LEARNING TENSORFLOW

INDEX

1.MACHINE LEARNING AND DEEP LEANING

Supervised Learning

Decision tree

Cart

CNN

RNN

Reinforced learning

LASSO

Logistic Regression

Unspervised Learning

Boltzman Machine

Apriori

KNN

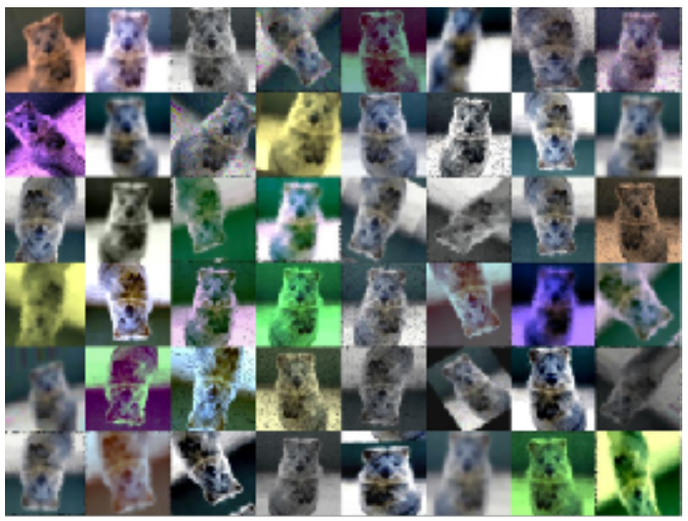
Hebbian Learning

GAN

K-means clustering

DATA SET AND PREPROCESSING

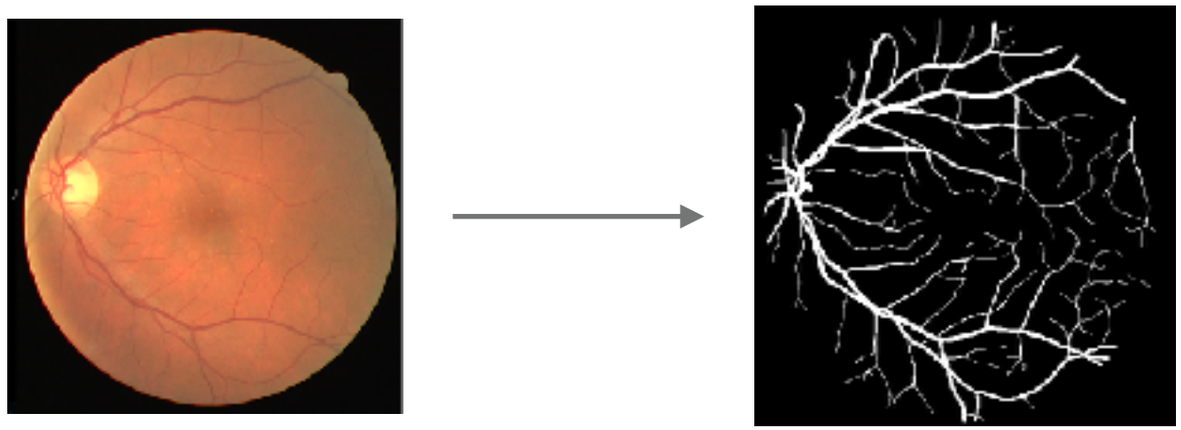
- Augmentation



- Normalize

$$\frac{x\text{-}min(x)}{max(x)\text{-}min(x)}$$

- Preprocessing



DATASET

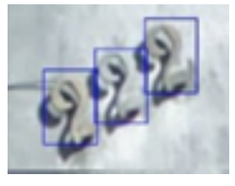
1.Mnist

http://yann.lecun.com/exdb/mnist/

```
from tensorflow.contrib.learn.python.learn.datasets.mnist import read_data_sets
```

2.SVHN

http://ufldl.stanford.edu/housenumbers/



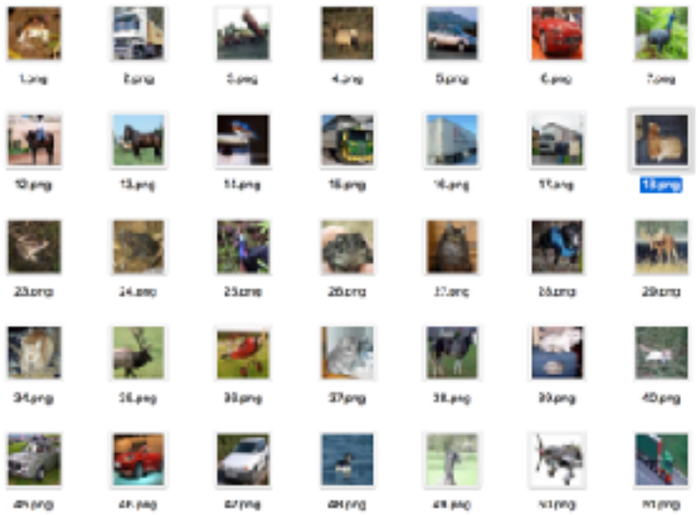
3.CIFAR 10 / CIFAR 100

https://www.cs.toronto.edu/~kriz/cifar.html

4.COCO

http://mscoco.org/

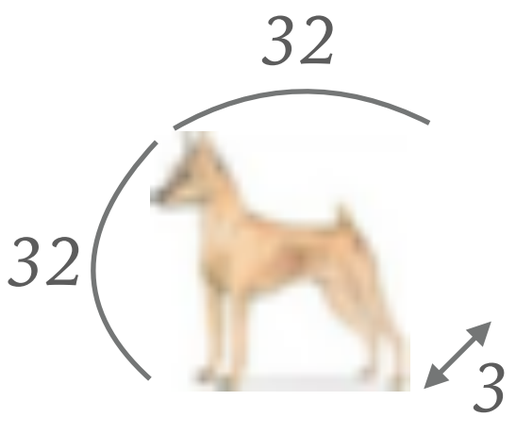
CIFAR-10



Bus *Horse* *Horse* *Bird* *Airplane* *Dog* *Cat* *Horse* *Deer* *Car*

10 Index

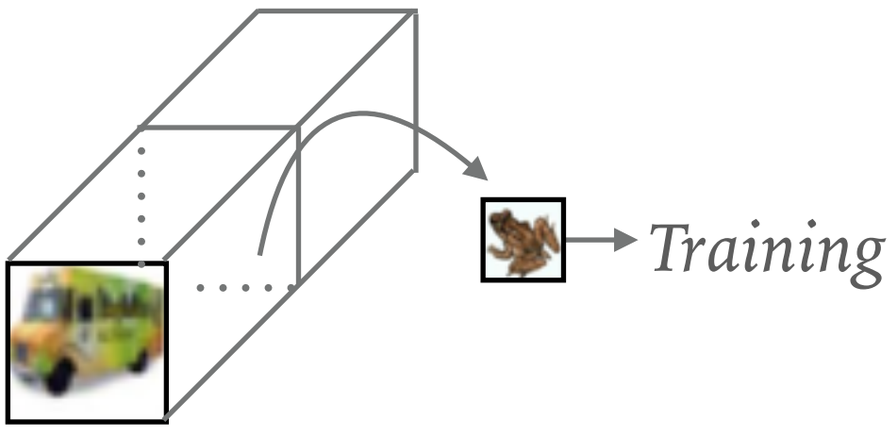
The Number of DataSet : 50000 *32x32 Pixel , color Image*



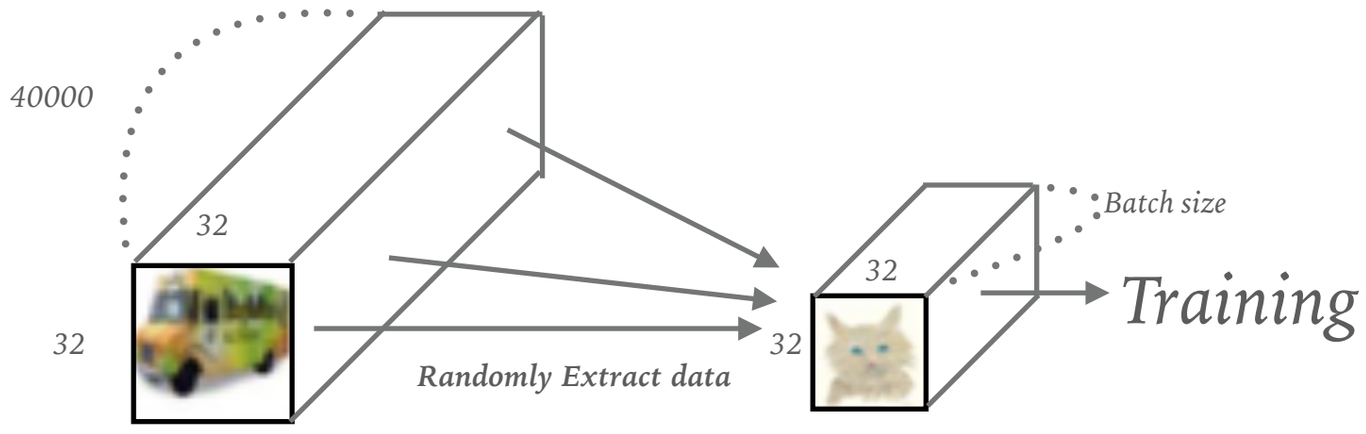
Color Image typically has 3 channel of pixel (R ,G B)

BATCH TRAINING

1.

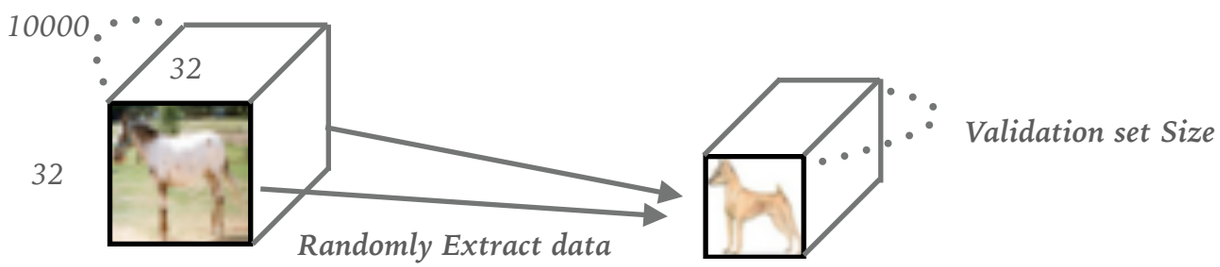


2. Batch Training



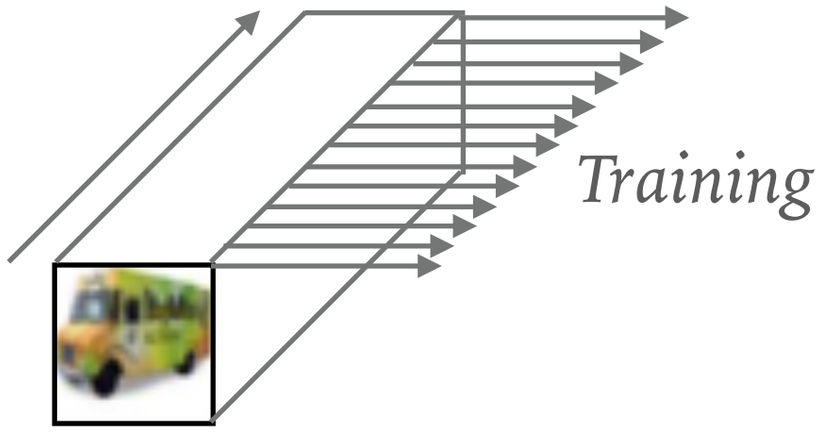
Train Data Set

3. Validation set and Test set



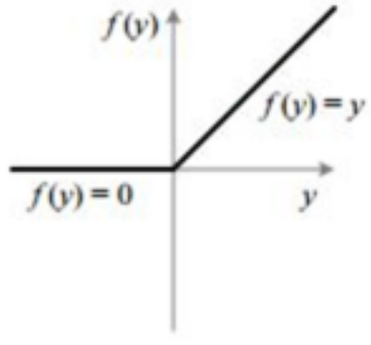
Test Data Set *Validation Set*

4. epoch



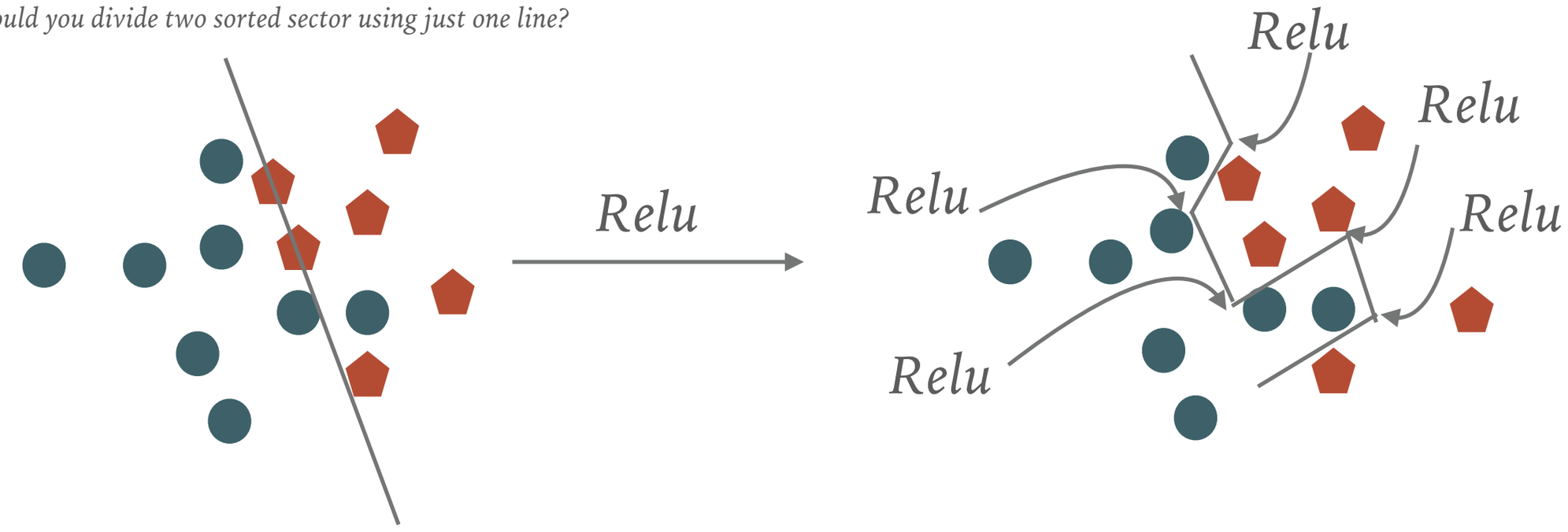
ACTIVATION FUNCTION

Let's talk about Relu



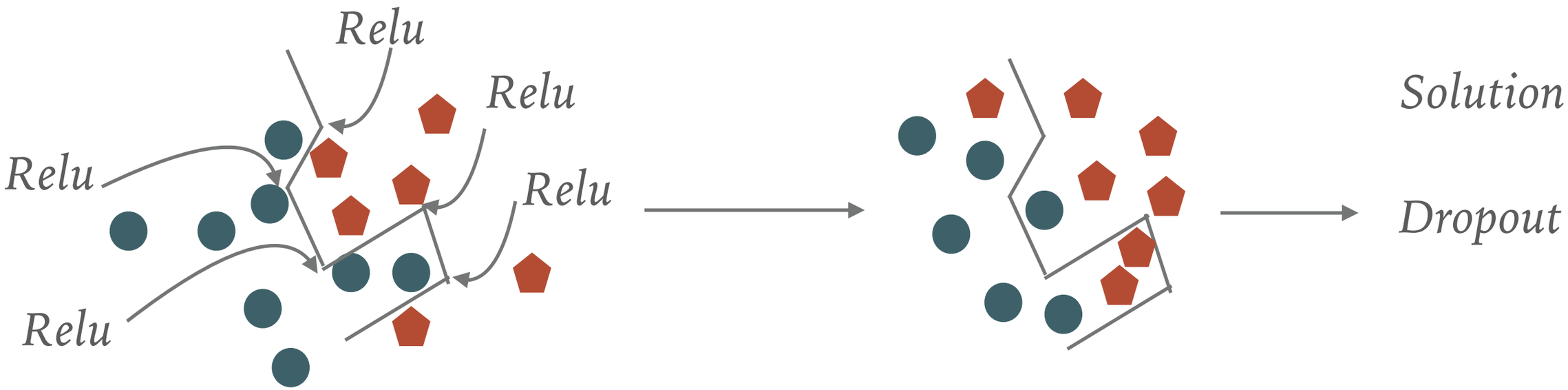
It's simple but It's Important

Could you divide two sorted sector using just one line?



OVERFITTING PROBLEM

More layer .. more layer



Training Set

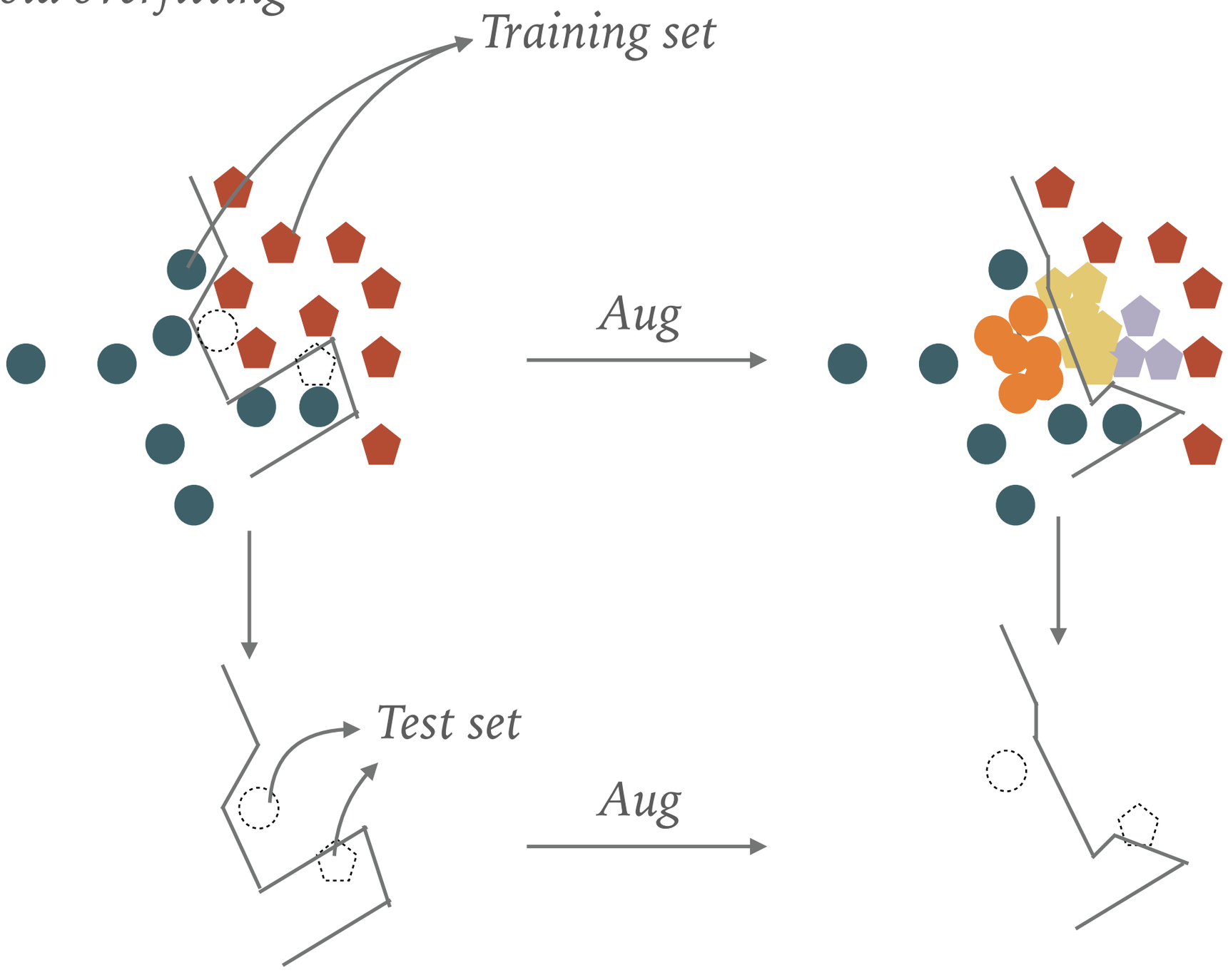
Accuracy : 100%

Test Set

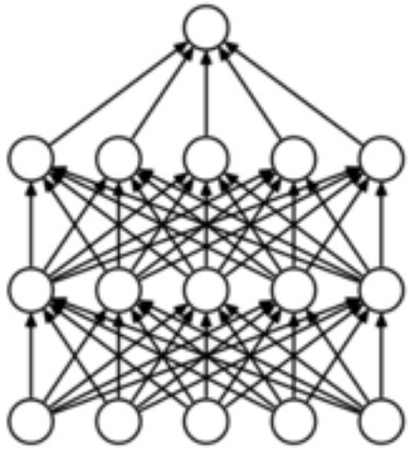
Accuracy :69%

WHY WE NEED AUGMENTATION

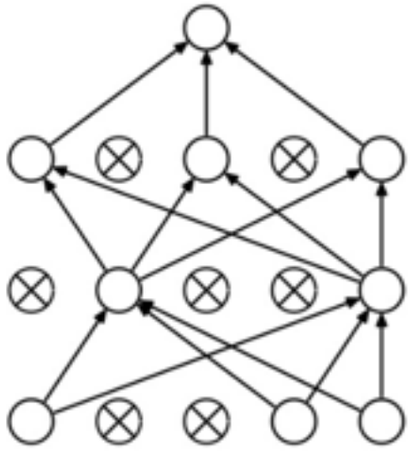
Avoid overfitting



DROPOUT

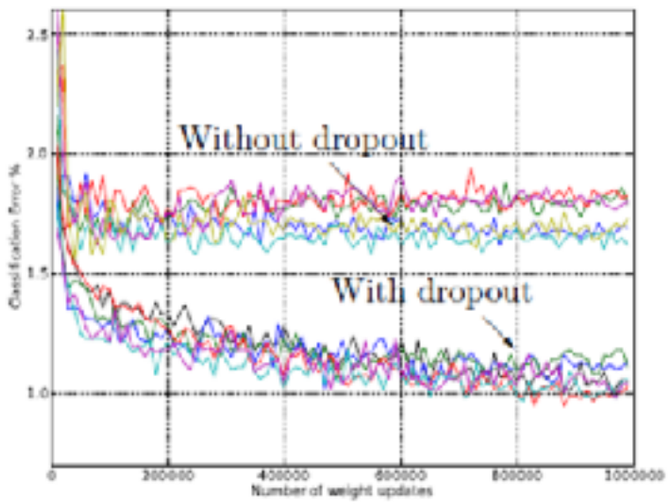


(a) Standard Neural Net



(b) After applying dropout.

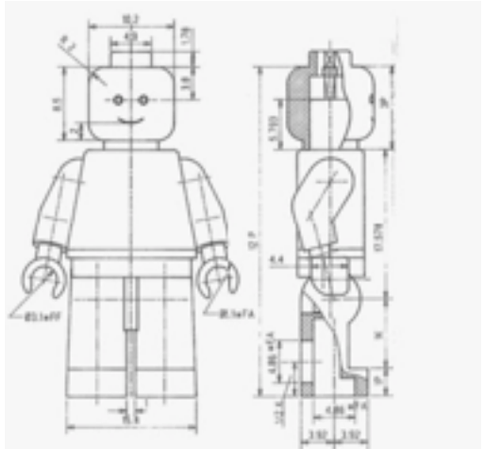
Randomly kill nodes , it result in avoiding overfitting problem
it allow in stacking more layer , we can more high accuracy and low loss



- Why? Tensor flow

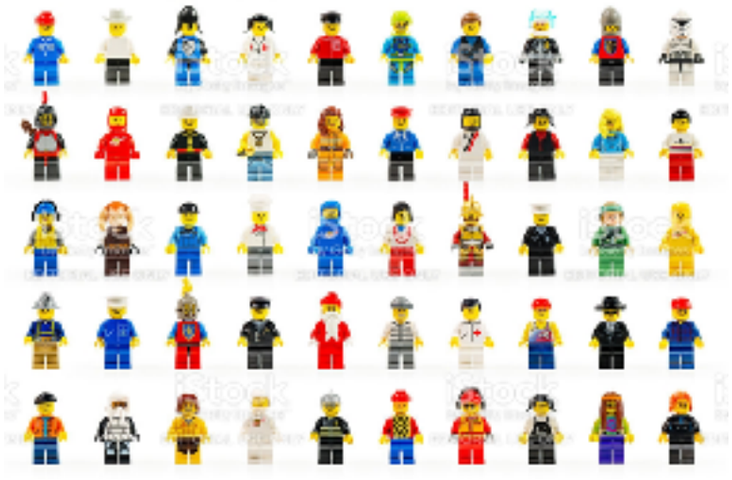
    1. *Easy to build Deep learning solution*
    2. *lot of Library for Developer*
    3. *Was built by Multi threads*
    4. *Support Multi GPU*
    5. *Many user*
    6. *Useful Analysis tool (tensor board)*
    7. *etc.*

TENSOR FLOW MECHANIISM



Draw Scheme

Input material



Get Scheme

Draw Graph

Make session

Input Data

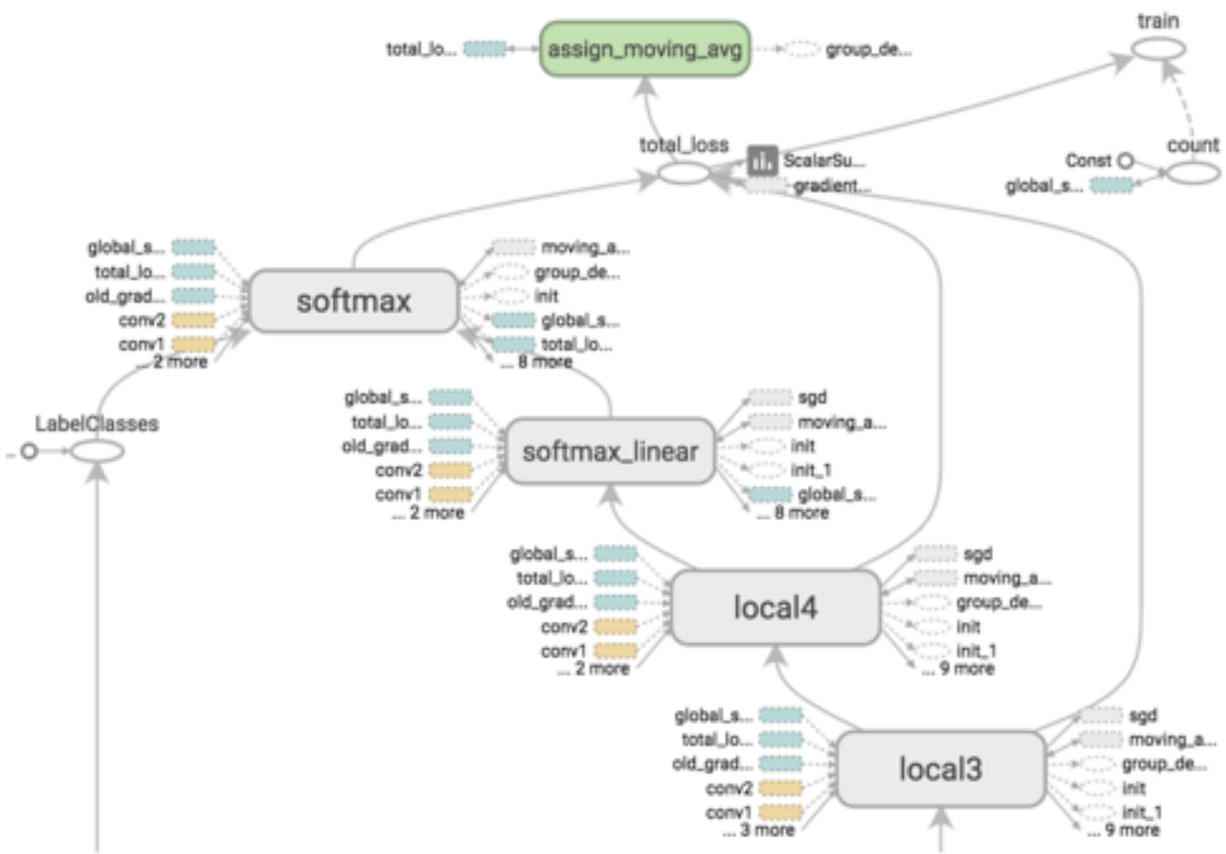
Accuracy : 97%

Get Result

SESSION

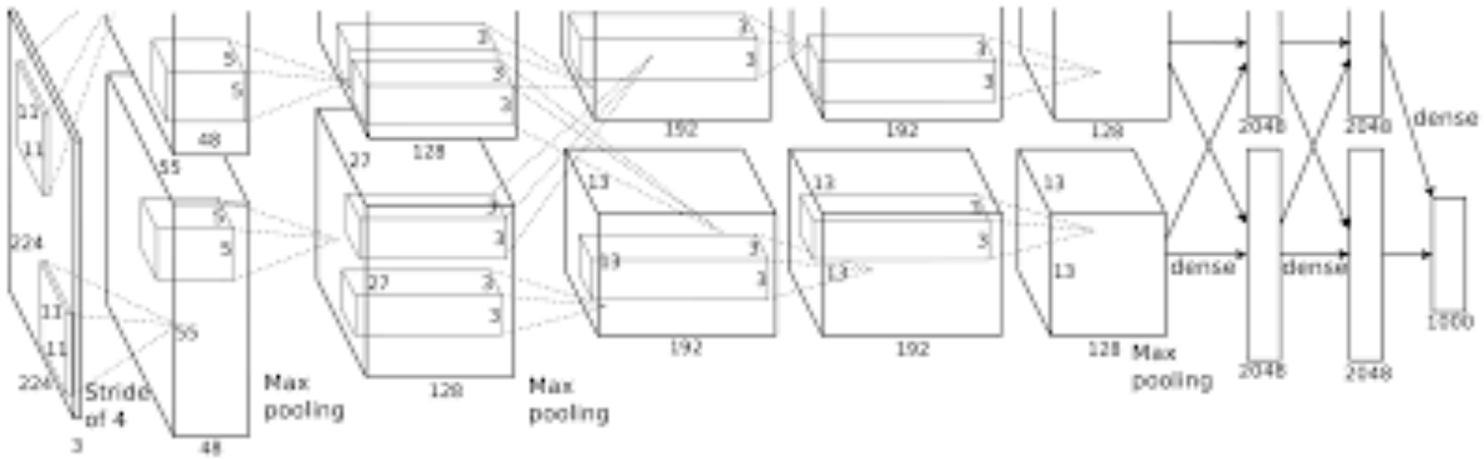
Through Session, upload graph to RAM , or GRAM

And run Graph , input Data



CONVOLUTION NEURAL NETWORK

- Convolution neural network (CNN) have the advantage of Image Detection and Classification



CONVOLUTION FILTER

Filter

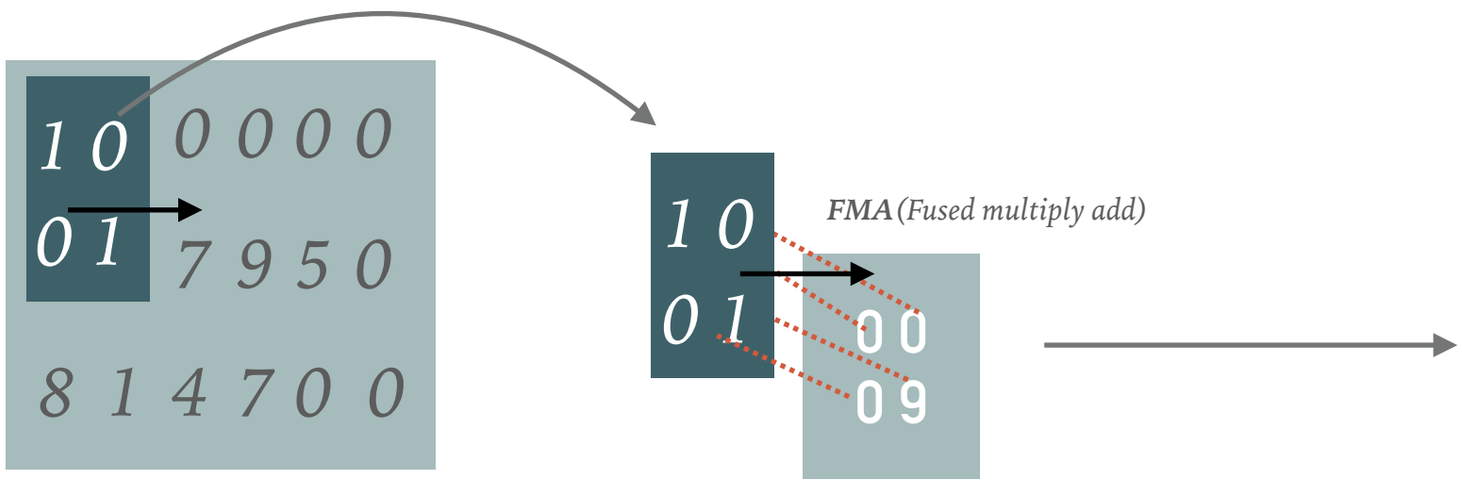
1	0
0	1

Background

0	0	0	0	0	0
0	9	7	9	5	0
8	1	4	7	0	0

Convolution filter hold Background Image's feature

Let's See how filter work



$$1x0 + 0x0 + 0x0 + 0x9$$

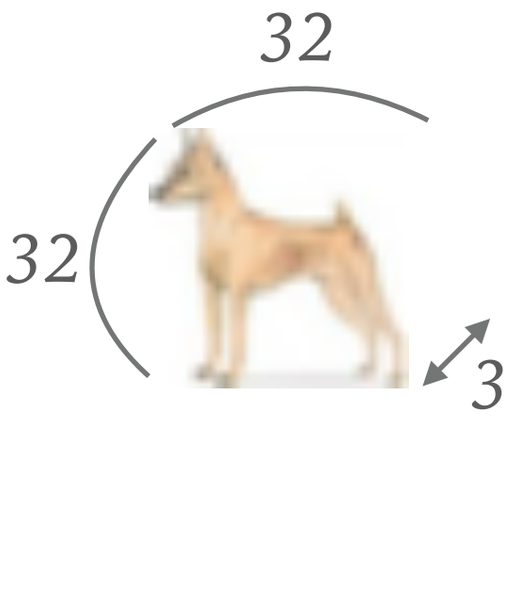
Output Image

9	7	9	5	0
1	13	14	9	5

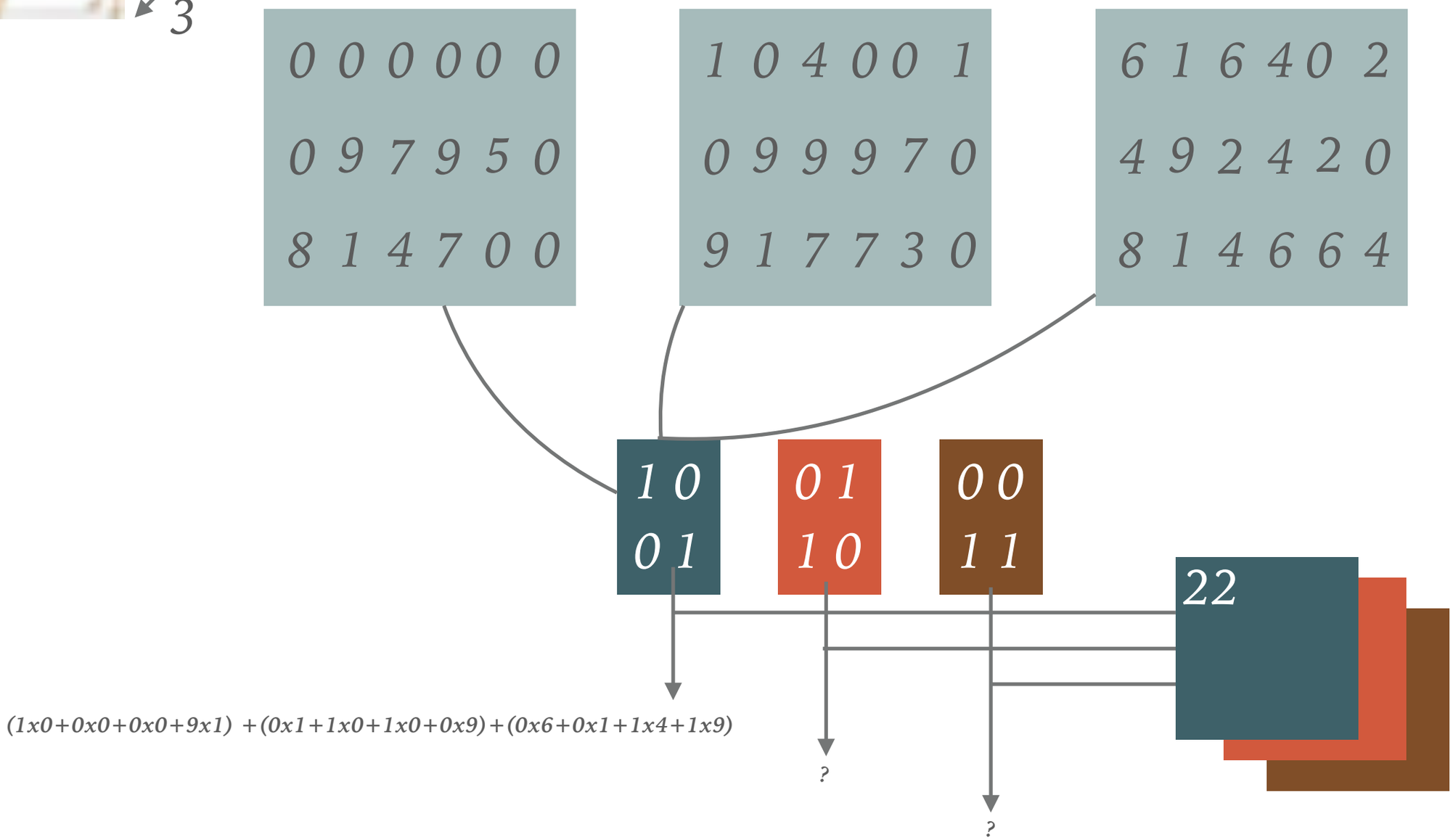
Filter holds feature of image

If filter and ROF are similar ,

the output pixel has great value than not

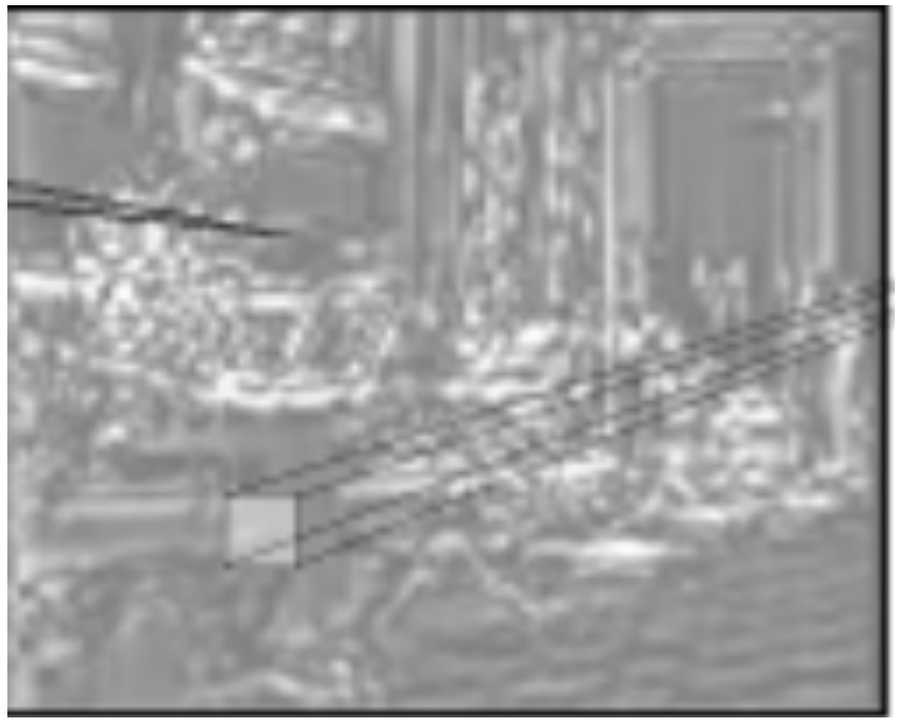


Color images have 3 channel , so how dose it work FAM at color image ?



SO IN CNN WHY WE NEED DEEP LEARNING?

Convolution filter hold Feature



Output value has more great value

If ROF has similar Feature to convolution filter

IN CNN ,

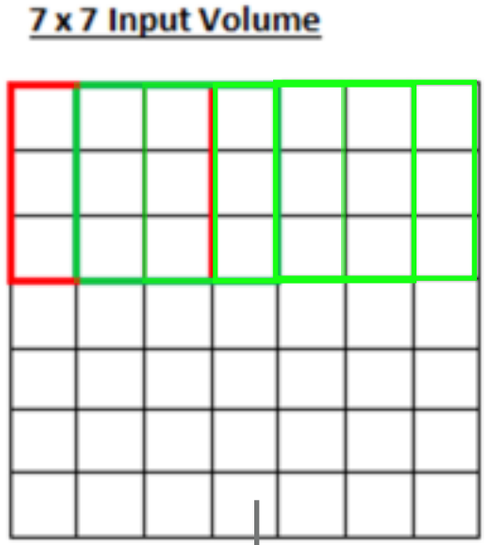
WE TRAIN CONVOLUTION FILTER

CAN GET IMAGE FEATURE

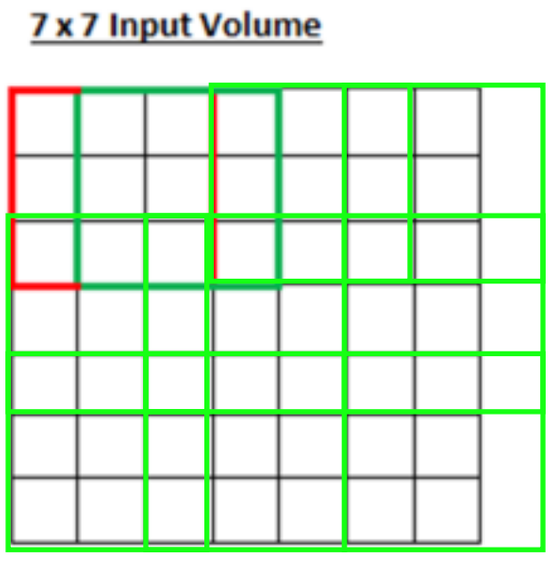
PADDING / STRIDE

Zero padding

Strides



Stride 1



Stride 2

PADDING / STRIDE

Tensor flow padding

VALID / SAME

VALID

```
matrix
[[ 0.  0.  0.]
 [ 1.  1.  1.]
 [ 2.  2.  2.]
 [ 3.  3.  3.]
 [ 4.  4.  4.]
 [ 5.  5.  5.]]
```

```
filter
[[ 0.  1.]
 [ 1.  0.]]
```

→

```
output
[[ 1.  1.]
 [ 3.  3.]
 [ 5.  5.]
 [ 7.  7.]
 [ 9.  9.]]
```

Convolution Output size is shrinked than original image

SAME

```
matrix
[[ 0.  0.  0.]  0
 [ 1.  1.  1.]  0
 [ 2.  2.  2.]  0
 [ 3.  3.  3.]  0
 [ 4.  4.  4.]  0
 [ 5.  5.  5.]] 0
```

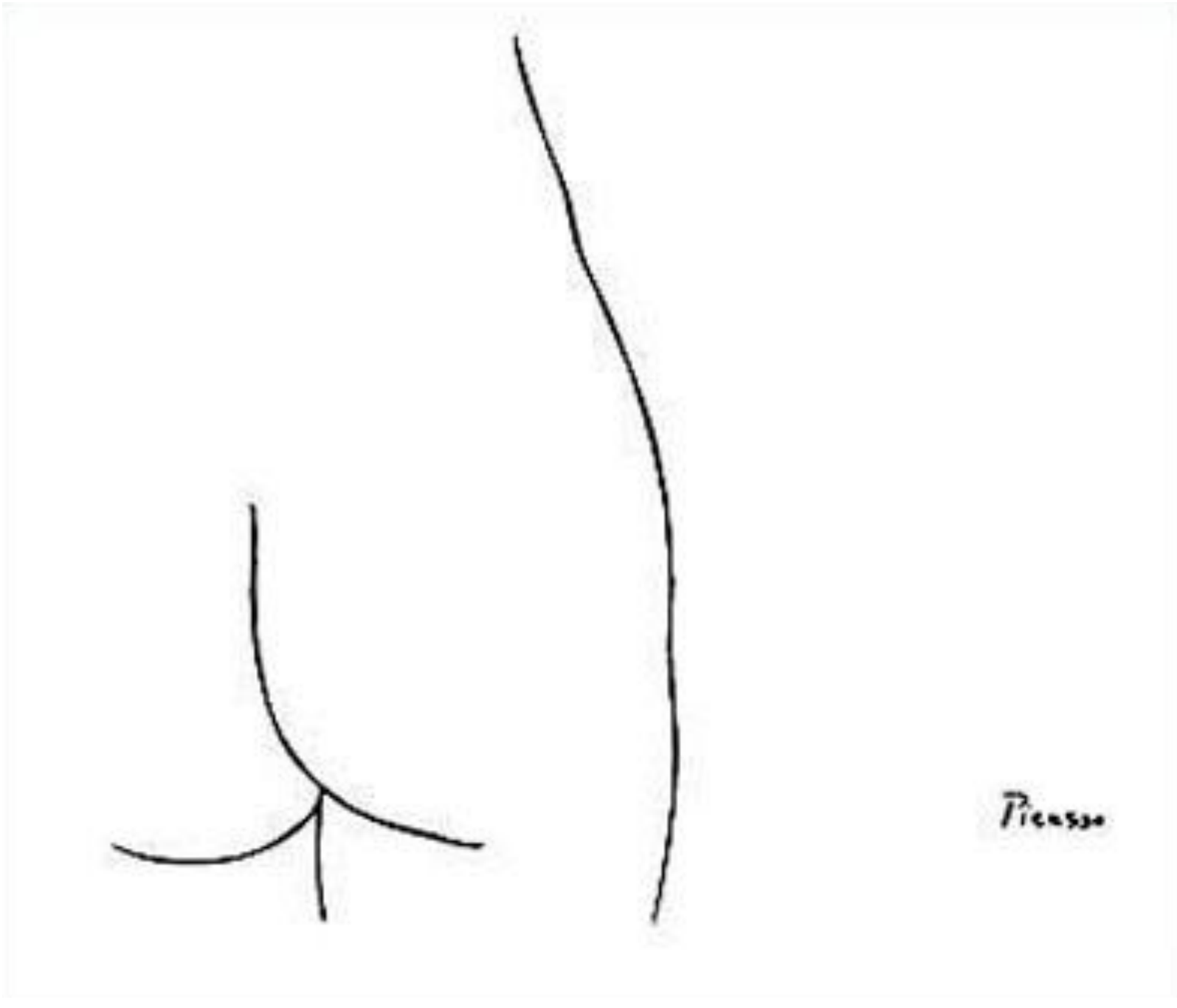
```
filter
[[ 0.  1.]
 [ 1.  0.]]
```

→

```
output
[[ 1.  1.  1.]
 [ 3.  3.  2.]
 [ 5.  5.  3.]
 [ 7.  7.  4.]
 [ 9.  9.  5.]
 [ 5.  5.  0.]]
```

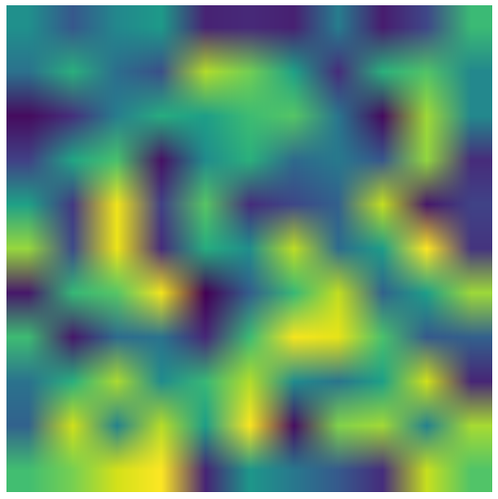
*Convolution Output size **same** as intput data*

Abstract?

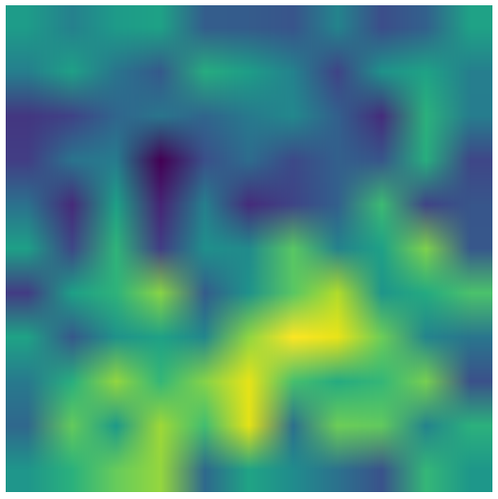


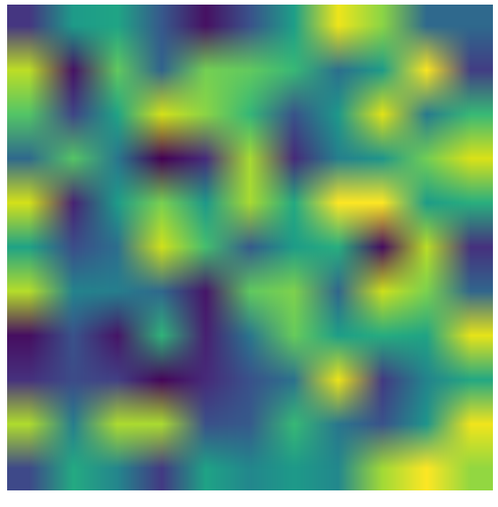
WHAT INFORMATION WAS EXTRACT FROM CONVOLUTION NEURAL NETWORK?

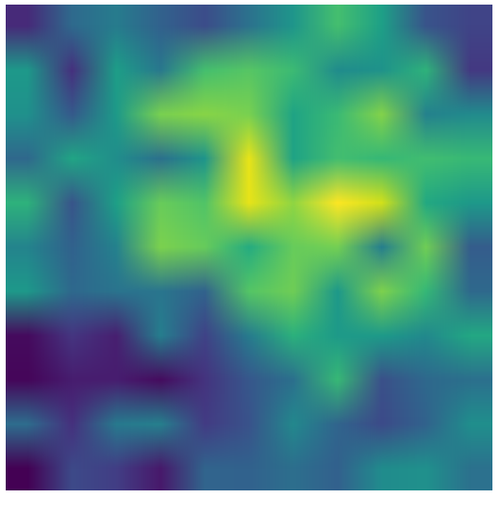
Iteration 1000



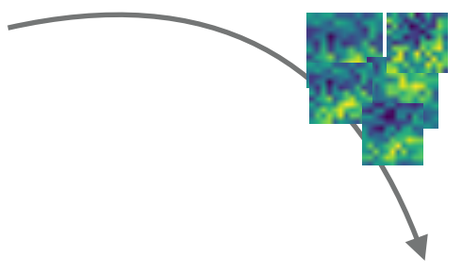
Iteration 200,000



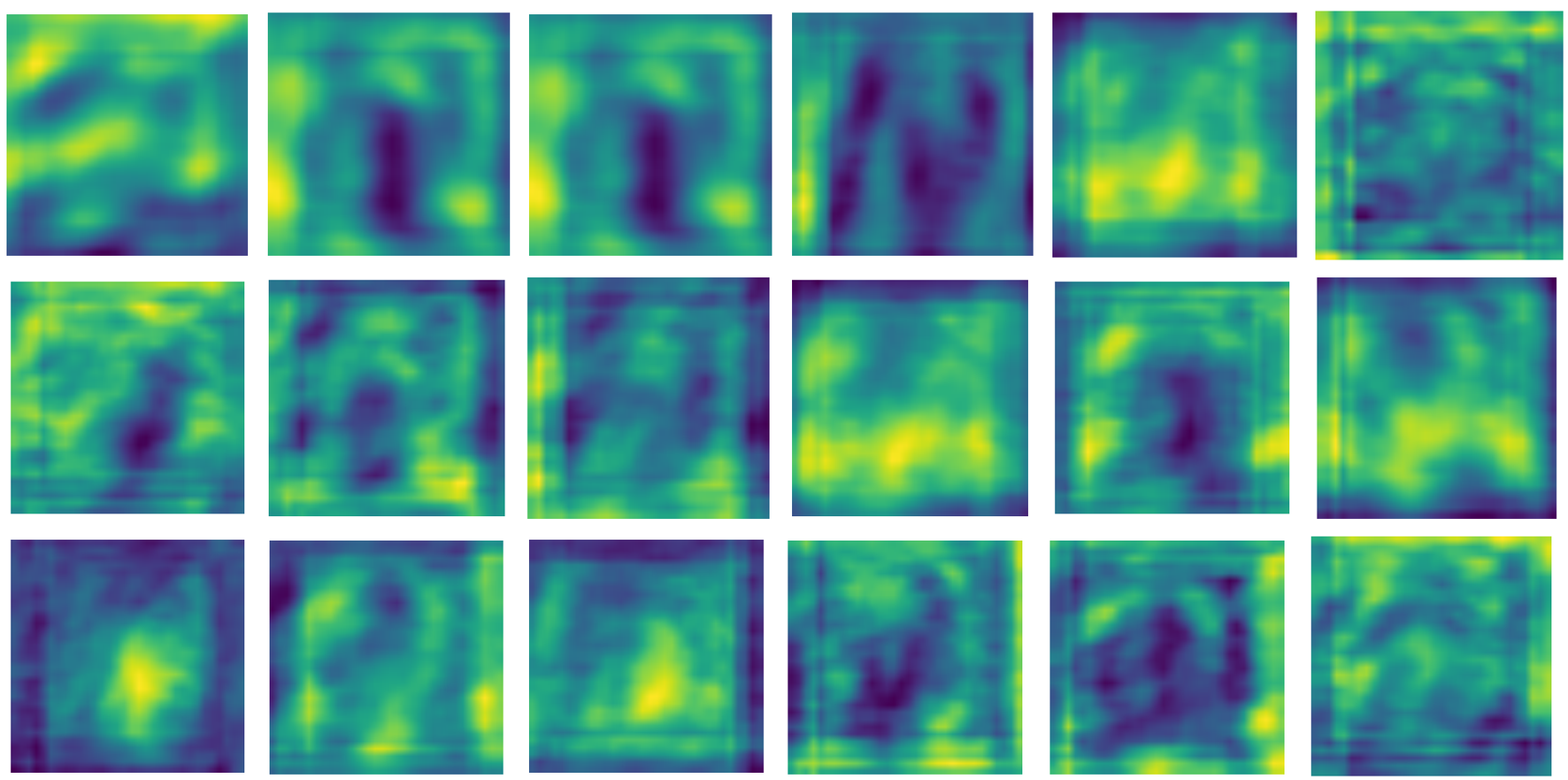




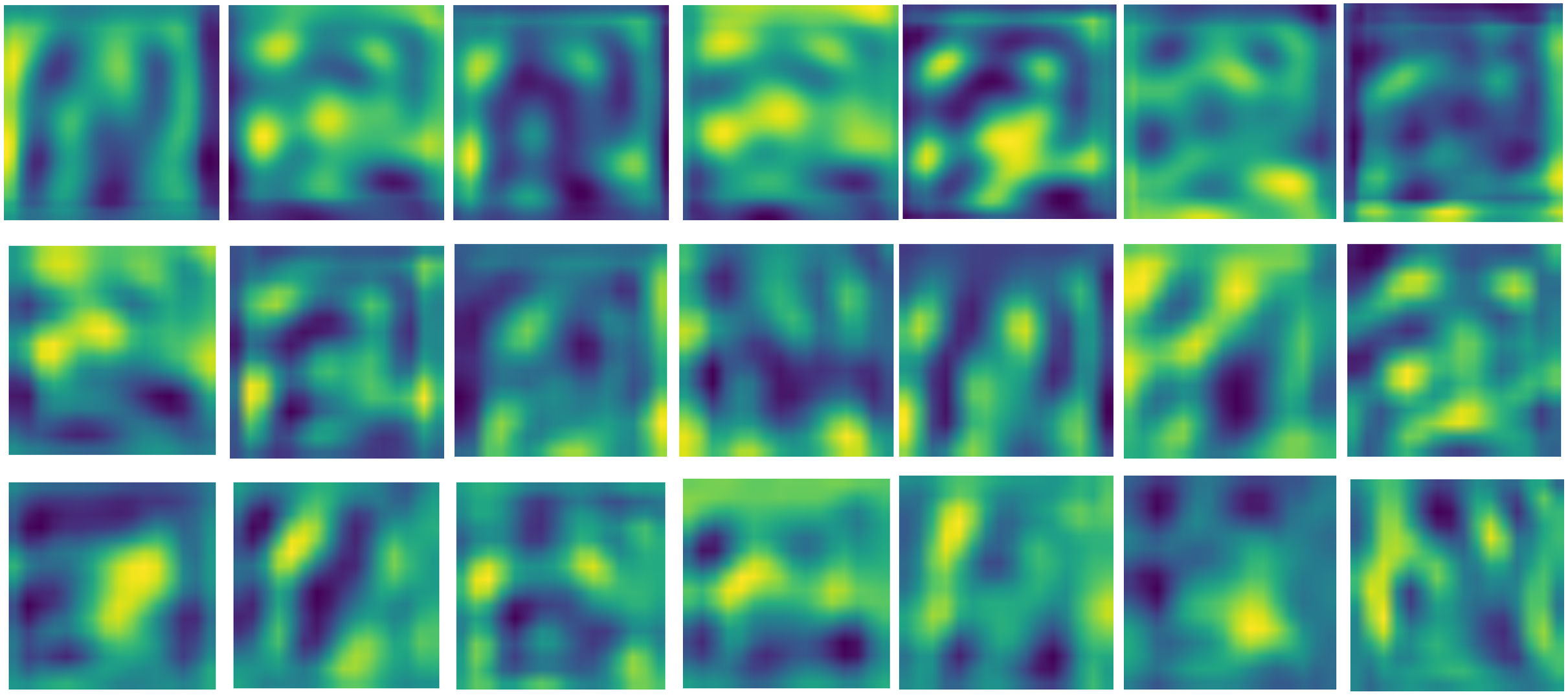
WHAT INFORMATION WAS EXTRACT FROM CONVOLUTION NEURAL NETWORK?



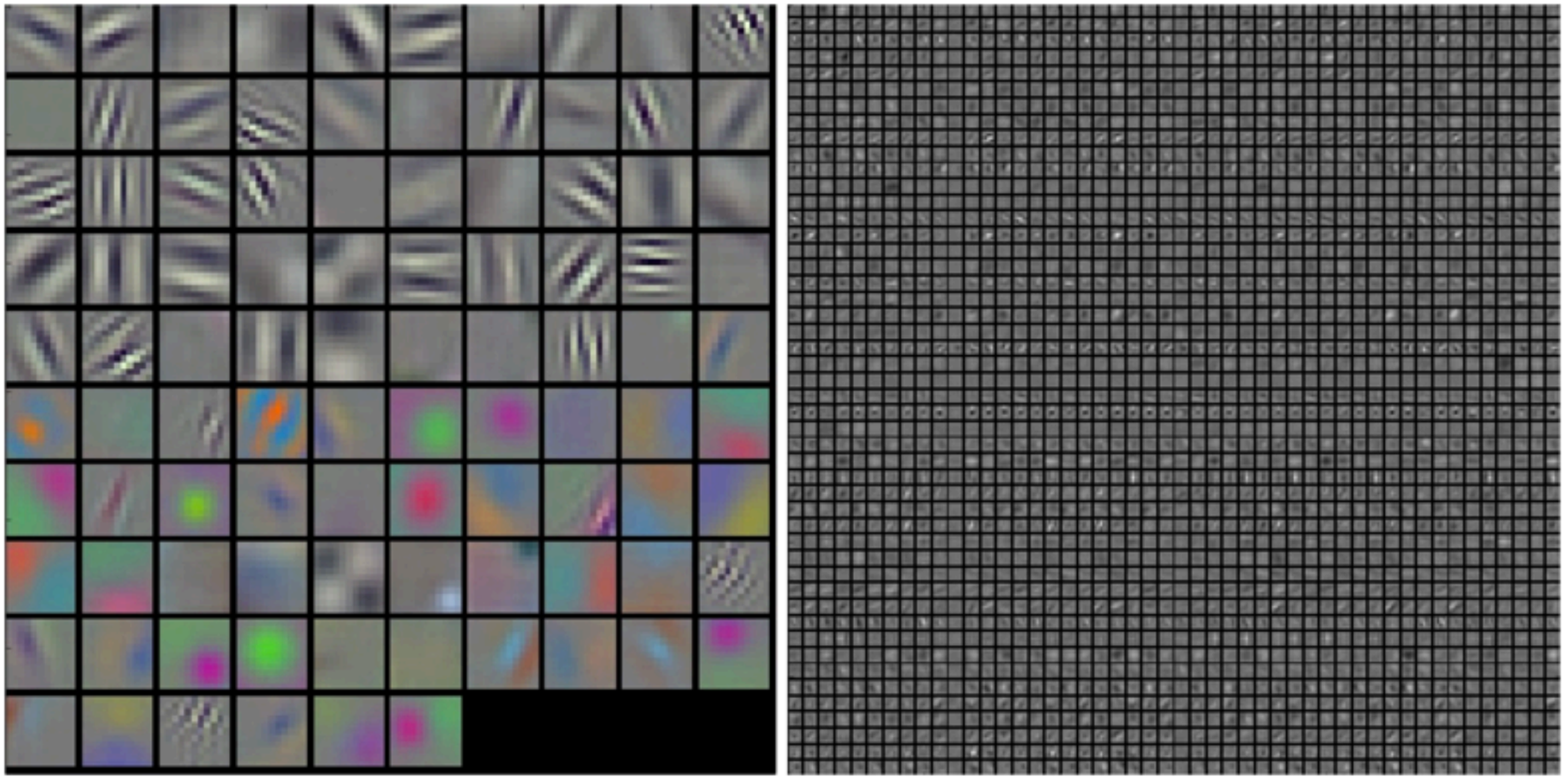
Filter abstracts image



MORE DEEP LAYER , GET MORE ABSTRACT IMAGE



WHAT INFORMATION WAS EXTRACT FROM CONVOLUTION NEURAL NETWORK?



“More deeper layer , get more abstract ,more complicated image”

LET'S DRAW SCHEME OF TENSORFLOW [CNN]

1.Define input data shape

Input data

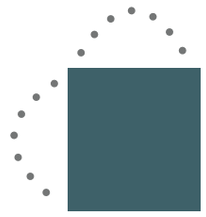
$x\_ = tf.placeholder(tf.float32, [\ None\ ,\ height\ ,\ width\ ,\ color\_ch\ ])$

$y\_ = tf.placeholder(\ tf.int32\ ,\ [\ None\ ,\ n\_classes\ ])$

2.Define Convolution filter

width

height



Usually **2d convolution*

Filter shape [1,height , width ,1]

$w1 = tf.get\_variable("w1"\ ,\ [7,7,color\_ch\ ,\ out\_ch]\ )$

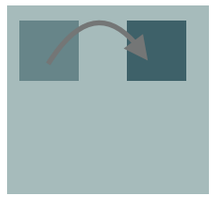
$b1 = tf.Variable(tf.constant(0.1)\ ,out\_ch)$

3.Define Strides , Pooling

**Usually 2d Convolution*

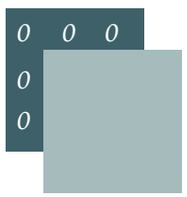
Strides shape : [height , width , 1, 1]

Pooling type = "SAME" , "VALID"



Stride

$strides = [1,1,1,1]$



Pooling

$padding = 'SAME'$

Combine

$layer2 = tf.nn.conv2d(layer1,\ w2\ ,\ s2,$

$padding = 'SAME') + b2$

HOW CAN WE CONNECT LAYER BY LAYER

Layer 1

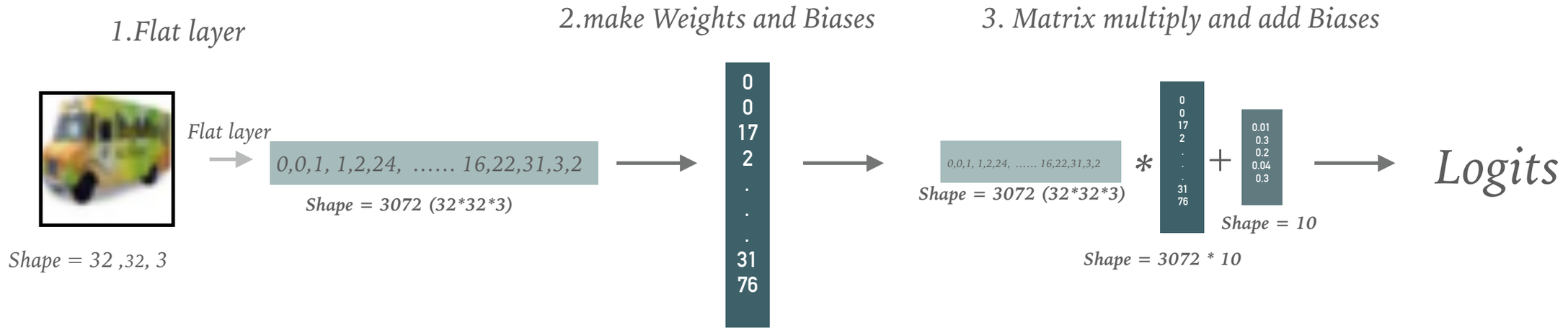
Layer 2

layer 1 output = layer 2 input

```
out_ch=28
w1=tf.get_variable("w1" , [7,7,color_ch , out_ch] , in
b1=tf.Variable(tf.constant(0.1) ,out_ch)
s1=[1,1,1,1]
p1='SAME'
layer1=tf.nn.conv2d(x_ , w1 , s1 , p1 )+b1
layer1=tf.nn.relu(layer1)
```

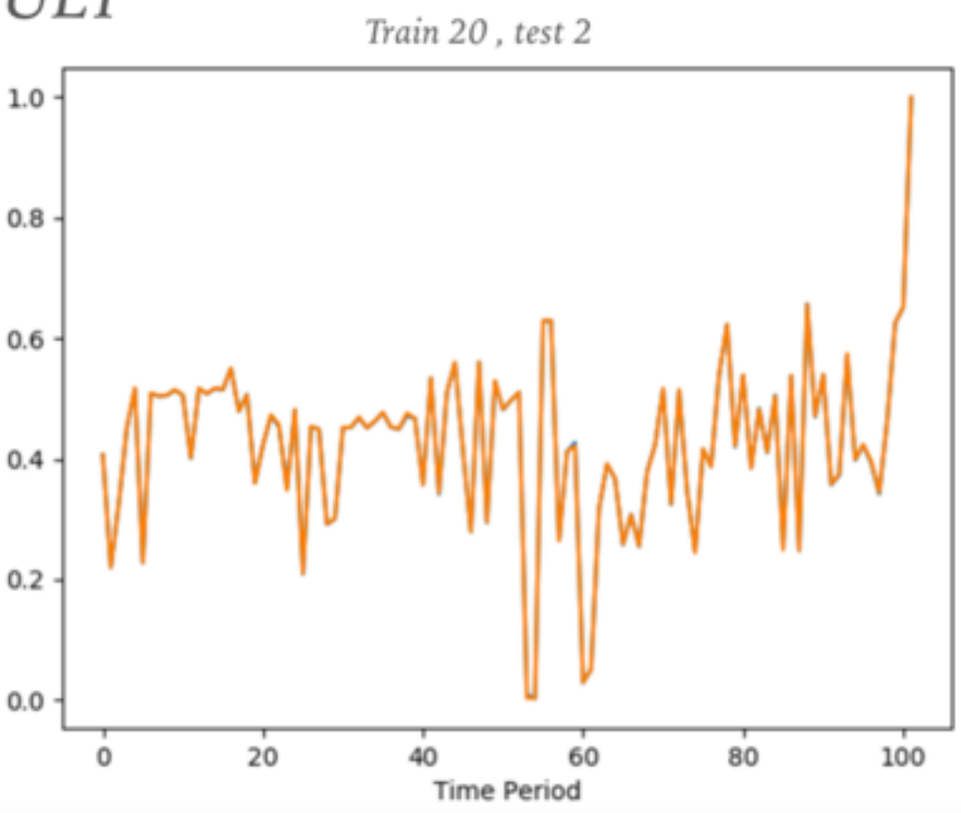
```
out_ch2=64
w2=tf.get_variable("w2" , [5,5,out_ch, out_ch2] , initialize
b2=tf.Variable(tf.constant(0.1) ,out_ch2)
s2=[1,1,1,1]
layer2=tf.nn.conv2d(layer1, w2 , s2, padding='SAME')+b2
layer2=tf.nn.relu(layer2)
end_conv_layer=layer2
```

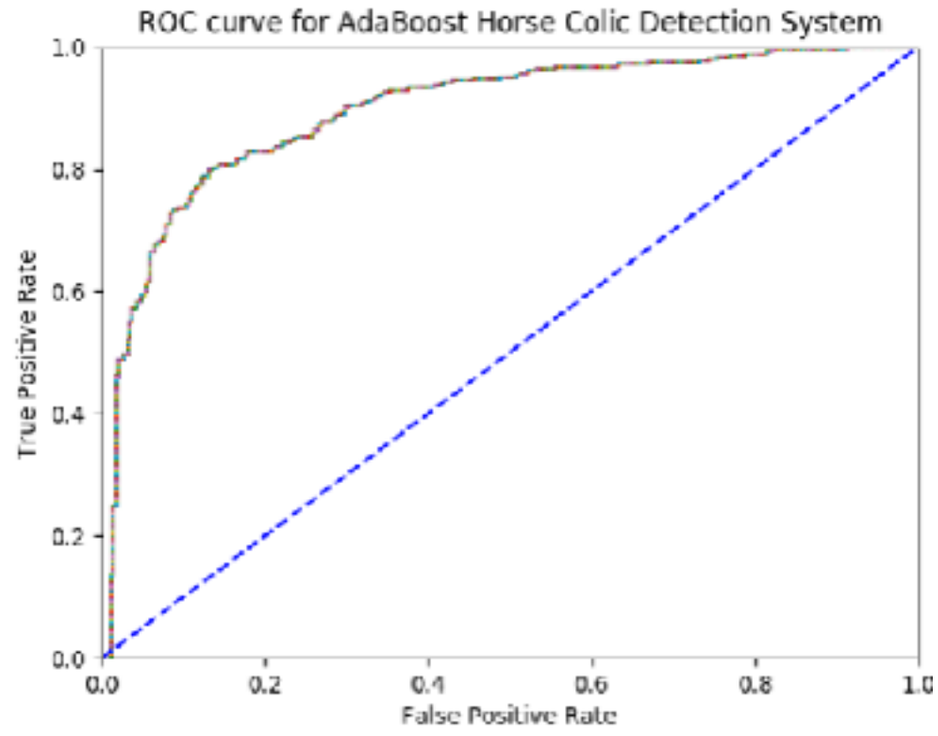
FULLY CONNECTED LAYER



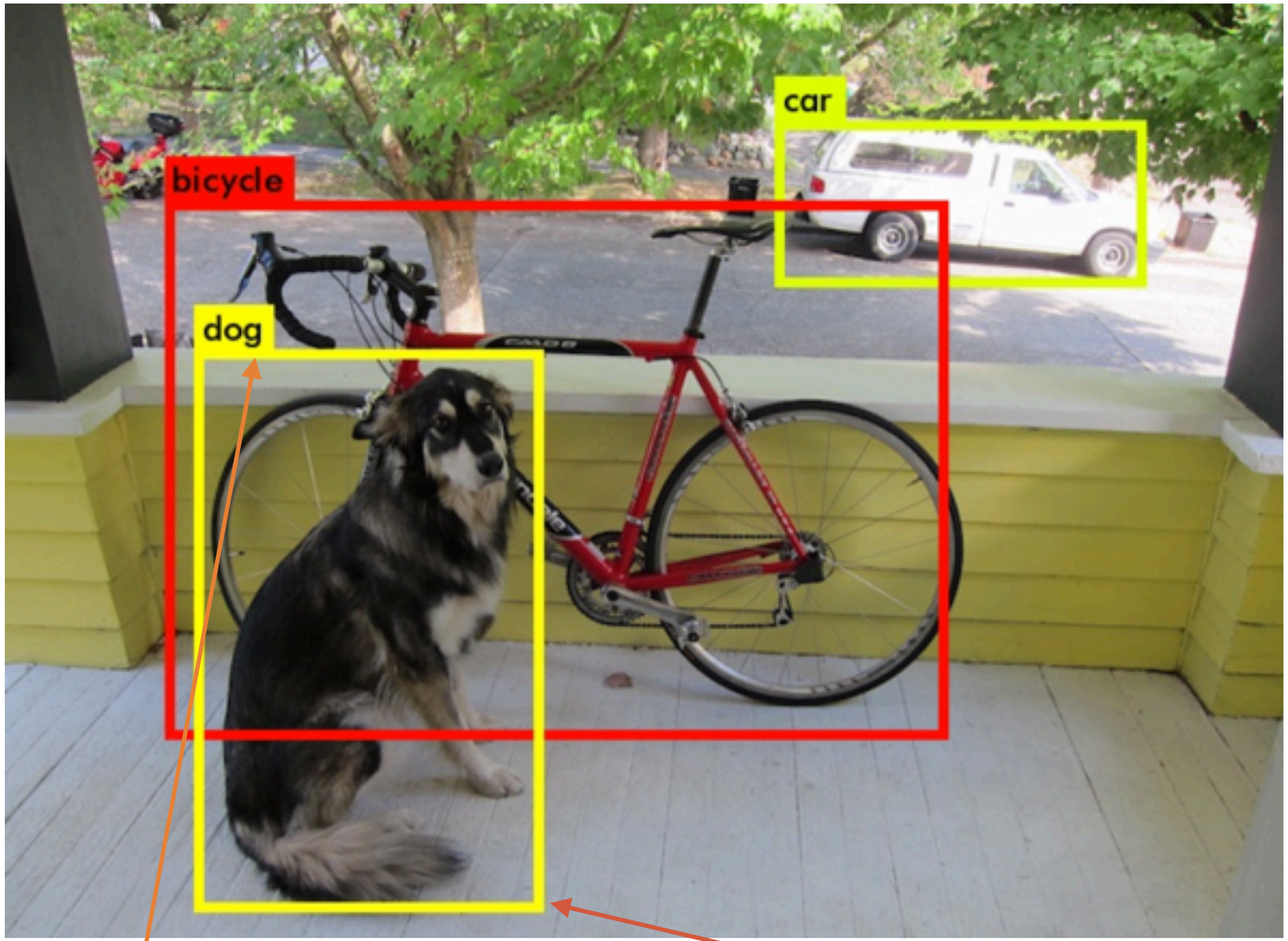
GET RESULT AND ANALYSIS

RESULT





OBJECT DETECTION & CLASSIFICATION



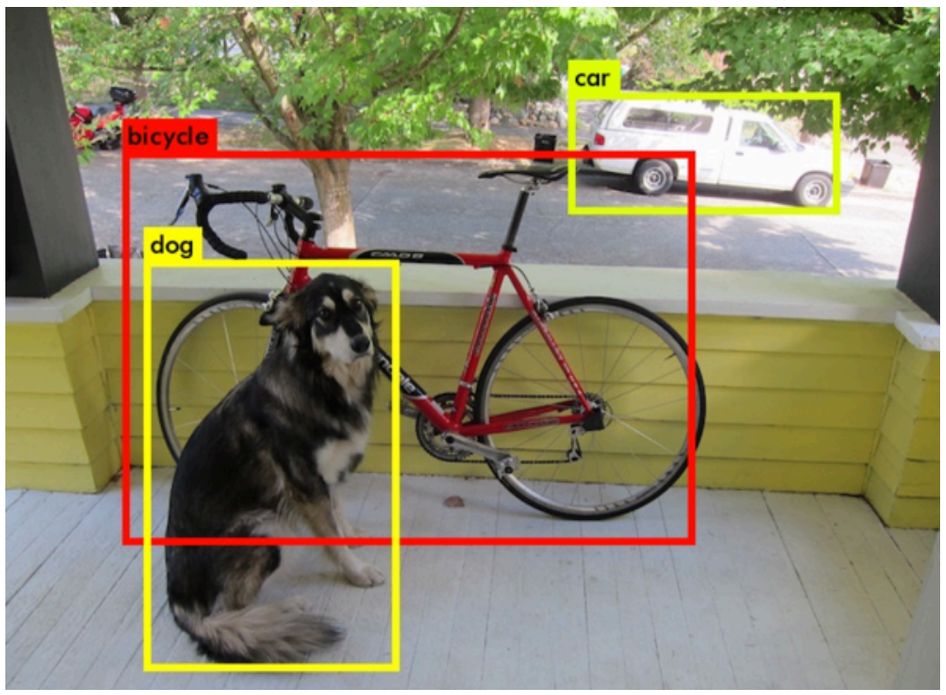
Classification

Define target

Detection

Find location of target

DETECTION



Need for Detection

Bounding box location coordinate

x , y , h , w

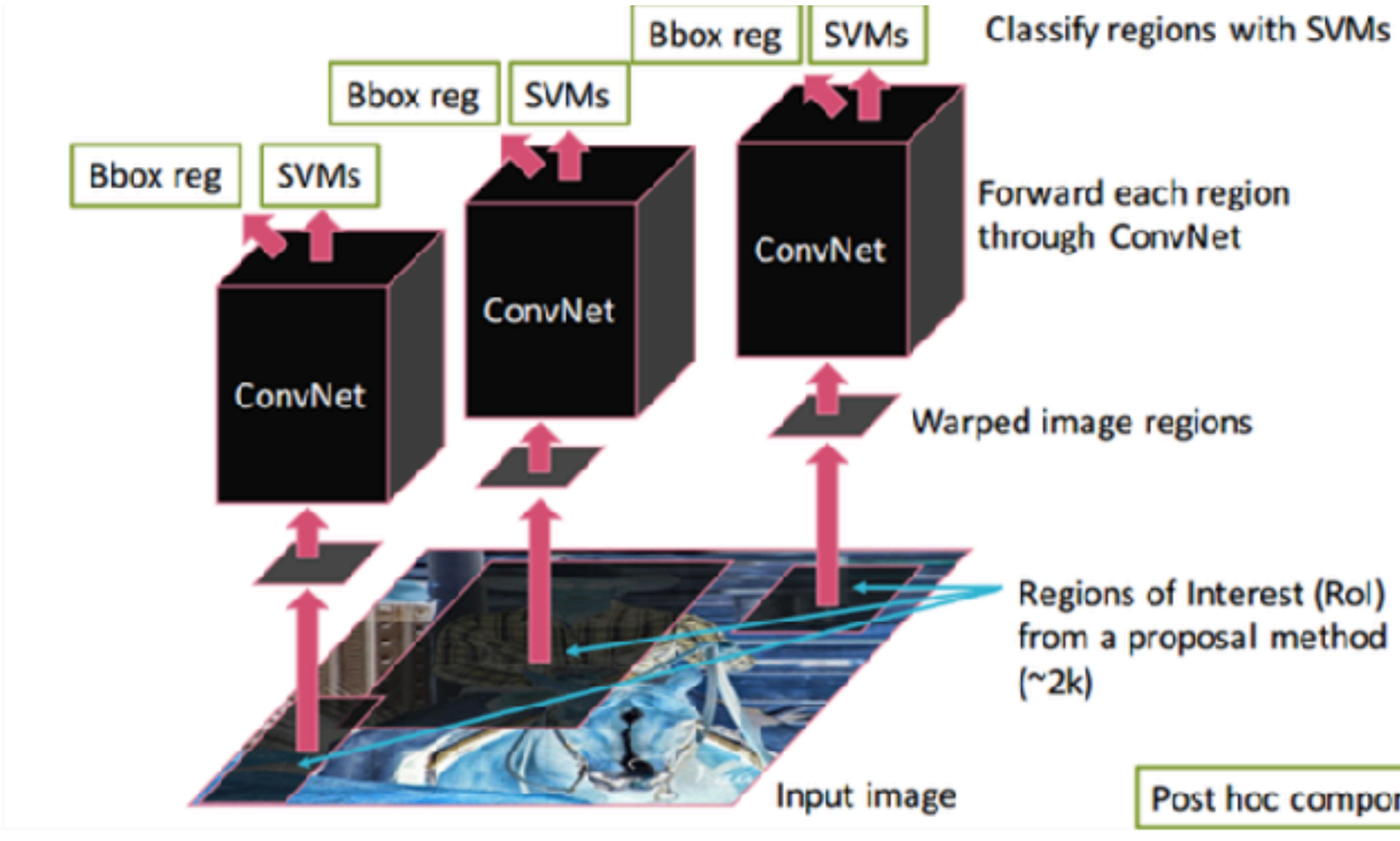
1.how we make bounding box

2.how we train box's location

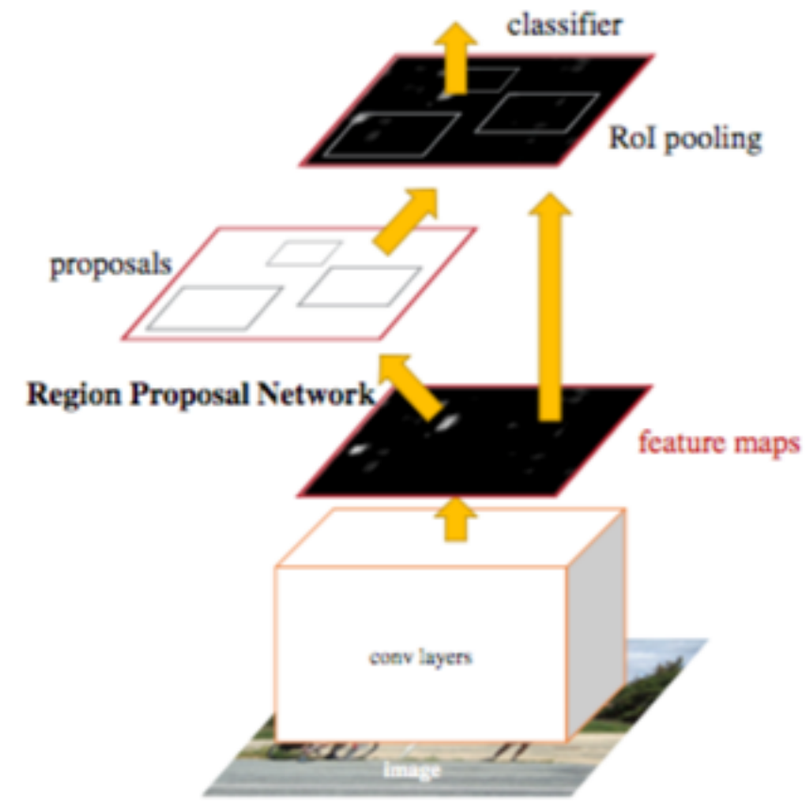
Detection Algorithm : SPP NET , RCNN , Fast -RCNN , Faster -RCNN , YOLO

HOW WE MAKE BOUNDING BOX

First we have to know where target is …



RCNN



Faster rcnn

FASTER-RCNN

(a)

(b)

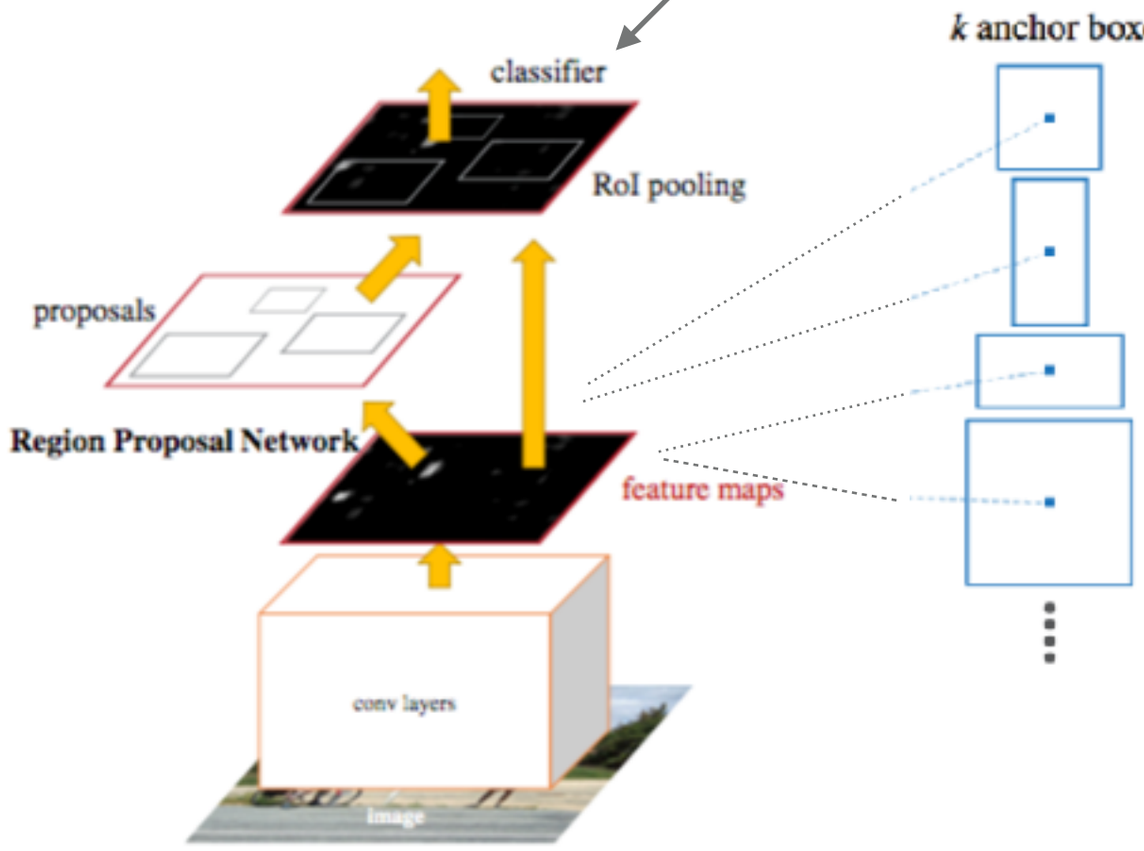
(c) Through deep fully convolutional network.

Predict x,y,h,w references

Fast R-CNN detector.

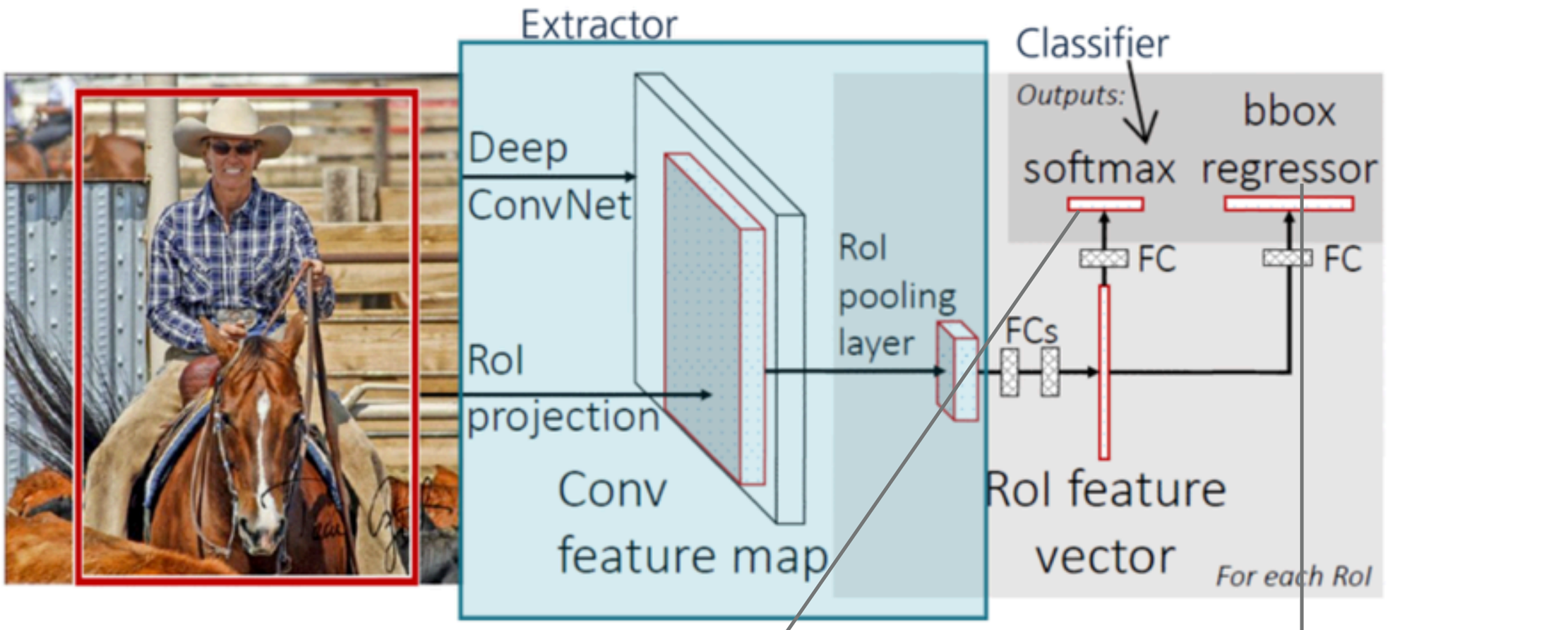
Using predicted reference

Training FCNN



Faster rcnn

FASTER CNN SHARE LAYER.



Class Layer

Object probability (cat? Dog? …)

Regression Layer

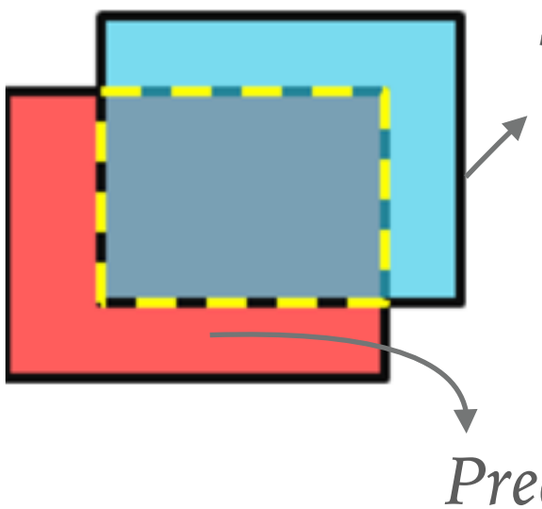
Object proposal (h,w,x,y)

BASIC DETECTION

- 1.CNN Fine Tuning
- 2.Region Search (SPPNet , RCNN , Faster RCNN , YOLO)
- 3.Training
- 4.Using NMS , get Bounding Box

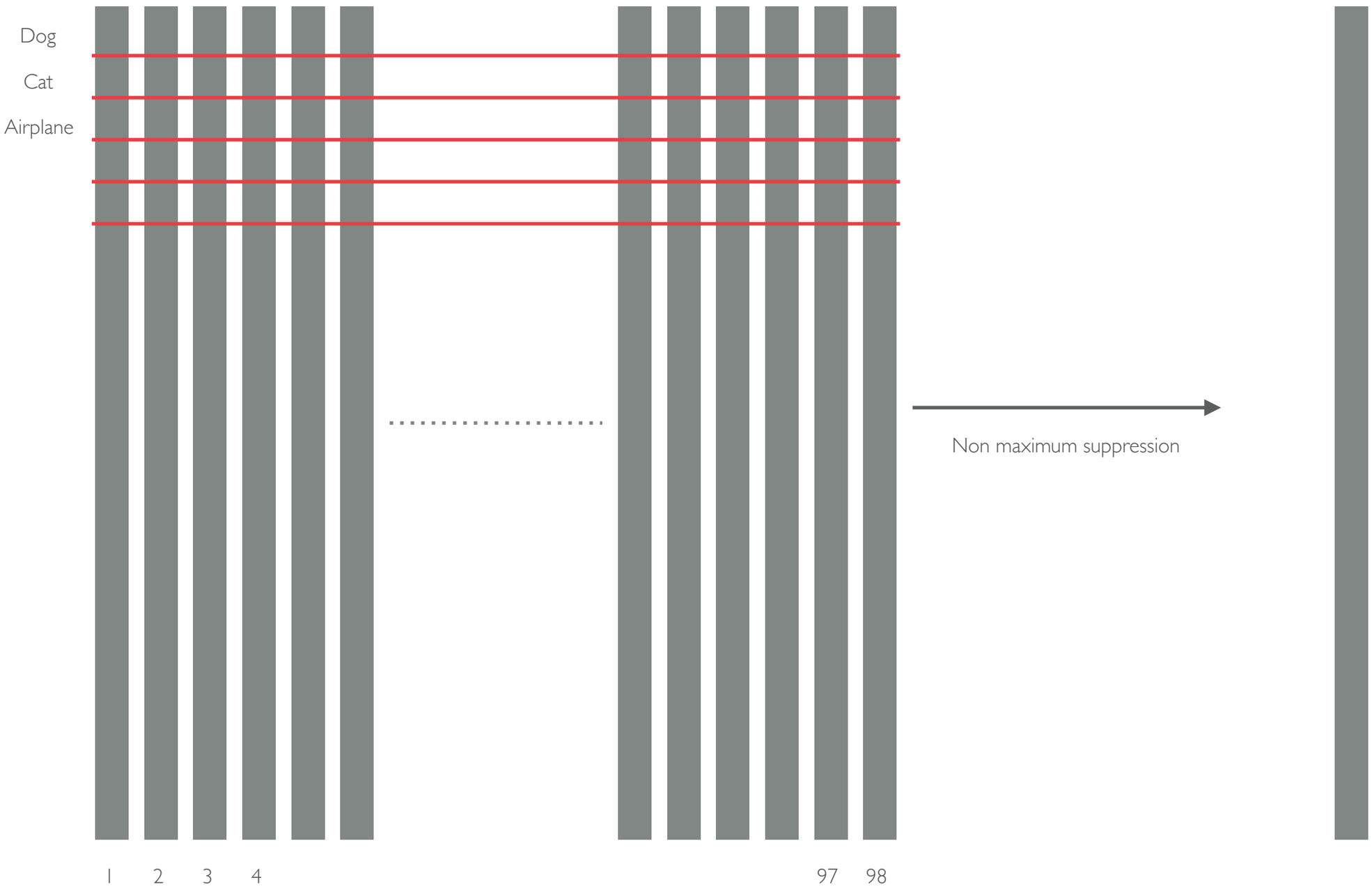
HOW WE TRAIN BOX LOCATION FACTOR

- IOU(intersection over union)

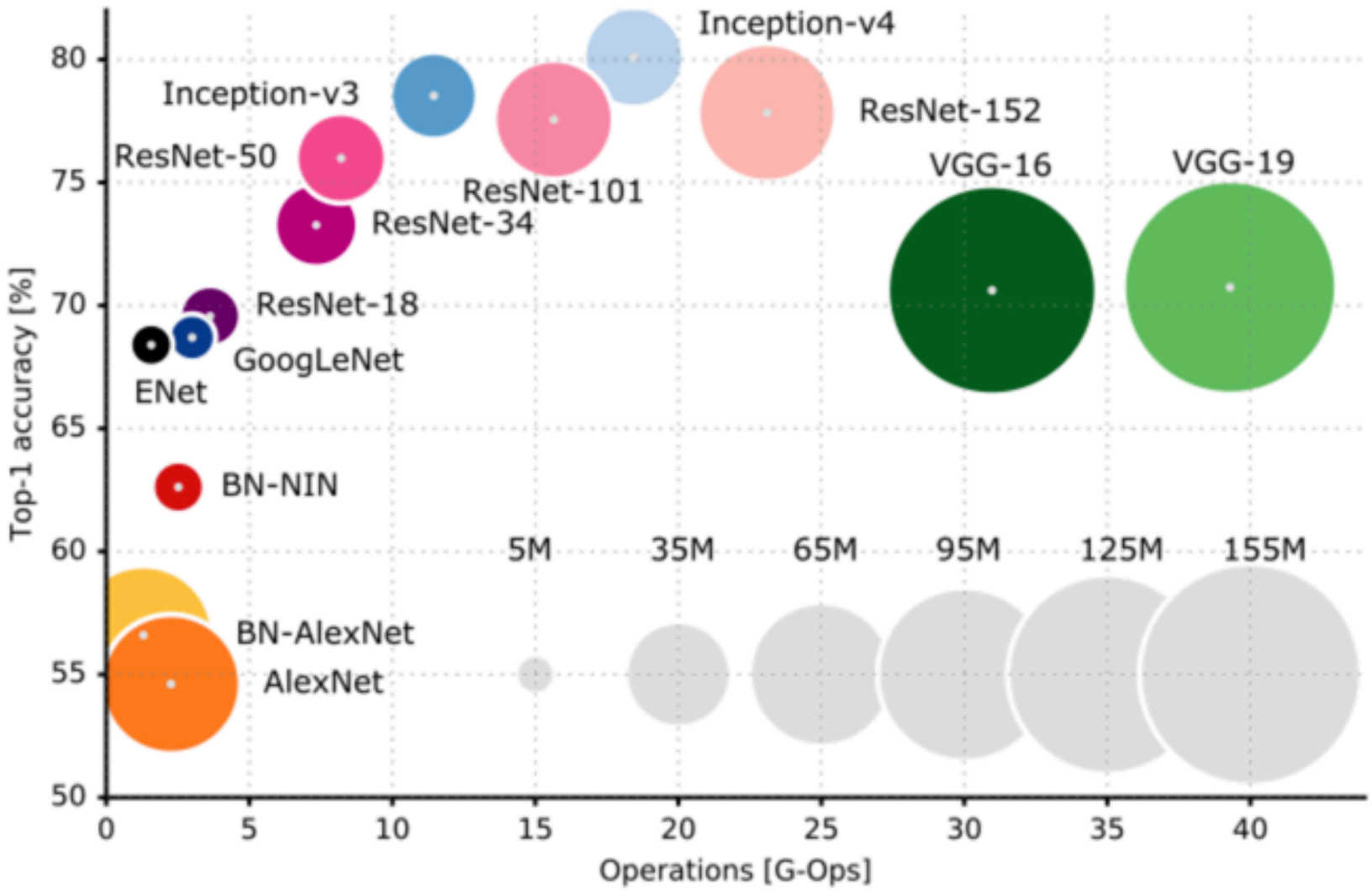


$$\frac{\textit{Predict box area}}{\textit{True box Area}} \longrightarrow \text{If IOU} > 0.5$$

NMS(NON MAXIMUN SUPPRESSION)



CONVOLUTION NETWORK ARCHITECTURES



Convolution Nueral Network

Lenet5 , *http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf*

Alexnet , *https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf*

ZFNet *https://arxiv.org/abs/1311.2901*

Overfit net , *https://arxiv.org/abs/1312.6229*

VGG , *https://arxiv.org/pdf/1409.1556.pdf*

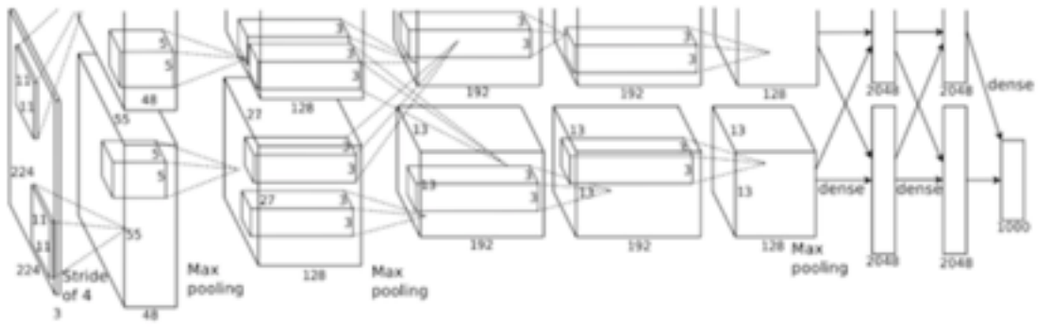
Resnet *https://arxiv.org/abs/1512.03385*

Enet *https://arxiv.org/abs/1606.02147*

Xeception net *https://arxiv.org/abs/1610.02357*

fractralnet *https://arxiv.org/abs/1605.07648*

ALEXNET



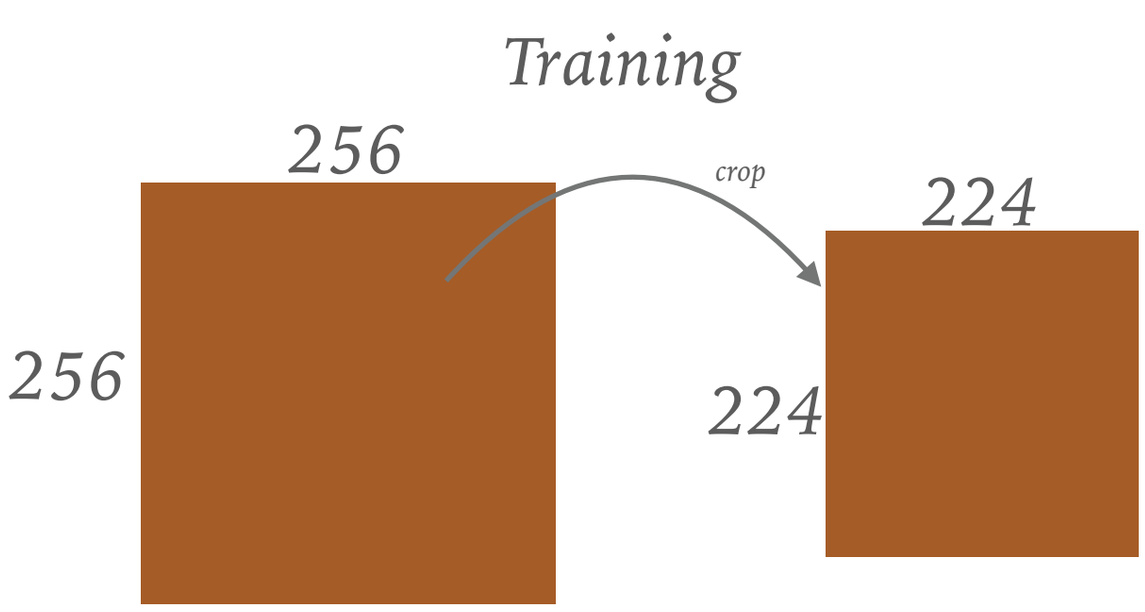
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

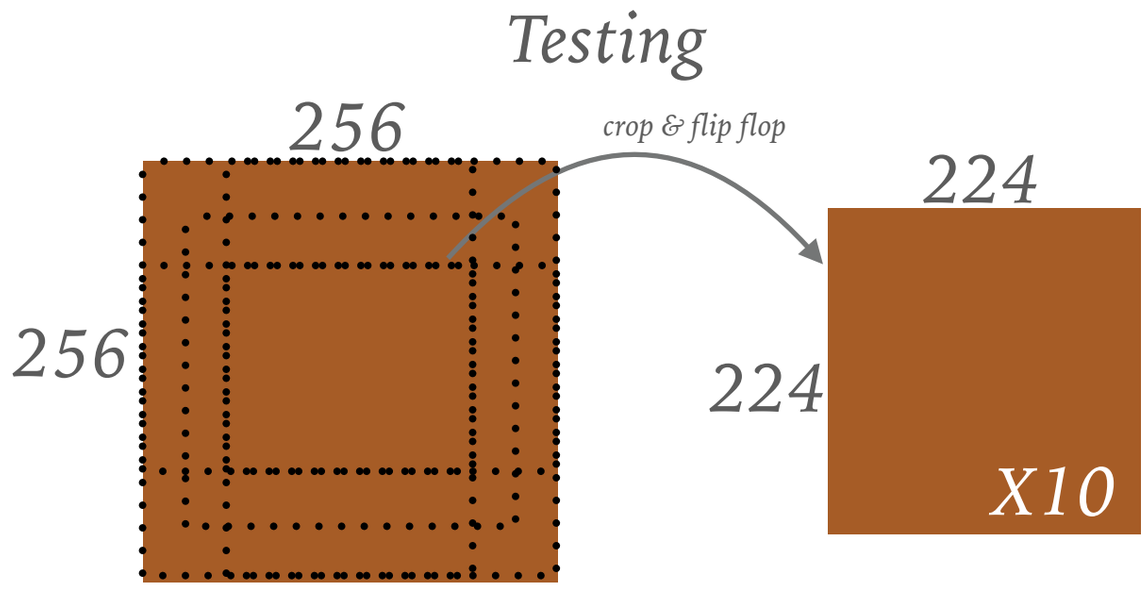
[227x227x3] INPUT
[55x55x96] CONV1 : 96@ 11x11, s = 4, p = 0
[27x27x96] MAX POOL1 : 3x3, s = 2
[27x27x96] NORM1 :
[27x27x256] CONV2 : 256@ 5x5, s = 1, p = 2
[13x13x256] MAX POOL2 : 3x3, s = 2
[13x13x256] NORM2 :
[13x13x384] CONV3 : 384@ 3x3, s = 1, p = 1
[13x13x384] CONV4 : 384@ 3x3, s = 1, p = 1
[13x13x256] CONV5 : 256@ 3x3, s = 1, p = 1
[6x6x256] MAX POOL3 : 3x3, s = 2
[4096] FC6 : 4096 neurons
[4096] FC7 : 4096 neurons
[1000] FC8 : 1000 neurons

RELU

DROP OUT

Reducing Overfitting - Augmentation



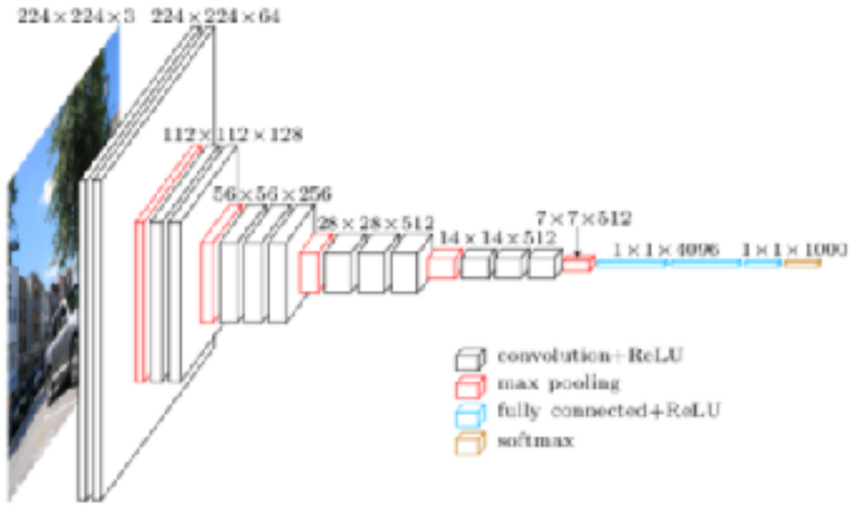


#2 ALEXNET HYPERPARAMETER

- SGD (Stochastic Gradient Descent) weight decay 0.0005
- Weight initialize Gaussian * 0.001
- Dropout 0.5
- Learning rate 0.01 —> 0.001
- 7 model ensemble

VGG #1

Inception modules we can use a high dimensional representation via multi-scale convolutional concatenation



PAPER *https://arxiv.org/abs/1409.1556*

- 3×3 filters in each convolutional layers

2 assumption to using convolution Kernel

1. Low level features are local
   -adjacent feature maps have highly correlated outputs
   -Which makes sense as natural images are known to exhibit some local statistical properties consistent with this
2. What's useful in one place will also be useful in other places

What is effect of kernel size

A larger size kernel can overlook at the features and could skip the essential details in the images
a smaller size kernel could provide more information leading to more confusion.
Thus there is a need to determine the most suitable size of the kernel/filter.

gaussian pyramids(set of different sized kernels) are generally used to test the efficiency of the feature extraction and appropriate size of kernel is determined.

Rule for selecting hyperpamameter

PAPER *http://www.robots.ox.ac.uk/~vgg/publications/2014/Chatfield14/chatfield14.pdf*
https://arxiv.org/pdf/1512.00567.pdf
http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf

1. Use small size of filter instead of using large filter e.g) 3x3 ~~7x7~~

Why?

1.Low Computation

2. More nonlinearity , more representational capacity

2.7x7 → 1x7 , 7x1

3. Balance depth with width

4. High dimensional representation via multi-scale convolutional concatenation

5.Use 1x1 conv layers (Network in Network style) to reduce dimensionality.

VGG #2

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**
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**
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**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

(3x3)x2 ->5x5
(3x3)x3 ->7x7
(3x3)x3 ->11x11

Notice that VGG-E: 256×256 and 512×512 3×3 filters are used multiple times in sequence to **extract more complex features**

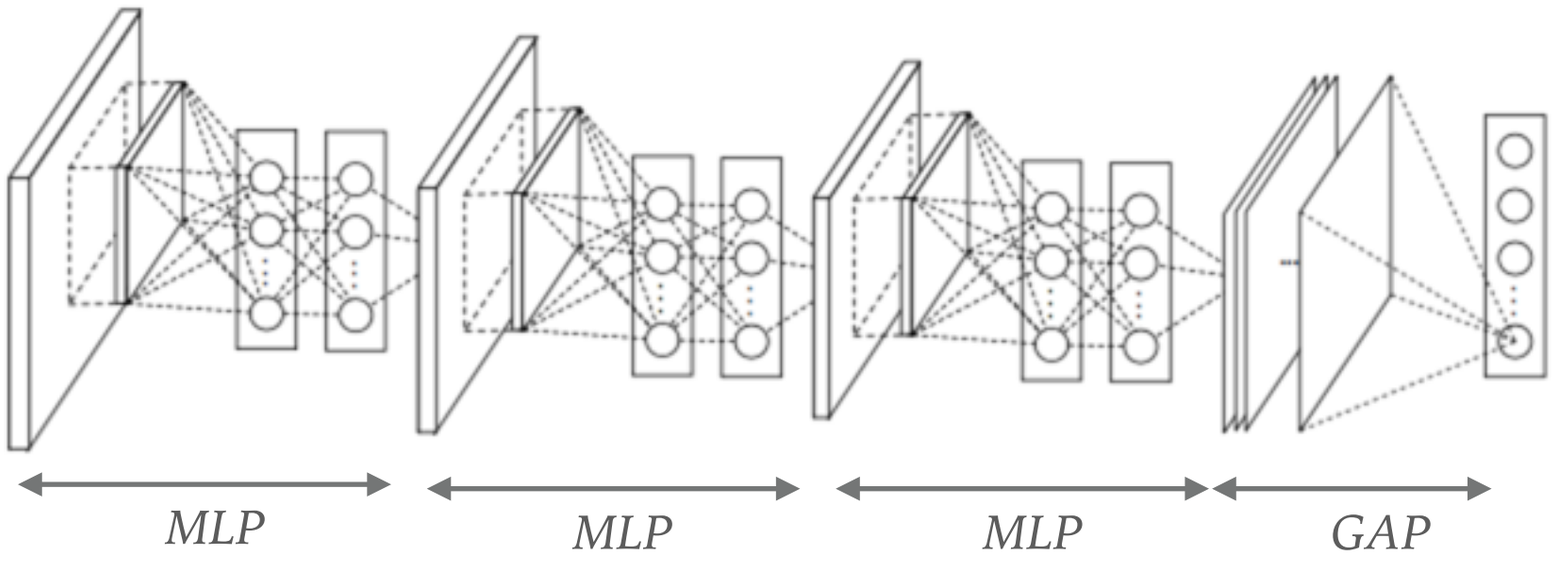
VGG #3 PARAMETER

, mini-batch 사이즈는 256
모멘텀은 0.9, Weight decay는 5∗10−4

- fc 2개 레이어에 dropout (p = 0.5)

weight 초기화는 AlexNet과 많이 다르다. 단순히 gaussian 분포를 통한 랜덤 초기화가 아니라 상대적으로 작은 네트워크 A를 AlexNet과 같은 초기화 한 후 트레이닝 한다. 그리고 트레이닝 된 작은 네트워크를 기반으로 다른 큰 네트워크를 fine-tuning 하는 기법을 사용한다.
이 때 첫 4개의 cov 레이어와 마지막 3개의 fc의 weight를 pre-train된 네트워크의 weigh로 사용하였으며, 나머지 레이어의 weight는 AlexNet과 동일하게 랜덤 초기화 하였다 (단, 바이어스는 0으로 초기화).

NETWORK IN NETWORK



convolution layer = GLM (Generalized Linear model) —*Non-linearity*→ *MLP*

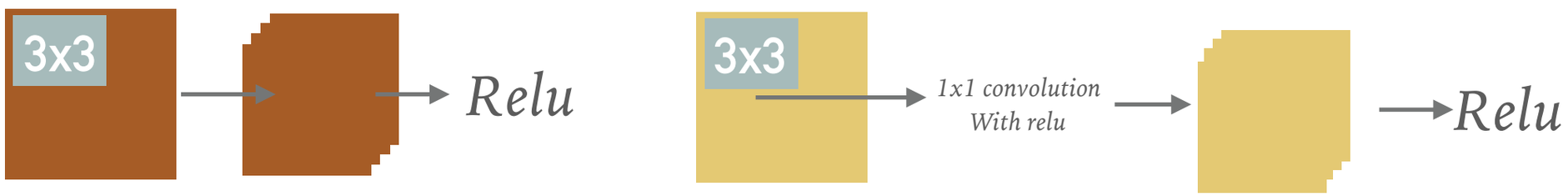
$$f^1_{i,j,k_1} = \max(w^{1\,T}_{k_1} x_{i,j} + b_{k_1}, 0).$$ weighted linear combination

$$\vdots$$

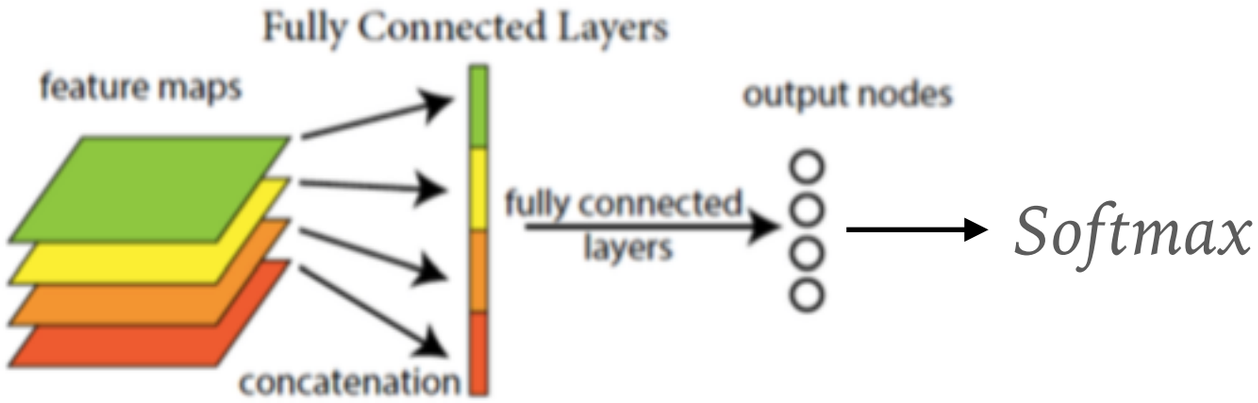
$$f^n_{i,j,k_n} = \max(w^{n\,T}_{k_n} f^{n-1}_{i,j} + b_{k_n}, 0).$$

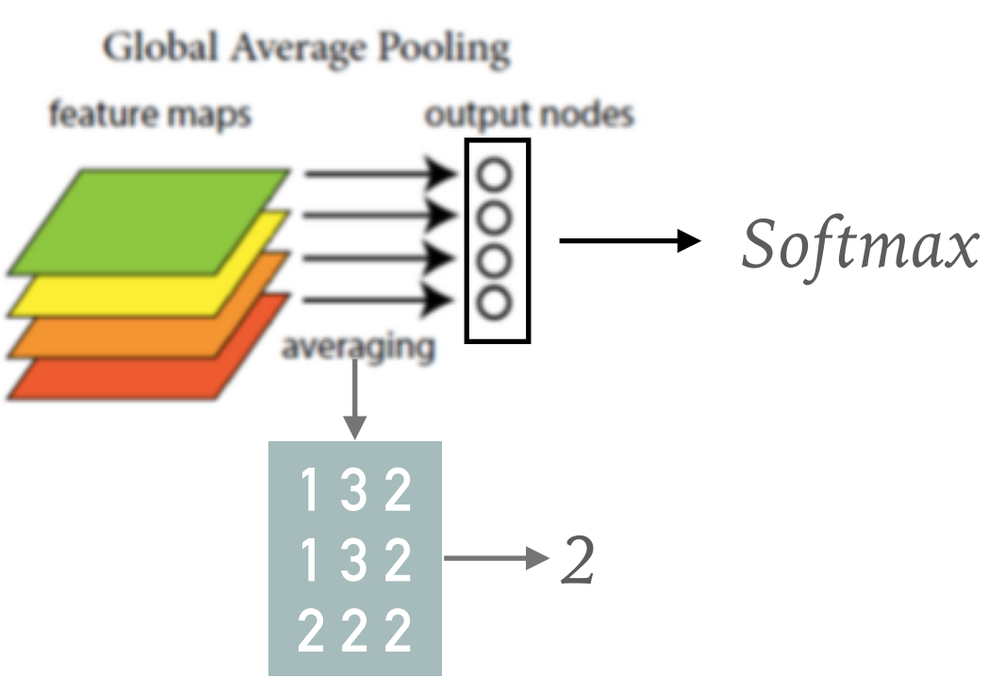
Cascaded Cross Channel Pooling (CCCP)

This cascaded cross channel parameteric pooling structure allows complex and learnable interactions of cross channel information".



NETWORK IN NETWORK #2 GLOBAL AVERAGE POOLING





1.avoid overfitting

2.filter already has spatial information

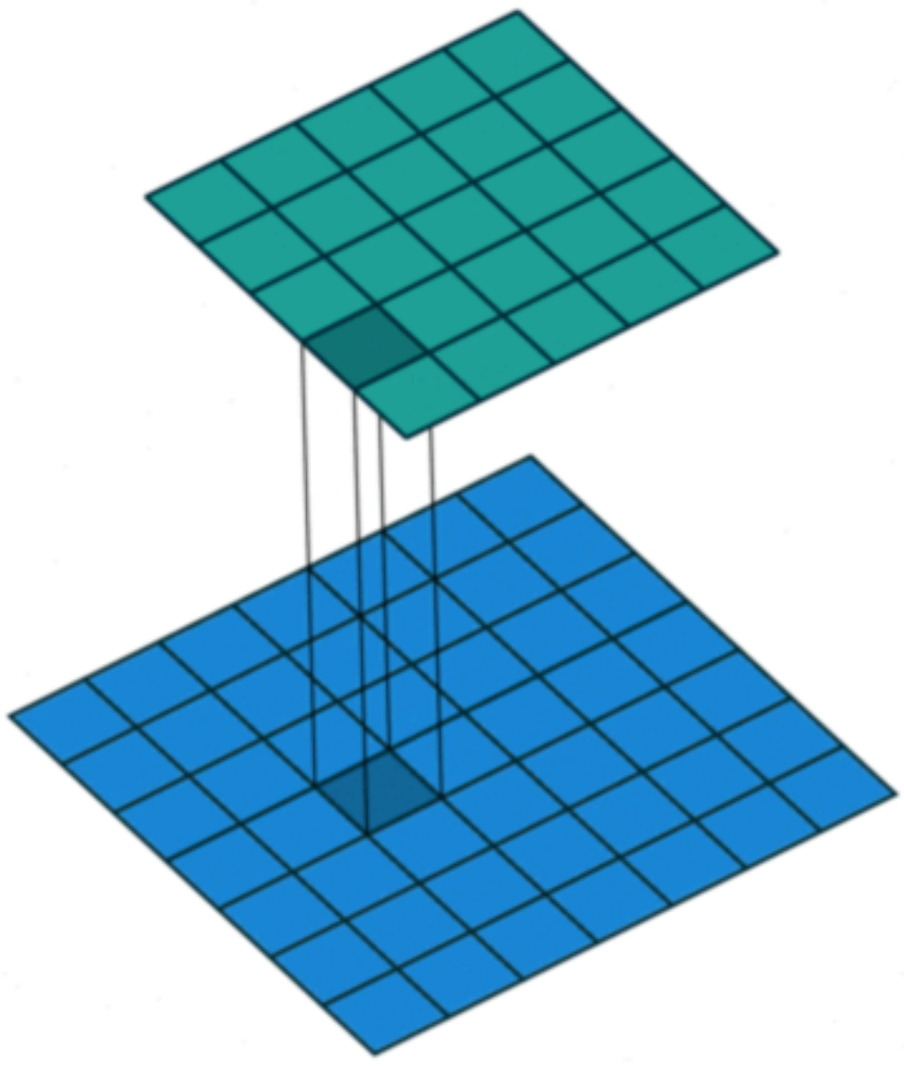
3.regularization

*4.**object localization*** → ***CAM (class Activation map)***

http://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf

 # Fc , dropout > gap > fc

NETWORK IN NETWORK 1X1 CONVOLUTION



1x1 convolution (feature pooling =Feature transformation)

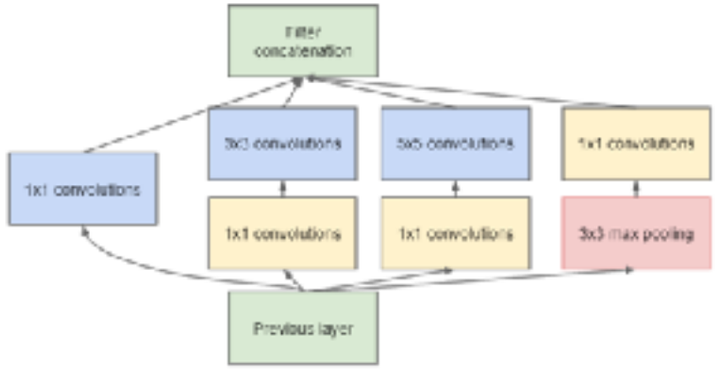
1.allow for making models deeper without impacting the receptive field,

2. decoupling the size of the representation from the size (and computational cost) of the model to some extent

3.This idea will be later used in most recent architectures as ResNet and Inception and derivatives.

4.dimension reductionality to reduce the number of feature

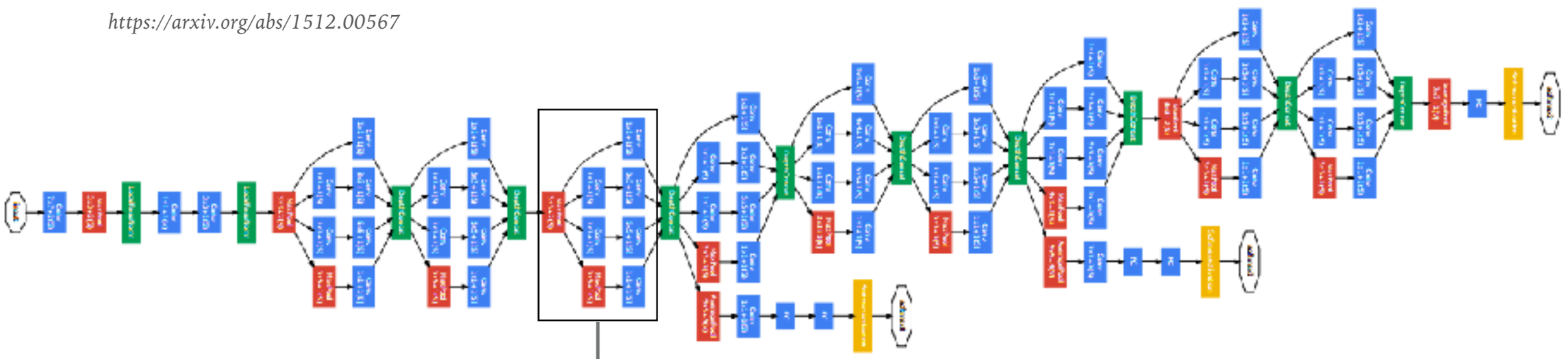
5. Can make layer more deeper

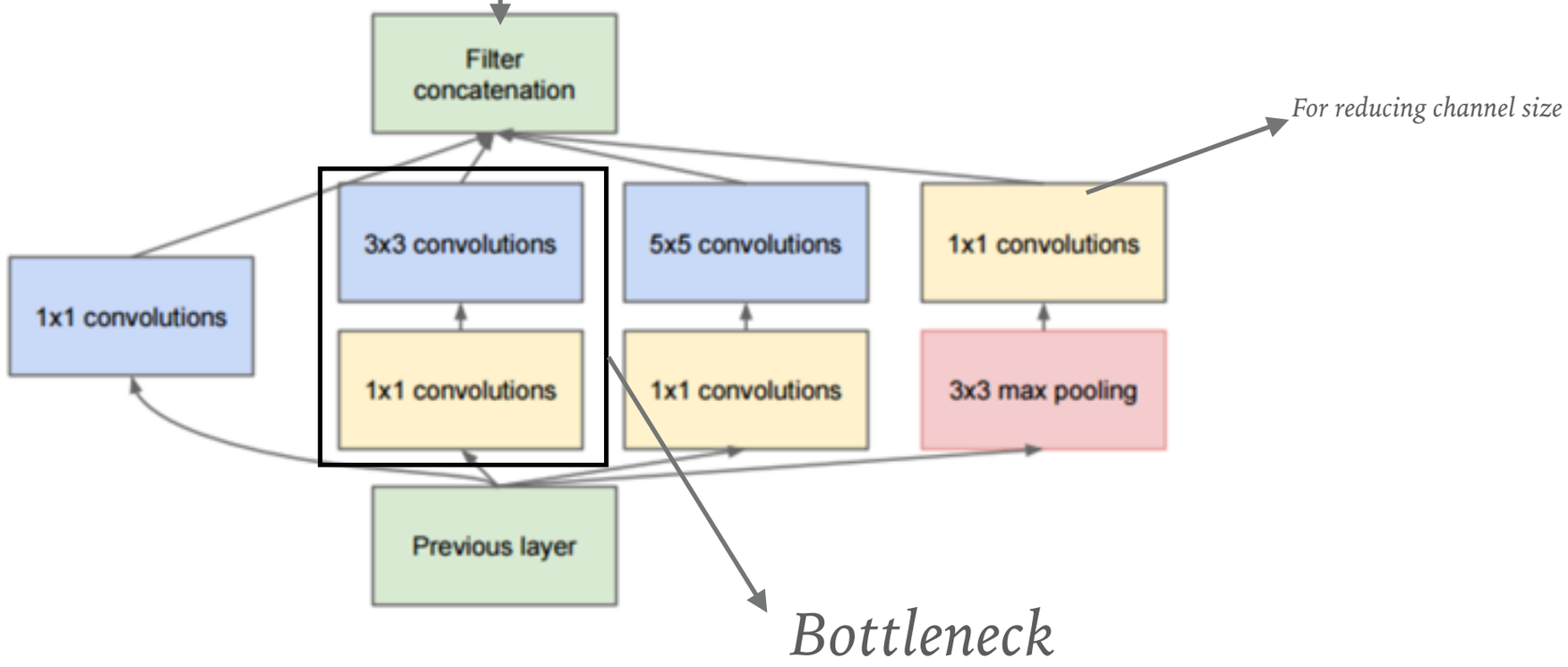


Google net

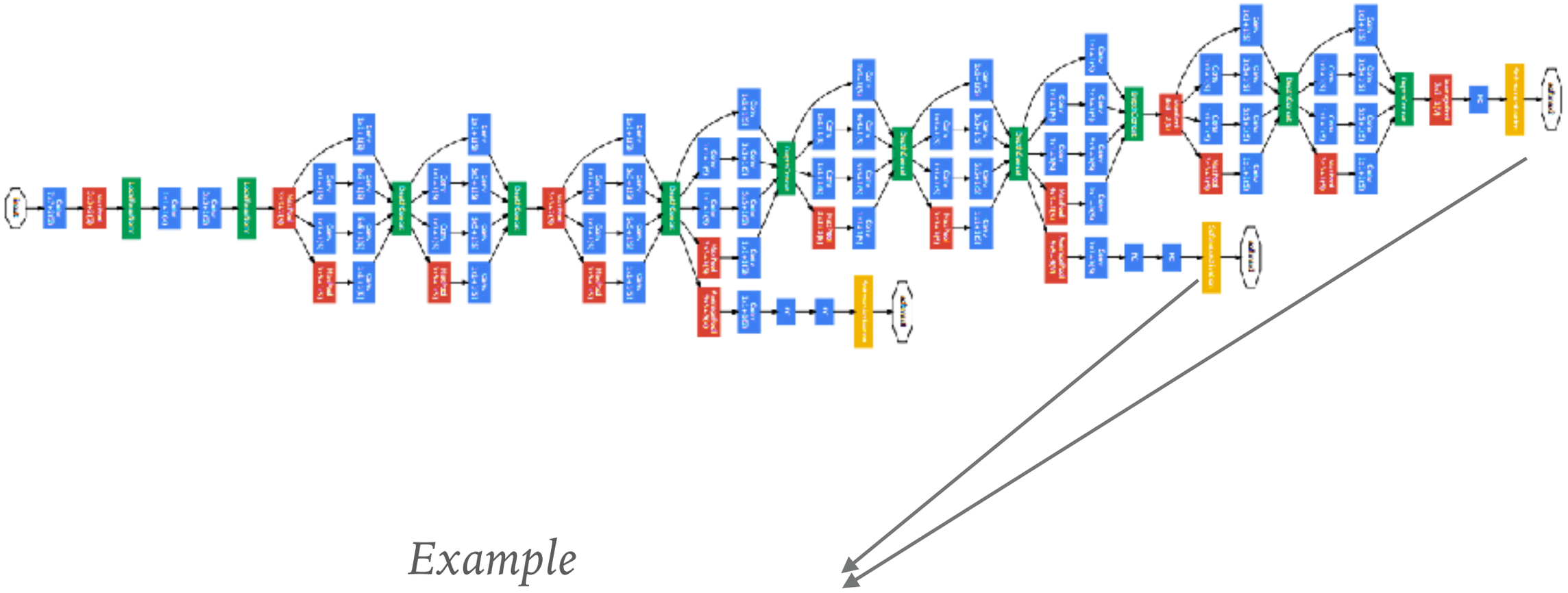
GOOGLE NET #1 INCEPTION V1 WITH BOTTLENECK

PAPER : https://arxiv.org/abs/1409.4842
v4:https://arxiv.org/abs/1602.07261
https://arxiv.org/abs/1512.00567





INCEPTION V1 #2 MULTI SOFTMAX



Example

0.1 | 0.05 | 0.1 | 0.4 | 0.2 | 0.05 | 0.1 | *x 0.3*

+

0.05 | 0.05 | 0.3 | 0.4 | 0.05 | 0.05 | 0.1 |

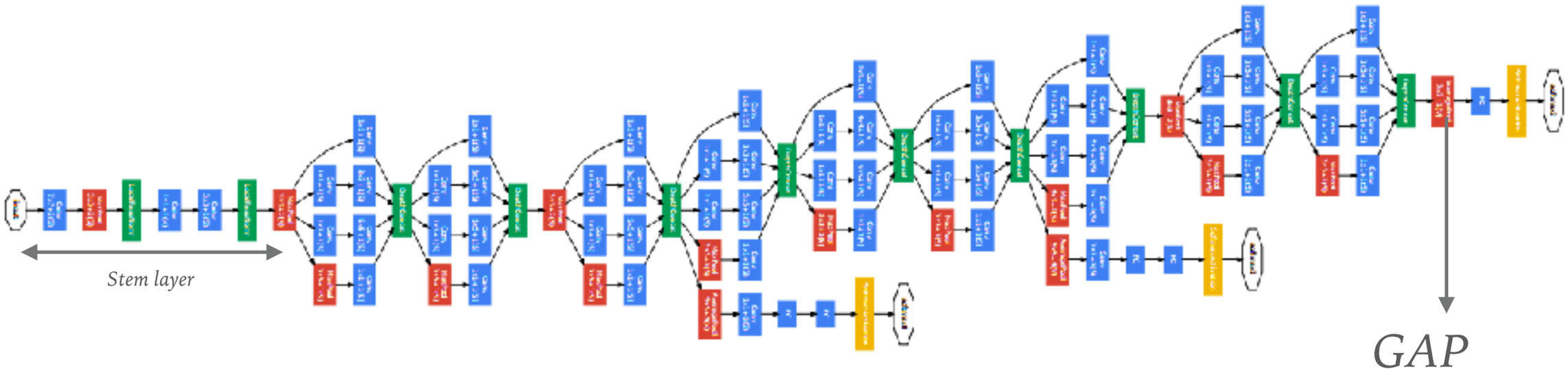
↓

0.15 | 0.1 | 0.4 | 0.8 | 0.25 | 0.1 | 0.2 |

Only used in training,

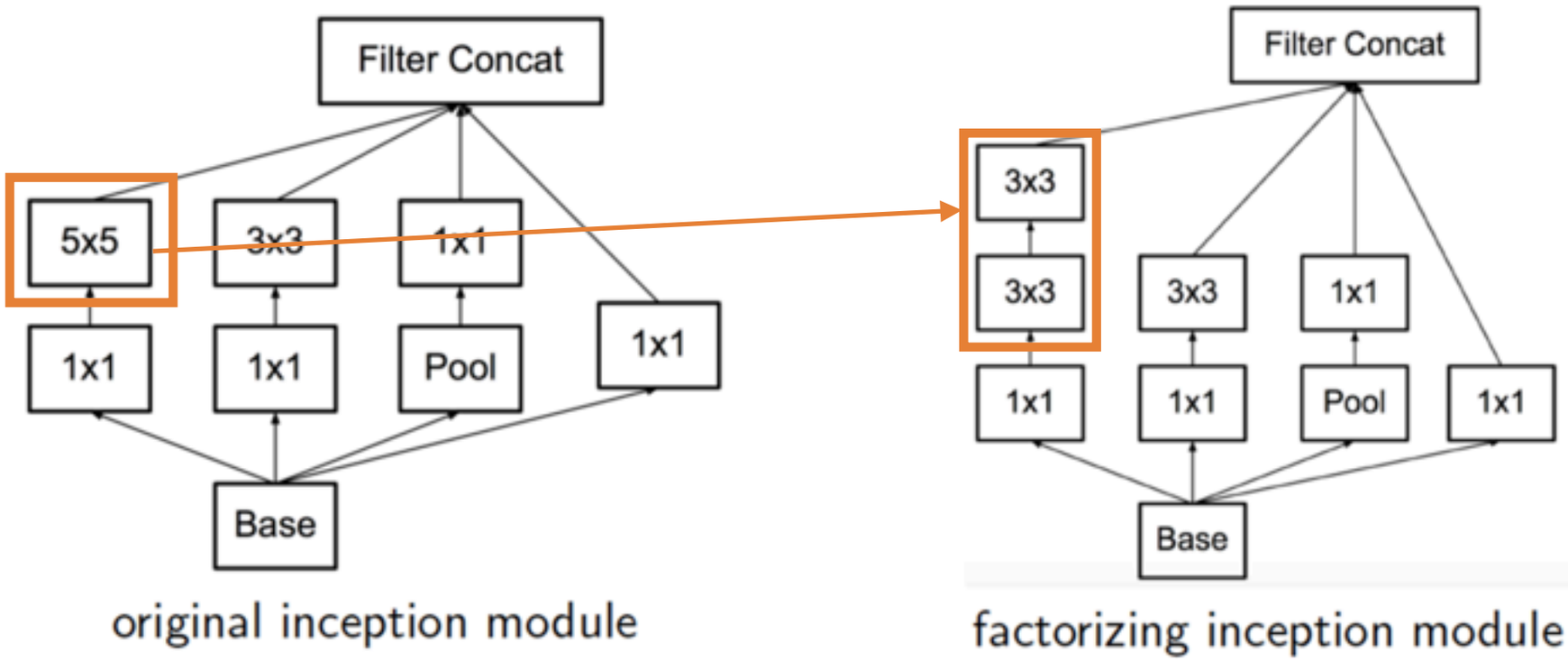
For test , ignore first and second softmax values

INCEPTION V1 #3 STEM LAYER | GAP



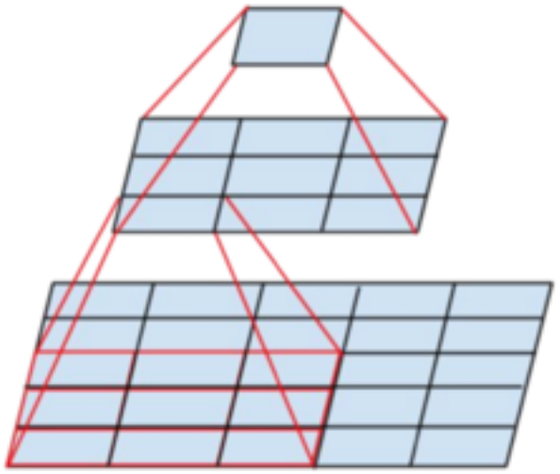
Without inception module as initial layer

INCEPTION V3 CONVOLUTION FATORIZING



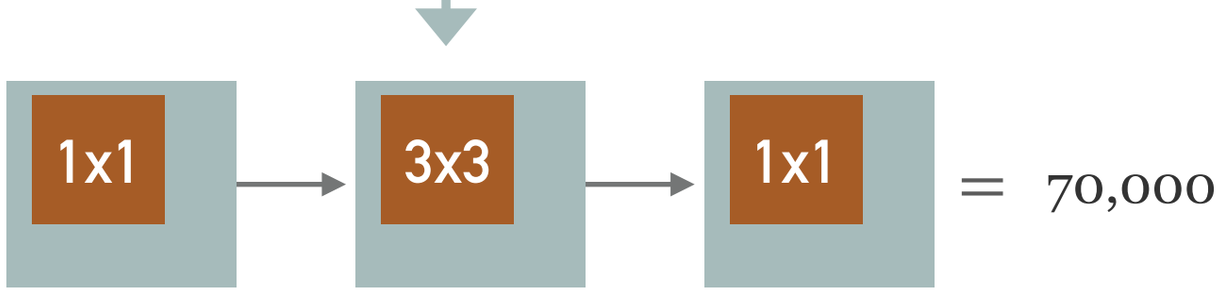
Same size of receptive field , low computation , more non linearity

INCEPTION V3 CONVOLUTION FATORIZING



256 features coming in, and 256 coming out,
and let's say the Inception layer only performs 3x3

Computation
256x256 x 3x3=589,000



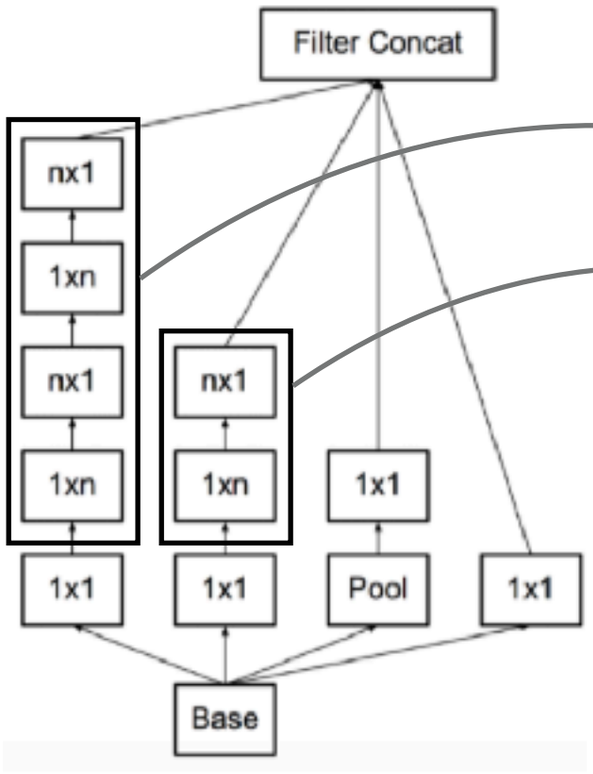
= 70,000

256×64 × 1×1 = 16,000s

64×64 × 3×3 = 36,

64×64 × 3×3 = 36,000s

ASYMMETIRC CONVOLUTION FACTORIZAING & POOLING

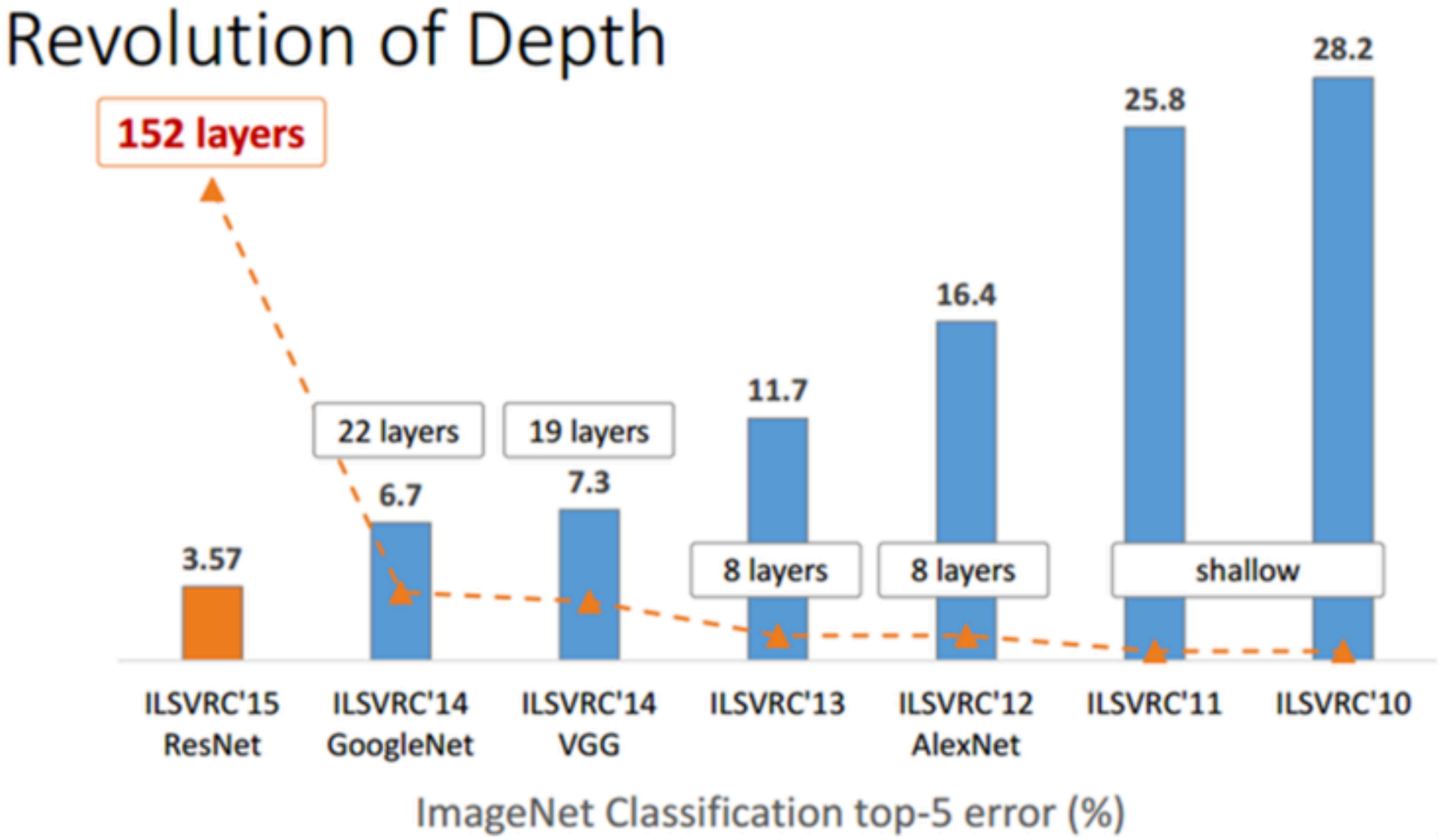


$256 \times 64 \times 3x1 + 64 \times 64 \times 3x1 + 64 \times 64 \times 3x1 + 64 \times 64 \times 3x1$

?

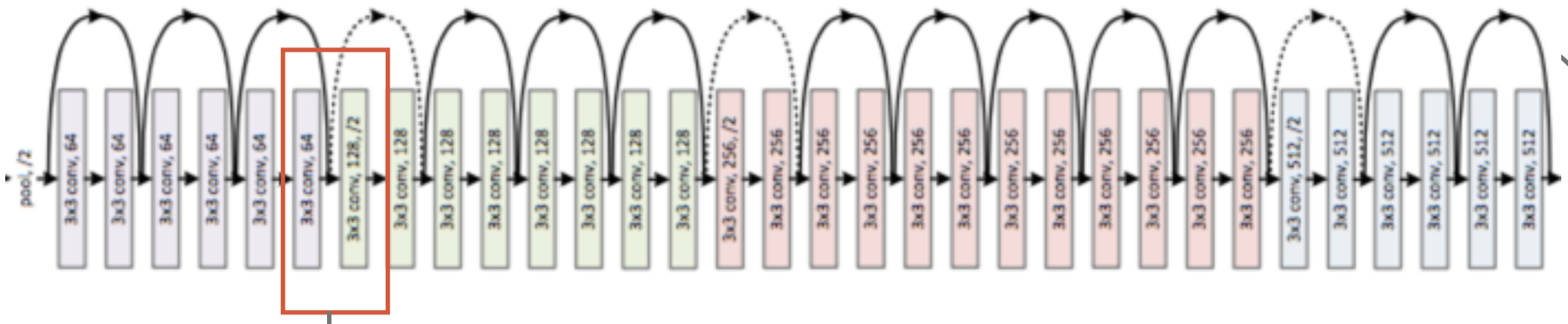
Network	Top-1 Error	Top-5 Error	Cost Bn Ops	
GoogLeNet [20]	29%	9.2%	1.5	
BN-GoogLeNet	26.8%	-	**1.5**	
BN-Inception [7]	25.2%	7.8	2.0	
Inception-v2	23.4%	-	3.8	
Inception-v2 RMSProp	23.1%	6.3	3.8	→ *RMS Optimizer*
Inception-v2 Label Smoothing	22.8%	6.1	3.8	→ *0 + e*
Inception-v2 Factorized 7×7	21.6%	5.8	4.8	→ *7x7 convolution layer —> 3x3 , 3x3 convolution*
Inception-v2 BN-auxiliary	**21.2%**	**5.6%**	4.8	→ *Adopt Batch normalization Only Fully connected layer*

RESNET


Revolution of Depth
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10
ImageNet Classification top-5 error (%)

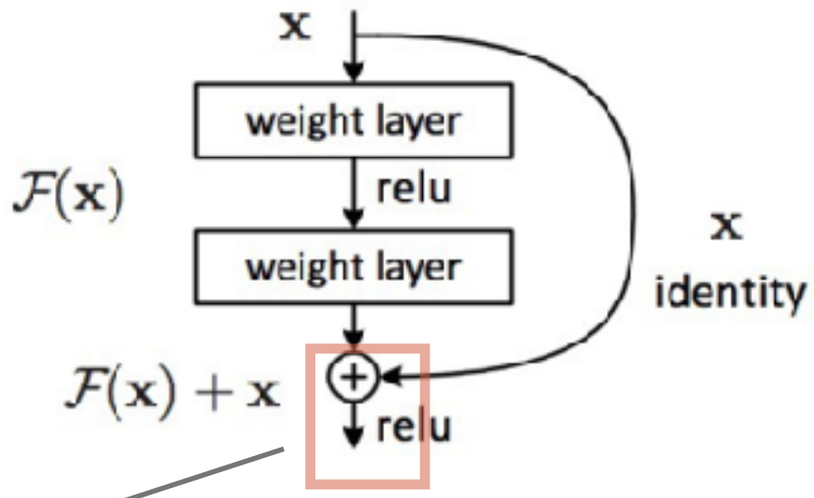
RESNET

https://arxiv.org/pdf/1512.03385.pdf



Increase the number of channel
decrease image size

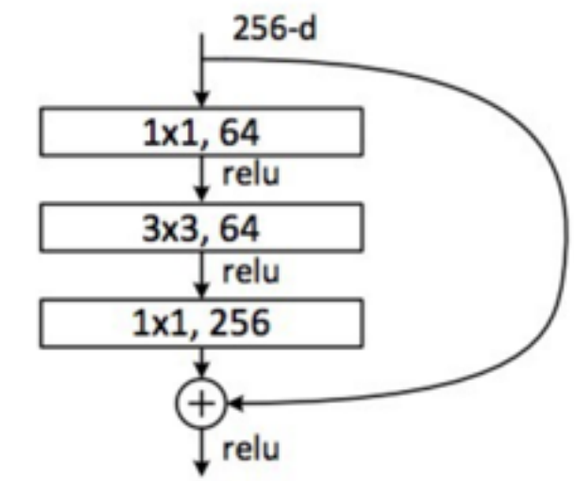
Short cut layer

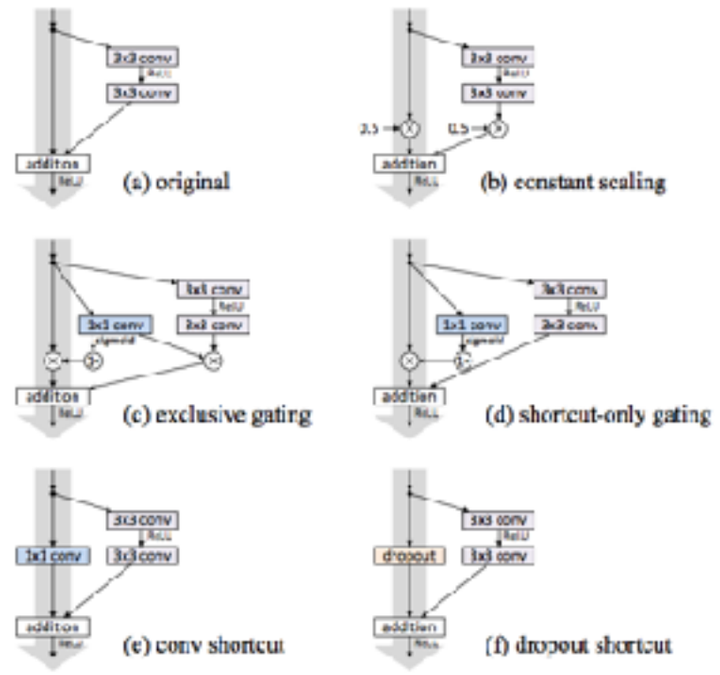


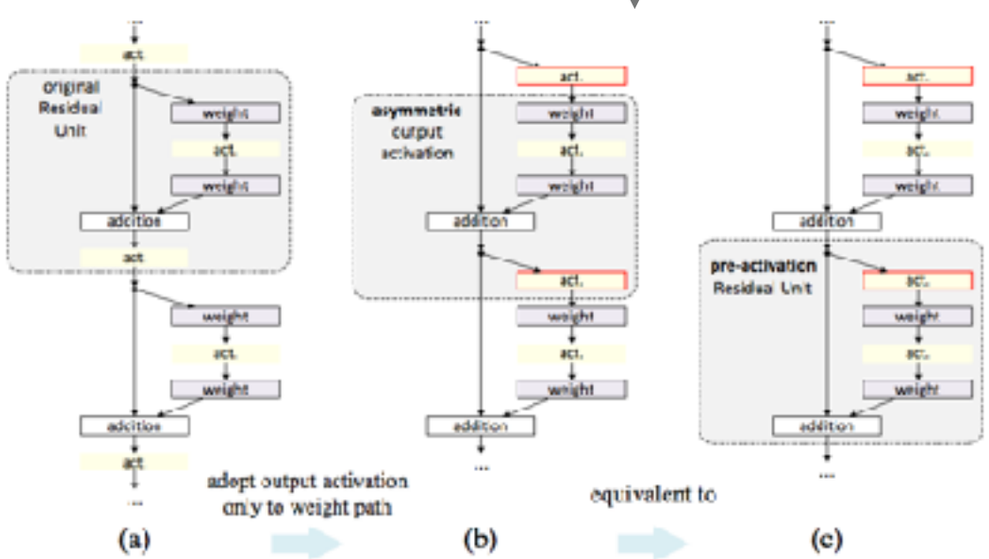
1	1	1		1	1	1		2	2	2
2	2	2	+	3	3	3	=	5	5	5
3	3	3		4	4	4		7	7	7

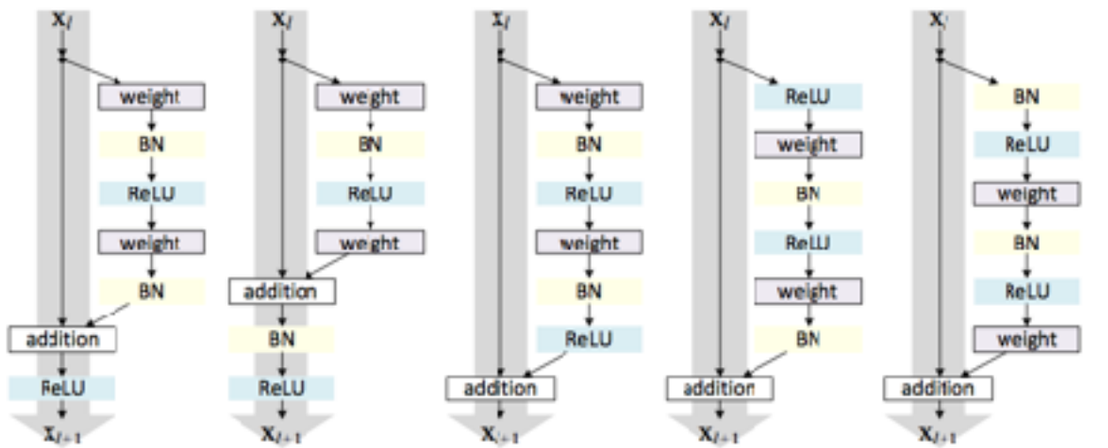
RESENT BOTTLECT LAYER

https://arxiv.org/abs/1603.05027









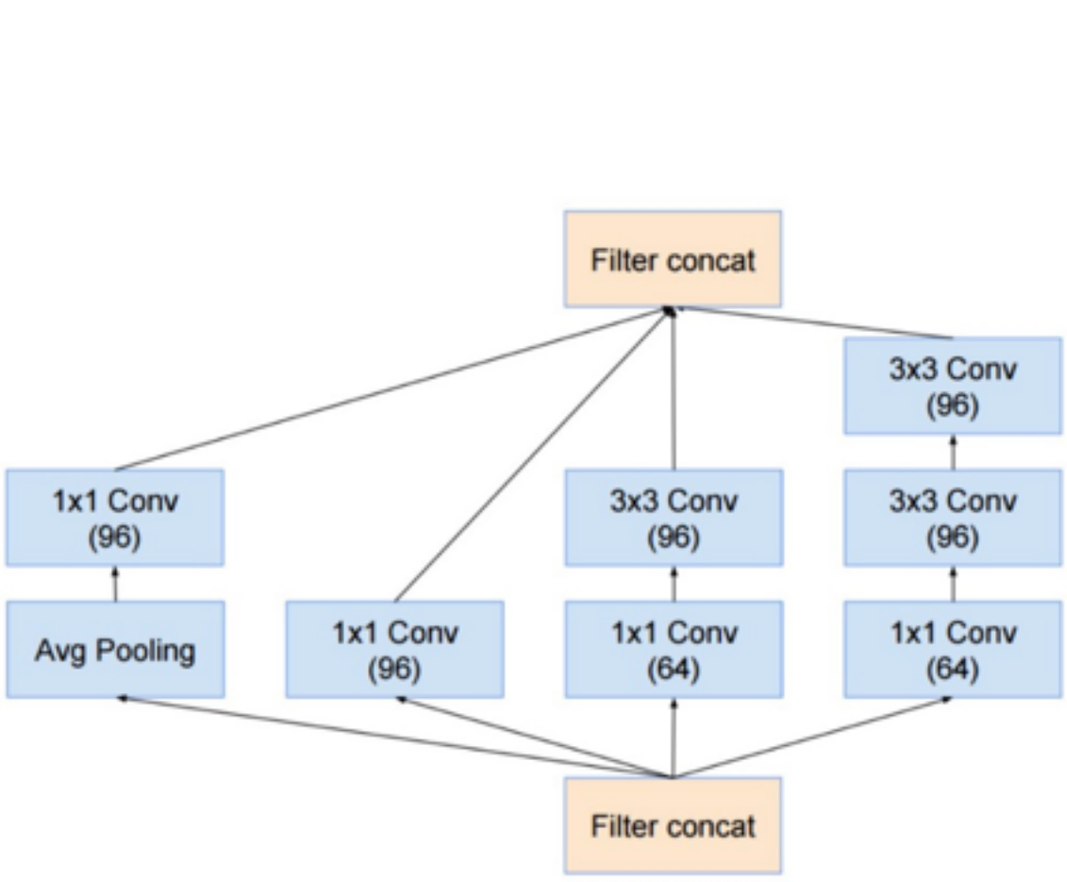
RESNET

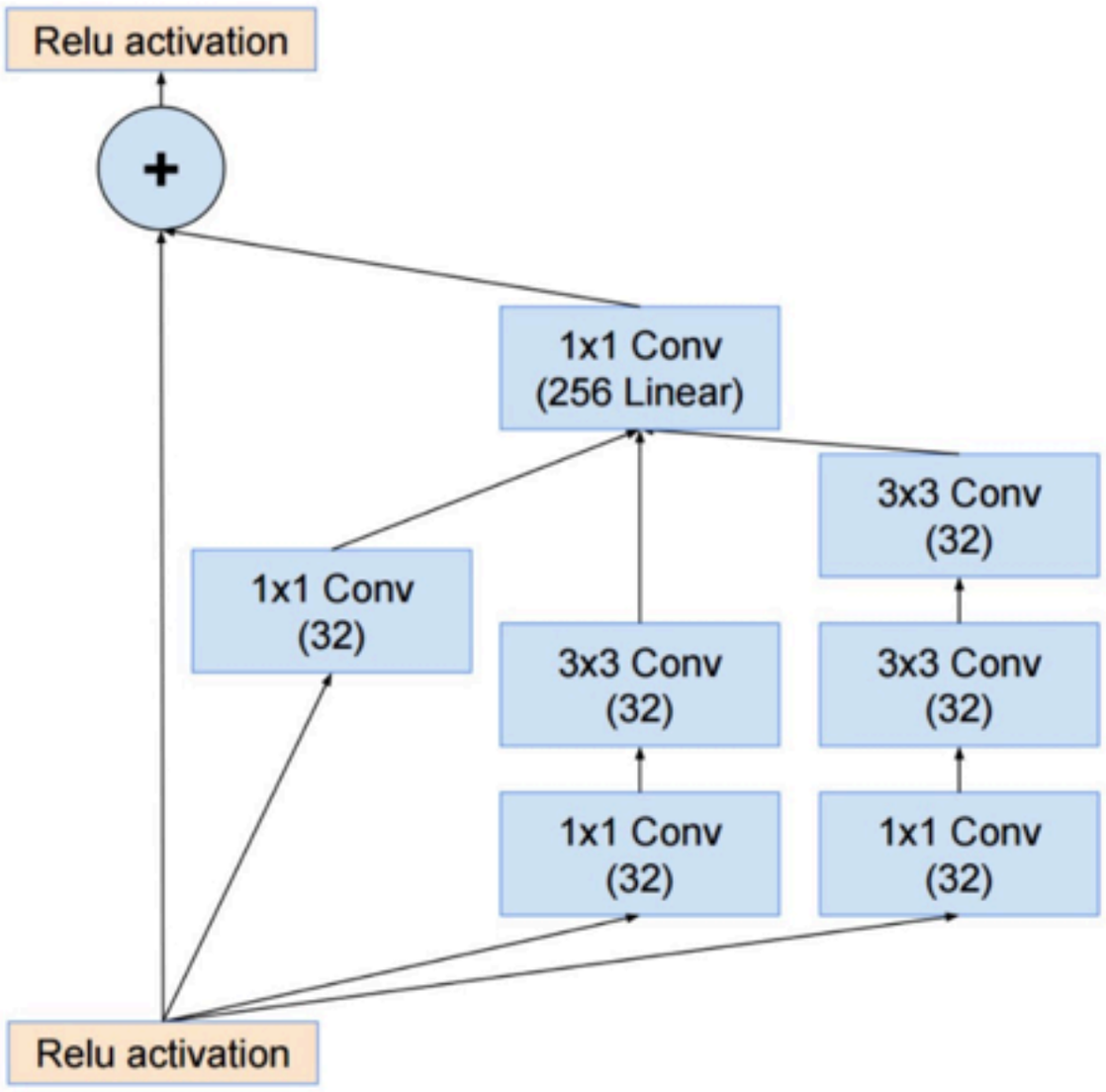
CIFAR-10	error (%)
NIN [15]	8.81
DSN [16]	8.22
FitNet [17]	8.39
Highway [7]	7.72
All-CNN [14]	7.25
ELU [12]	6.55
~~FitResNet, LSUV [18]~~	~~5.84~~
ResNet-110 [1] (1.7M)	6.61
ResNet-1202 [1] (19.4M)	7.93
ResNet-164 [ours] (1.7M)	5.46
ResNet-1001 [ours] (10.2M)	4.92 (4.89±0.14)
ResNet-1001 [ours] (10.2M)†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

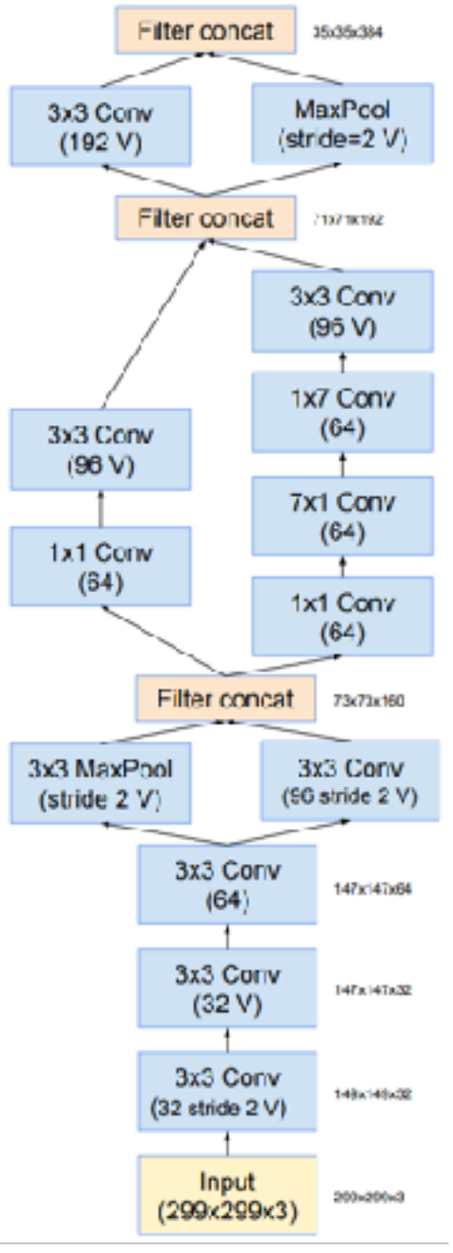
method	augmentation	train crop	test crop	top-1	top-5
ResNet-152, original Residual Unit [1]	scale	224×224	224×224	23.0	6.7
ResNet-152, original Residual Unit [1]	scale	224×224	320×320	21.3	5.5
ResNet-152, **pre-act** Residual Unit	scale	224×224	320×320	21.1	5.5
ResNet-200, original Residual Unit [1]	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





INCEPTION V4



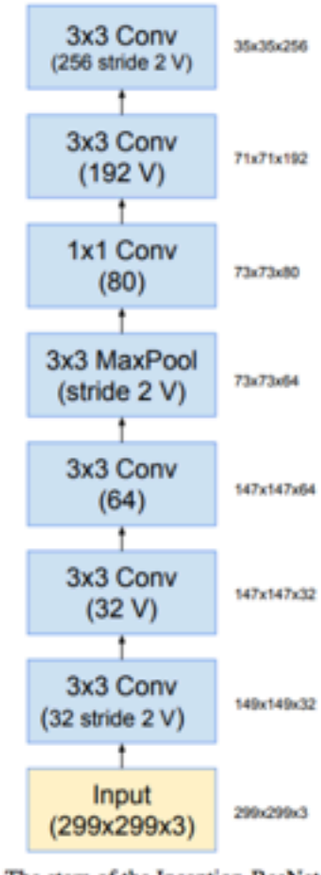


Figure 14. The stem of the Inception-ResNet-v1 network.

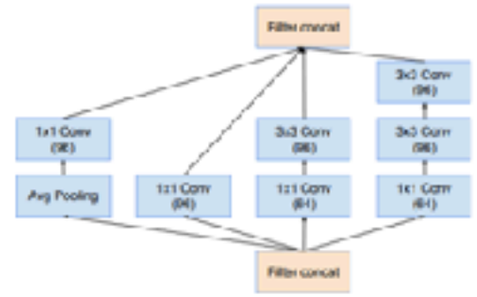


Figure 4. The schema for 35 × 35 grid modules of the pure Inception-v4 network. This is the Inception-A block of Figure 9.

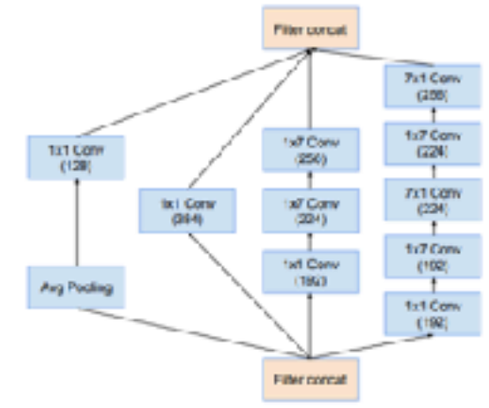


Figure 5. The schema for 17 × 17 grid modules of the pure Inception-v4 network. This is the Inception-B block of Figure 9.

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

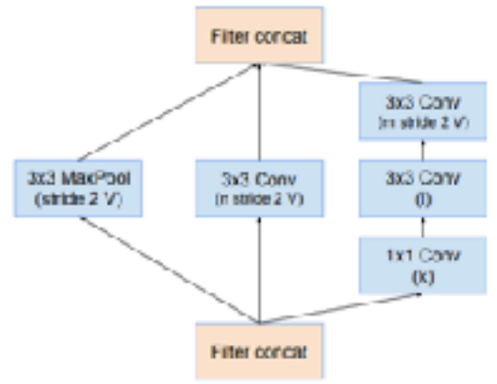


Figure 7. The schema for 35 × 35 to 17 × 17 reduction module. Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The *k, l, m, n* numbers represent filter bank sizes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

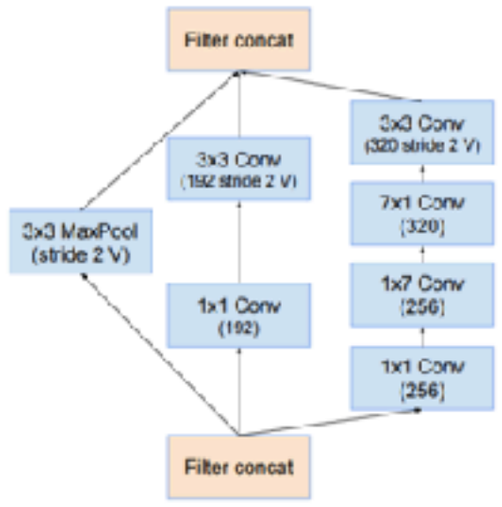


Figure 8. The schema for 17 × 17 to 8 × 8 grid-reduction module. This is the reduction module used by the pure Inception-v4 network in Figure 9.

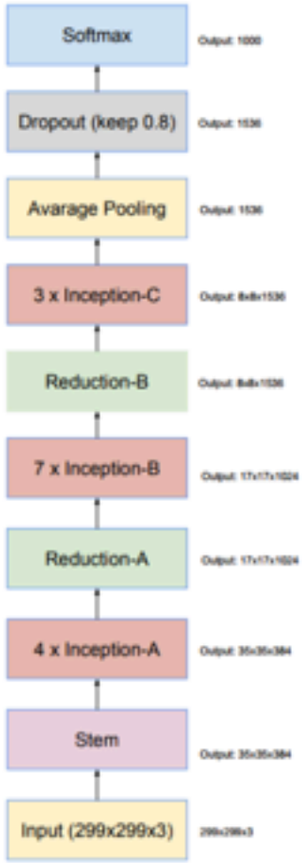


Figure 9. The overall schema of the Inception-v4 network. For the detailed modules, please refer to Figures 3, 4, 5, 6, 7 and 8 for the detailed structure of the various components.

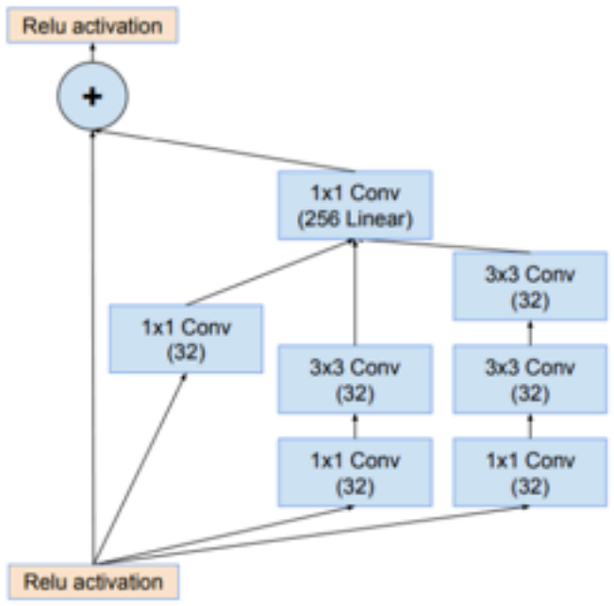


Figure 10. The schema for 35 × 35 grid (Inception-ResNet-A) module of Inception-ResNet-v1 network.

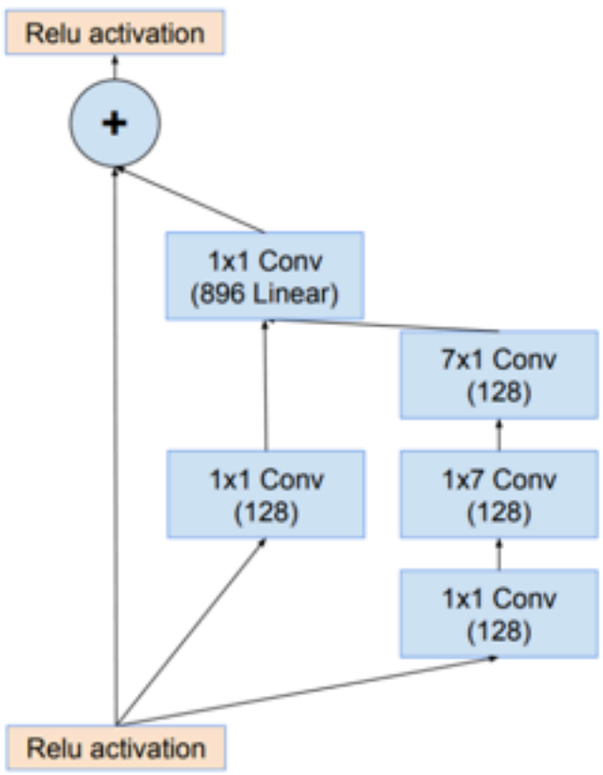


Figure 11. The schema for 17 × 17 grid (Inception-ResNet-B) module of Inception-ResNet-v1 network.

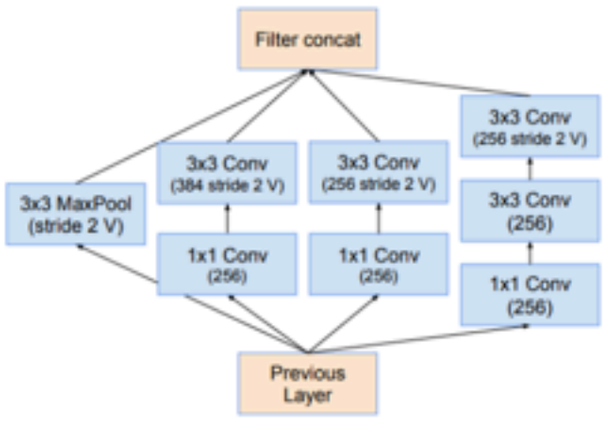


Figure 12. "Reduction-B" 17 × 17 to 8 × 8 grid-reduction module. This module used by the smaller Inception-ResNet-v1 network in Figure 15.

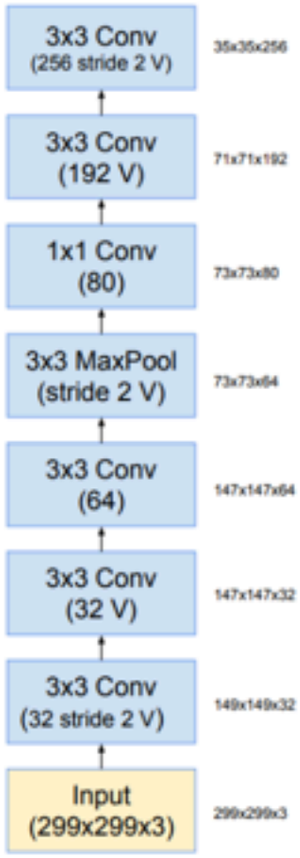


Figure 14. The stem of the Inception-ResNet-v1 network.

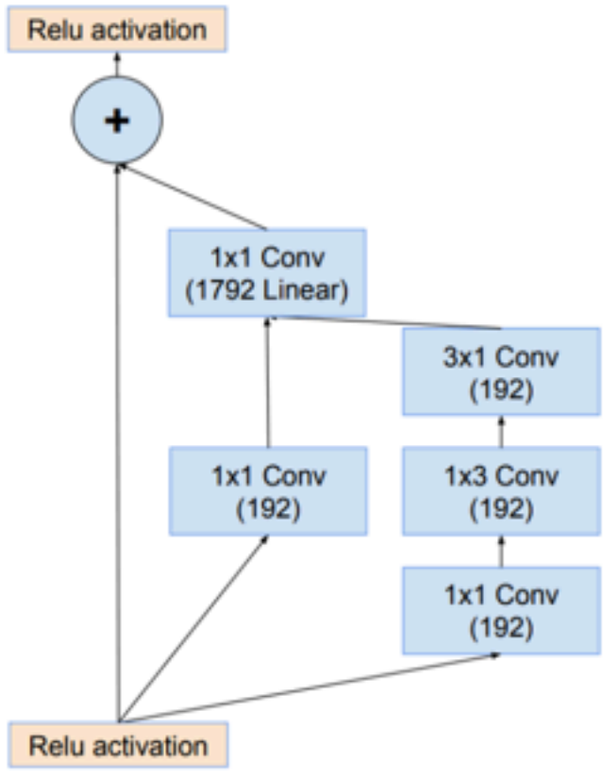


Figure 13. The schema for 8×8 grid (Inception-ResNet-C) module of Inception-ResNet-v1 network.