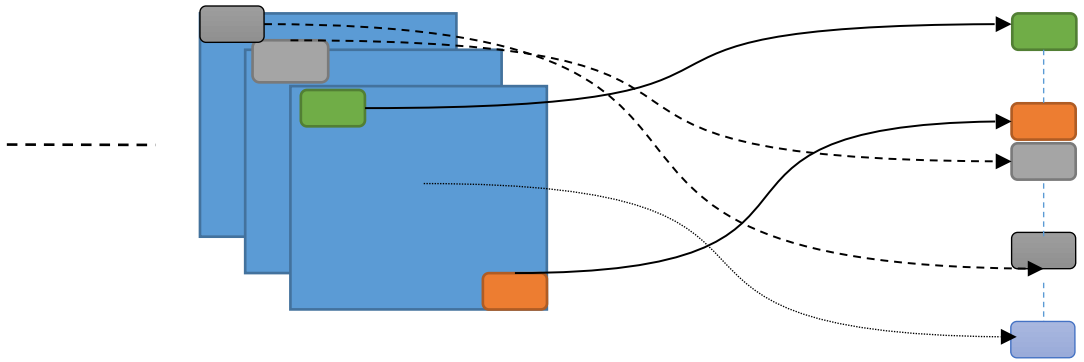
Correlation coefficient (0.3 ▭ 0.6)

About Convolution and Pooling layers

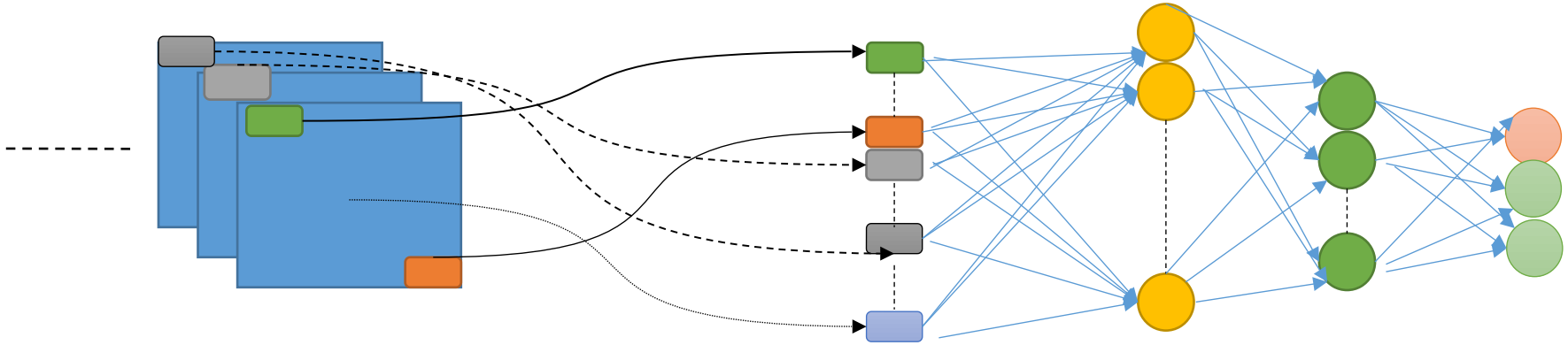
- Each convolution Layer (in a certain kernel size) remove infrequent sub-patterns (**Disturbances**) and extract frequent sub-patterns (**Features**) ,.
- At each convolution Layer several feature maps may be generated . Each feature map extract an independent frequent sub-pattern.
- Each Pooling Layer reduces the dimension of the feature maps by sub-sampling. After each pooling operation, all features points which make a common frequent sub-pattern (from different input patterns) come nearer together: their scales will be more similar together . Also, one can say that after applying a pooling layer, the extracted features will be more invariant to size, position and perspective of the input pattern.
- Repeating the Convolution and Pooling layers allows (1) to filter disturbances from lower to upper sizes and concurrently (2) to extract each feature with a certain scale.
- Sub-Patterns which appear in the first convolution layer are the main edges of the inserted pattern; In next layers, sub-patterns become like corners and smaller sub-skeletons and finally in the last convolution layer they form the main sub-skeletons (effective features).

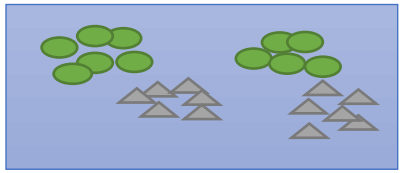
Fully Connected Layers

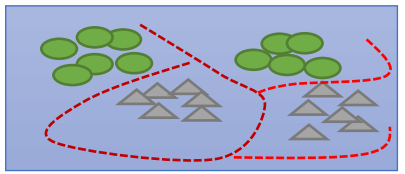
1. Units of last (Convolution Pooling) layer are arranged in a line :



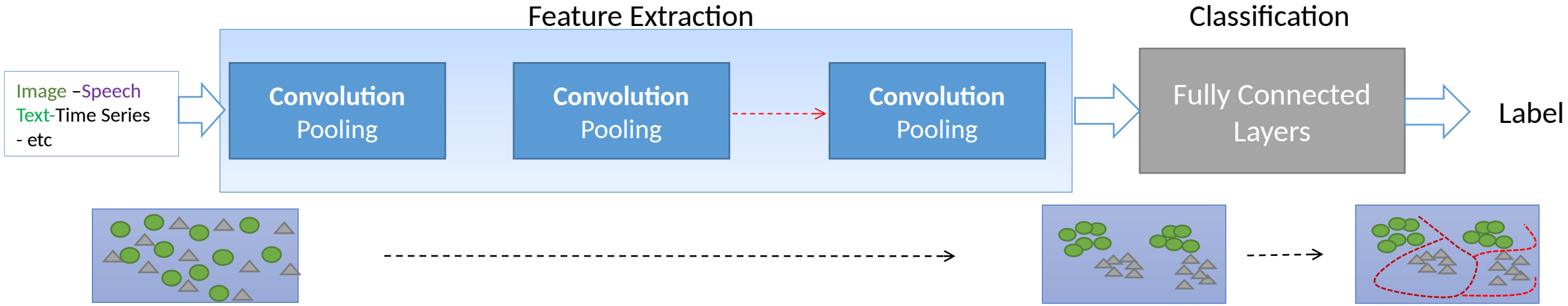
2. Then, one or two fully connected layers are defined and Finally, the output visible layer of CNN are defined.

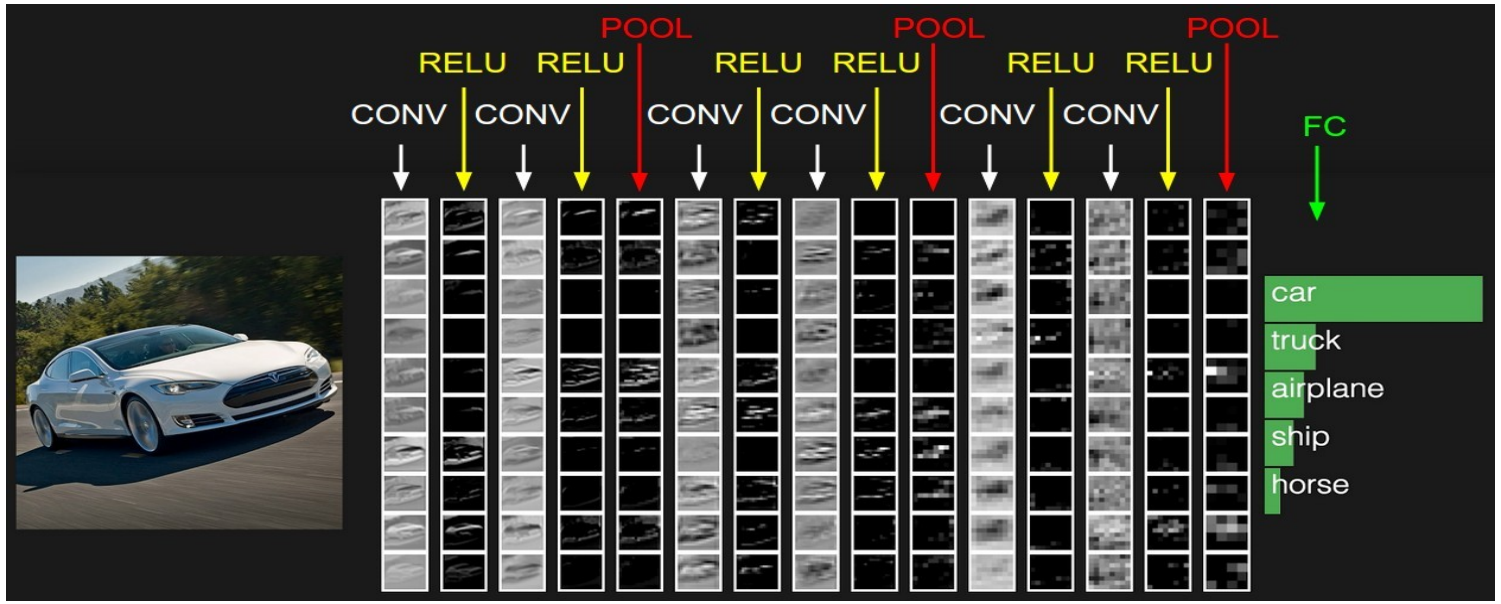






CNN Architecture

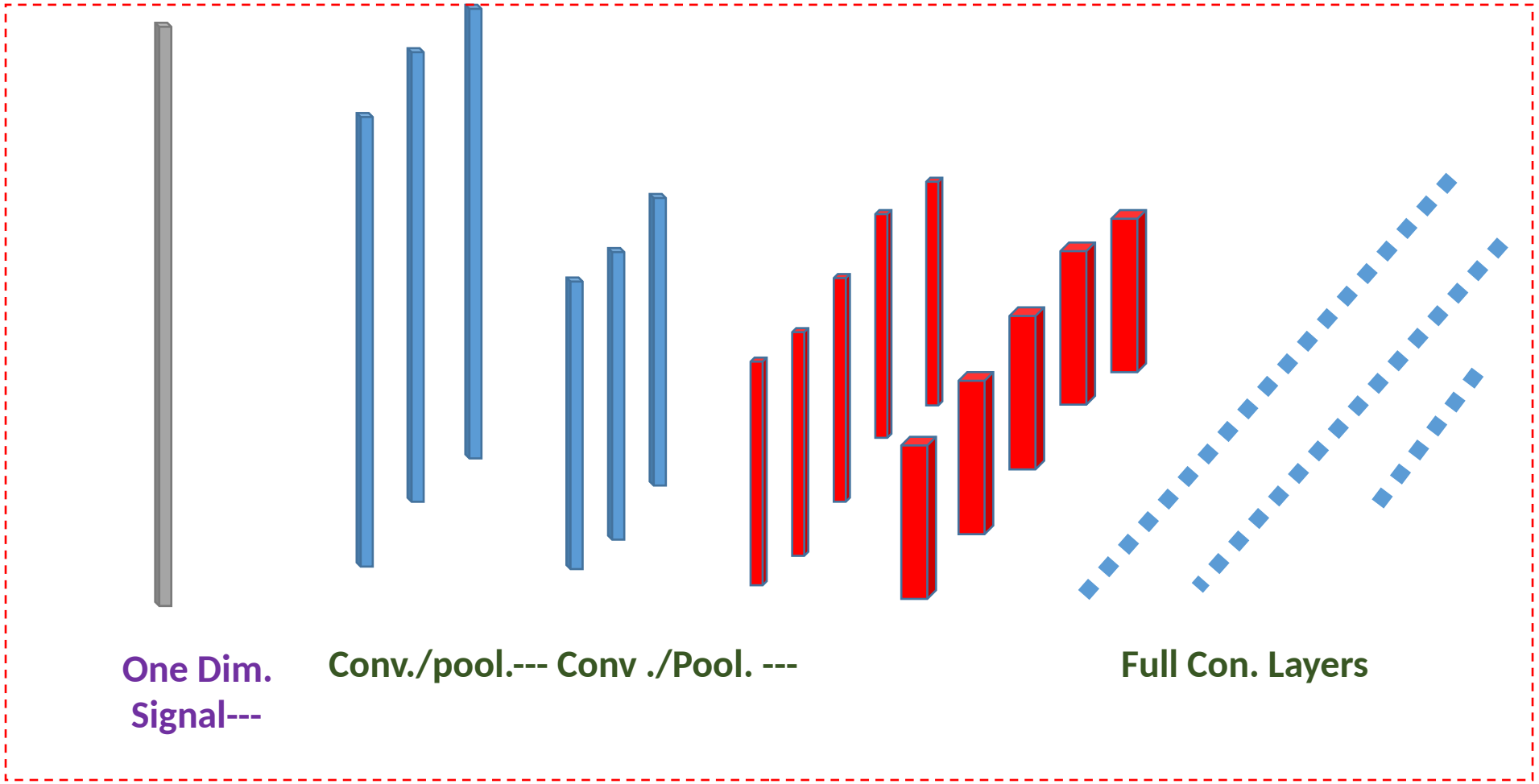




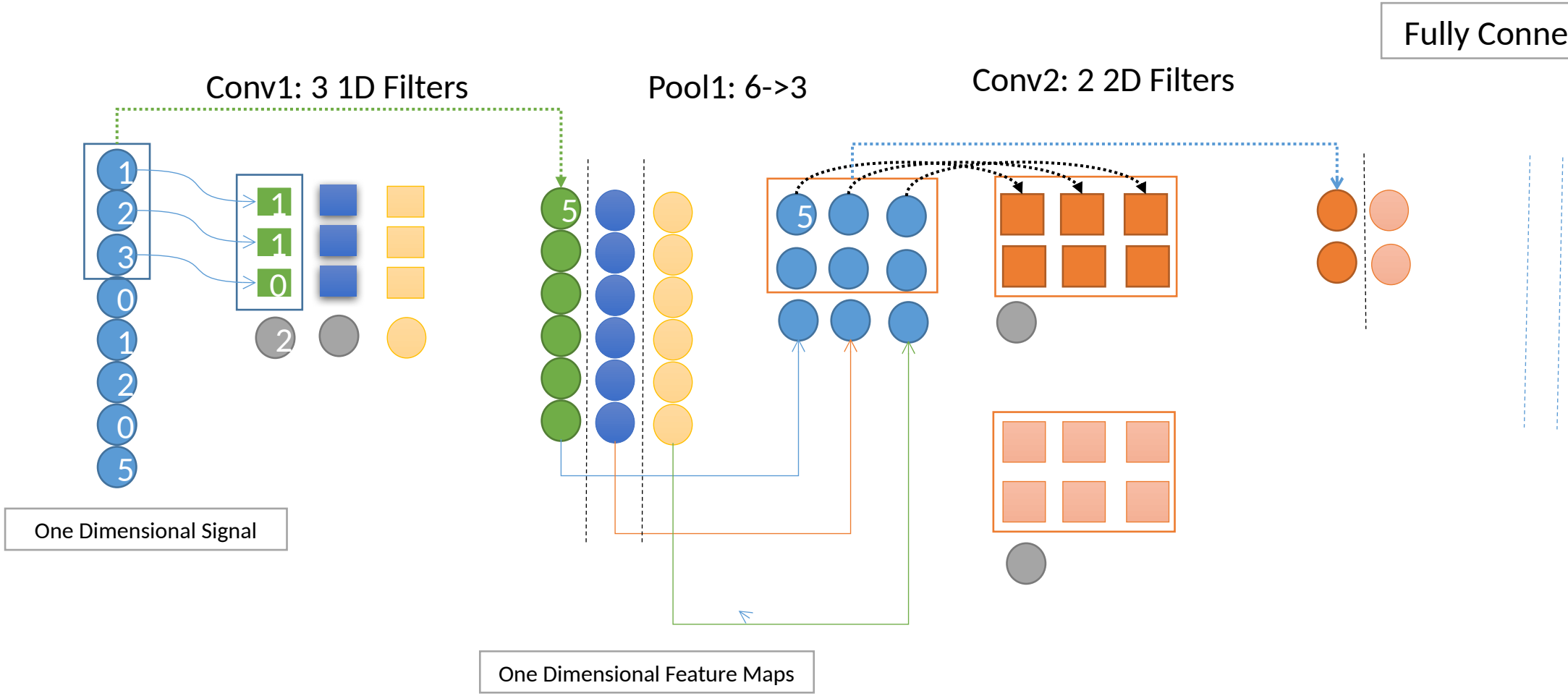
* CNNs can be utilized for regression applications, too. It is enough to (1) extract features in conv./pooling layers and (2) use fully connected layers for mapping purpose.

Convolution and Pooling for one-dimensional signals

- CNNs can extract features from one dimensional signals such as time series.
- The size of convolution filters in the first layer is “1” but for further layers is “2”.



An illustrative example



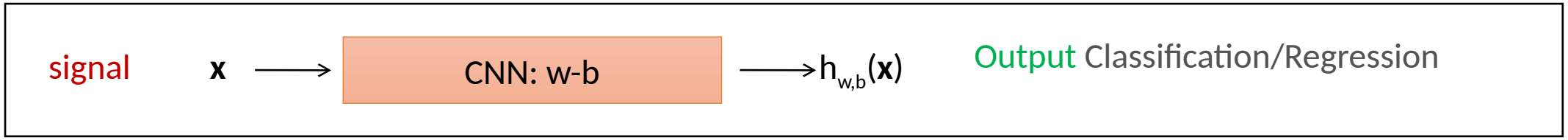
Learning in CNN

Real Data Set: Image Set, Speech, Text, Time Series

1- Training set $\{x^q, y^q\}$, $q=1..Q_{tr}$

2- Test set $\{x^q, y^q\}$, $q=Q_{tr}+1..Q_{total}$

The CNN Network:



Goal: Design the CNN in order that $y^q \sim h_{w,b}(x^q)$

0 (MSE or Cross Entropy)

- **Structural Parameters**
  1. **Number of Convolutions/Pooling layers**
  2. **Number of Filters at each Layer**
  3. **Size of each Convolution and Pooling**
  4. **Number of fully connected layers and their numbers of neurons**
- Parameters
  1. **Filters: Windows-Biases**
  2. **Weights and biases of fully connected layers**

Learning Methods of CNNs

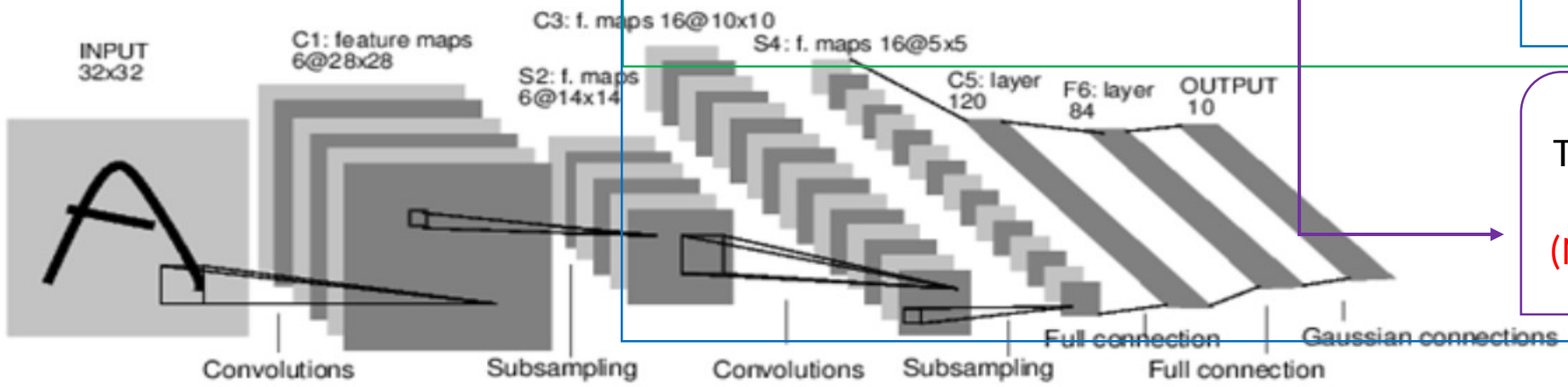
1. Error Back Propagation (Local Search- the same method explained in MLNN)
2. Transfer Learning (From other pre-trained CNNs)
3. Evolutionary-Based Methods (Global Search)
4. Geometric Based Methods

Different Layers in CNNs

1. Convolutional Layers
2. Rectified Layers (**Rectified Linear Unit**)
3. Pooling Layers(**Subsampling layers**)
4. Fully Connected Layers (**Dense Layers**)
5. Soft-Max Layer (**Gaussian Connections**)
6. Loss Function (Point base/Mini-Batch based/Batch based)
7. (Batch) Normalizing Layers
8. Dropout Layers
9. Network in Network Layers
10. Deconvolution Layers **(Deconvolution and Unpooling Layers)**

Main Layers

1. Convolutional Layers
2. Rectified Layers
3. Pooling Layers
4. Fully Connected Layers
5. Soft-Max Layer (Gaussian Connections)



A Full Convolutional Neural Network (LeNet)

To pass frequent patterns and remove disturbances in different scales

To subsample feature maps and remove nullities in order to provide invariances against different distortions

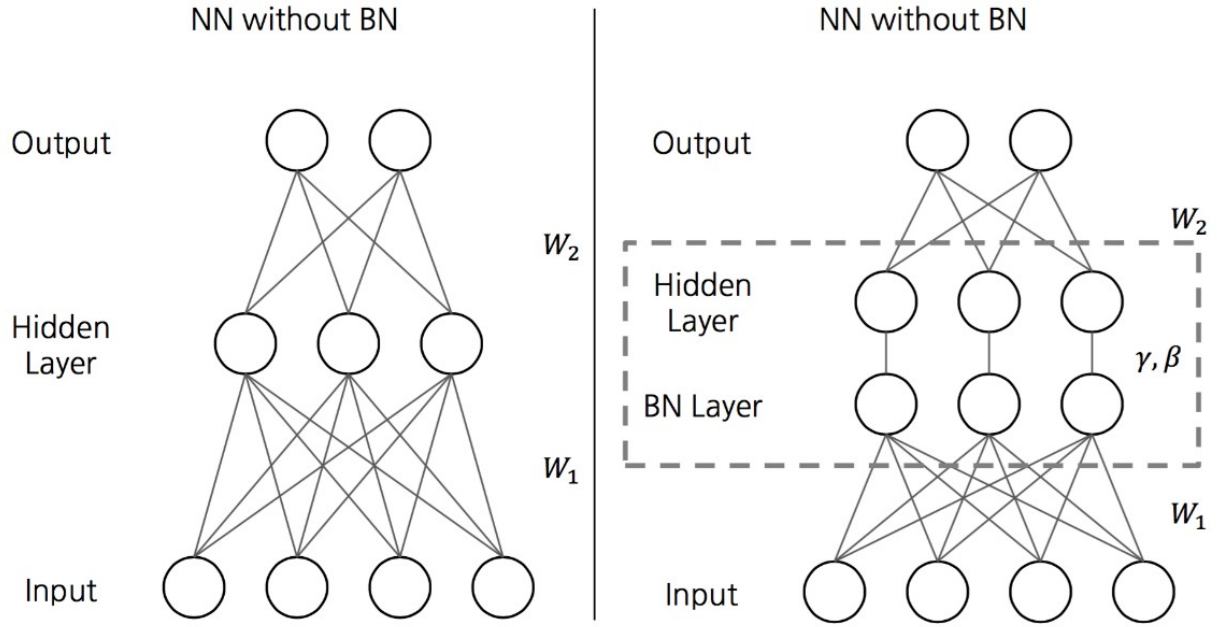
To partition (or map) the extracted feature space in a classification (or regression) problem

To present a probability function for different classes (Normalized Gaussian functions)

7. (Batch) Normalizing Layers

Batch Normalization is a method to reduce internal covariate shift in neural networks leading to the possible usage of higher learning rates.

In principle, the method adds an additional step between the layers, in which the output of the layer is normalized.



Batch Normalizing Algorithm

Batch normalization:

The scalar features are normalized independently, making their means 0 and the variance 1. This means in case of a d-dimensional input x = {$x^{(1)}$...$x^{(d)}$}, each dimension of the input x is normalized independently, thus every variable x, y, μ, σ becomes $x^{(k)}$, $y^{(k)}$, $\mu^{(k)}$, $\sigma^{(k)}$.

γ and β are introduced as scale and shift parameters

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

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

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

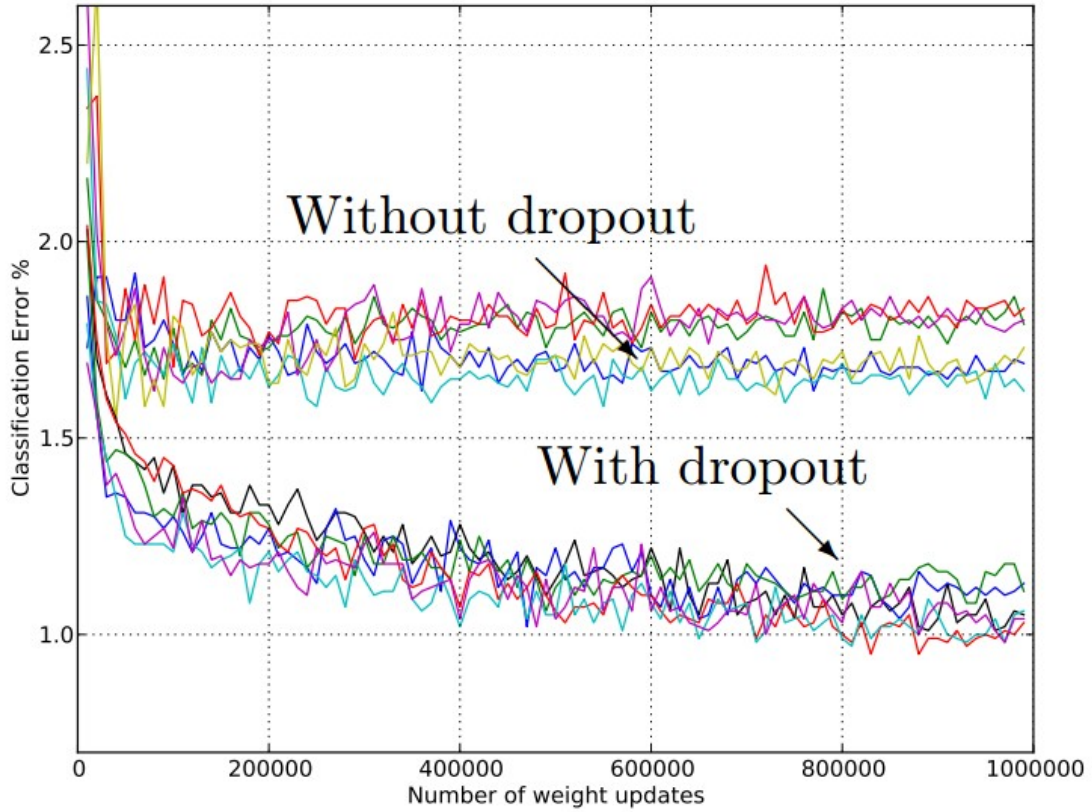
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

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

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

8. Dropout Layers, 2014 Nitish Srivastava and et al

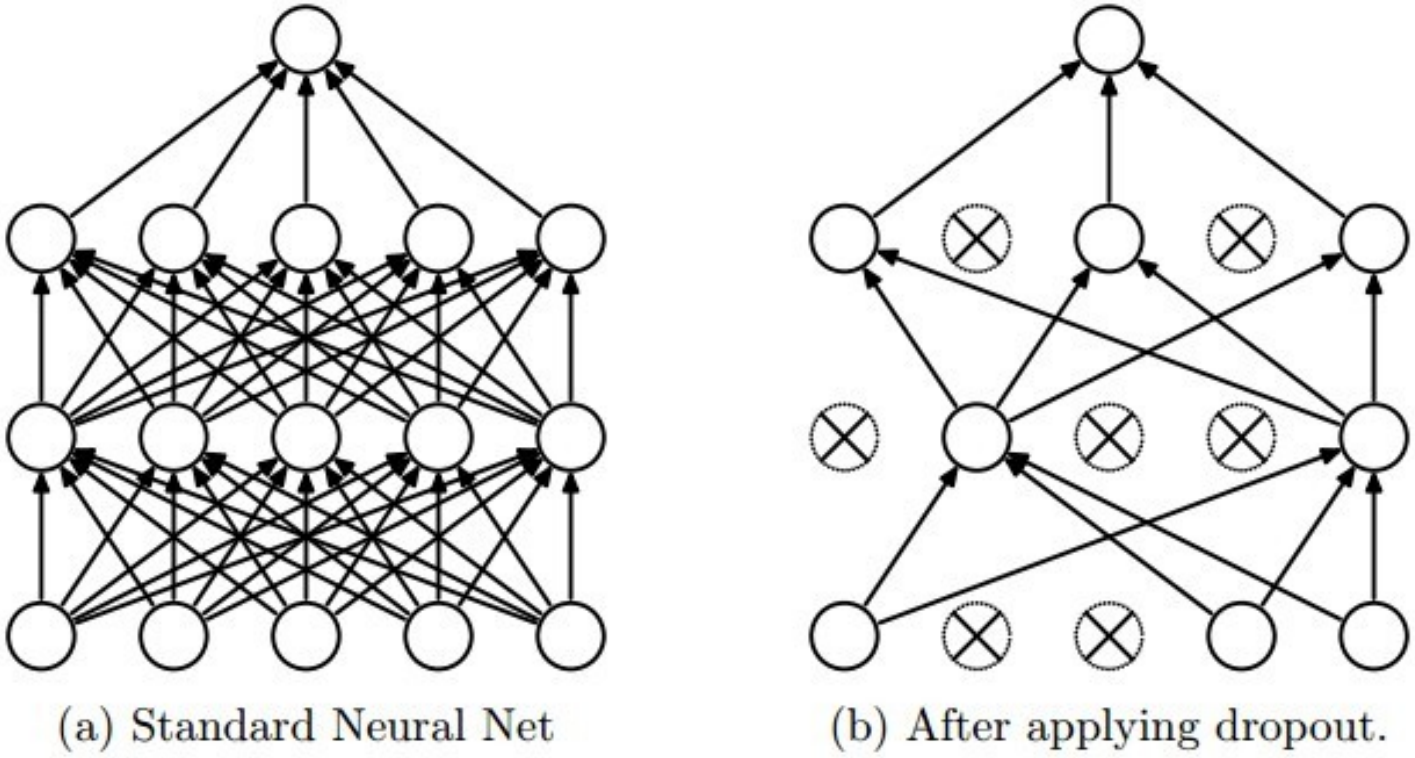
- Dropout is a technique used to improve over-fit on neural networks.
  you should use Dropout along with other techniques like L2 Regularization



As other regularization techniques the use of dropout also make the training loss error a little worse. But that's the idea, basically we want to trade training performance for more generalization. Remember that's more capacity you add on your model (More layers, or more neurons) more prone to over-fit it becomes.

How it works?

Basically during training a percentage of neurons on a particular layer will be deactivated. This improve generalization because force your layer to learn with different neurons the same "concept".
During the prediction phase the dropout is deactivated



(a) Standard Neural Net

(b) After applying dropout.

Where to use Dropout layers?

Normally some deep learning models use Dropout on the fully connected layers, but is also possible to use dropout after the max-pooling layers, creating some kind of image noise augmentation.

Implementation

In order to implement this neuron deactivation, we create a mask(zeros and ones) during forward propagation. This mask is applied to the layer outputs during training and cached for future use on back-propagation. As explained before this dropout mask is used only during training.
On the backward propagation we're interested on the neurons that was activated (we need to save mask from forward propagation). Now with those neurons selected we just back-propagate. The dropout layer has no learnable parameters, just it's input (X). During back-propagation we just return "dx".

Dropout in training and testing

When applying dropout in artificial neural networks, one needs to compensate for the fact that at training time a portion of the neurons were deactivated. To do so, there exist two common strategies:

1. **scaling the activation at test time (suggested in the reference paper)**

A unit at training time that is present with probability p and is connected to units in the next layer with weights w. Right: At test time, the unit is always present and the weights are multiplied by p.

For example if 30% of neurons are deactivated then the output of neurons at test time should be decreased to 30% of their complete values.

2. **Inverting the dropout during the training phase.**

For example if 30% of neurons are randomly deactivated in training time then the output of activated neurons should be divided to (1-0.3).

9. Network in Network Layers

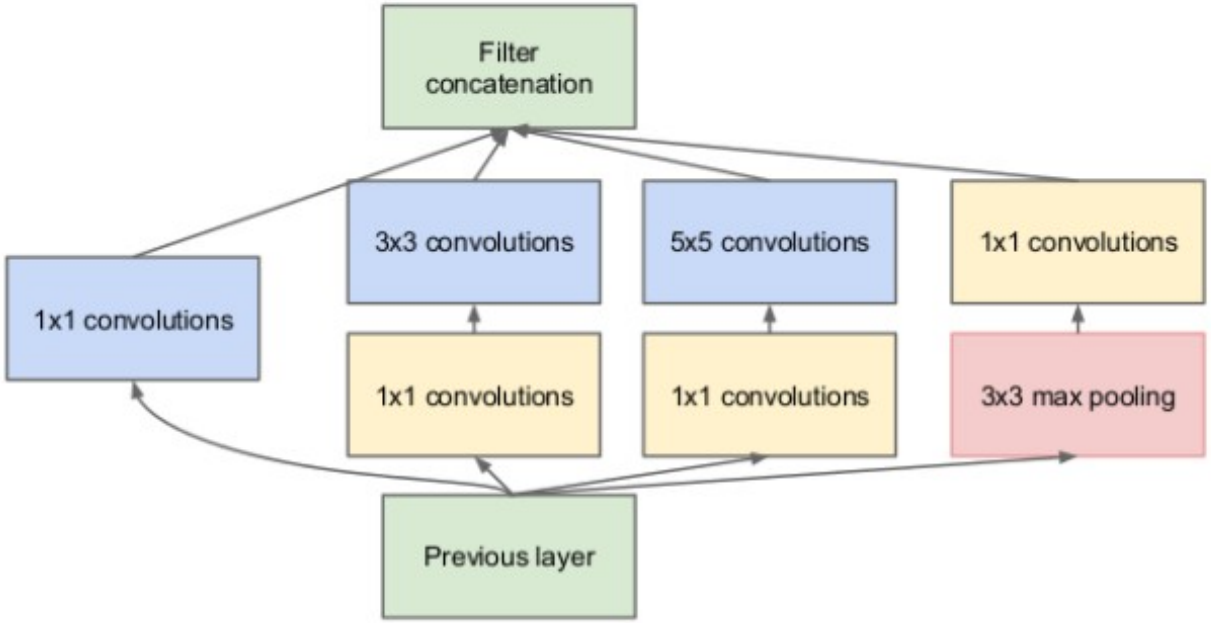
A network in network layer refers to a conv. layer where a 1 x 1 size filter is used.

Assume there are N1>N feature maps in a certain layer:

Actually, 1x1xN convolutions are used to reduce the number of feature maps from N1 to N before applying the expensive 3x3 and 5x5 convolutions to them.

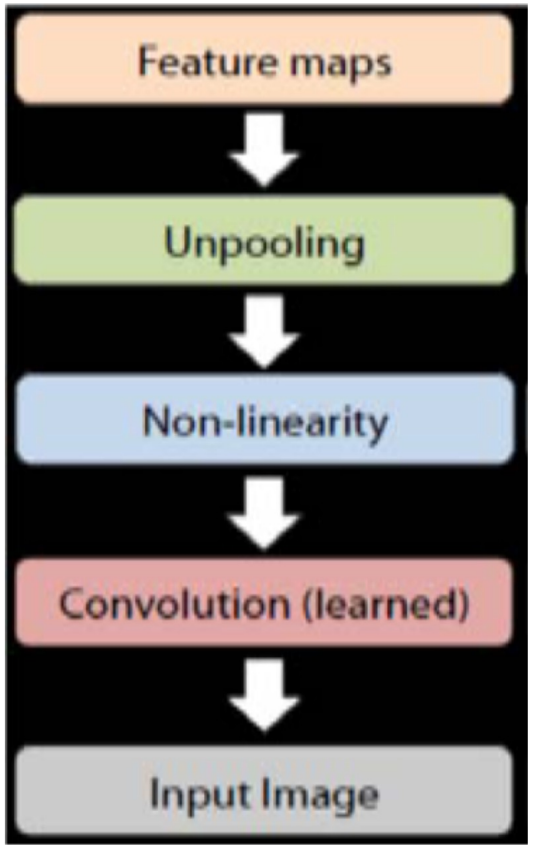
An Inception module

In Google Net



10. Deconvolutional Networks (Deconvolutional and unpooling layers)

Deconvolutional networks are convolutional neural networks (CNN) that work in a reversed process.



Applications:

1. To analyze a certain CNN

to study that which parts of the input image or signal have been amplified through convolution and pooling layers.

2.Utilized in pattern generation purposes particularly in Generative part of GANs.

3. Also, for detection and segmentation purposes.

Reference

[Zeiler11] M. Zeiler, G. Taylor, and R. Fergus: **Adaptive Deconvolutional Networks for Mid and High Level Feature Learning. ICCV 2011**

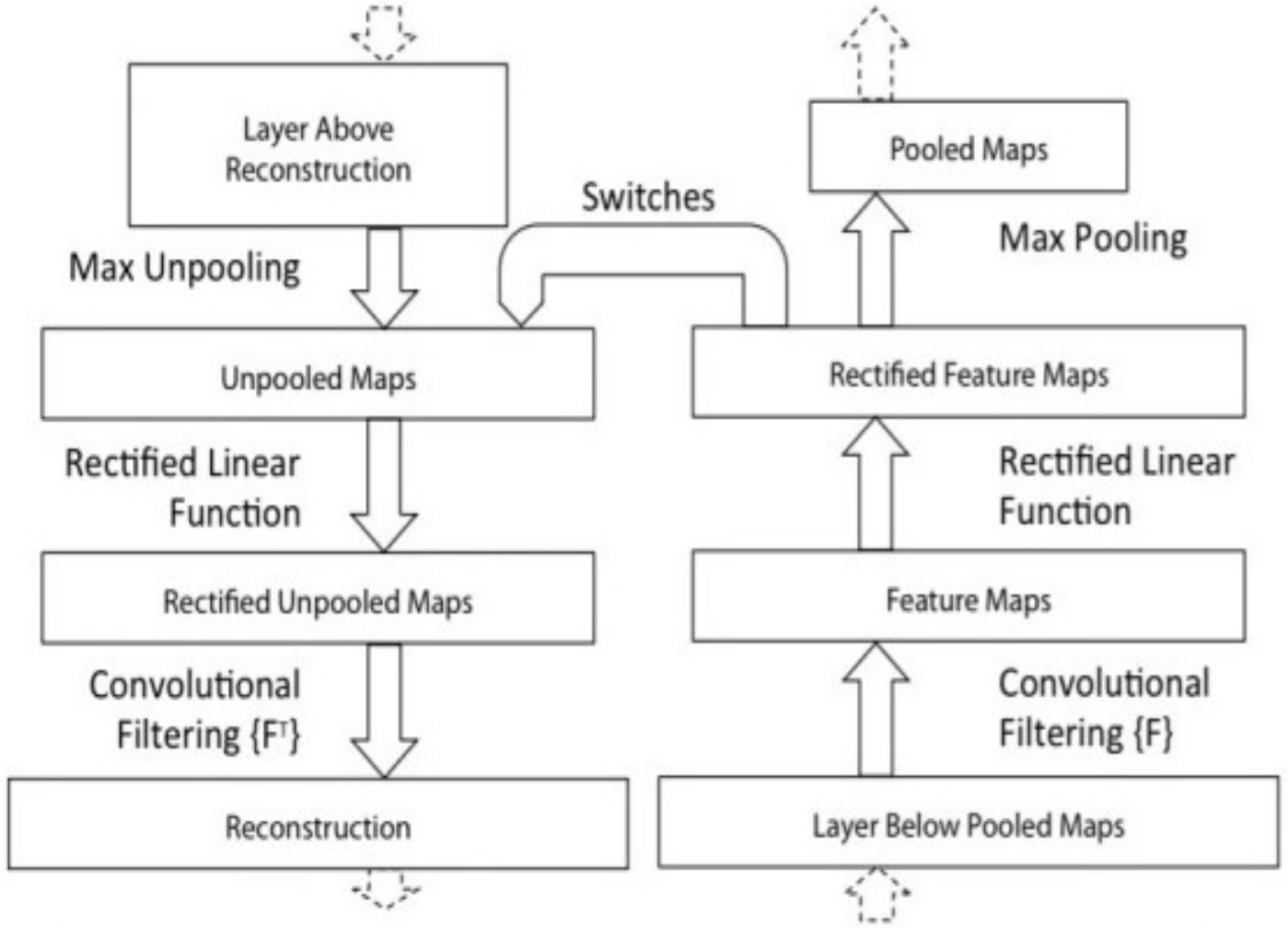
Main idea and operations

Main idea

- Mapping activations at high layers back to the input pixel space
- Showing what input patterns originally caused a given activation in the feature maps

Same operations as CNNs, but in reverse

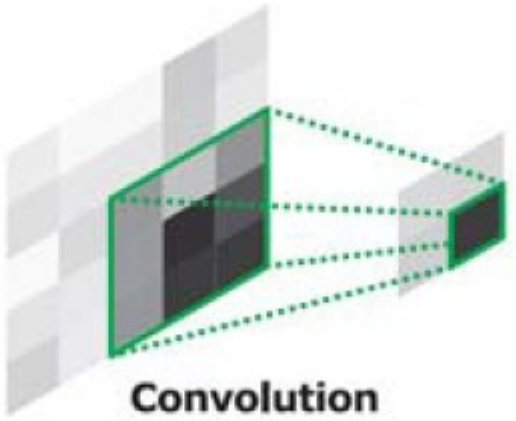
- Unpool feature maps
- Convolve unpooled maps

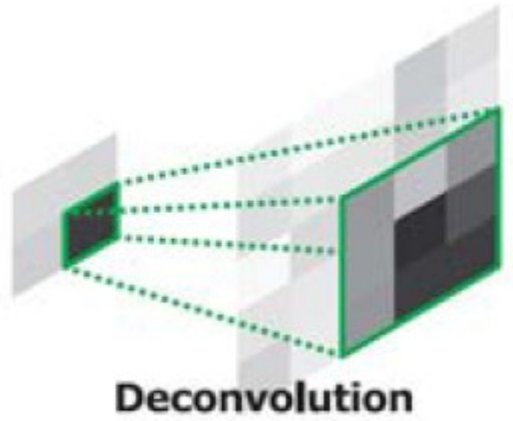


Deconvolution layers

After the convolution section (i.e., Convolution layer, Relu, Pooling), it is common to have more than one feature map output, which would be treated as input channels to successive layers (Deconv..). How could these feature maps be combined together in order to achieve the activation map with same resolution as original input?

- Deconvolution
  - The size of output layer is larger than that of input.
  - Densify sparse activations
  - Conceptually similar to convolution
  - Bases to reconstruct shape

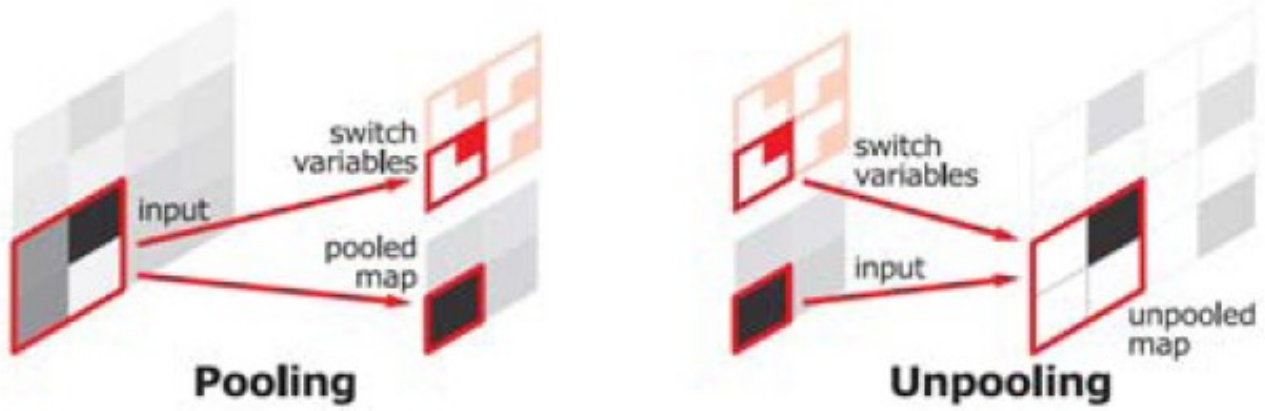




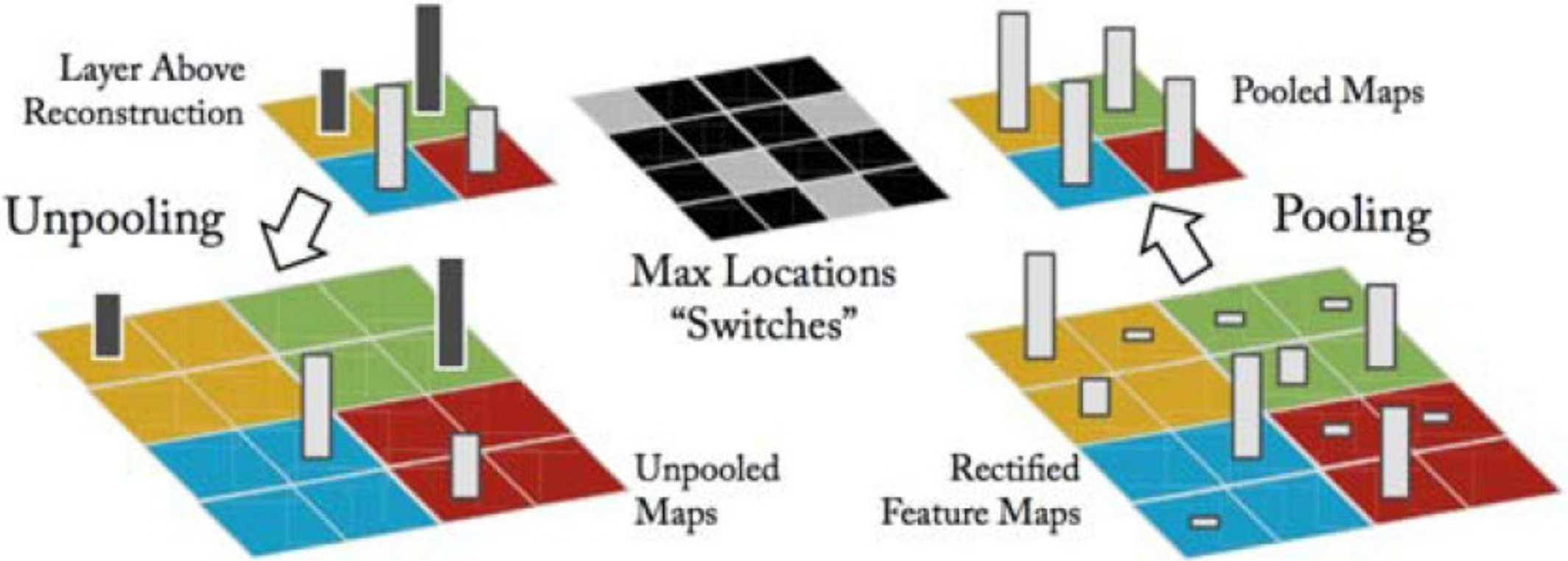
Unpooling Layers

While during the unpooling stage, the activations are restored back to the locations of maximum activation selections, which makes sense, but what about the remaining activations? Do those remaining activations need to be restored as well or interpolated in some way or just filled as zeros in unpooled map.

- Unpooling
  - Place activations to pooled location
  - Preserve structure of activations



- Unpooling
  - Approximate inverse: Max pooling operation is non-invertible
  - Switch variables: recording the locations of maxima



Applied Techniques in CNNs

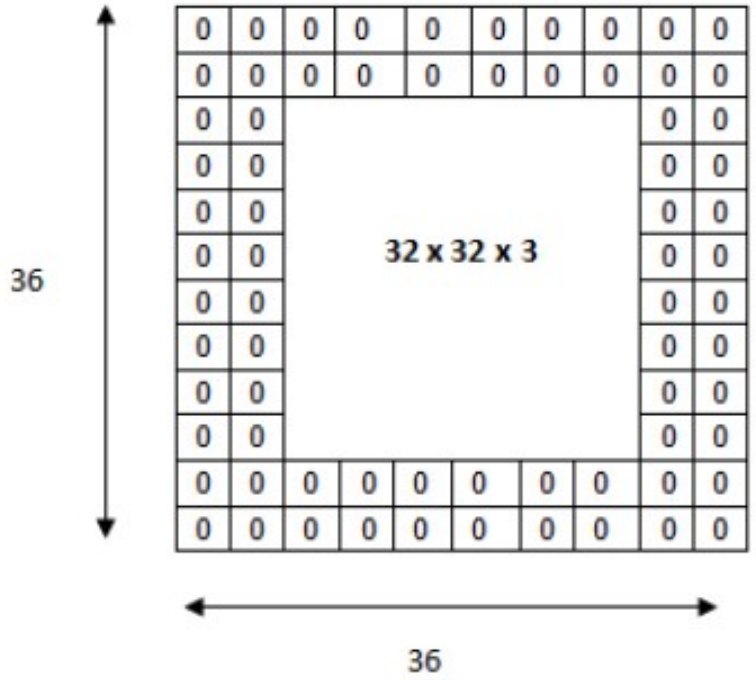
- In convolution layers

1. Padding (to save spatial size after convolution)
2. Stride>1 (to provide a fast convolution and dimensionality reduction)

- In Learning process

1. Data Augmentation (to increase the training set)
2. Transfer Learning (to use a pertained CNN)

Padding



The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x3 output volume.

If you have a stride of 1 and if you set the size of zero padding to

$$Zero\ Padding = \frac{(K-1)}{2}$$

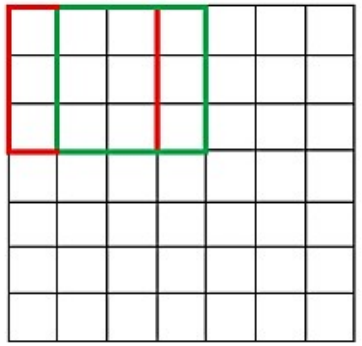
where K is the filter size, then the input and output volume will always have the same spatial dimensions.
The formula for calculating the output size for any given convolutional layer is

$$O = \frac{(W-K+2P)}{S} + 1$$

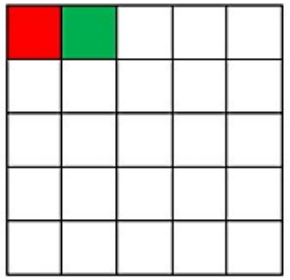
where O is the output height/length, W is the input height/length, K is the filter size, P is the padding, and S is the stride.

Stride

7 x 7 Input Volume

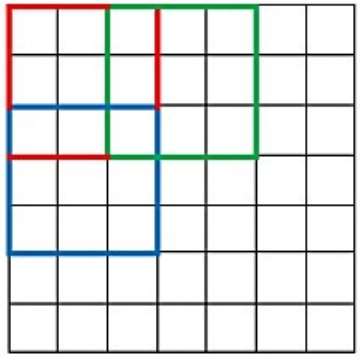


5 x 5 Output Volume

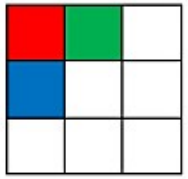


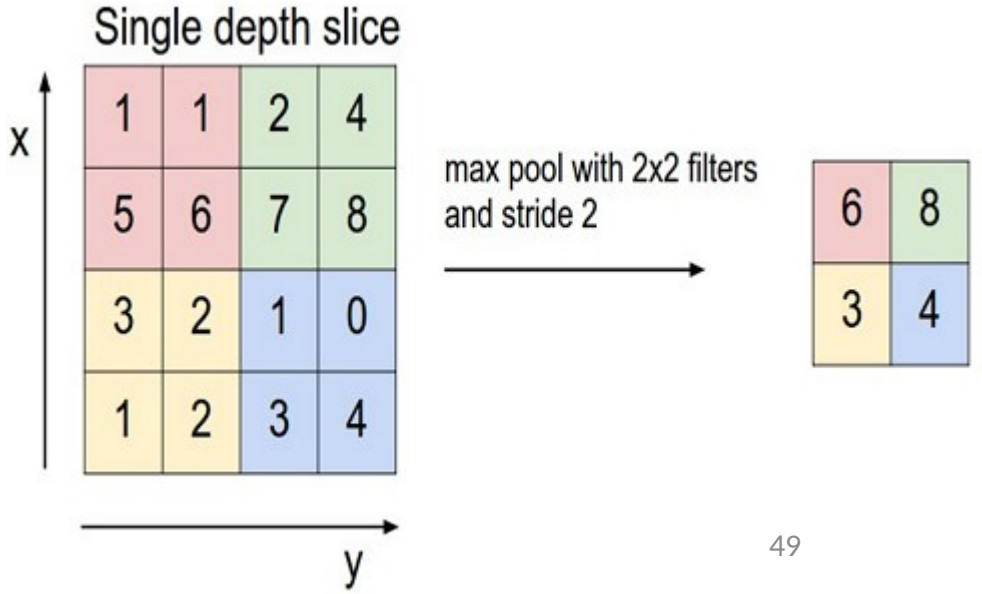
Same old, same old, right? See if you can try to guess what will happen to the output volume as the stride increases to 2.

7 x 7 Input Volume



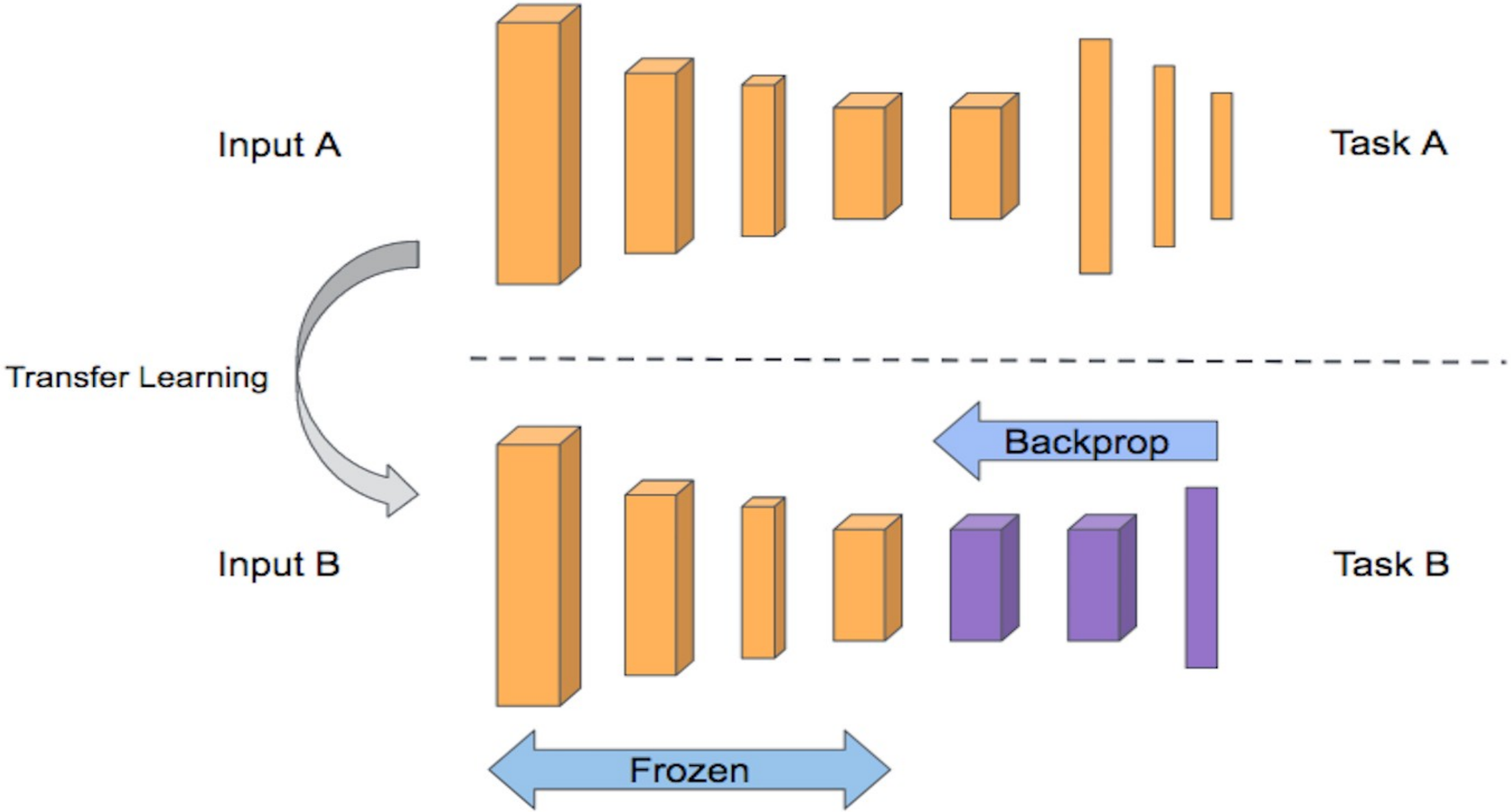
3 x 3 Output Volume

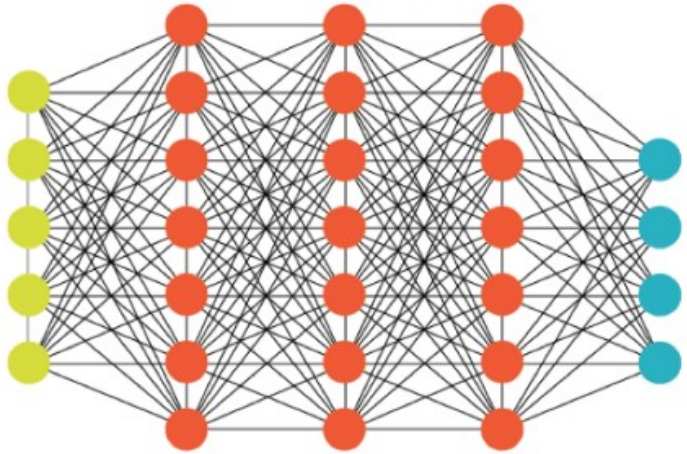




Transfer Learning

- Transfer learning is the process of taking a pre-trained model (the weights and parameters of a network that has been trained on a large dataset by somebody else)
  and
- “fine-tuning” the model with your own dataset. The idea is that this pre-trained model will act as a feature extractor
- You will remove the last layers of the network and replace it with your own classifier (depending on what your problem space is). You then freeze the weights of all the other layers and train the network normally (Freezing the layers means not changing the weights during gradient descent/optimization).

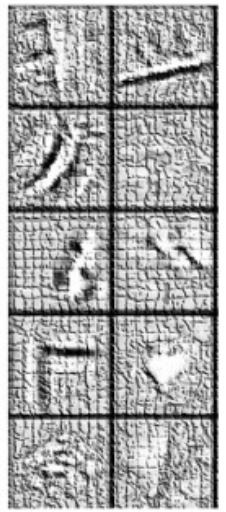

Input A
Task A
Transfer Learning
Backprop
Input B
Task B
Frozen



SoftMax

GIANT PANDA 0.9
RED PANDA 0.05
RACCOON 0.01
...

DeepImageFeaturizer



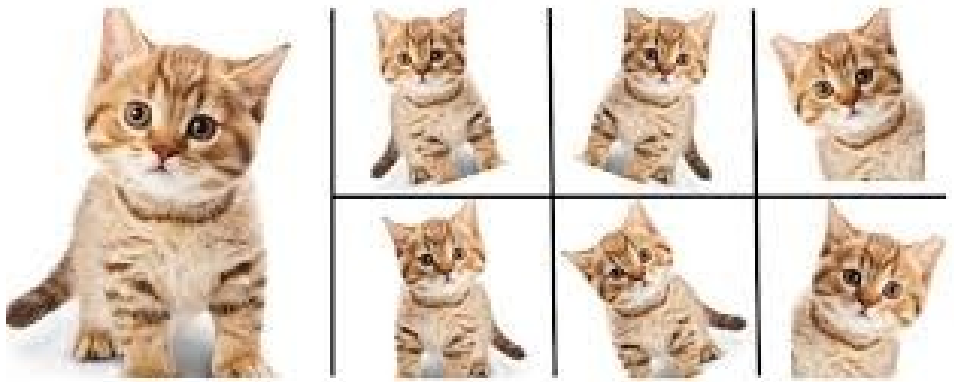
Classifier

Chihuahua

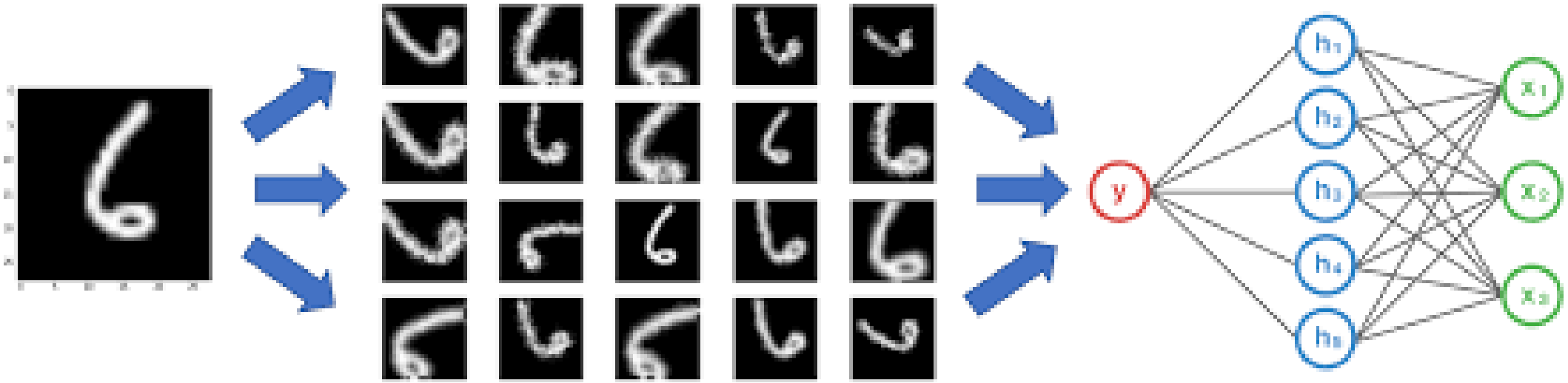
Data Augmentation Techniques

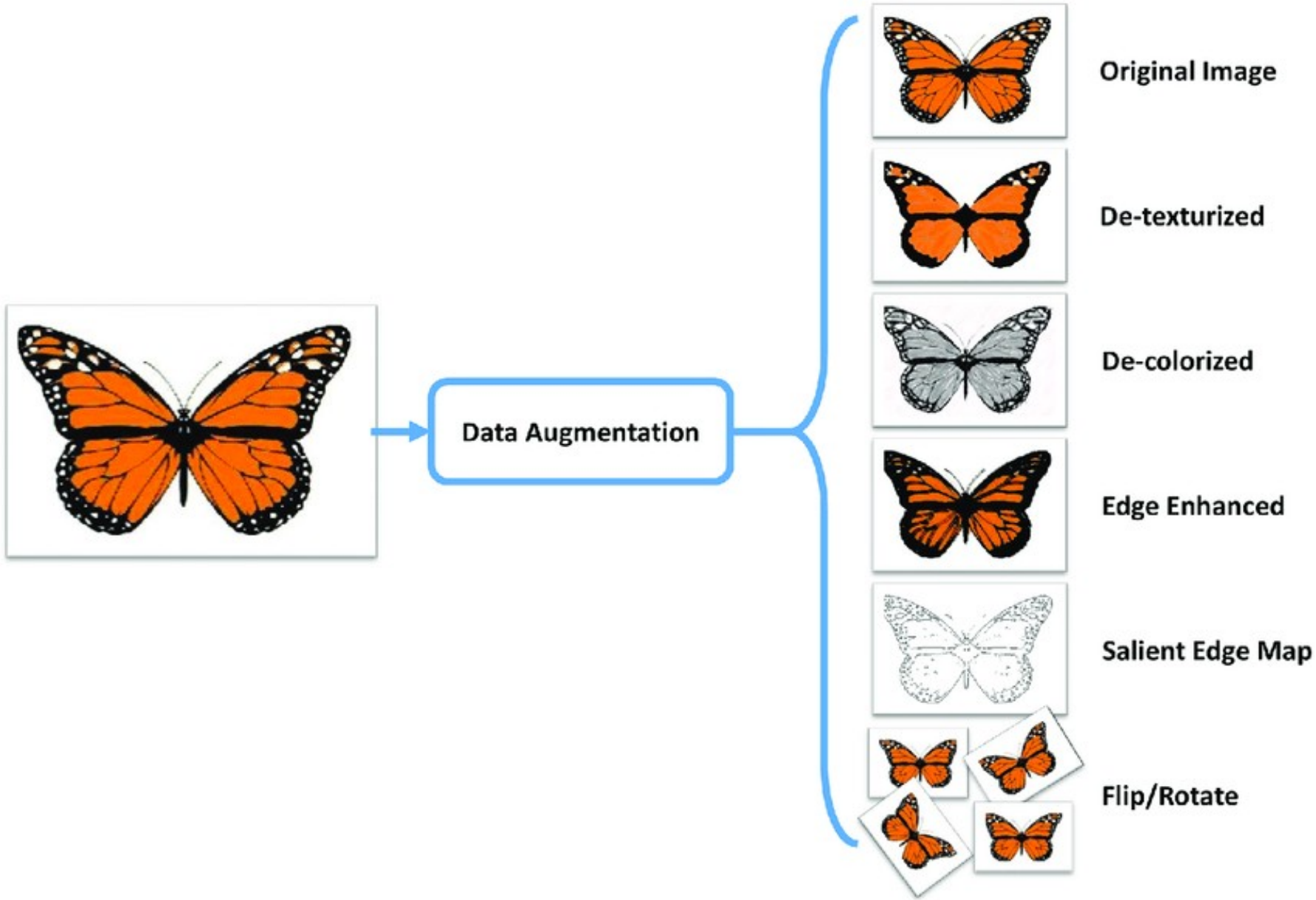
By now, we're all probably numb to the importance of data in ConvNets, so let's talk about ways that you can make your existing dataset even larger, just with a couple easy transformations. Like we've mentioned before, when a computer takes an image as an input, it will take in an array of pixel values. Let's say that the whole image is shifted left by 1 pixel. To you and me, this change is imperceptible. However, to a computer, this shift can be fairly significant as the classification or label of the image doesn't change, while the array does. Approaches that alter the training data in ways that change the array representation while keeping the label the same are known as data augmentation techniques. They are a way to artificially expand your dataset. Some popular augmentations people use are grayscales, horizontal flips, vertical flips, random crops, color jitters, translations, rotations, and much more.

By applying just a couple of these transformations to your training data, you can easily double or triple the number of training examples.



Enlarge your Dataset

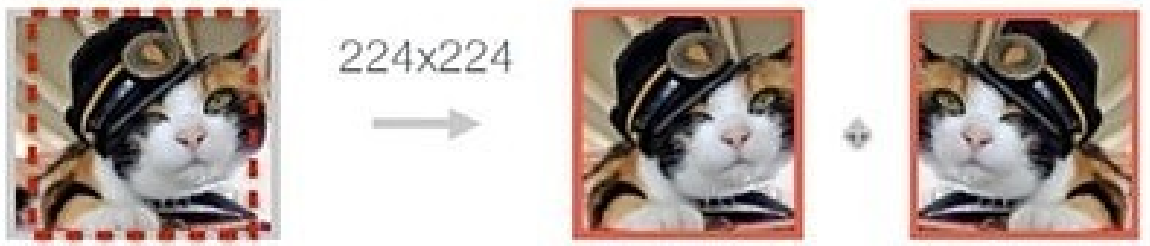



Data Augmentation
Original Image
De-texturized
De-colorized
Edge Enhanced
Salient Edge Map
Flip/Rotate

Data Augmentation:

a. No augmentation (= 1 image)

b. Flip augmentation (= 2 images)



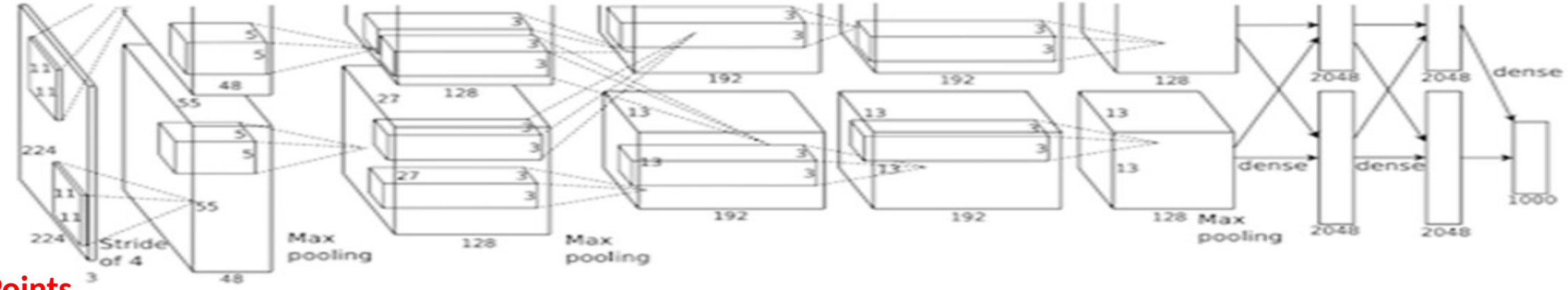
c. Crop+Flip augmentation (= 10 images)

Some Important Developments in CNNs

(Tutorial from Deshpande - UCLA)

1. Alex-Net (Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton - 2012)
2. ZF Net (Matthew Zeiler and Rob Fergus from NYU – 2013)
3. VGG Net (Karen Simonyan and Andrew Zisserman of Oxford- 2014)
4. Google LeNet (Google company 2015)
5. Microsoft ResNet (Microsoft company 2015)
6. **Efficient Net**

1. Alex-Net (Oxford-2012)



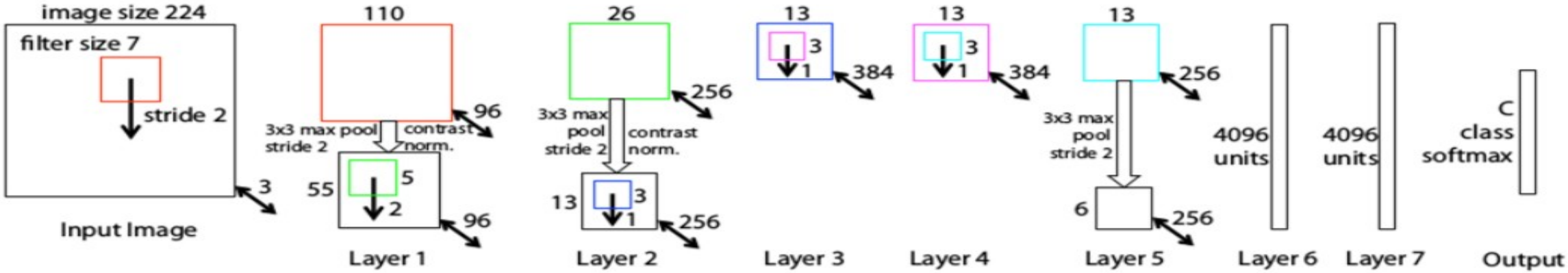
Main Points

1. Trained the network on ImageNet data with over 15 million images from a total of over 22,000 categories.
2. Used Relu for the nonlinearity functions.
3. Used data augmentation techniques to increase number of data points.
4. Implemented dropout layers to avoid overfitting to the training data.
5. Trained the model using batch stochastic gradient descent.
6. Trained on two GTX 580 GPUs for five to six days.

Why It's Important

- Record breaking performance in the competition.
- This was the first time a model performed so well on a historically difficult ImageNet dataset.
- Utilizing technique like data augmentation and dropout.

2. ZF Net (NYU-2013)



ZF Net Architecture

Main Points

Very similar architecture to AlexNet, except for a few minor modifications:

1. AlexNet trained on 15 million images, while ZF Net trained on only 1.3 million images.
2. Instead of using 11x11 sized filters in the first layer , ZF Net used filters of size 7x7 and a decreased stride value.
3. As the network grows, we also see a rise in the number of filters used.
4. Used Relus for their activation functions, cross-entropy loss for the error function, and trained using batch stochastic gradient descent.
5. Trained on a GTX 580 GPU for twelve days.
6. Developed a visualization technique named Deconvolutional Network(the opposite of what a convolutional layer does)

Why It's Important

- ZF Net was the winner of the competition in 2013.
- It provided great intuition as to the workings on CNNs.
- A visualization approach(Deconvolutional Network) described helps not only to explain the inner workings of CNNs but also provides insight for improvements to network architectures.

3. VGG Net (2014)

A 19 layer CNN that strictly used 3x3 filters with stride and pad of 1 along with 2x2 MaxPooling layers.

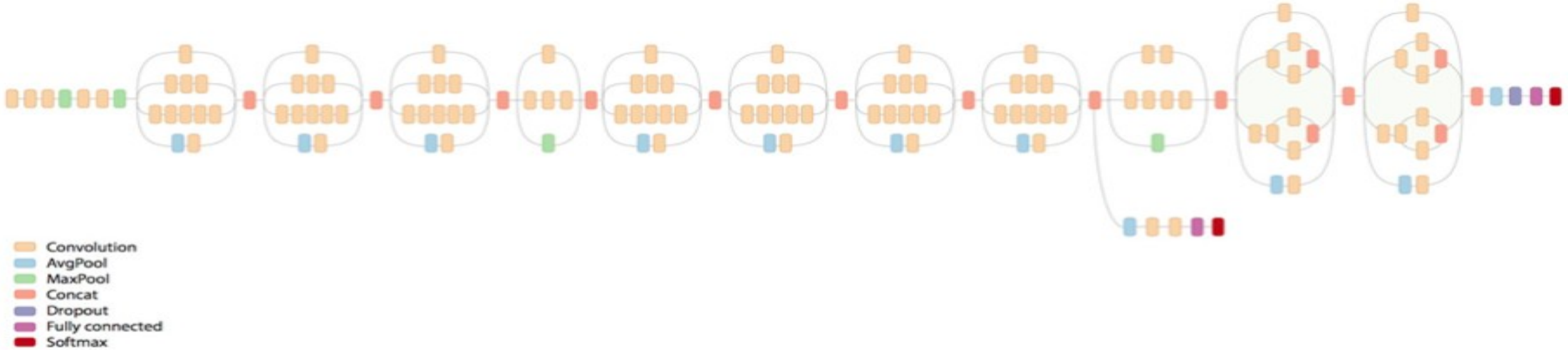
Main Points:

1. The use of only 3x3 sized filters is quite different from AlexNet's 11x11 filters in the first layer and ZF Net's 7x7 filters.
2. With two conv layers and two ReLU layers, instead of one.
3. The network is deeper than former developments.
   The Number of filters are increased while we go to more depth layers.
4. Data augmentation technique has been used during training.
5. Used ReLU layers after each conv layer and trained with batch gradient descent.
6. Trained on 4 Nvidia Titan Black GPUs for two to three weeks.

Why It's Important

its most important results is: Keep it deep. Keep it simple.

4. GoogLeNet (2015)



Main Points

1. over 100 layers in total!
2. Without any fully connected layers!
3. They use an Mean-pool instead, to go from a 7x7x1024 volume to a 1x1x1024 volume.
4. This saves a huge number of parameters. Uses 12x fewer parameters than AlexNet.
   Trained on “a few high-end GPUs within a week”.

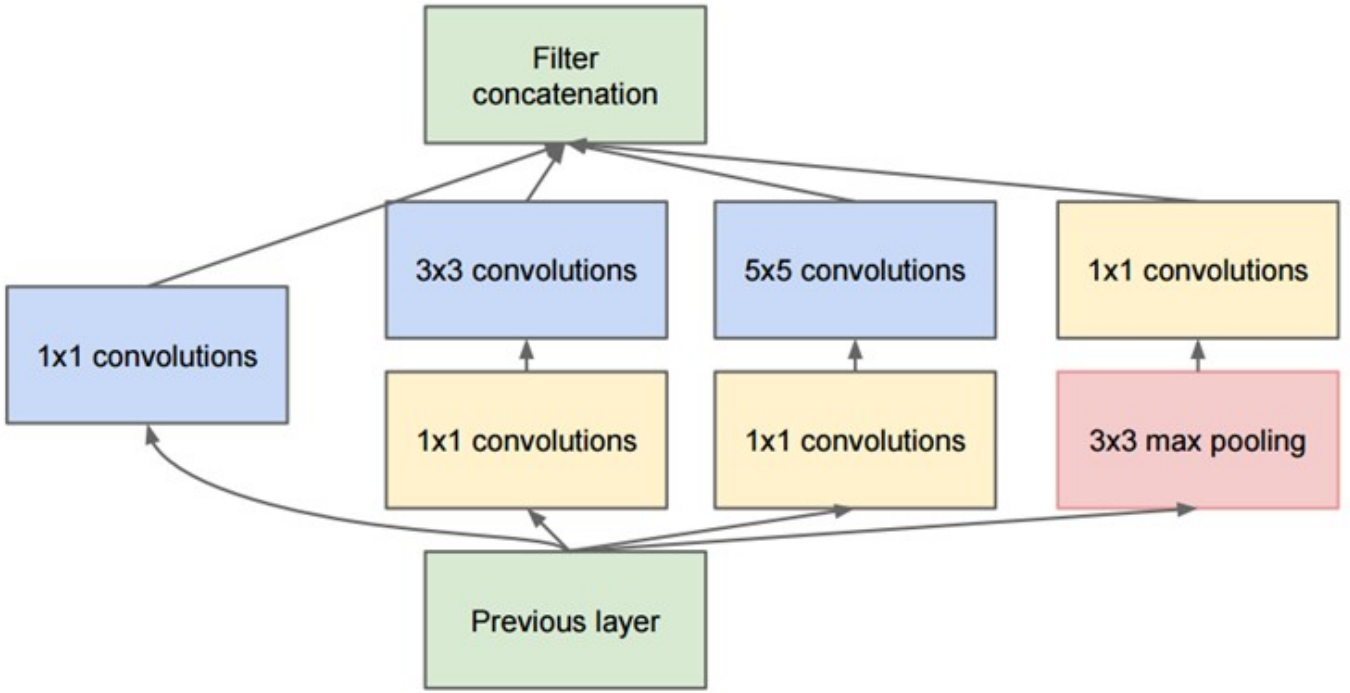
Why It’s Important

- the idea that CNN layers didn’t always have to be stacked up sequentially.
- there is a creative structuring of layers which lead to improved performance and computationally efficiency.

The key concept :Inception Module

Inception Module

When we first take a look at the structure of GoogLeNet, we notice immediately that not everything is happening
sequentially, as seen in previous architectures. We have pieces of the network that are happening in parallel.



Full Inception module

5. Microsoft ResNet (2015)

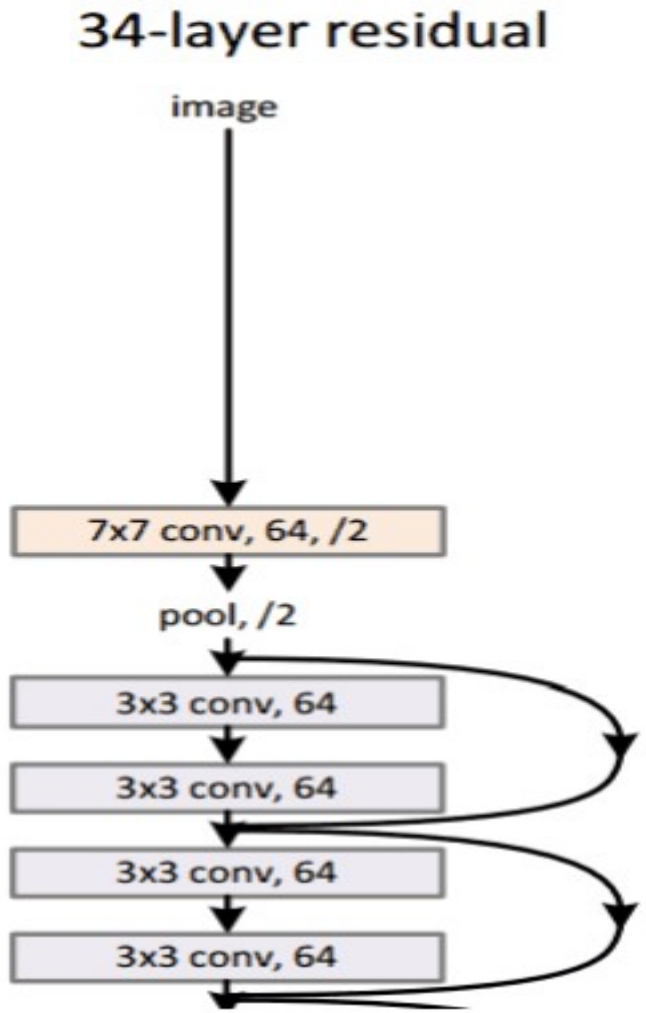
Res-Net is a new 152 layer network architecture that set new records in classification, detection, and localization through one incredible architecture.

Main Points

1. “Ultra-deep” , 152 layers…
2. After only the *first 2* layers, the spatial size gets compressed from an input volume of 224x224 to a 56x56 volume.
3. Authors claim that a naïve increase of layers in plain nets result in higher training and test error
4. The group tried a 1202-layer network, but got a lower test accuracy, presumably due to overfitting.
5. Trained on an 8 GPU machine for two to three weeks.

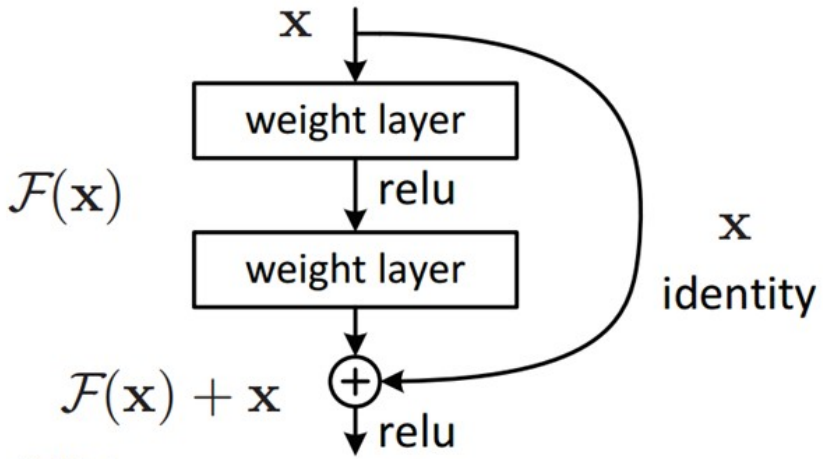
Why It’s Important

3.6% error rate. That itself should be enough to convince you.
The ResNet model is one of the best CNNs architecture that we currently have and is a great innovation for the idea of residual learning.



The key concept : **Residual block**

Residual Block
The idea behind a residual block is that you have your input x go through conv-relu-conv series. This will give you some F(x). That result is then added to the original input x. Let's call that H(x) = F(x) + x. In traditional CNNs, your H(x) would just be equal to F(x) right? So, instead of just computing that transformation (straight from x to F(x)), we're computing the term that you have to *add*, *F(x)*, *to your input*, *x. Basically, the mini module shown below is computing a "delta" or a slight change to the* original input x to get a slightly altered representation (When we think of traditional CNNs, we go from x to F(x) which is a completely new representation that doesn't keep any information about the original x). The authors believe that "it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping".



A residual block

Another reason for why this residual block might be effective is that during the backward pass of back propagation, the gradient.
will flow easily through the graph because we have addition operations, which distributes the gradient.

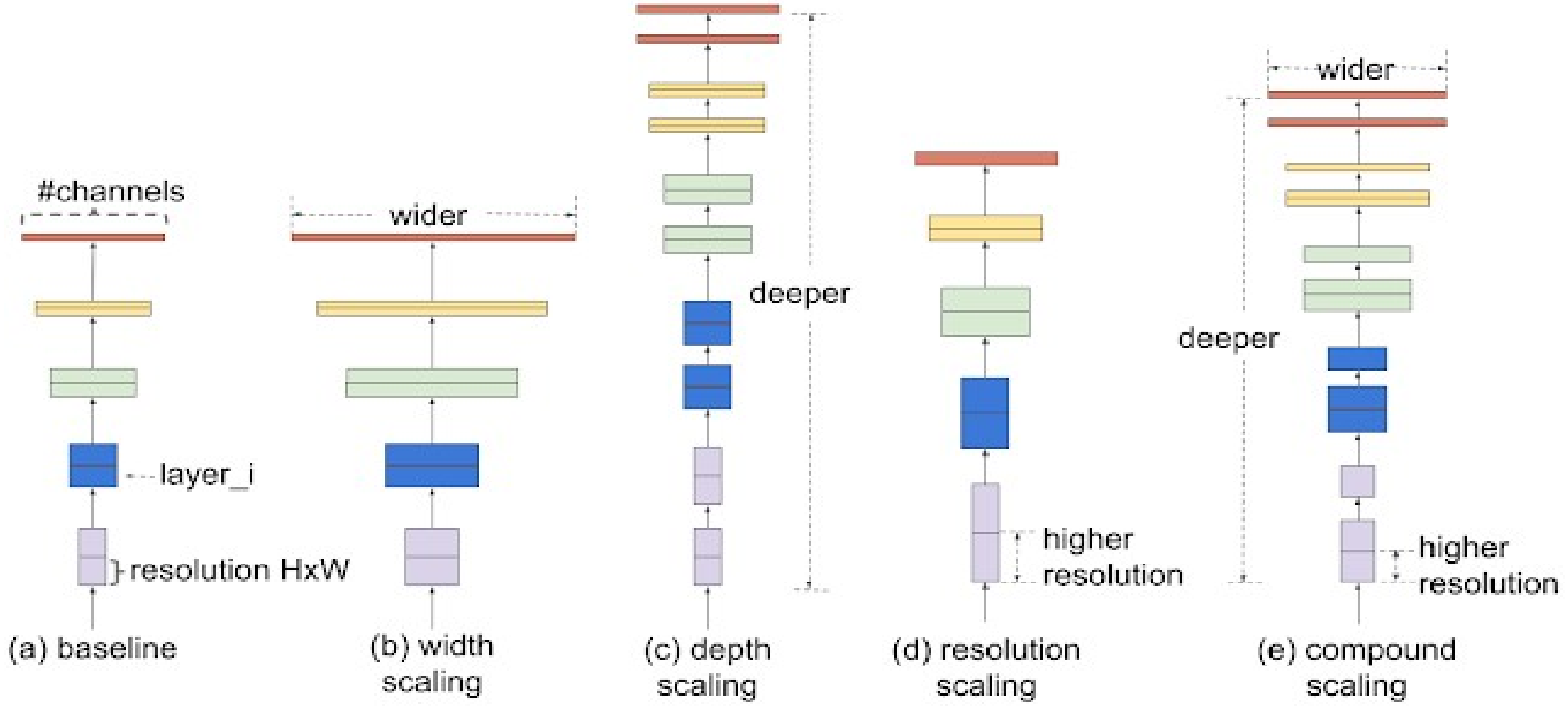
6. Efficient Net(2020)

Rethinking Model Scaling for Convolutional Neural Networks

From the author of paper, "EfficientNet : Rethinking Model Scaling for Convolutional Neural Networks" :

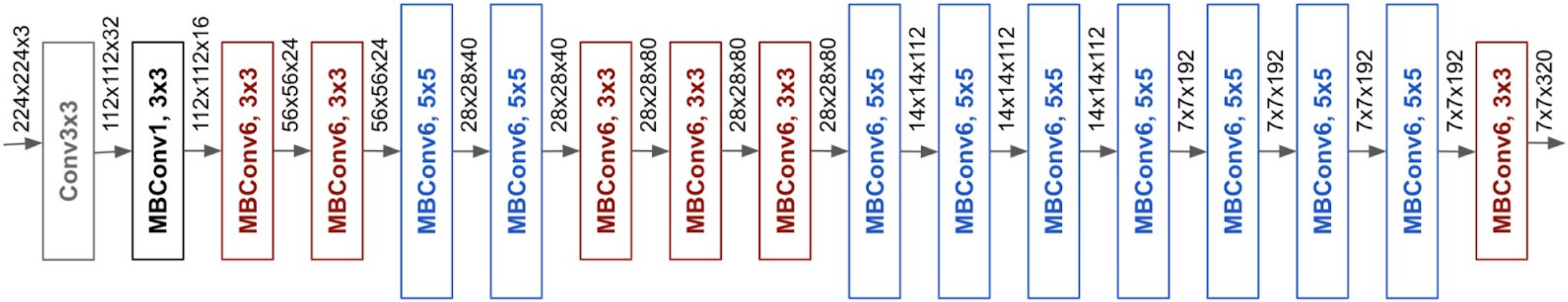
- we propose a novel model scaling method that uses a simple yet highly effective *compound coefficient* to scale up CNNs in a more structured manner. Unlike conventional approaches that arbitrarily scale network dimensions, such as width, depth and resolution, our method uniformly scales each dimension with a fixed set of scaling coefficients.
- we have developed a family of models, called EfficientNets, which superpass state-of-the-art accuracy with up to 10x better efficiency (smaller and faster).

EfficientNet



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

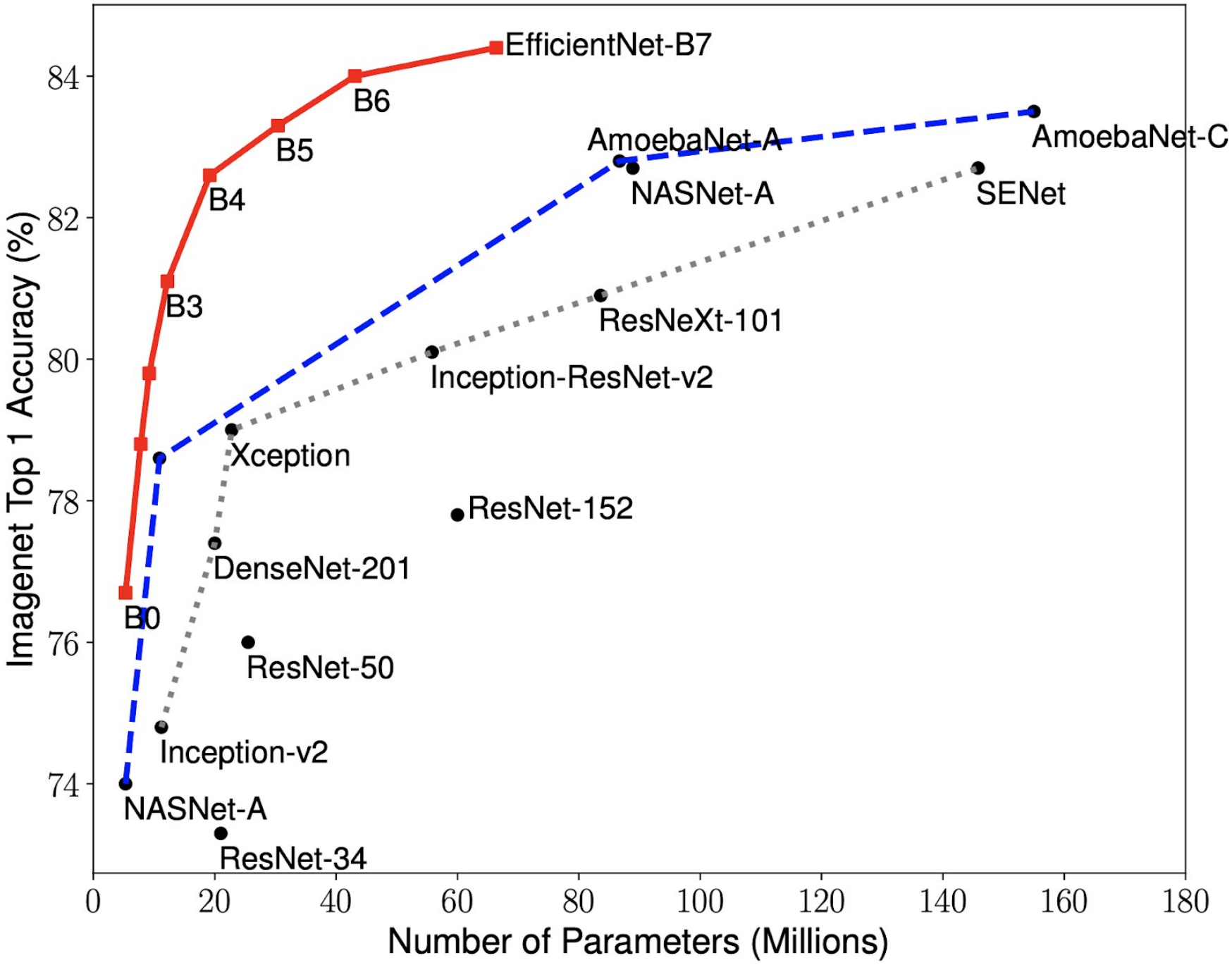
EfficientNet



The architecture for the baseline network EfficientNet-B0

EfficientNe

Model Size vs.
Accuracy Comparison.



End of Chapter3

Thank you