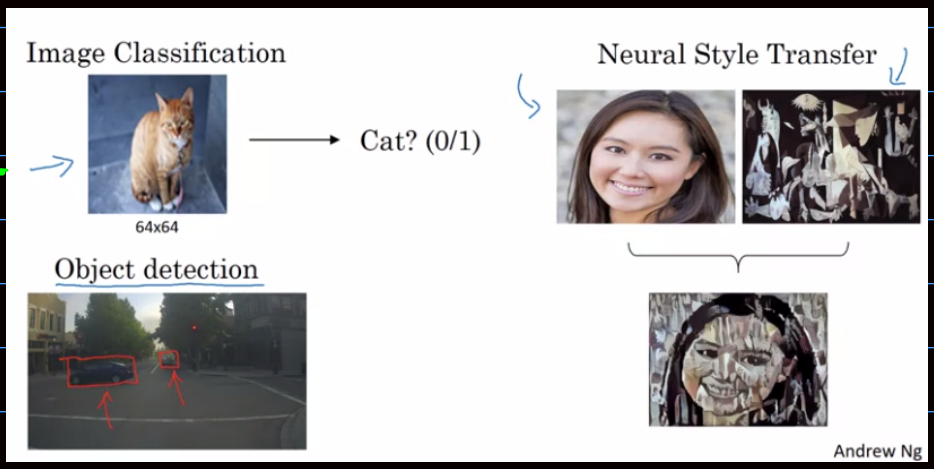
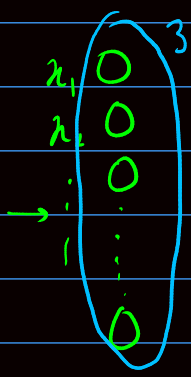
# Convolutional Neural Networks

## Week 1

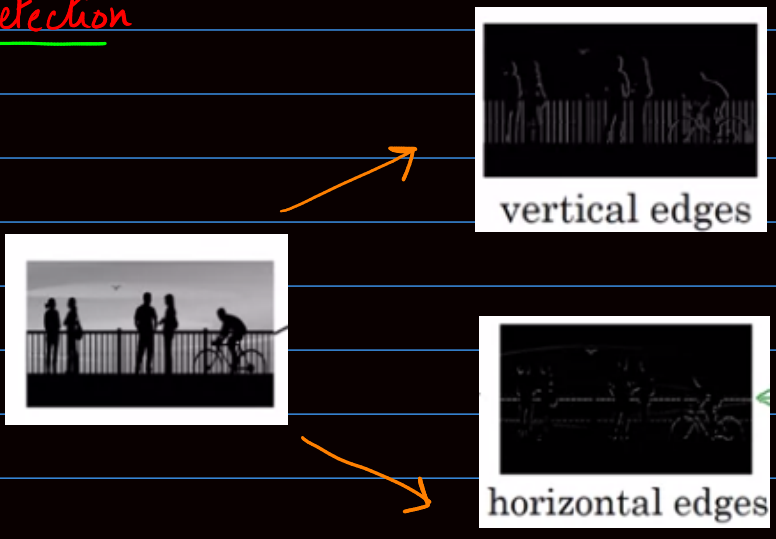
Examples : Image Classification , Object Detection , Neural Style Transfer



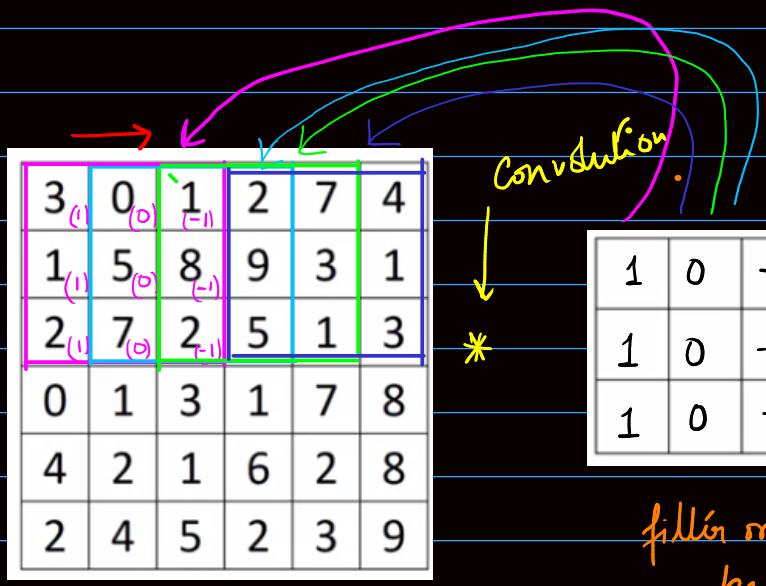

If we use 1000 x 1000 x 3 pixels then




## Edge Detection



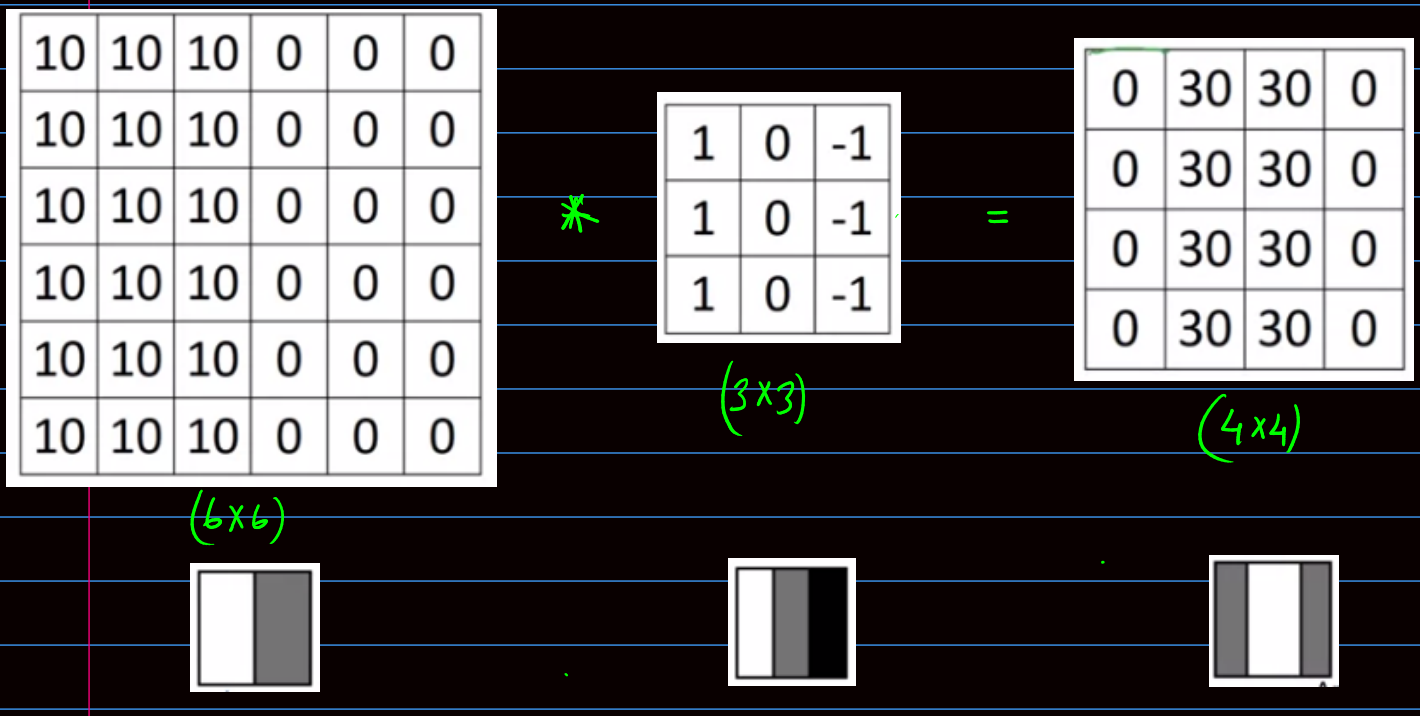
# Vertical Edge Detection




(6 x 6 x 1) * (3 x 3) = (4 x 4 x 1)

filter or kernal

$3\times1 + 1\times1 + 2\times1 + 0\times0 + 5\times0 + 7\times0 + 1\times(-1) + 8\times(-1) + 2\times(-1) = -5$

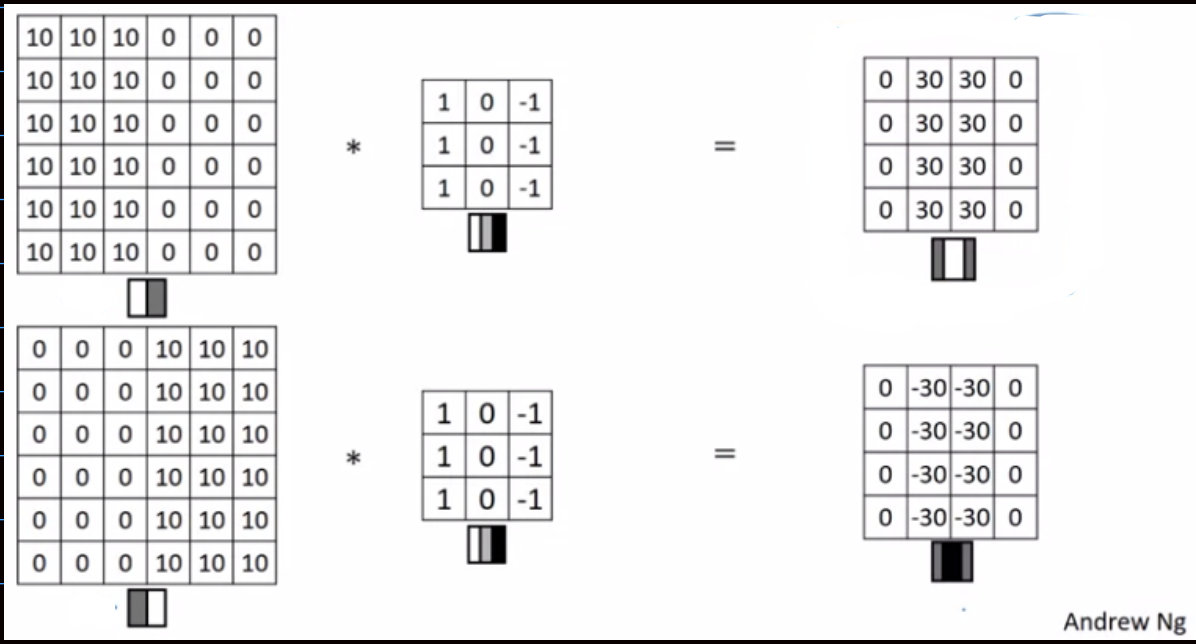



(6x6) * (3x3) = (4x4)

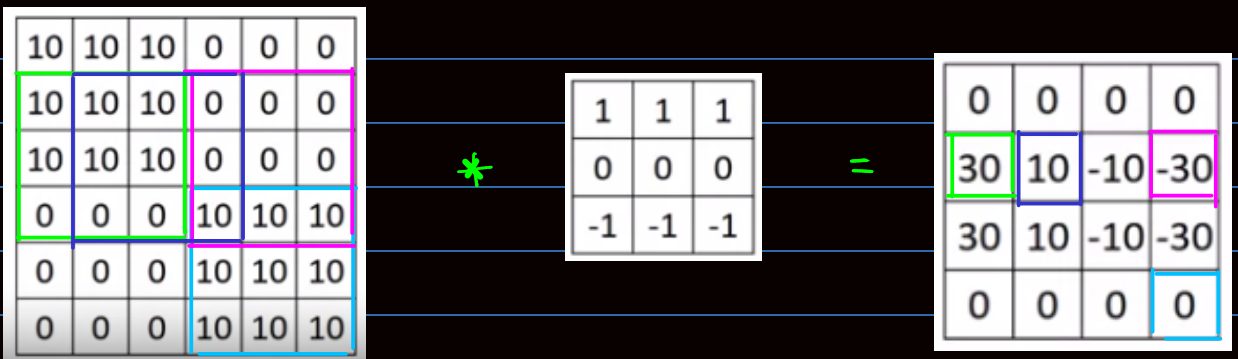


Since we are using very small sized image the edges seems very thick but when we use very large image the results does a pretty good job

# Vertical . Edge Detection Examples.








→ Other filters

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 2 | 0 | -2 |
| 1 | 0 | -1 |

Sobel filter

| | | |
|---|---|---|
| 3 | 0 | -3 |
| 10 | 0 | -10 |
| 3 | 0 | -3 |

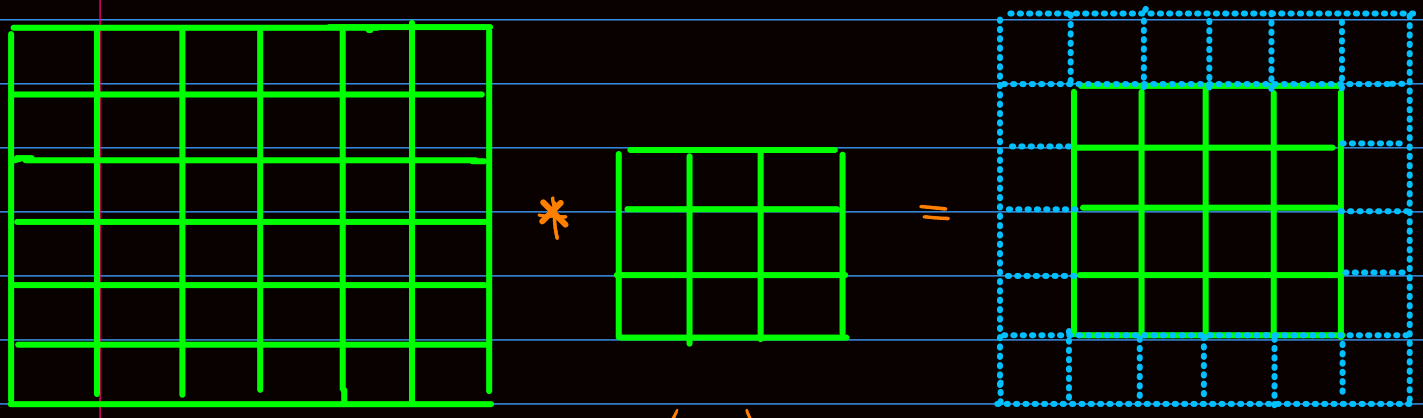
Scharr filter

But in neural networks, these filters are not handpicked, instead they are treated as parameters in a network which is automatically figured out by the network

| | | |
|---|---|---|
| $w_1$ | $w_2$ | $w_3$ |
| $w_4$ | $w_5$ | $w_6$ |
| $w_7$ | $w_8$ | $w_9$ |

With this apart from vertical & horizontal parameters we can detect 45°, 70°, 73°, 60° edges as well even more robustly.

## PADDING




* The o/p size will go on shrinking as we apply more filters after the other.
* The edge pixels will be used much lesser during computation than those in the middle

* Padding is used (.....) in the figure to get the same sized.

image & pad with 0 values (generally)
Here Padding = 1

## Valid & Same Convolutions

Valid : $(n \times n) * (f \times f) \longrightarrow (n-f+1) \times (n-f+1)$ → No padding

Same : Pad so that the o/p size = i/p size

$$(n+2p-f+1) = n$$

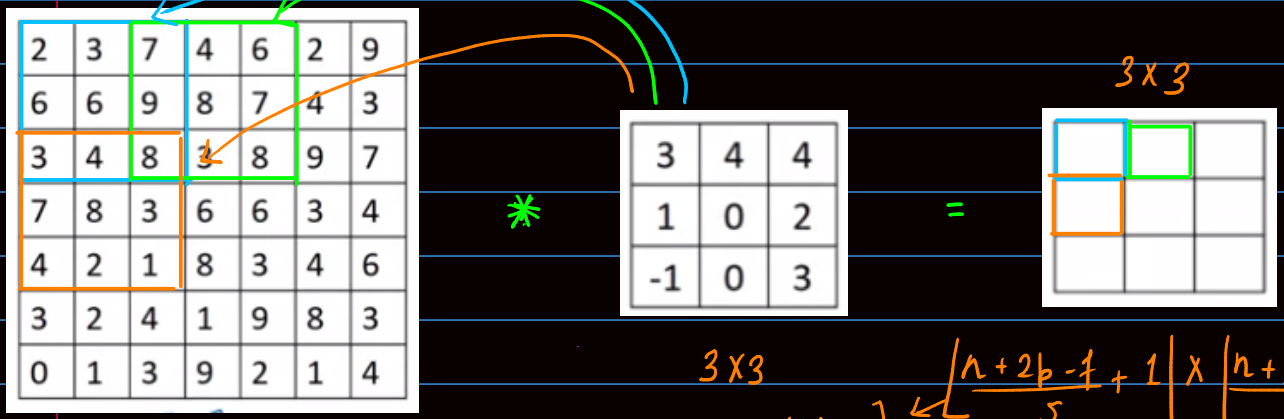
$$p = \frac{(f-1)}{2}$$

Now

$$(n \times n) * (f \times f) = (n+2p-f+1) \times (n+2p-f+1)$$

* f is usually odd. (3x3, 5x5, 7x7 ....)

## STRIDED CONVOLUTIONS




$$\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right]$$

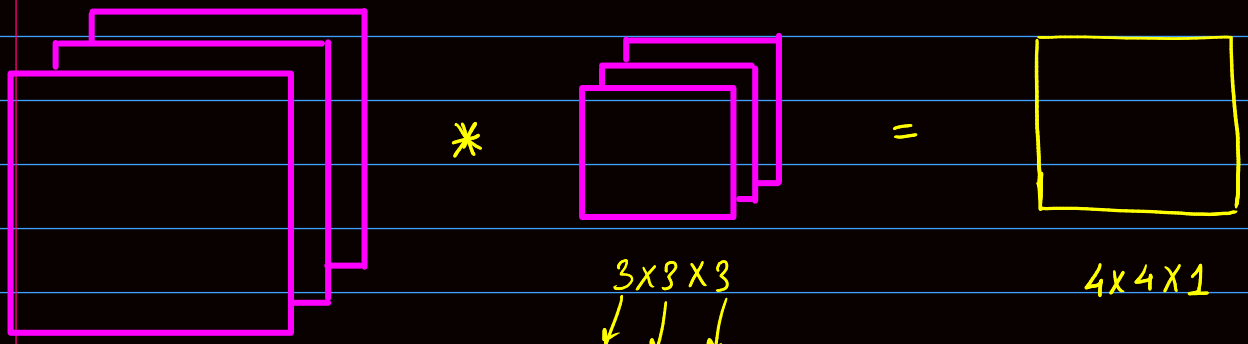
[floor] [floor]

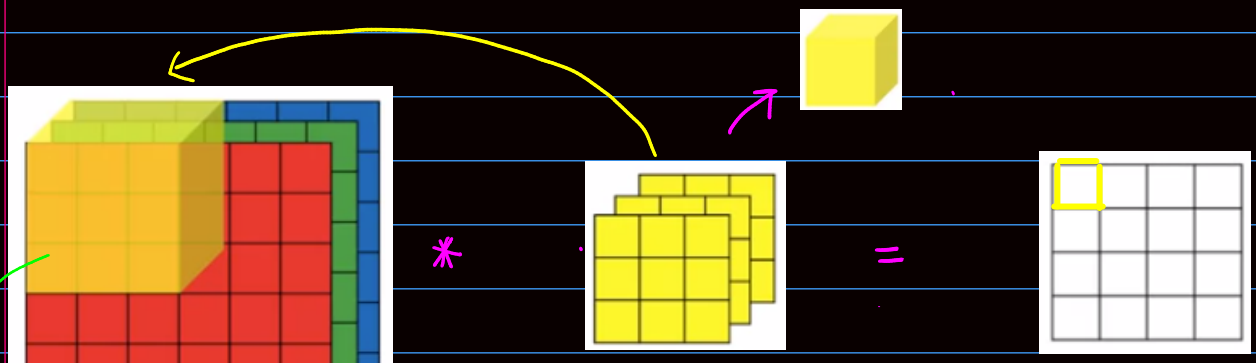
Stride = 2 [Scan by jumping 2 steps at a time]

Eg: $\left(\frac{7-0-3}{2}+1=3\right)$

* Caution : Don't follow the math & SS textbook for convolution formula.
In ML actually convolution is cross-convolution.

# Convolutions over Volumes




27 terms in the multiplication

Eg

R

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

G

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

B

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

↳ Red channel edge detector

R

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

G

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

B

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

All channel edge detector

## Multiple filters

$$n \times n \times n_c \quad * \quad (f \times f \times n_c) \rightarrow (n-f+1) \times (n-f+1) \times n_c'$$

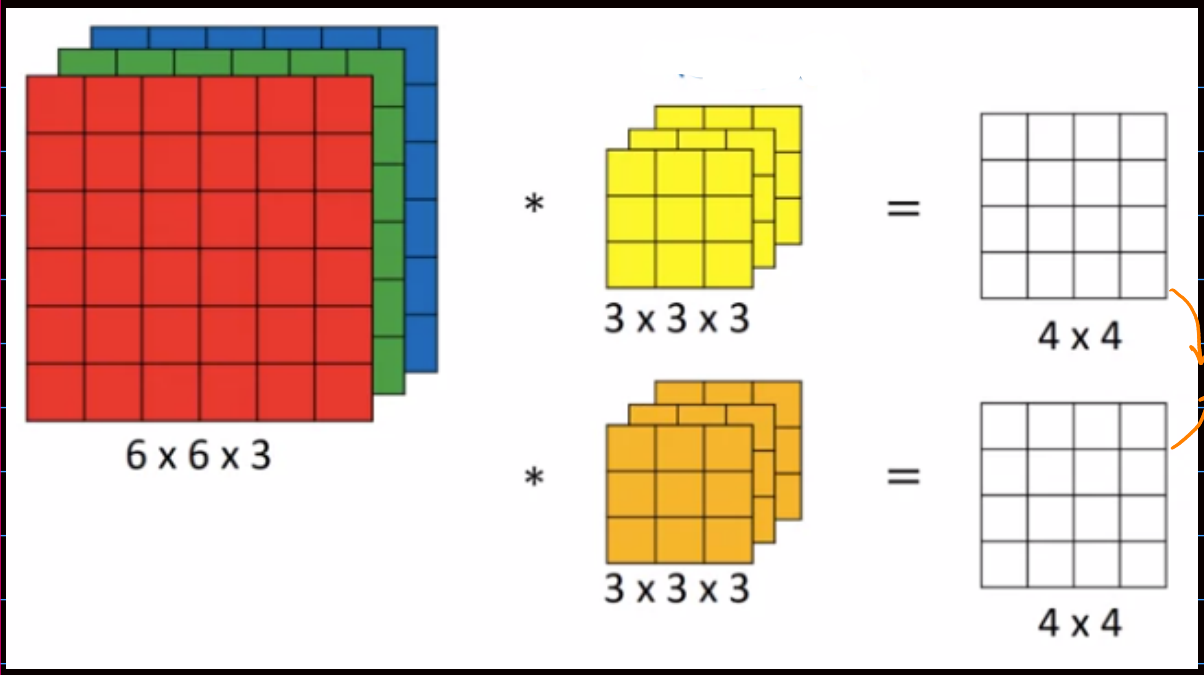
$$(f \times f \times n_c)_1, (f \times f \times n_c)_2$$

no. of filters used

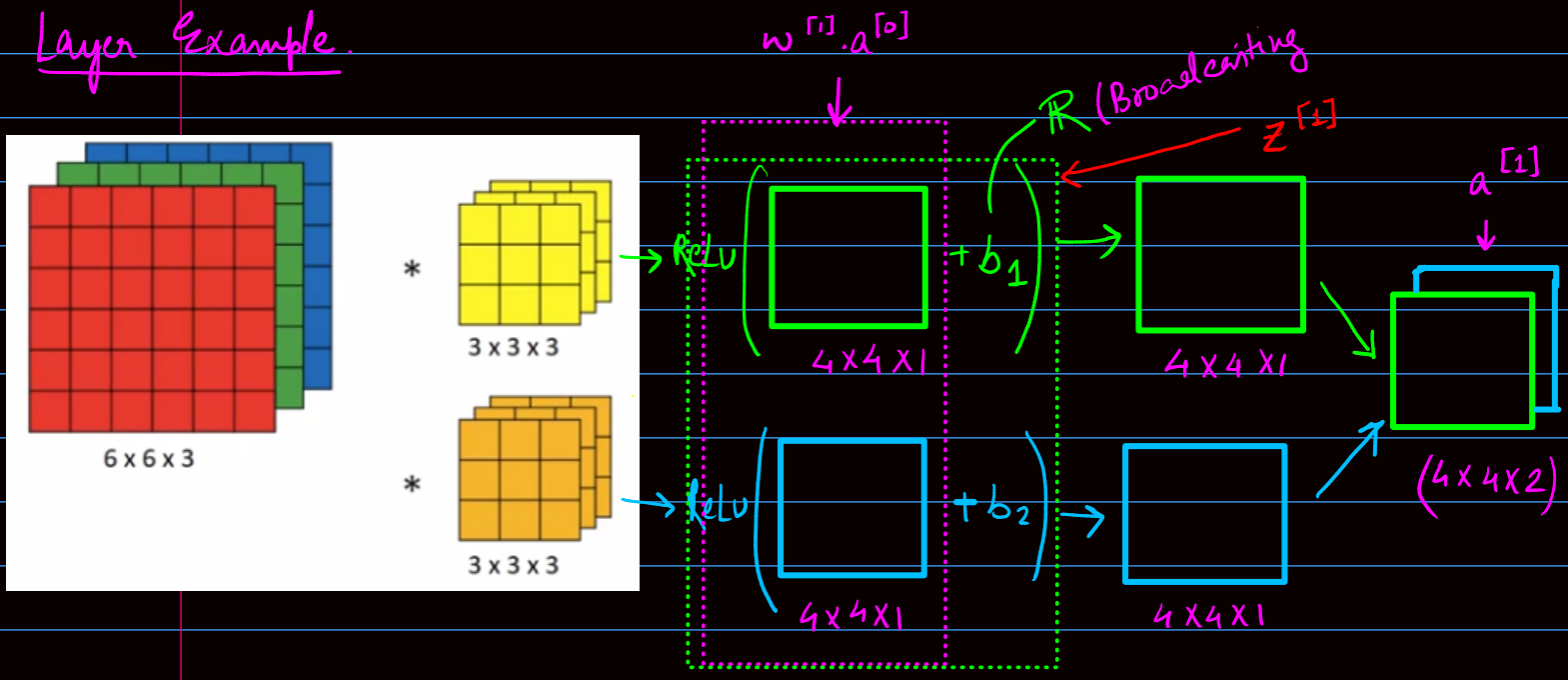
$6 \times 6 \times 3$

$3 \times 3 \times 3 \times (2)$

$4 \times 4 \times 2$




## Layer Example.




Q If you have 10 filter that are 3x3x3 in one layer of a neural network how many parameters does that layer have?

Ans: $10 \times 3 \times 3 \times 3 + 10$ (bias) $= 280$ parameters

* Notation
  1. $f^{[l]}$ = filter size
  2. $p^{[l]}$ = padding

③ $s^{[l]}$ = Stride

④ Input $\rightarrow$ $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

⑤ Output $\rightarrow$ $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

for H $\quad n^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$

⑥ $n_c^{[l]} \rightarrow$ no. of filters

⑦ Each filter is $= f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

⑧ Activation : $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

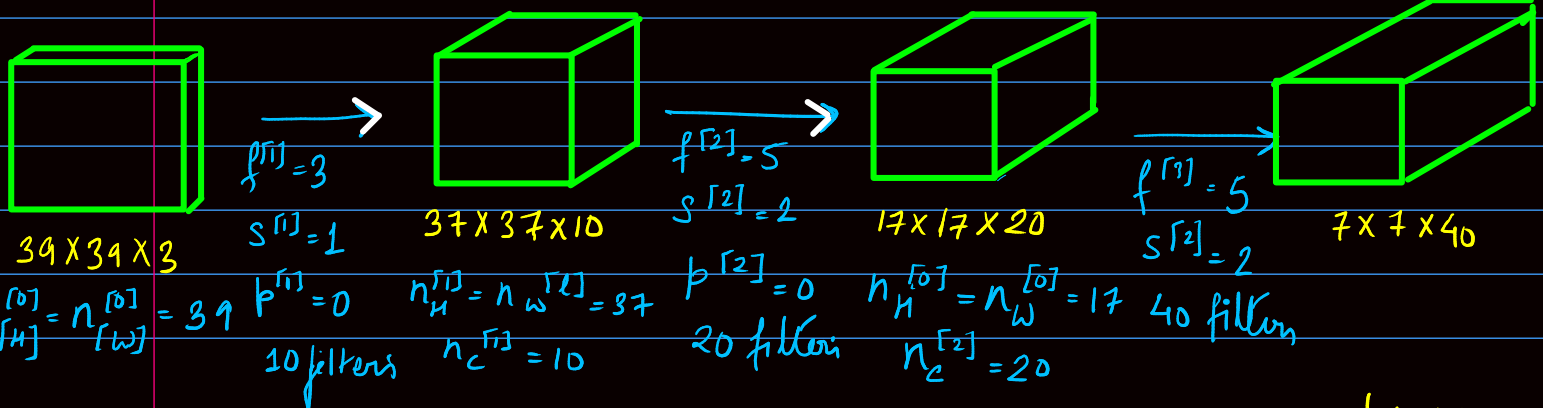
$A = m \times n_H^{[l]} \times n_H^{[w]} \times n_c^{[l]}$

⑨ Weights : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

↳ total no. of filters

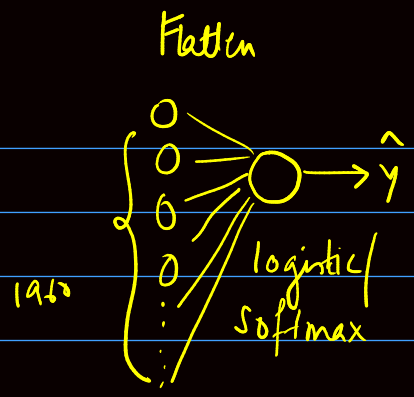
⑩ Bias : $n_c^{[l]} \rightarrow (1, 1, 1, n_c^{[l]})$

* Simple Convolutional Network example :




$\left\{ \frac{n + 2p - f}{s} + 1 \right\} = 37 \qquad \frac{37 - 5}{2} + 1$

7 X 7 X 40 = 1960

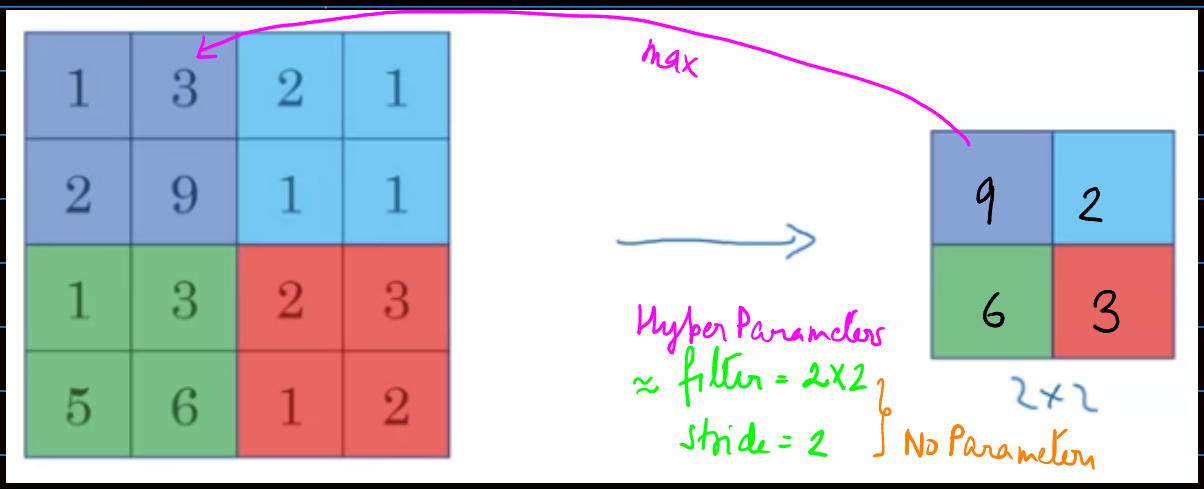


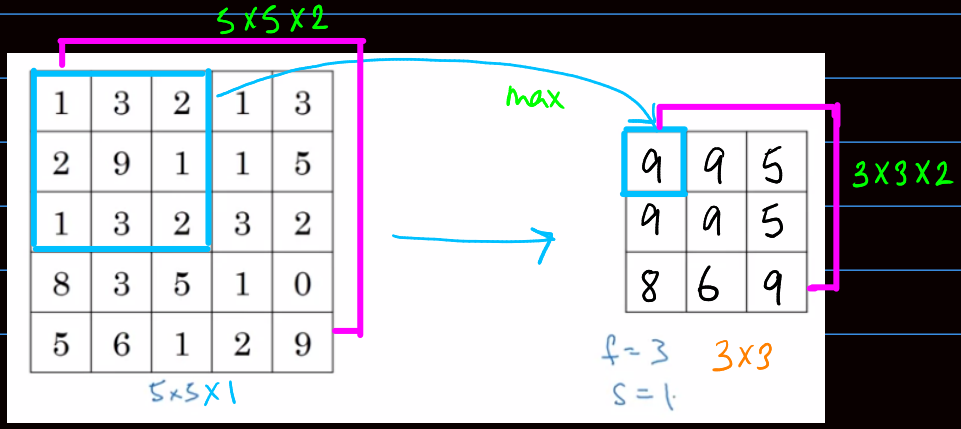

## Types of Convolutional Network

1. Convolution (CONV)
2. Pooling (POOL)
3. Fully Connected (FC)

### POOLING LAYERS

1. Max Pooling

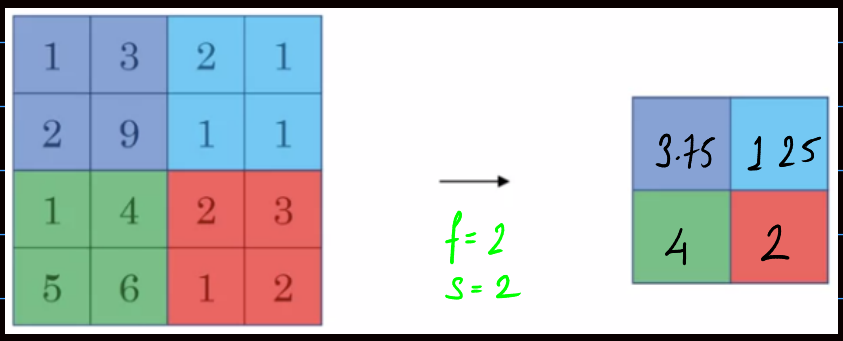



$f = 3, s = 1$

\# Max Pooling in done independently on each of the channels i.e $5 \times 5 \times n_c \rightarrow 3 \times 3 \times n_c$

## ② Average Pooling



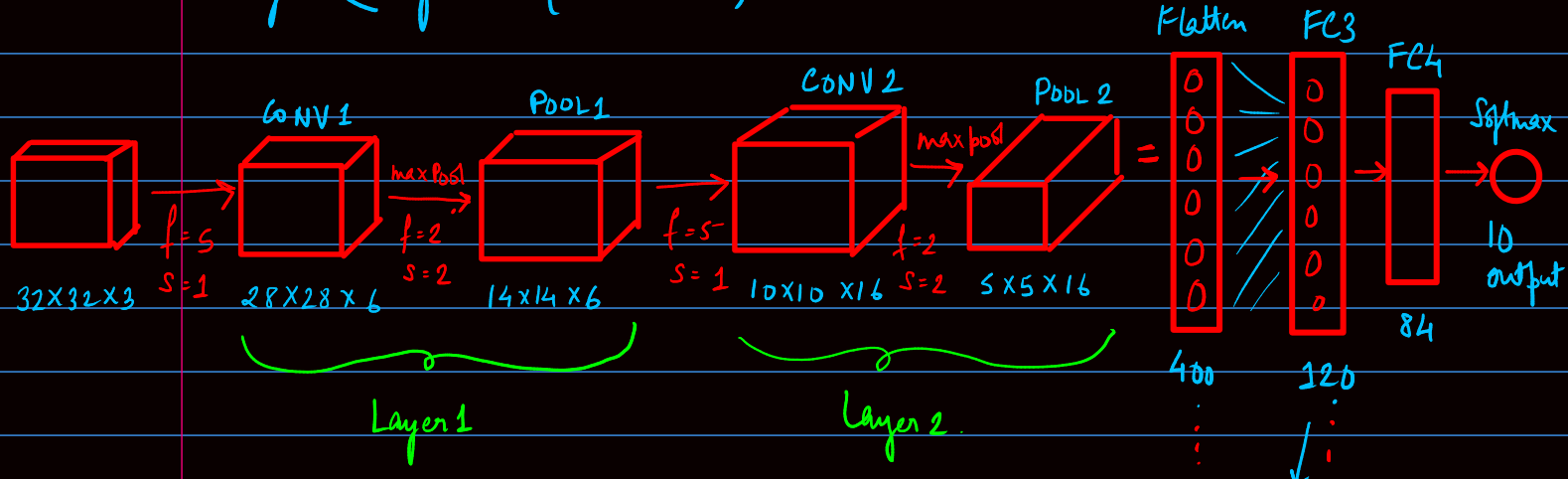

* Max Pooling is most commonly used.

Hyperparameters Used: f = filter size. { common f=2, s=2 / f=3, s=1
s = stride
Max or Average pooling

* No Learnable Parameters in Pooling Layers

## CNN Example

* Hand Digit Recognition (LeNet-5)




$W^{[3]} = (120, 400)$

* As we get deeper $n_H$ & $n_W$ ↓ & $n_C$ ↑
* CONV — POOL — CONV — POOL — FC — FC — SOFTMAM { Common Pattern

## On the basis of the LeNet :

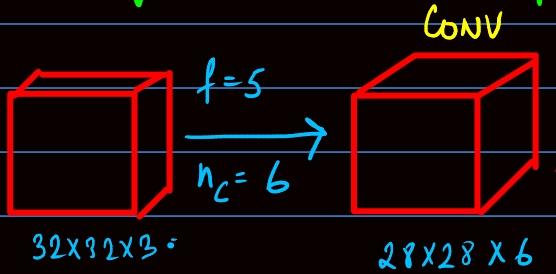
| | Activation shape | Activation Size | # parameters |
|---|---|---|---|
| Input: | (32,32,3) | 3,072 $a^{[0]}$ | 0 |
| CONV1 (f=5, s=1) | (28,28,8) | 6,272 | 608 |
| POOL1 | (14,14,8) | 1,568 | 0 |
| CONV2 (f=5, s=1) | (10,10,16) | 1,600 | 3216 |
| POOL2 | (5,5,16) | 400 | 0 |
| FC3 | (120,1) | 120 | 48120 |
| FC4 | (84,1) | 84 | 10164 |
| Softmax | (10,1) | 10 | 850 |

Inference : ① Pooling layers have no parameters

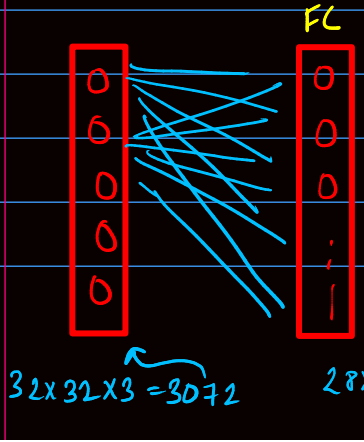
② CONV layers have very less parameters as compared to FC layers

③ Activation size goes down gradually
( If it drops very quickly it is not a great performance

## Why convolutions are useful ?



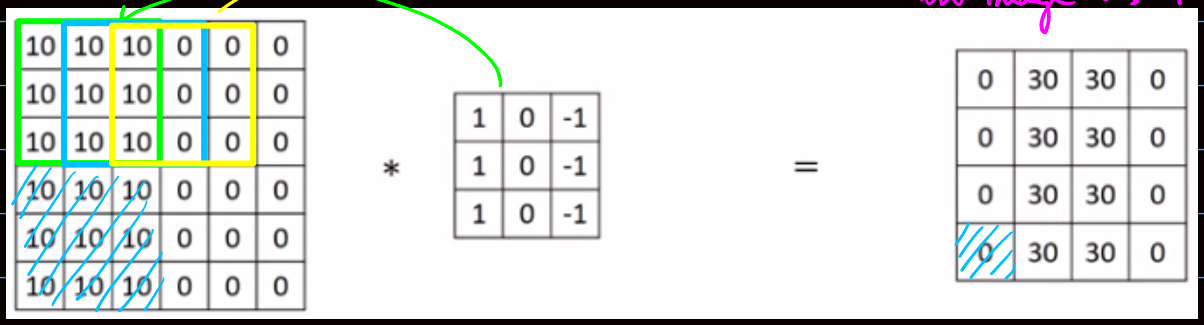

Parameters = 5X5X6 + 1 = 156 // Very small




Parameters = 3072 X 4074
= 14 Million in one layer and a low pixel image

∴ Convolutional layers has much lesser parameters

Reason: i.e same filter is used for all the time all around the image ∴ 9 parameters are shared

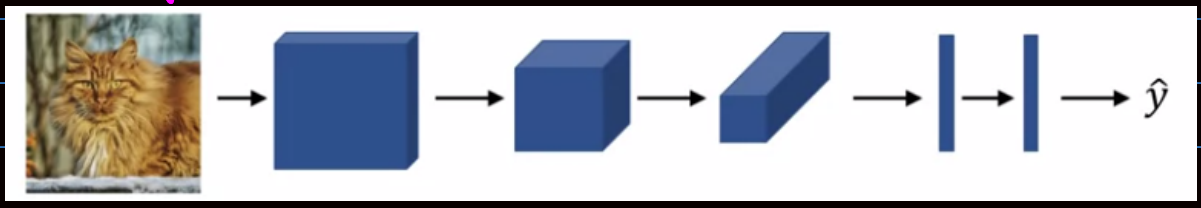



Parameter Sharing: A feature detector (such as vertical edge detector) that's useful in one part of the image is probably useful in another part of the image

Sparsity of Connections: In each layer, each output value depends on small no. of i/p's

Translation Invariance is captured very well in CNN.

Training Set: $(x^{(1)}, y^{(1)}) \ldots (x^{(m)}, y^{(m)})$

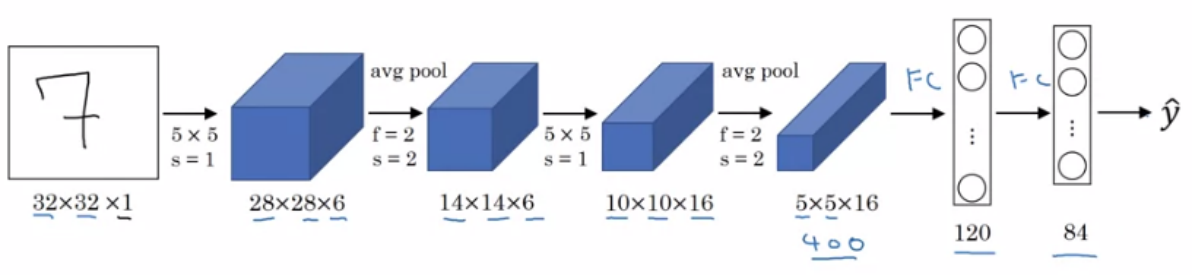



$$\text{Cost } J = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

* Use gradient descent to optimize J.

# Week2

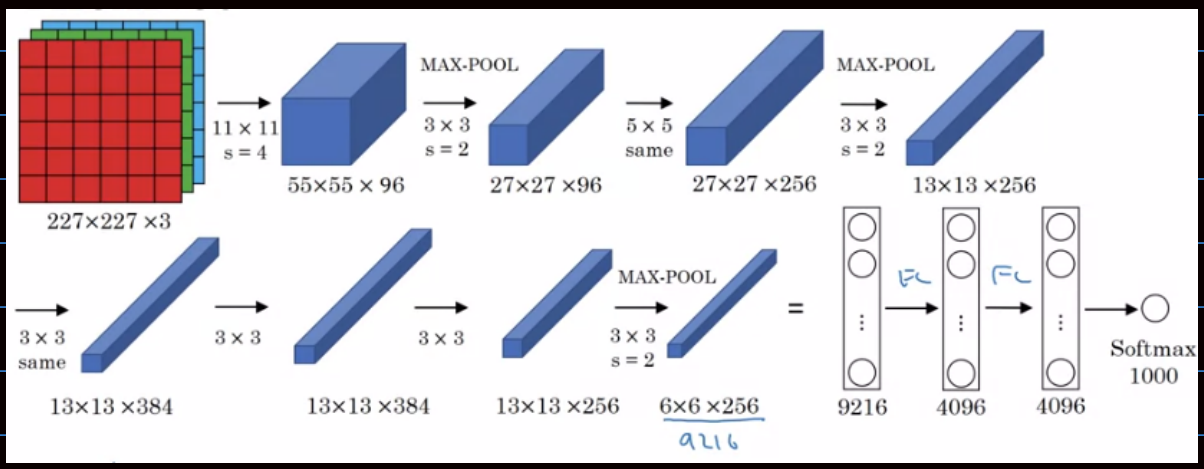
## ① LeNet-5




* 60K Parameters
* $n_H, n_W \downarrow$
* CONV - POOL — CONV — POOL — FC — FC — OUTPUT

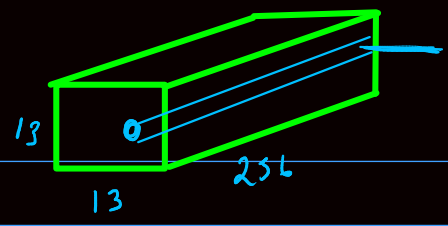
{ Original Paper → used sigmoid / tanh
→ used non-linearity after pooling }

## ② ALEX Net.




* Around 60 M Parameters
* Used multiple GPU's
* ReLU
* Used Local Response Normalization Layer (LRN)
↑ * Not used in practise

## LRN

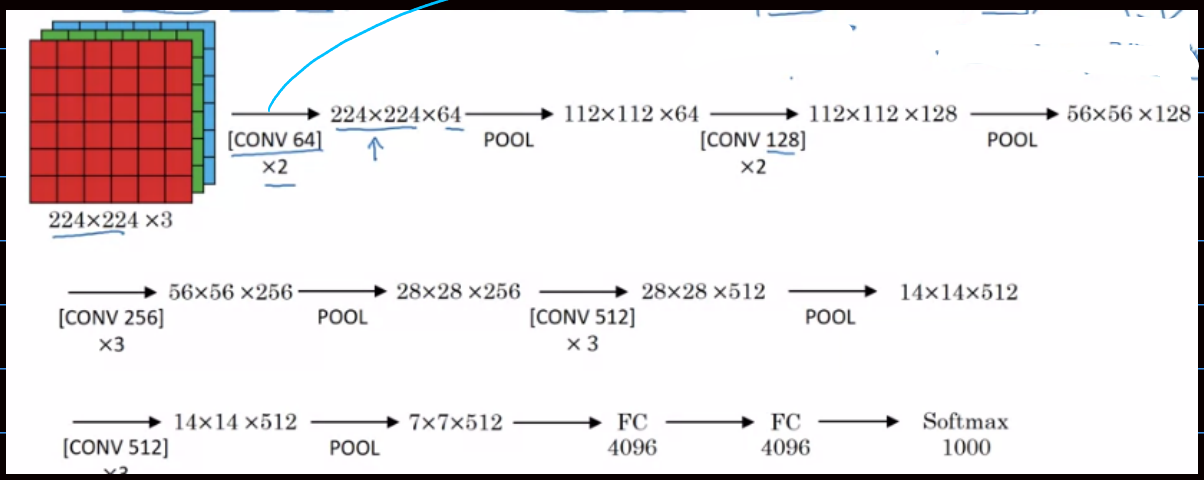



Normalize a position in height & weight face & normalize accross the depth

## ③ VGG-16 (16 layers

* CONV = 3×3 filter, $s=1$, same.
  MAX POOL = 2×2, $s=2$

Means 2 Conv 2D layers with 64 filters




* 138 M Parameters
* $n_H, n_W \downarrow$ but $n_C \uparrow$

## RESNETS

skip connection

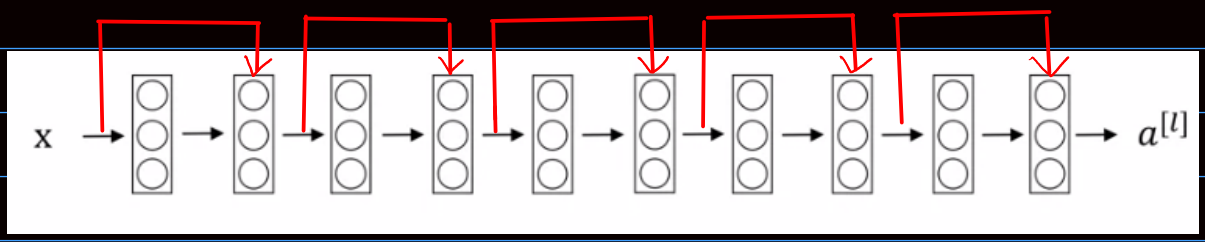
$a^{[l]} \rightarrow$ Linear $\rightarrow$ ReLU $\rightarrow a^{[l+1]} \rightarrow$ Linear $\rightarrow$ (+) $\rightarrow$ ReLU $\rightarrow a^{[l+2]}$

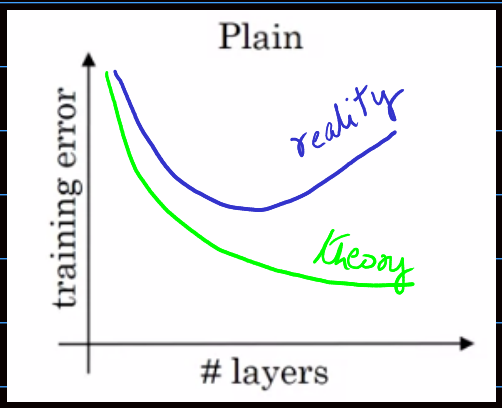
$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]} \rightarrow a^{[l+1]} = g(z^{[l+1]}) \rightarrow z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

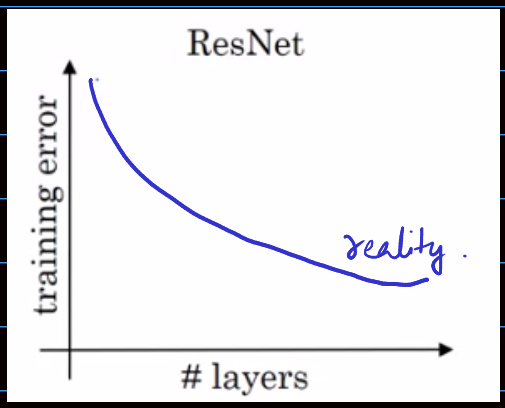
$$a^{[l+2]} = g(z^{[l+2]})$$

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

* It is useful when we build very very deep network.




Why RESNETs works so well?

$X \rightarrow$ Big NN $\rightarrow a^{[l]}$

$X \rightarrow$ Big NN $\rightarrow a^{[l]} \rightarrow a^{[l+2]}$

ReLU

$$a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$$

$$= g\left(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]}\right) = g\left(a^{[l]}\right)$$

"after some wt. decay"

$$= a^{[l]}$$

If $w^{[l+2]} = 0 = b^{[l+2]}$

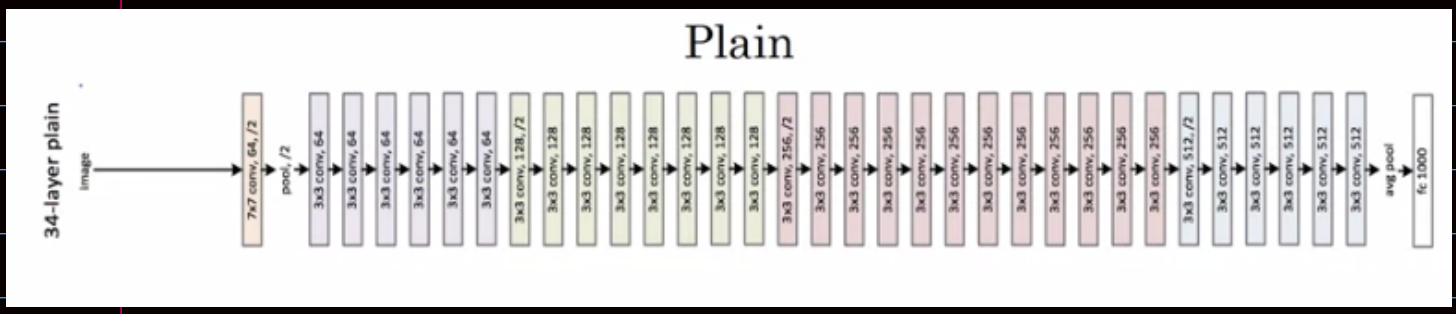
Identity fun$^n$ $\longrightarrow$ easy for R. Block to learn

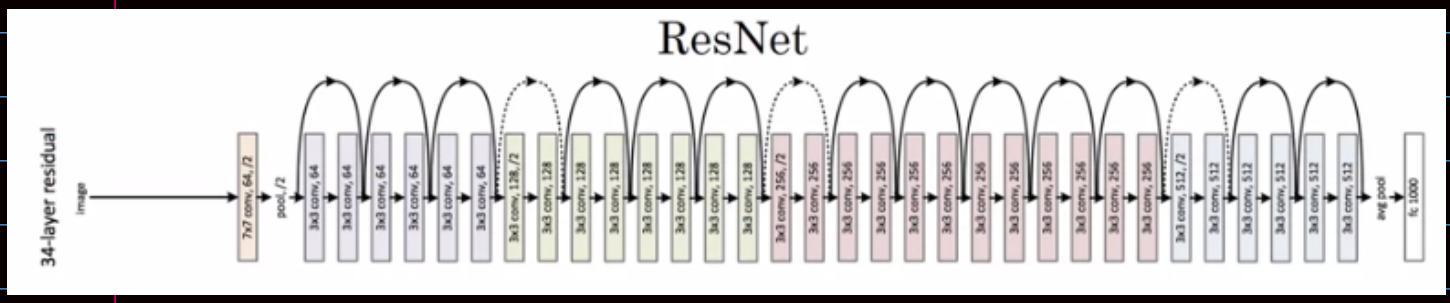
* Adding the 2 extra FC layers doesn't hurt the network at all
* Here dimensions of $z^{[l+2]}$ and $a^{[l]}$ may differ. Thus in all of Resnet we prefer using same padding

or

$$a^{[l+2]} = g\left(z^{[l+2]} + W_s a^{[l]}\right), \quad W_s \in \mathbb{R}^{256 \times 128}$$

256 (rows of $W_s$) × 128 (columns of $W_s$)



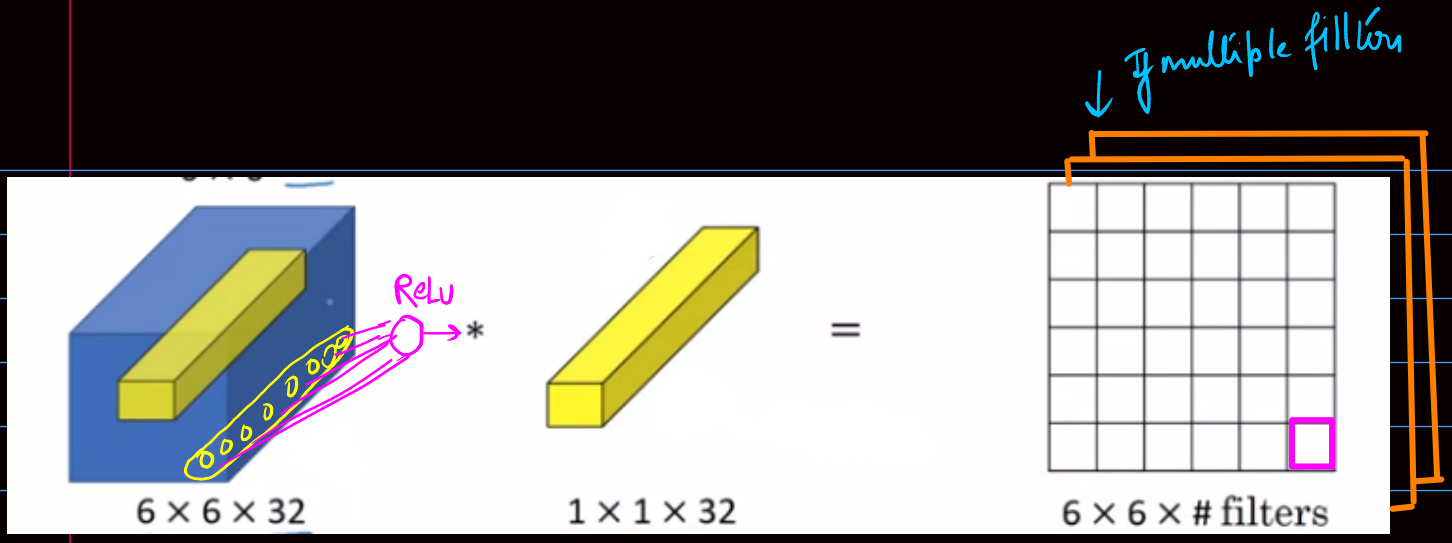

## Network in Networks & 1X1 Convolutions.

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

$6 \times 6 \times 1$

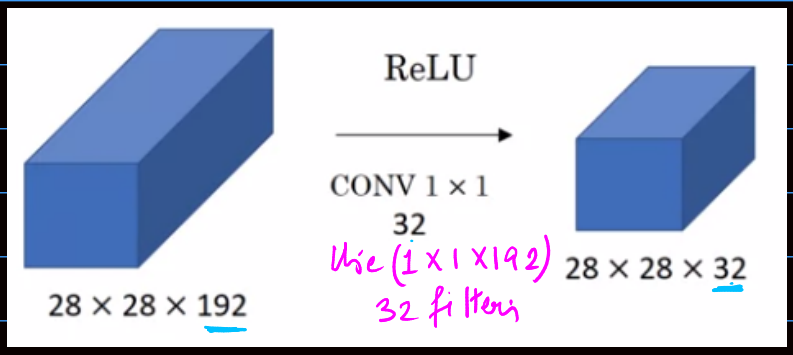
$*$ $\boxed{2}$ $=$

| | | | | | |
|---|---|---|---|---|---|
| 2 | 4 | 6 | 12 | . | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |



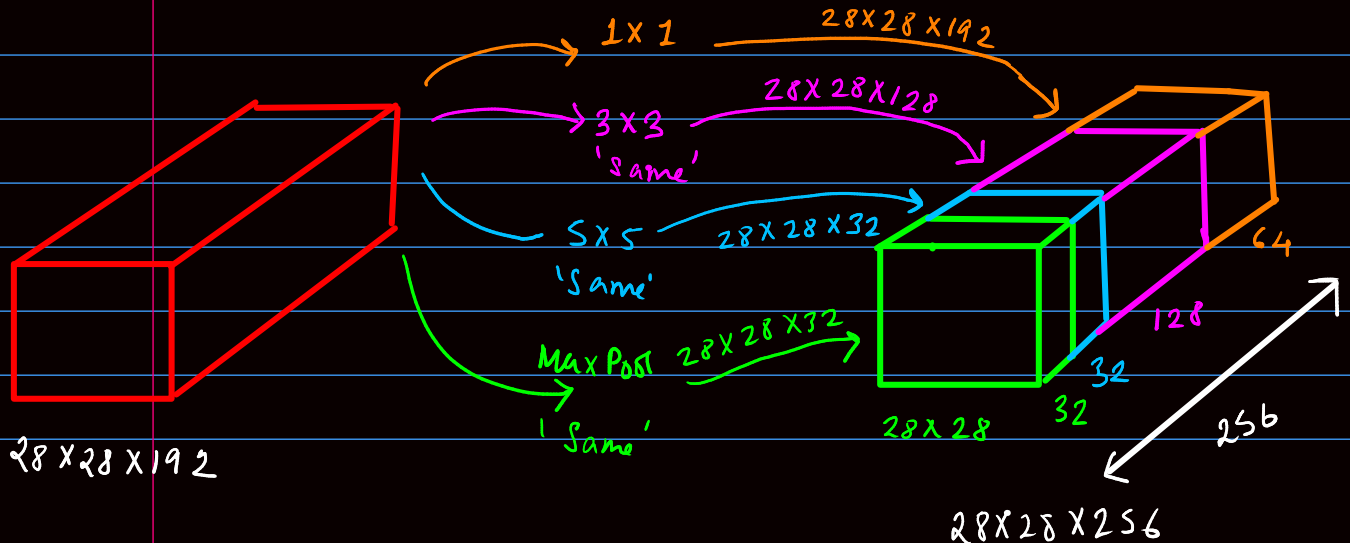

This is called 1X1 convolution or Network in Networks

Example :



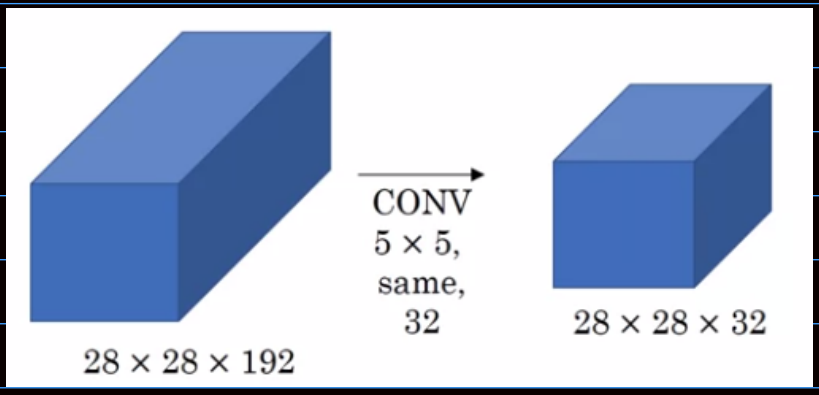

* Helps to shrink the filter size effectively & also saves on computation
* It also adds non-linearity (ReLU) to the n/w

Inception Network



Here instead of picking one filter sizes or pool we want we can pick them all & concatenate the outputs & let the network learn what parameters it wants to use or which filter sizes it wants

* Problem with computational cost

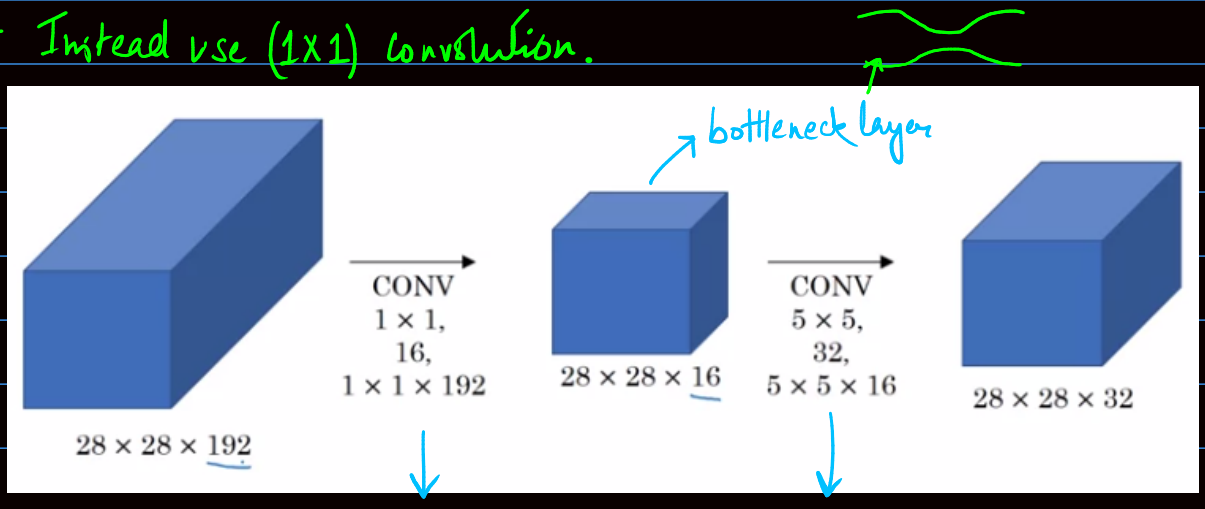



Parameters : 5X5X192X32

No. of Multiplications = 28X28X32X5X5X192 = 120M

Very Expensive Operation

* Instead use (1X1) convolution.

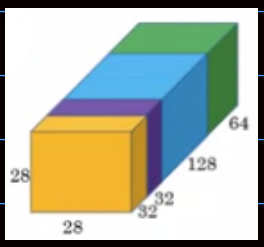



28X28X16X192 = 2.4M

28X28X32X5X5X16 = 10.0M

∴ 2.4M + 10M = 12.4M << 120M

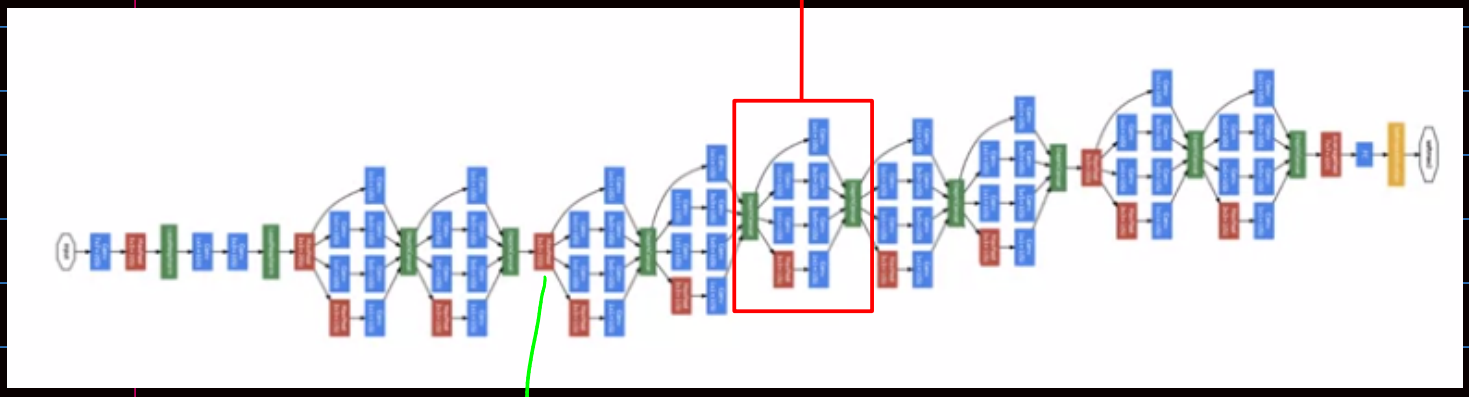
* Shrinking down the layers doesn't hurt the performance of the model but saves a lot of computation, until & unless we are use bottleneck layers





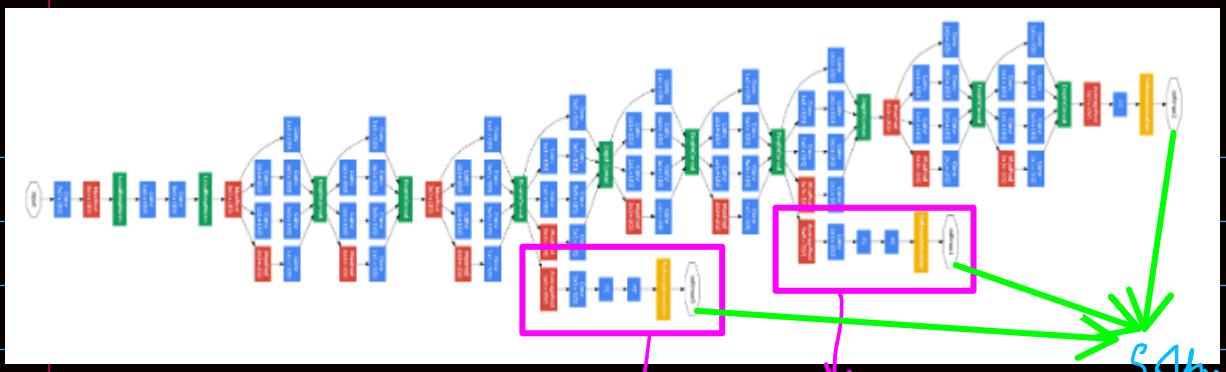


* Inception Module as a whole.



Max. Pooling Layers

Inception Network is a lot of these blocks



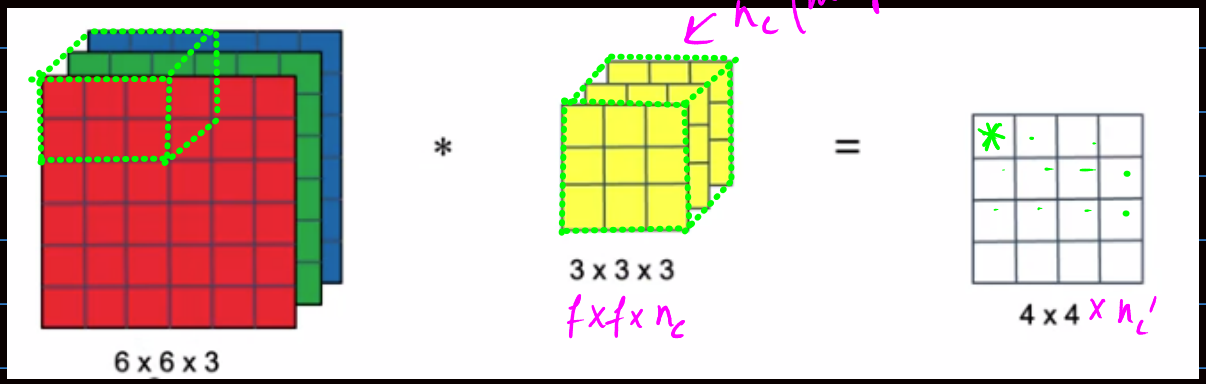

In the original paper we have these side blocks which are classifiers (consisting of a few FC layers & a softmax layer) to predict the output at different stages of the network.

* Also called GoogleNet

## MobileNet

* Low computational cost at deployment.
* Useful for mobile & embedded vision application.
* Normal vs Depthwise Seperable Convolutions.

### Normal Convolution

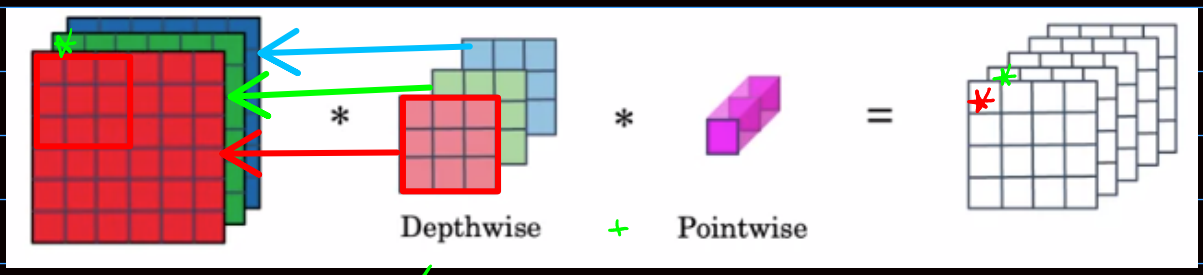



Computational Cost

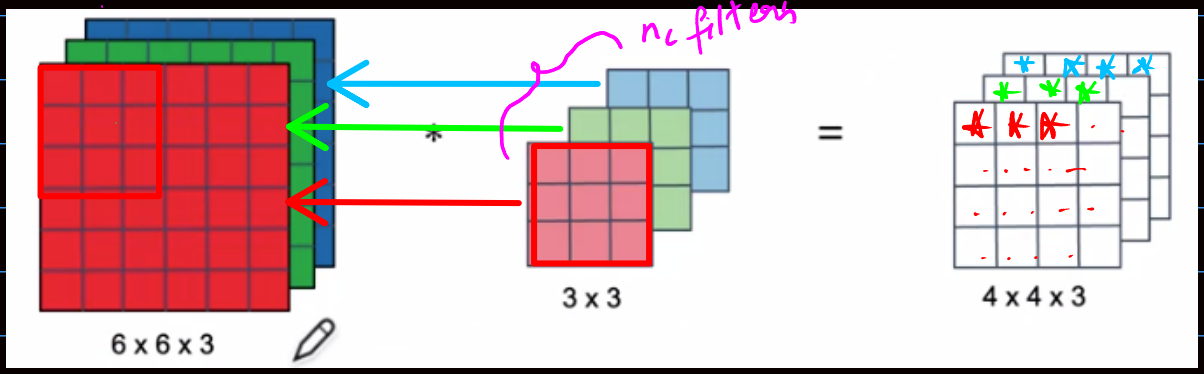
= # filter params x # filter positions x # of filters

(3x3x3) x (4x4) x 5 = 2160

# Depthwise Seperable Convolution




## ① Depthwise Convolution



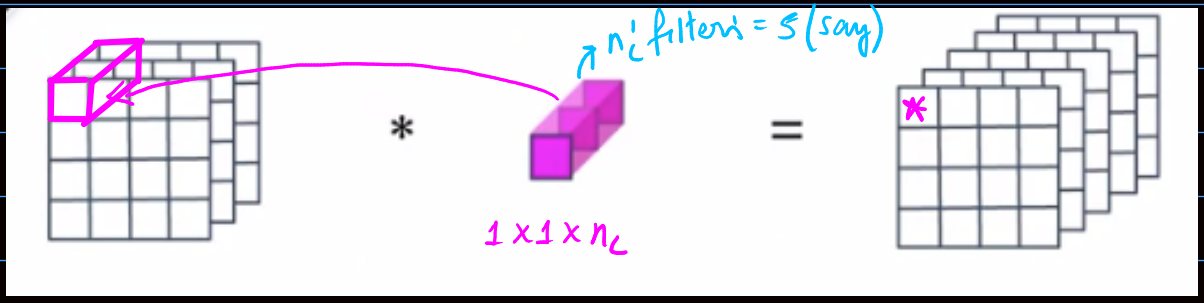

$n \times n \times n_c$ $\quad$ $f \times f$ $\quad$ $n_{out} \times n_{out} \times n_c$

**Computational Cost**

= #filter params X #filter positions X #of filters

3x3 $\quad$ 4x4 $\quad$ 3 $\quad$ = 432 — ①

## ② Pointwise Convolution




$n_{out} \times n_{out} \times n_c$ $\qquad$ $n_{out} \times n_{out} \times n_c'$

## Computational Cost

= # filter. params × # filter positions × # of filters

$1 \times 1 \times 3$ × $4 \times 4$ × $5$ = $240$ — (1)

## Cost Summary

Cost of normal convolution = 2160

Cost of depthwise seperable convolution = 432 + 240

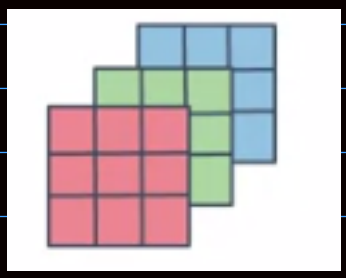
= 670 $\ll$ 2160

Generally,

$$\frac{\text{Cost of normal convolution}}{\text{Cost of depthwise seperable convolution}} = \frac{1}{n_c'} + \frac{1}{f^2}$$

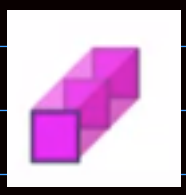
In reality $\approx$ 10 times cheaper

** For notes purpose (Notation)



↑

Denotes Depthwise convolution in general (not necessarily three channels)

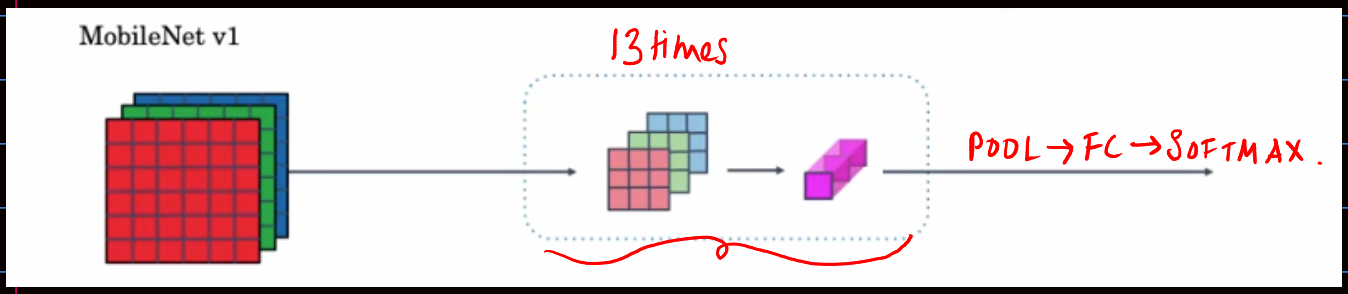


↑

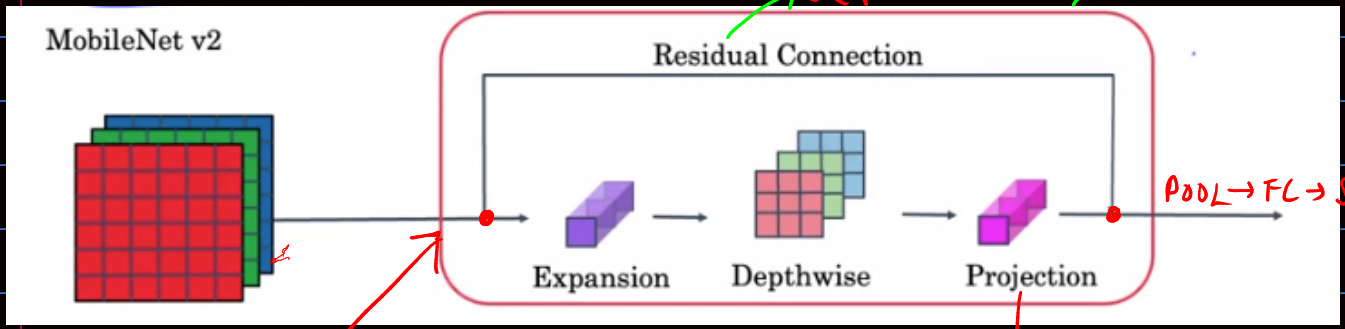
Denotes Pointwise convolution in general (not necessarily 3 channels)

## MobileNet Architecture.

(1)

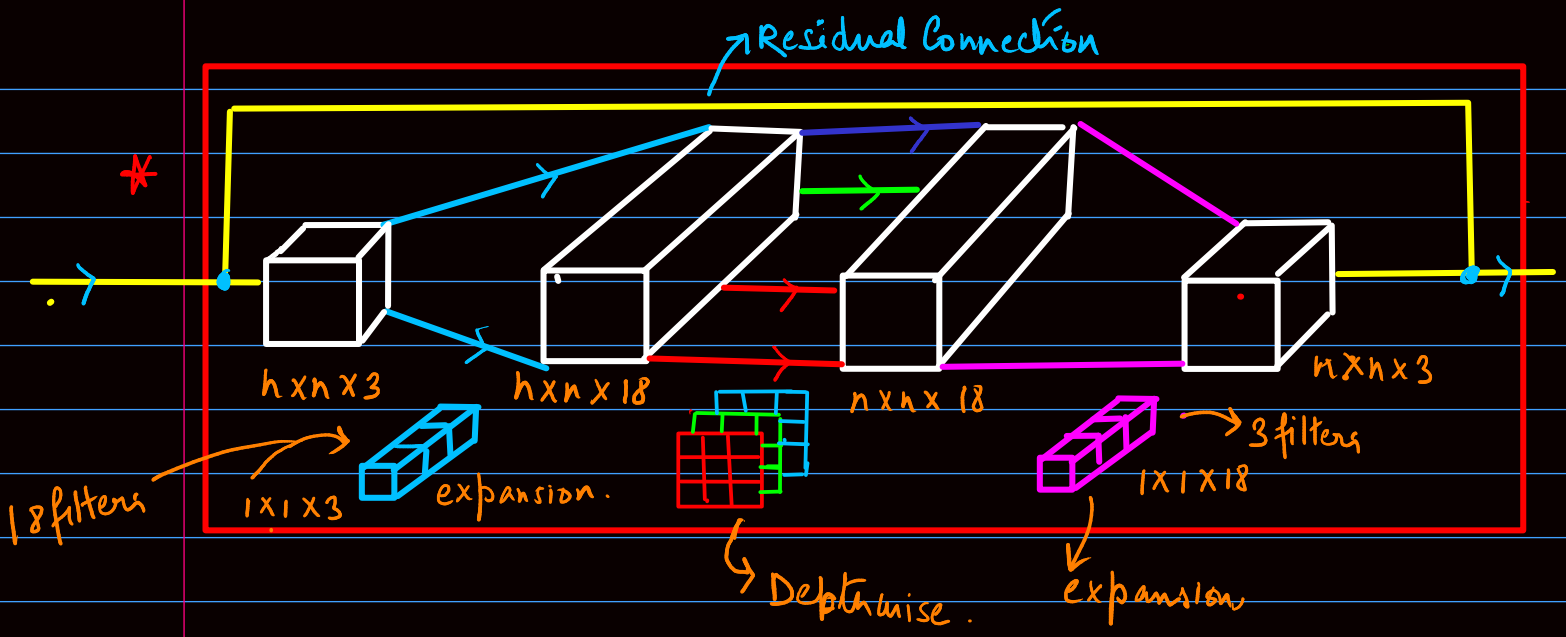


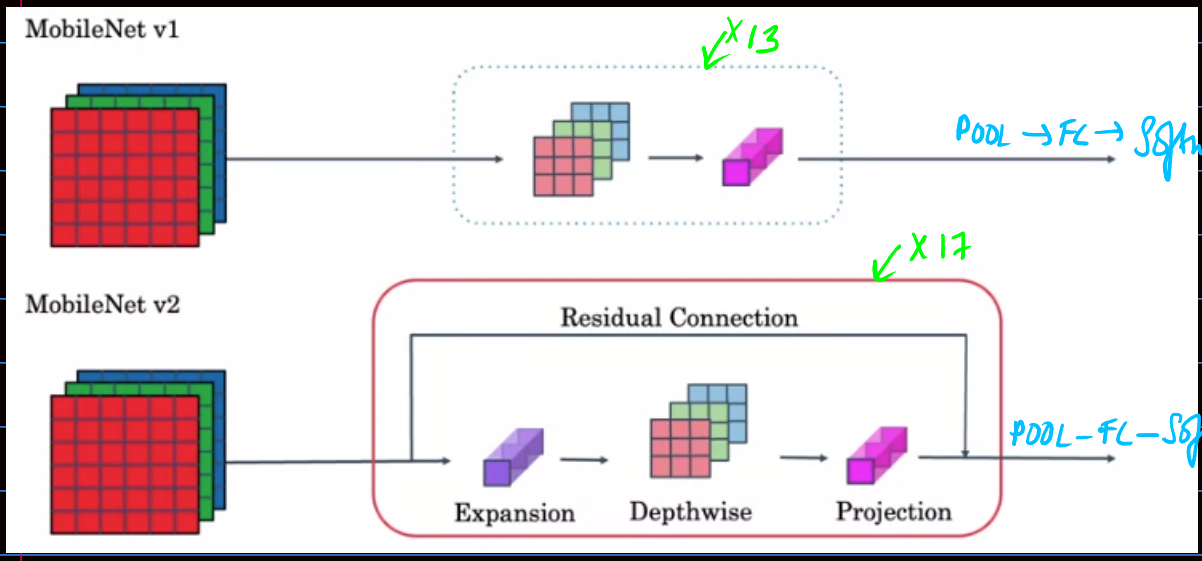

(2)



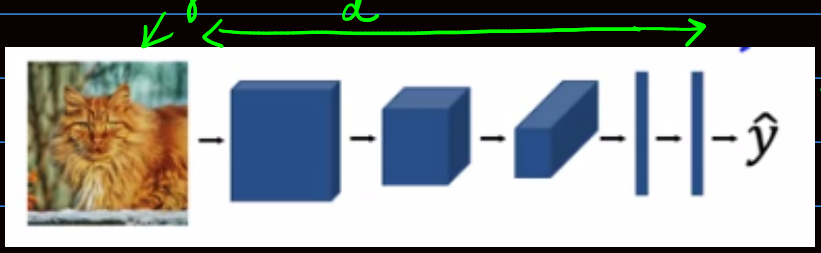

Bottleneck Block.

Pointwise Conv.




## Efficient Net




r = resolution

To scale the components up we can:

① $r\uparrow$ ② $d\uparrow$ ③ $w\uparrow$

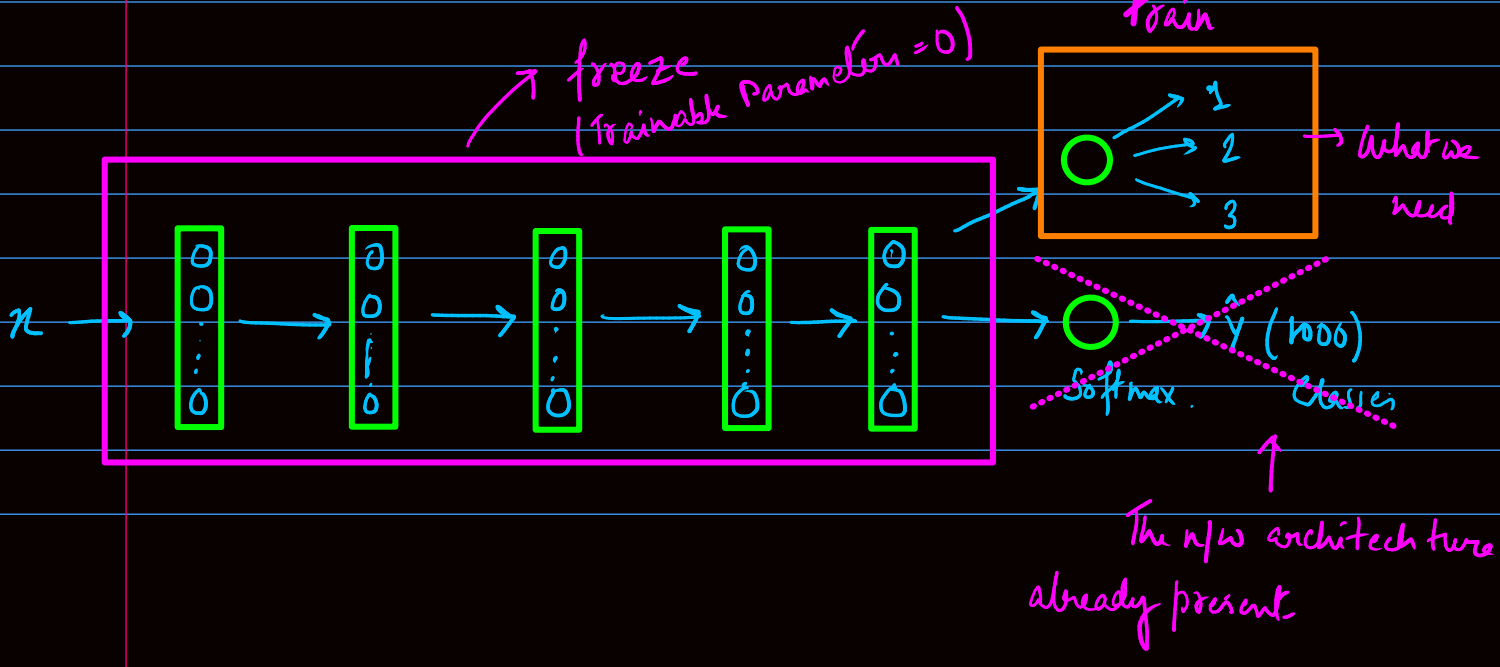
Q What is the best tradeoff betn $r$, $d$ & $w$ to be most computa-tionally efficient?

Ans: With efficient net we can scale up or down depending on resources we are working on.

## Practical Implementation

* Using Pretrained networks for similar tasks saves a lot of time
* Use Transfer learning to compensate for the small datasets (Transfer learning discussed in the previous course)

Eg.

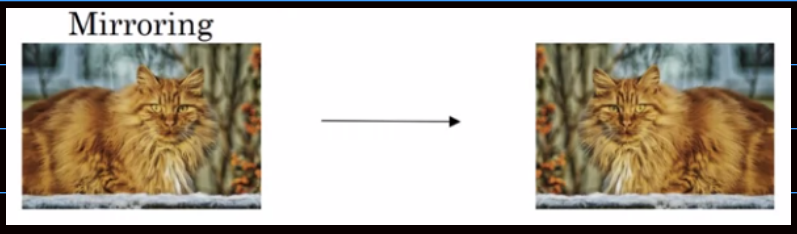


* If we have a slightly significant dataset then we can try freezing fewer layers

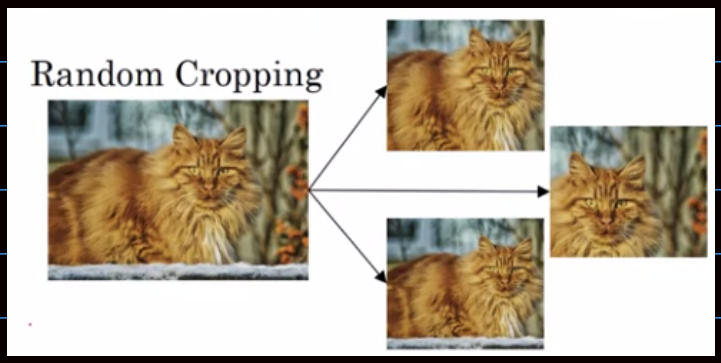
  * dataset size ↑, freezed layer ↓
* We can also download a model & just use the initialization & train the entire model from scratch.

## Data Augmentation

1.

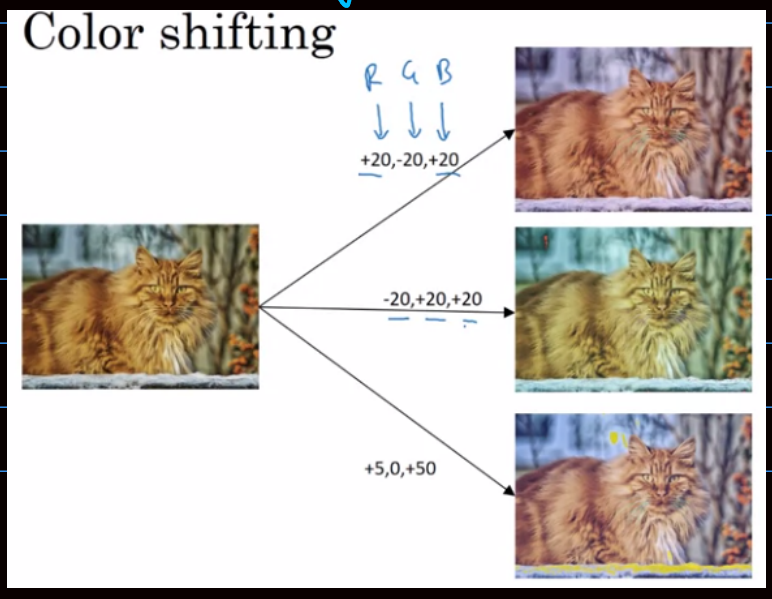



2.

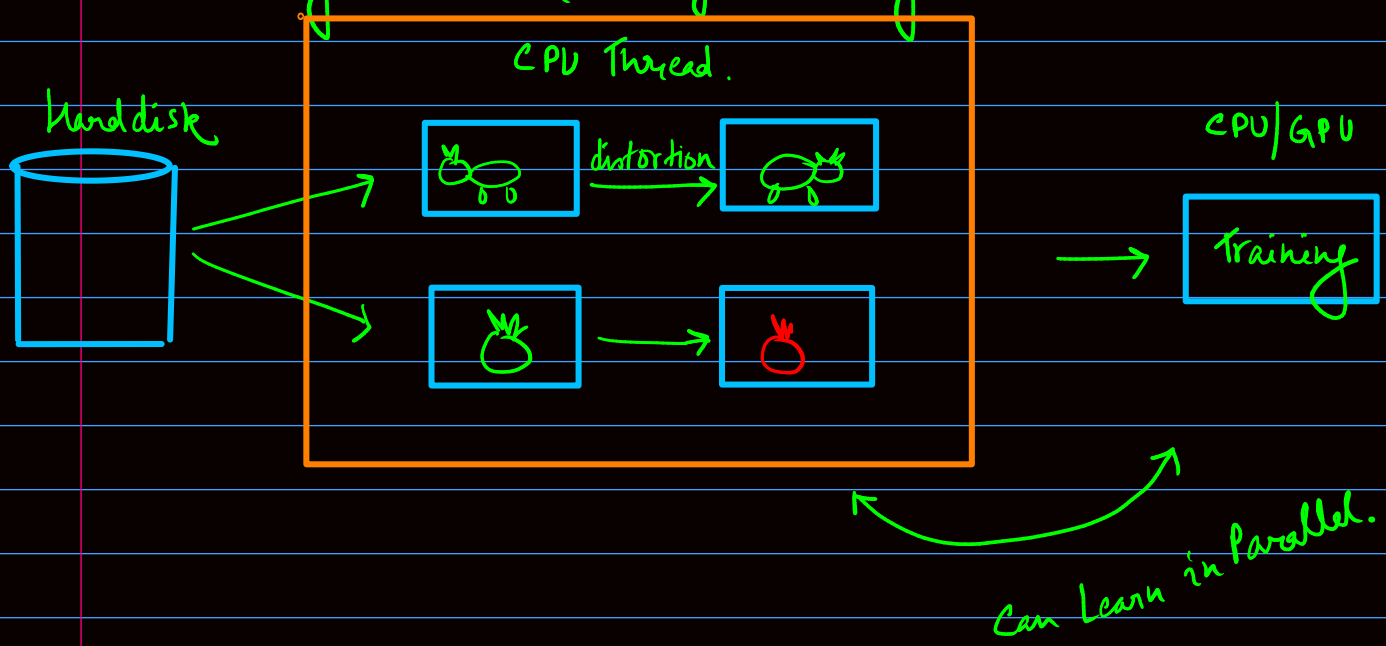



3. Rotation
4. Shearing
5. Local Warping

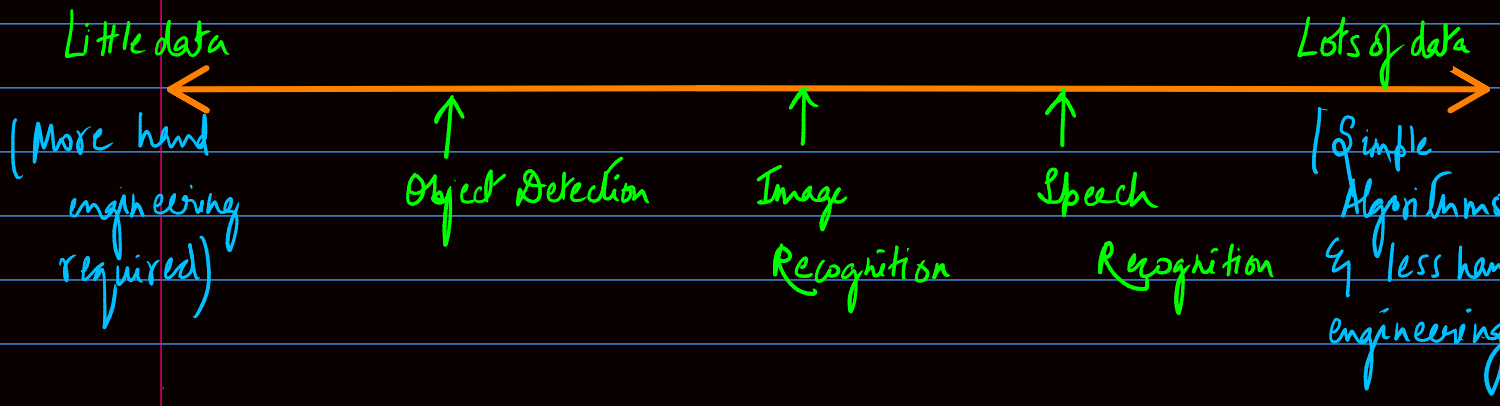
6.



* Implementing Distortions during training




State of Computer Vision




Two sources of knowledge.
- Labelled data.
- Hand engineered features/network architecture & other components

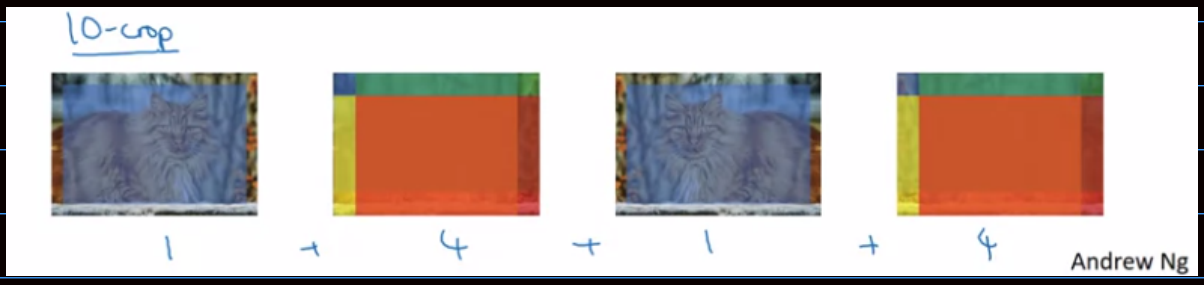
* C.V has more hand engineered data due to the problems of less data readily available.

* But for less data we have transfer learning to rescue

# Tips for doing well on benchmarks / winning competitions

→ Ensembling
- Train several networks independently & average their o/p's. ($\hat{y}$)
  (3-15 n/w's).
  → Not preffered in production.

→ Multi-crop at test time
- Run classifier on multiple versions of test images & average results

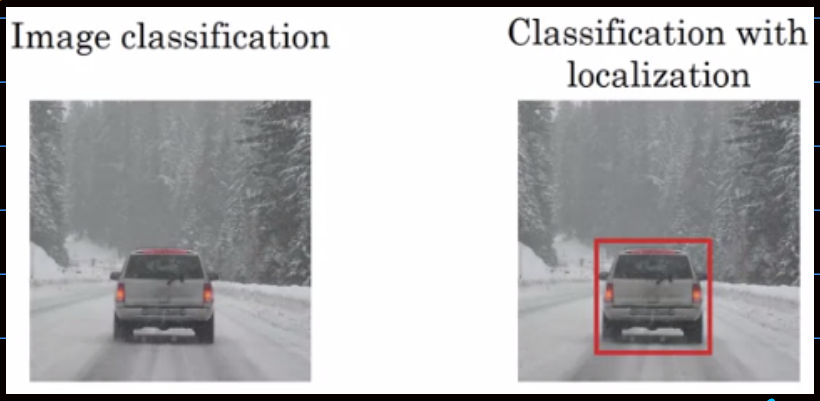



→ Not preffered in production

* Use architecture of networks published in the literature
* Use open source implementation if possible.
* Use pretrained models & finetune on your dataset.

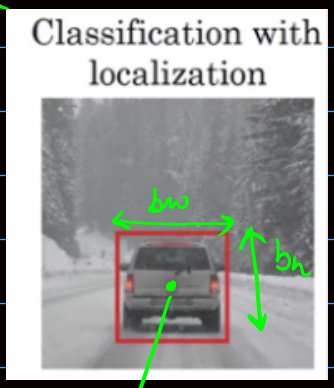
## Week 3:

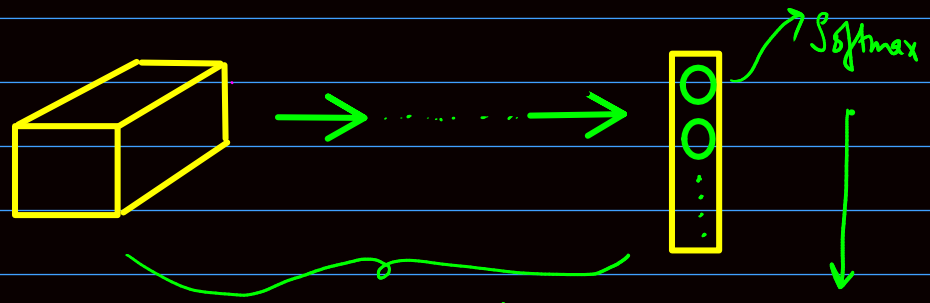
## Object Localization




One central object

multiple object



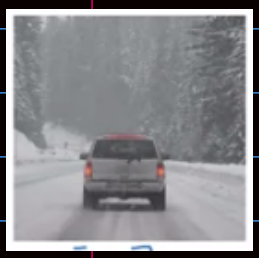

need to output
① 4 classes (say)
② $b_x, b_y, b_h, b_w$

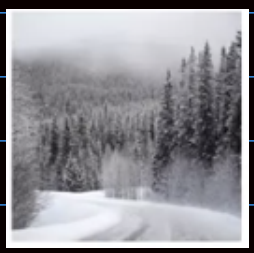
Eg. $b_x = 0.5$, $b_y = 0.7$, $b_h = 0.3$, $b_w = 0.4$

Let the classes be, ① Pedestrian ($c_1$)
② Car ($c_2$)
③ Motorcycle ($c_3$)
④ Background

$$y = \begin{bmatrix} P_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$P_c$ → Is there any object (1/0)





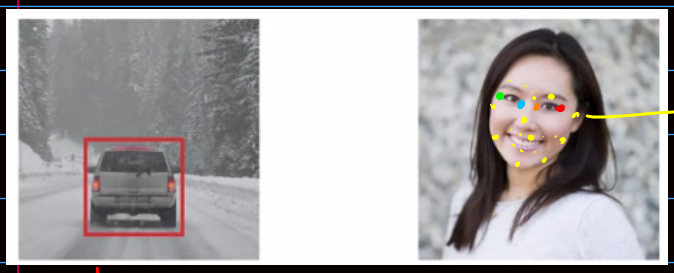
Eg

$$y = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$ don't care

Loss from

$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \cdots + (\hat{y}_8 - y_8)^2 & \text{if } y_1 = 1 \\ (\hat{y}_1 - y_1)^2 & \text{if } y_1 = 0 \end{cases}$$

# Landmark Detection



→ 64 landmarks

$b_x, b_y, b_h, b_w$

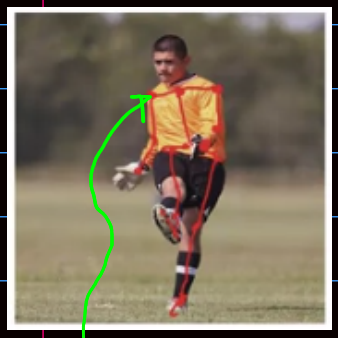
Net

$l_{1x}, l_{1y}$
$l_{2x}, l_{2y}$
$l_{3x}, l_{3y}$
$l_{4x}, l_{4y}$

:) → ConvNet → [0 0 ⋮ 1]

face?
$l_{1x}$
$l_{1y}$
⋮
$l_{64x}$
$l_{64y}$

129 classes



etc.

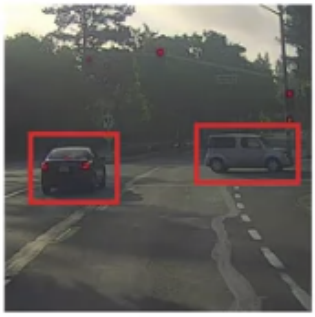
$(l_{1x}, l_{1x}) \cdots (l_{2'x}, l_{2'y})$

# Object Detection

Training set:

| x | y |
|---|---|
|  | 1 |
|  | 1 |
|  | 1 |
|  | 0 |
|  | 0 |

Cropped →



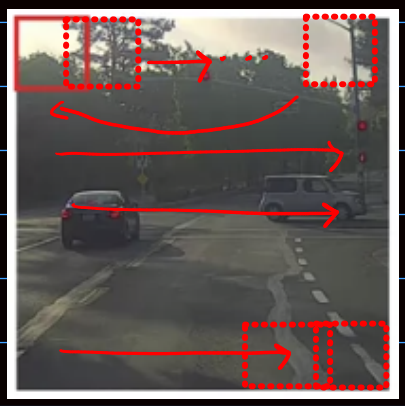
Conv Net →

Testing set

(1/0)

Detect whether car or not

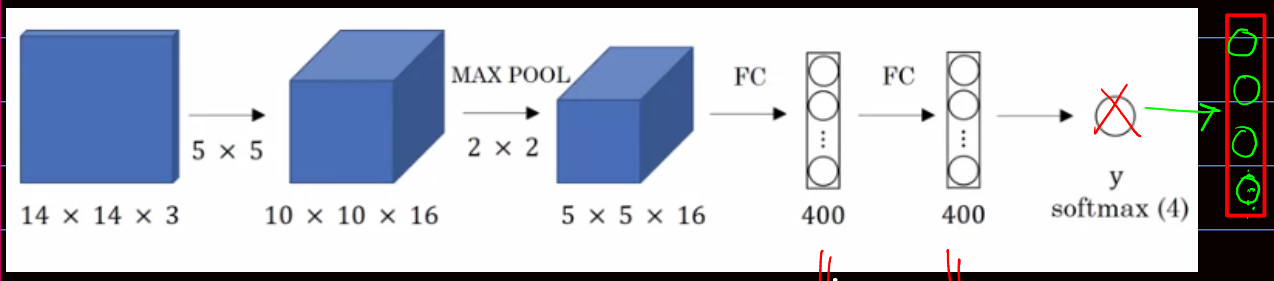
# Sliding Windows Detection

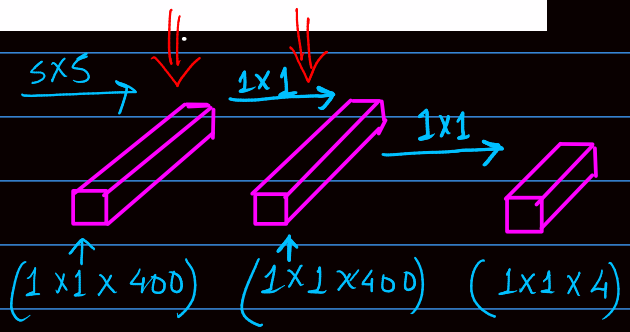


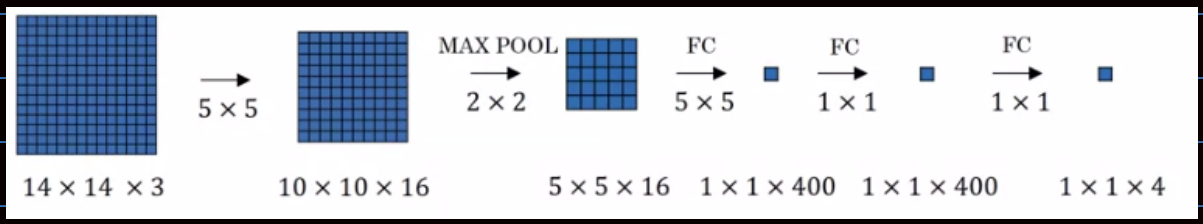
Slide a kernel (windows) of varying sizes & strides all over the image

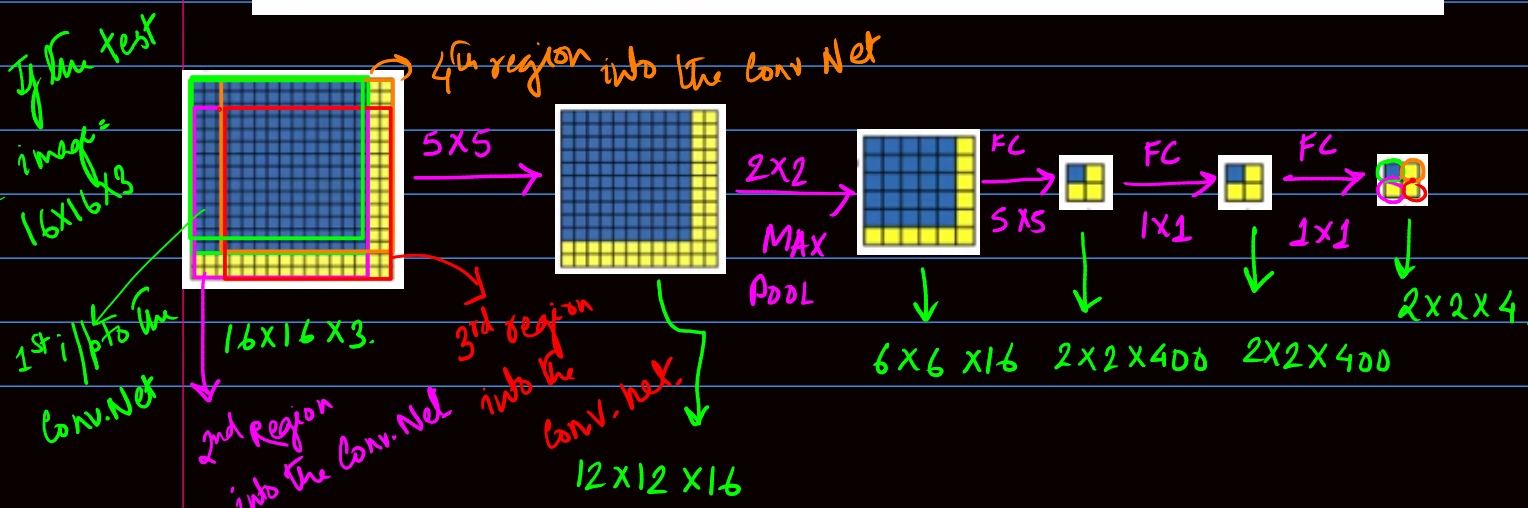
* Disadvantages: Larger computation as we are cropping a lot throughout the image and much slow

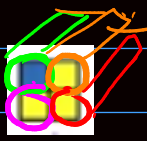
# Convolutional Implementation of Sliding Windows



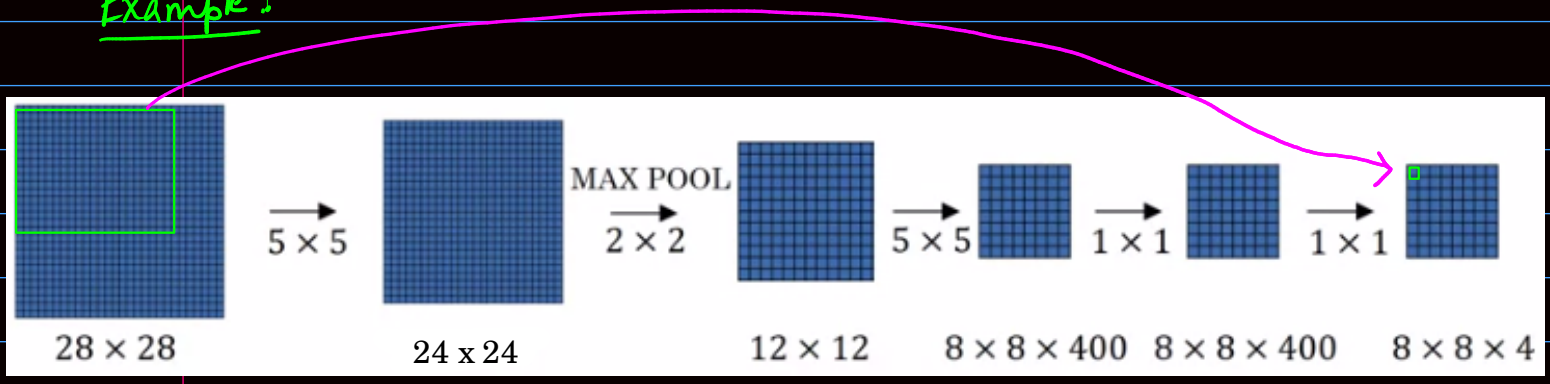



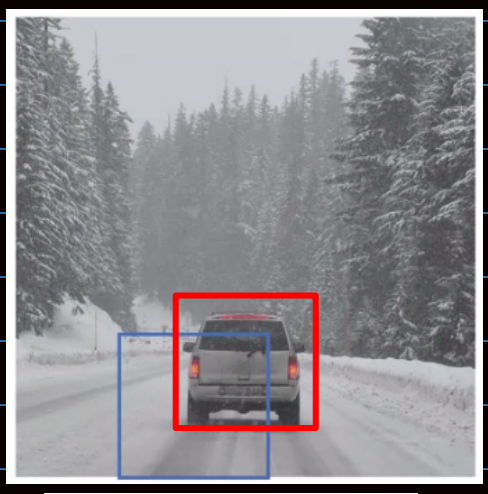
* So individually passing all the four regions into the ConvNet is expensive rather we can pass the entire net & the output volume gives us the representation of all the four nets  Volumes, thus sharing the computations

Example:




∴ Instead of many forward pass for each seperate sliding window we do all the predictions together.

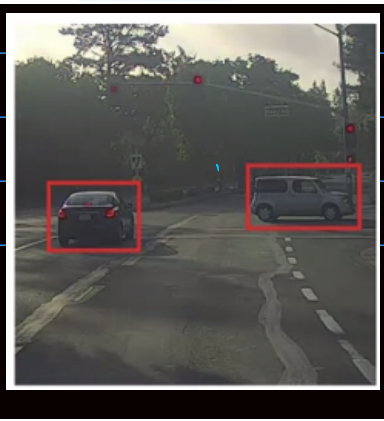
## Bounding Box Predictions



→ None of the bounding box fits the object well.

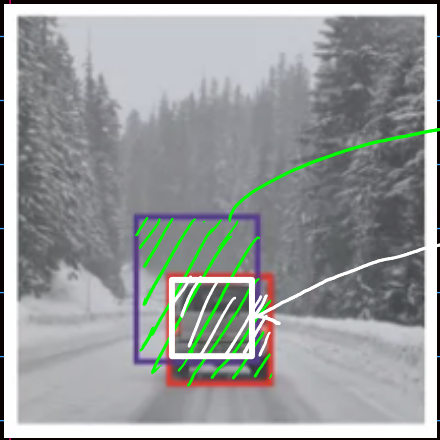
□ → Ground Truth.

* With normal window slides we might not get the bounding box even (ground truth)



* Or in this type of example with different dimensional bounding box, this normal convolutional sliding becomes inefficient.

# Intersection over Union (IOU)



Union

Intersection

$$IOU = \frac{\text{Size of } \boxslash}{\text{Size of } \boxslash}$$

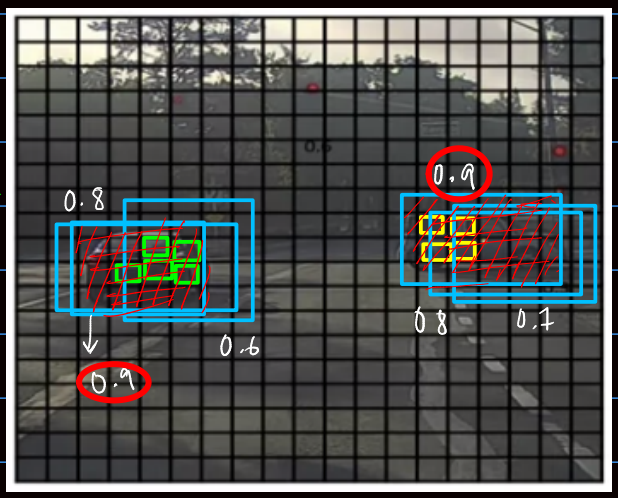
"Correct" if IOU $\geq 0.5$

Higher the IOU more accurate is the bounding box

IOU is the measure of the overlap bet" two bounding boxes.

## Non-max Suppression

It is a way to make sure that the algorithm detects the object only once




→ Multiple detections per car. ($P_c$ → Prob)

↳ only max prob will remain & others will get suppressed.

The max. prob bounding boxes will remain

19X19

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$$ {$c_1, c_2$ are absent since we have only one object}

## Algorithm

1. Describe the output prediction i.e' y'
2. Discard all boxes with $p_c \leq 0.6$
3. While there are any remaining boxes —
   - → Pick the box with largest $p_c$ & output that as prediction
   - → Discard any remaining box with IOU $\geq 0.5$ with the box o/p in the previous step.

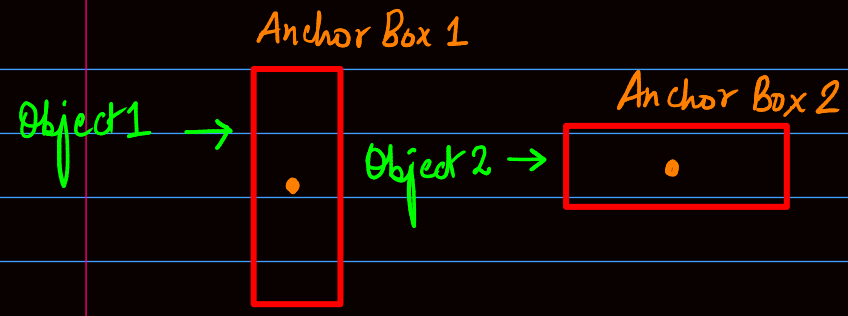
* If more than one class we apply non-max suppression to all the objects seperately.

## Anchor Boxes

$$y_1 = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

(Say)
$c_1 \rightarrow$ Car.
$c_2 \rightarrow$ Person
$c_3 \rightarrow$ Cycle.

→ Same centroid for both the objects (Person & Car)




$$y = \begin{bmatrix} p_c \\ b_x \\ \vdots \\ c_1 \\ p_c \\ b_x \\ \vdots \\ c_1 \\ \vdots \end{bmatrix}$$

anchor box 1

anchor box 2

Each object in training image is assigned to grid cell that contain object midpoint & anchor box for the grid cell with highest IoU.

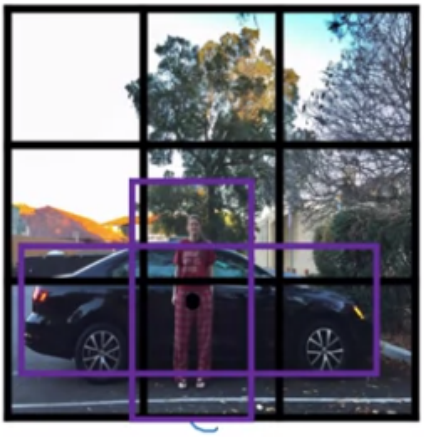
(grid cell, anchor box)

Output box: $y = 3 \times 3 \times 2 \times 8$

2 Anchor, 8D

Box anchor box.

## Anchor box example



Anchor box 1: Anchor box 2:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

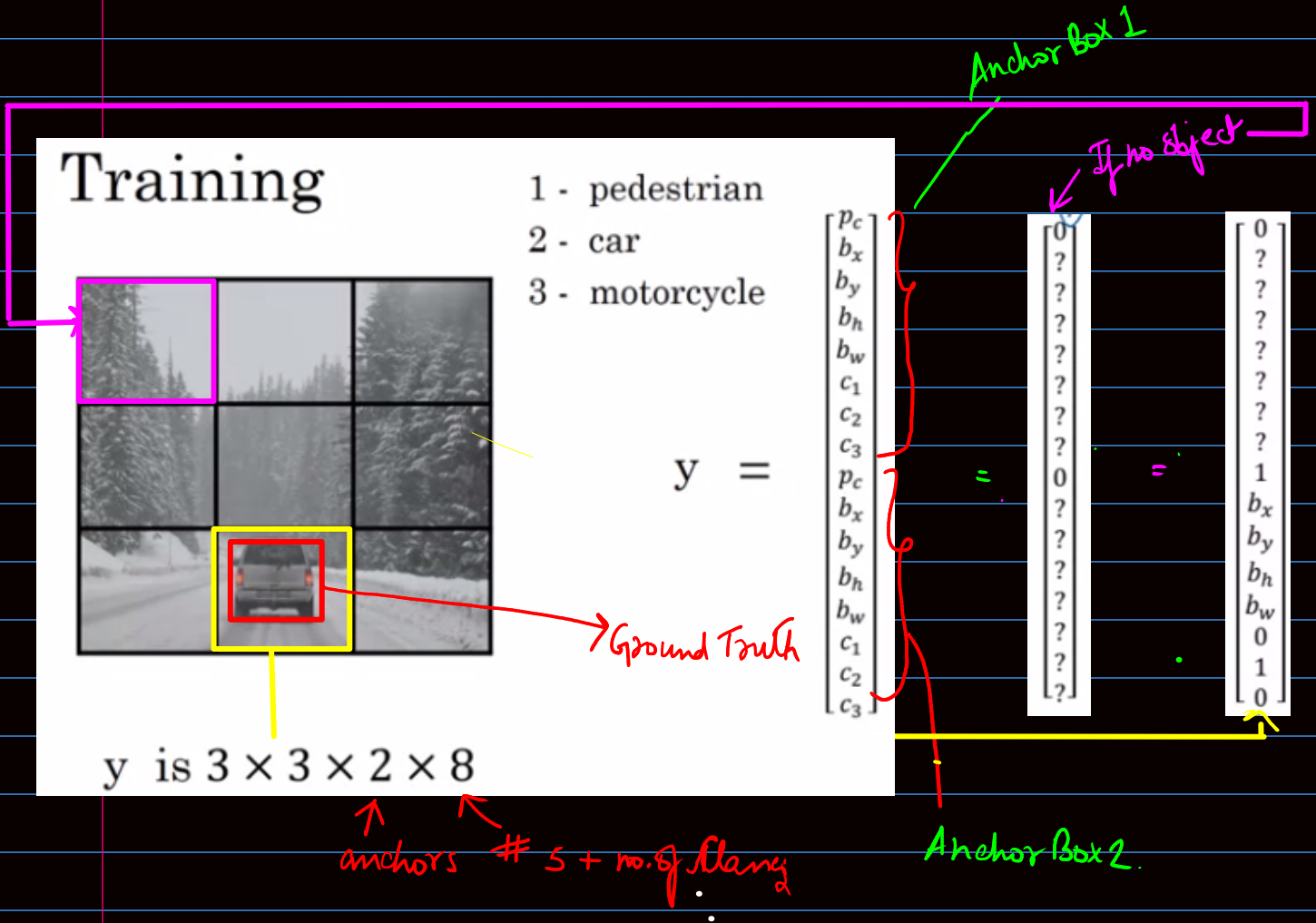
2 Object

Only 1 obj (car)

$$= \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

* Anchor Box Algorithm does not work well when,
  ① 3 Objects & 2 Anchor Boxes
  ② 2 Objects & 1 Anchor Box.

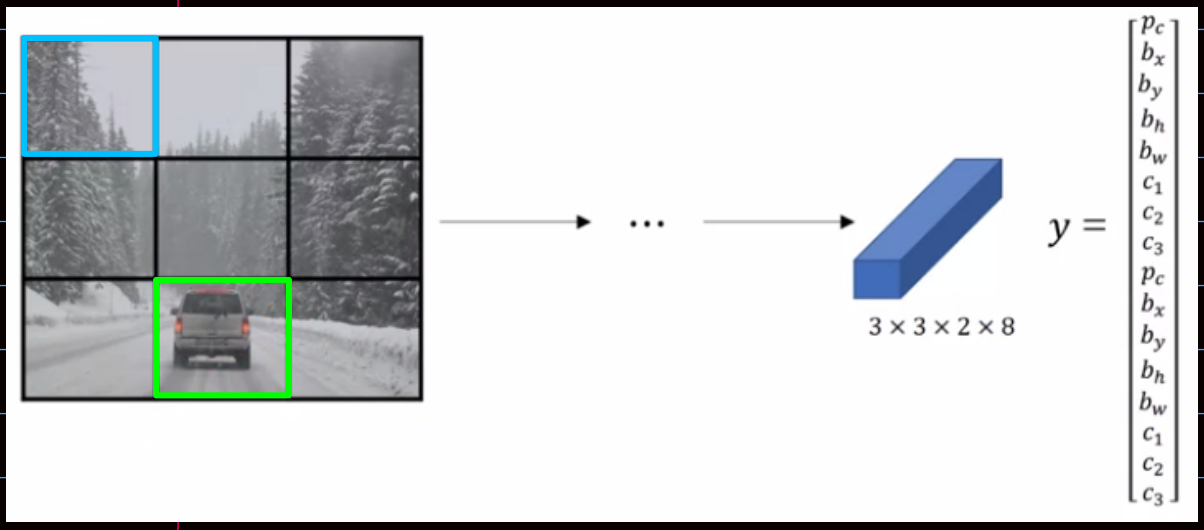
## YOLO ALGORITHM




If

Anchor Box 1 →

Anchor Box →




## Outputting the non-max supressed outputs.

① For each of the nine grid cell, we get 2 predicted bounding boxes

② Get rid of low probability predictions

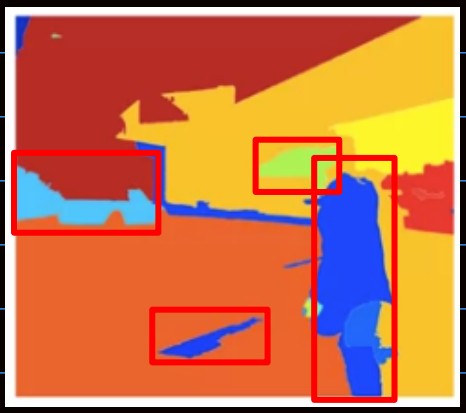
③ For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

# Region Proposals (R-CNN's)

This algorithm chooses some particular regions which makes sense to pass through the Conv Net

Rather than running the classifier through all windows we select some specific windows

Segmentation →



① Propose Regions
② Classify Proposed Regions one at a time
③ Find the accurate — Output label
Bounding Box

Selects the blob regions
(Say) find 2000 blob's & running the classifier on only those blobs.

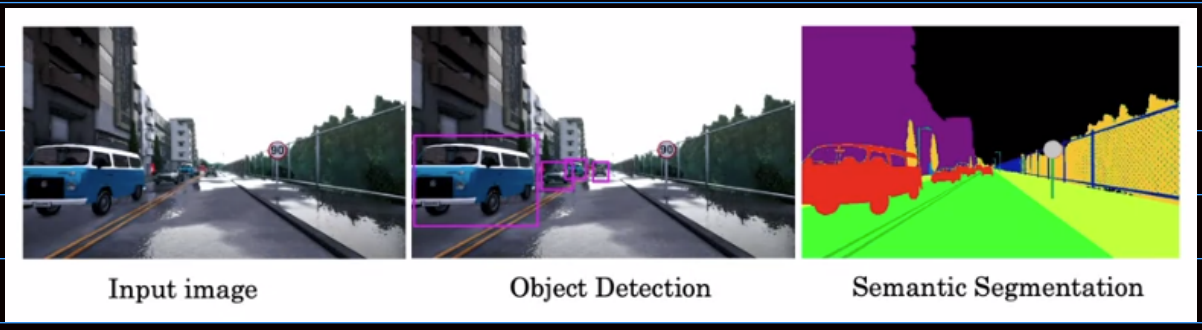
Drawbacks: The algorithm is quite slow.

Fast-(R-CNN) → Propose Regions Use Convolution implementation of sliding windows to classify all the proposed regions

Faster R-CNN: Use convolutional network to propose regions

*Andrew prefers YOLO

## Semantic Segmentation with U-Net

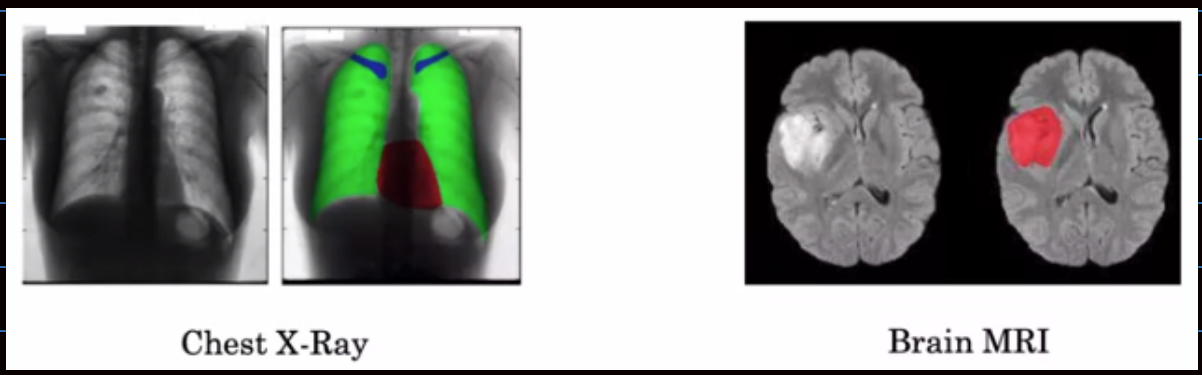


Input image | Object Detection | Semantic Segmentation

↑ Indicates every single pixels

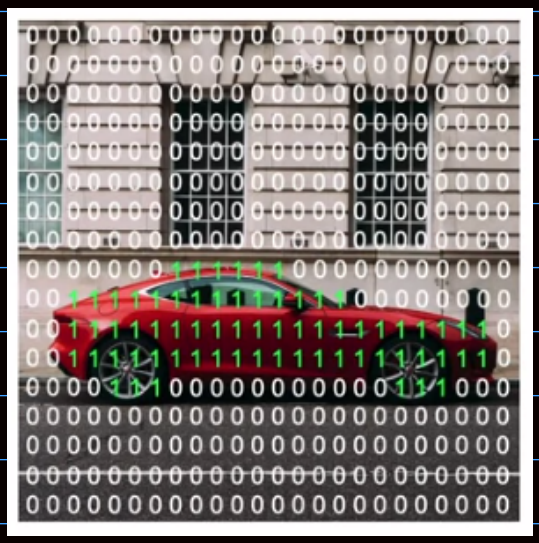
Uses: Self Driving car engg. to know about the drivable surface.

## Other uses:

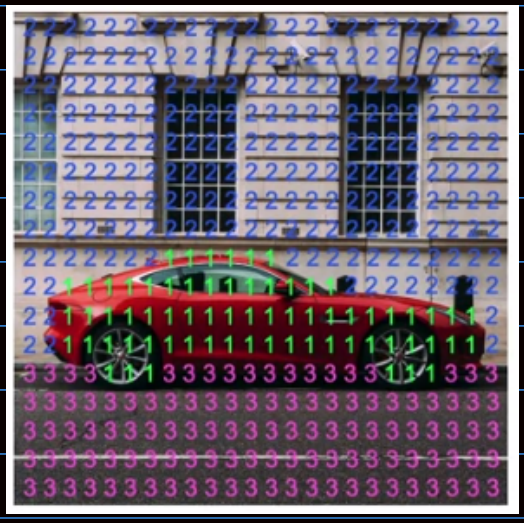


Chest X-Ray | Brain MRI

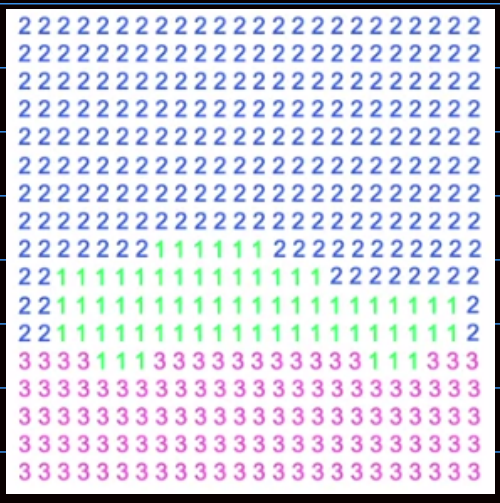
# Per Pixel Class Labels



1 → Car , 0 → not Car.

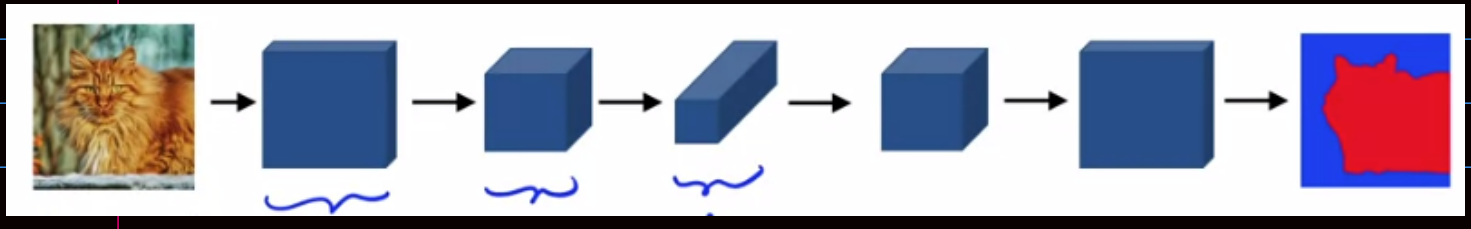


1: Car, 2: Building, 3: Road

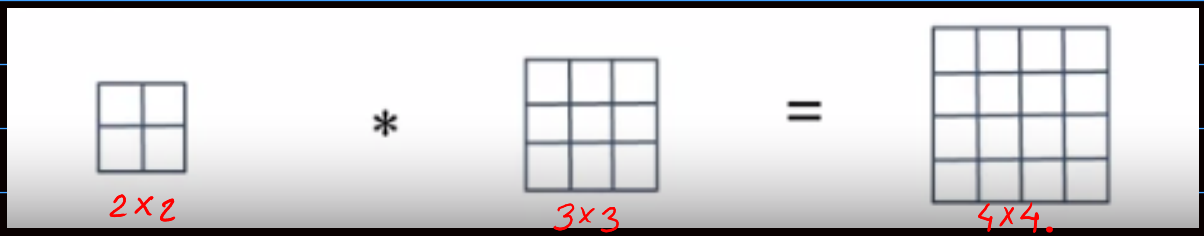


Segmentation Map.

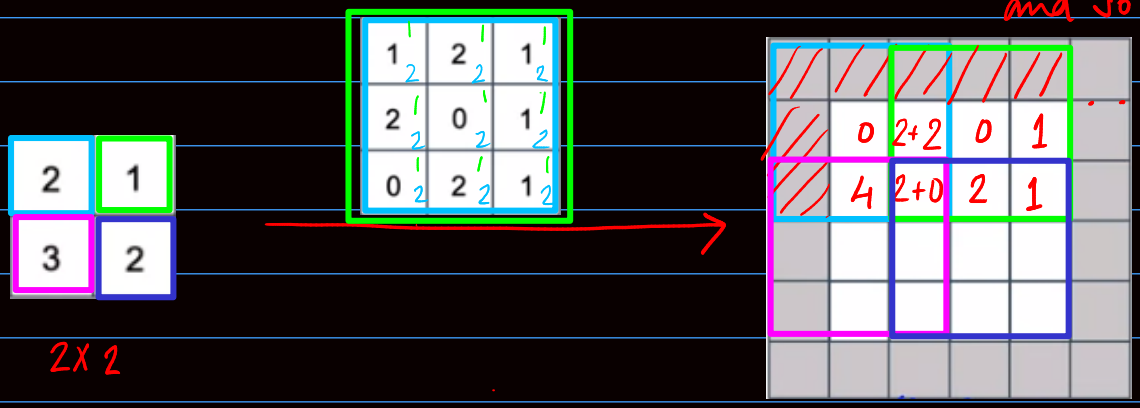
## UNet Implementation:



# Transpose Convolution



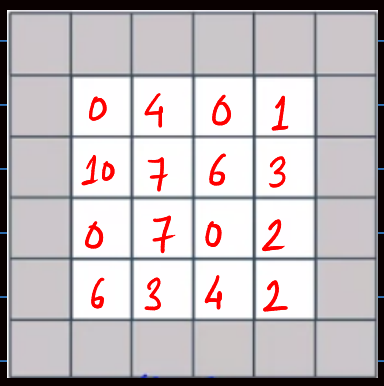

Upsampling




filter = f x f = 3 x 3 (say)

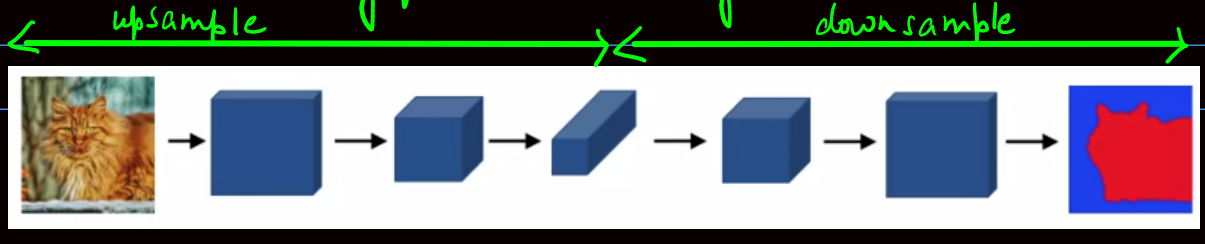
padding = 1

stride = 2



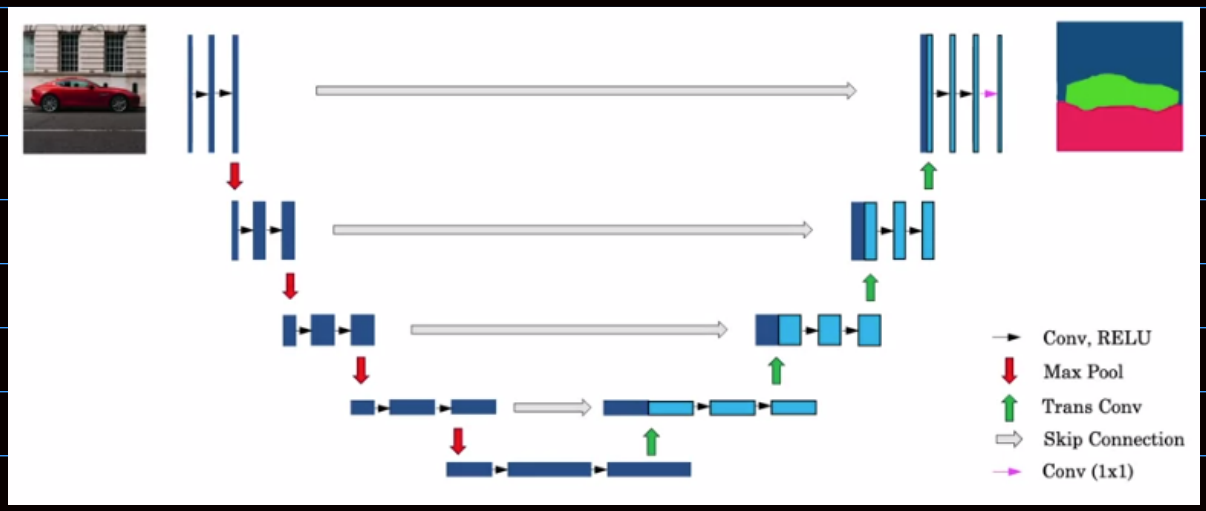

## U-Net Architecture

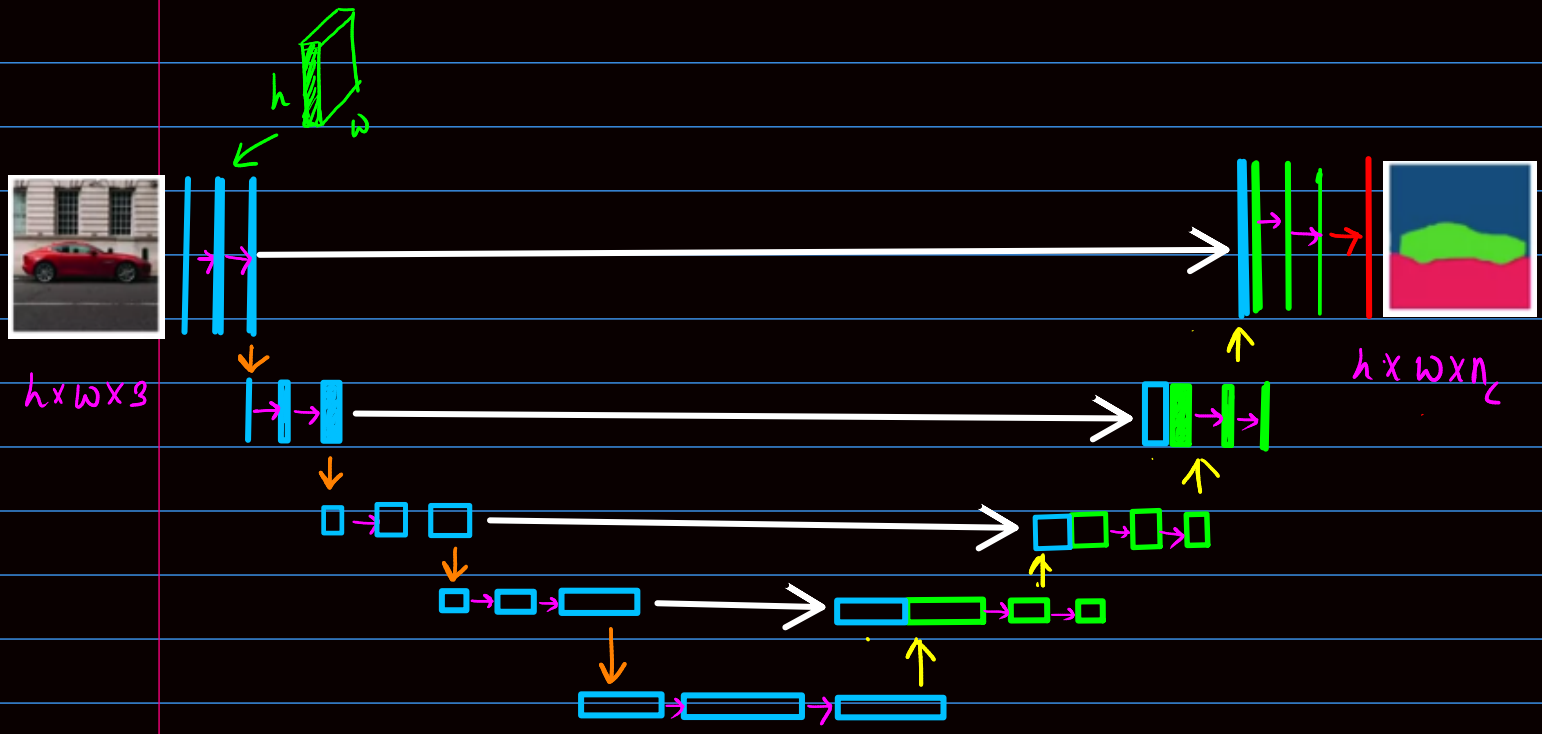
Deep Learning for Semantic Segmentation



Skip Connections are used in U-Net to get the low level information which was downsampled to get the more complex high level features

## Architecture




→ Conv, ReLU ↓ MaxPool ↑ Trans Conv ⇒ Skip Connections → Conv = 1x1

# Week 4

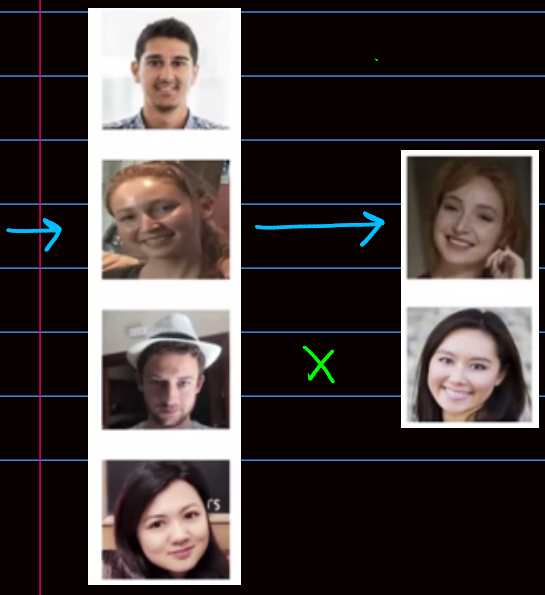
## Face Recognition

**Verification** — I/p image, name ID
O/p whether the input image is that of the claimed person

**Recognition** — Has a database of K persons
Get an i/p image.
O/p ID if the image is any of the K persons

## One Shot Learning Problem.

Learning from one example to recognise the person again



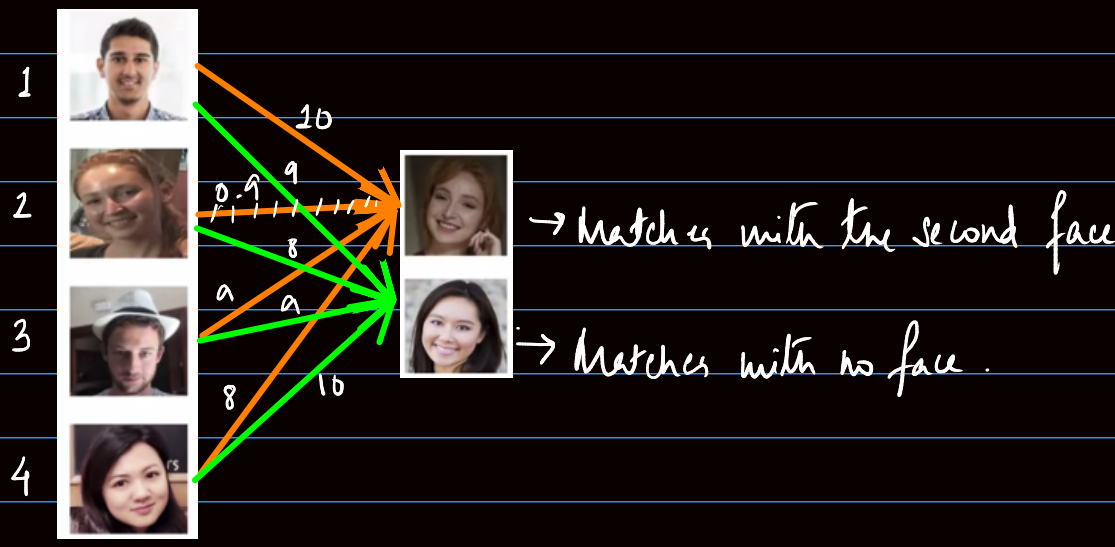
## Learning a 'similarity' function

$d(\text{img1}, \text{img2})$ = degree of difference between images

$d(\text{img1}, \text{img2}) \leq \tau$ → some Threshold → 'same'

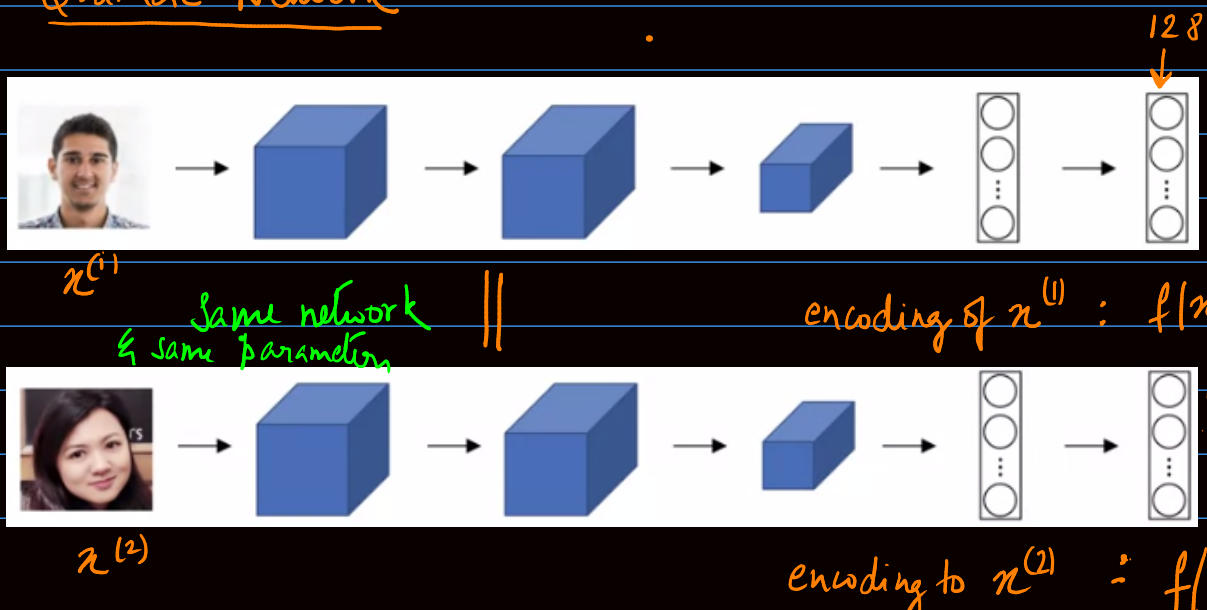
$> \tau$ → 'different'

] Face Verification Problem



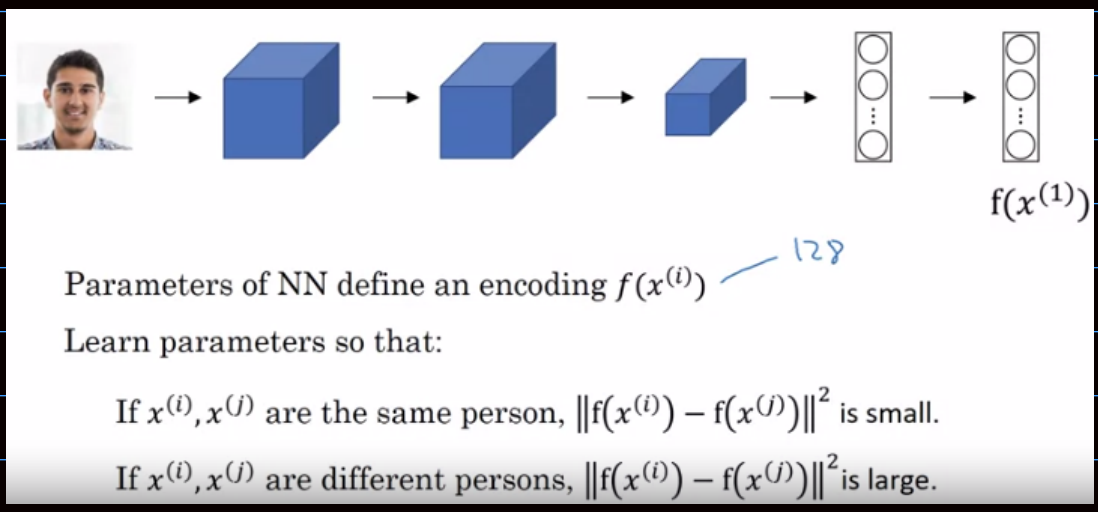

∴ In one shot learning adding extra images to the data set will not effect the face verification.

## Siamese Network

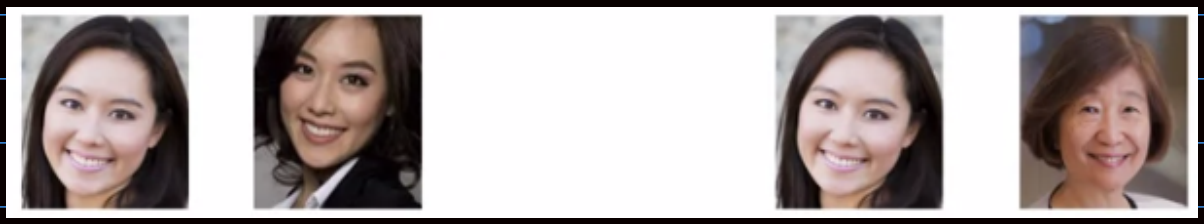


$$d(x^{(1)}, x^{(2)}) = \| f(x^{(1)}) - f(x^{(2)}) \|_F^2$$

This way of calling two NN's for 2 different i/p's & then comparing them → Siamese network.




## Triplet Loss



Anchor A | Positive P | Anchor A | Negative N

$$\underbrace{\| f(A) - f(P) \|^2}_{d(A,P)} \leq \underbrace{\| f(A) - f(N) \|^2}_{d(A,N)}$$

$$\| f(A) - f(P) \|^2 - \| f(A) - f(N) \|^2 \leq 0 - \alpha$$

$$\Rightarrow \| f(A) - f(P) \|^2 - \| f(A) - f(N) \| + \alpha \leq 0$$

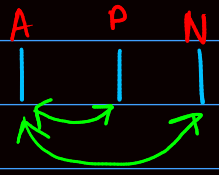
$\alpha$ → margin (pushes the AP & AN gap further)

## Loss Function

Given 3 images A, P, N

$$L(A,P,N) = \max\left(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0\right)$$

$$J = \sum_{i=1}^{m} L(A^{(i)}, P^{(i)}, N^{(i)})$$




eg. Training Set of 10K Pictures & 1K persons

(A P) → We need multiple pictures of the same person ∴ 1K persons & 10K images

## Choosing the triplets A, P & N

1. During training if A, P & N are chosen randomly

$d(A,P) + \alpha \leq d(A,N)$ is easily satisfied

$$\|f(A) - f(P)\|^2 + \alpha \leq \|f(A) - f(N)\|^2$$

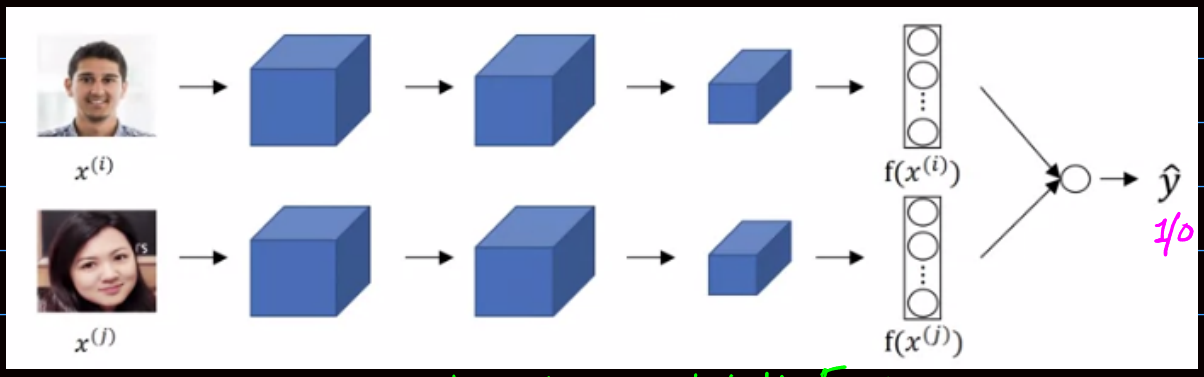
2. Choose triplets that are hard to train on.

s.t $d(A,P) \simeq d(A,N)$

so that to satisfy.

$d(A,P) + \alpha \leq d(A,N)$ the classify finds it hard

# Face Verification as a Binary Classification Problem




element wise diff bet$^n$ the two encodings

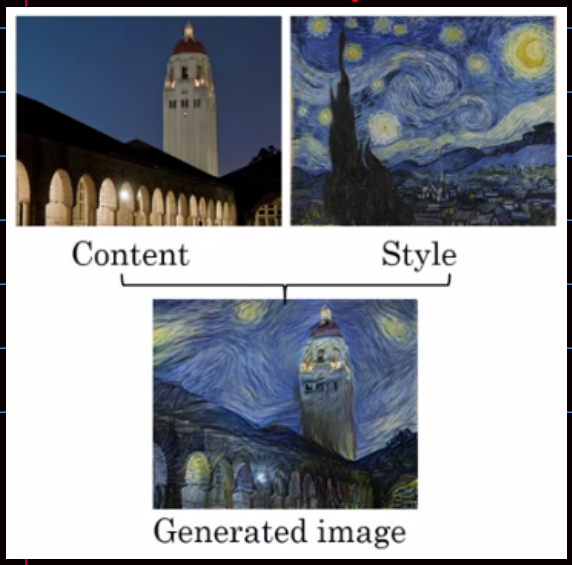
$$\hat{y} = \sigma\left(\sum_{k=1}^{128} w_k \left| f(x^{(i)})_k - f(x^{(j)})_k \right| + b\right)$$

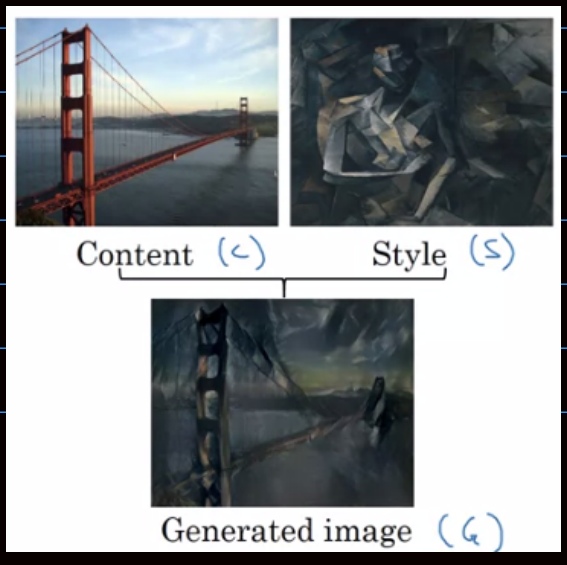
or

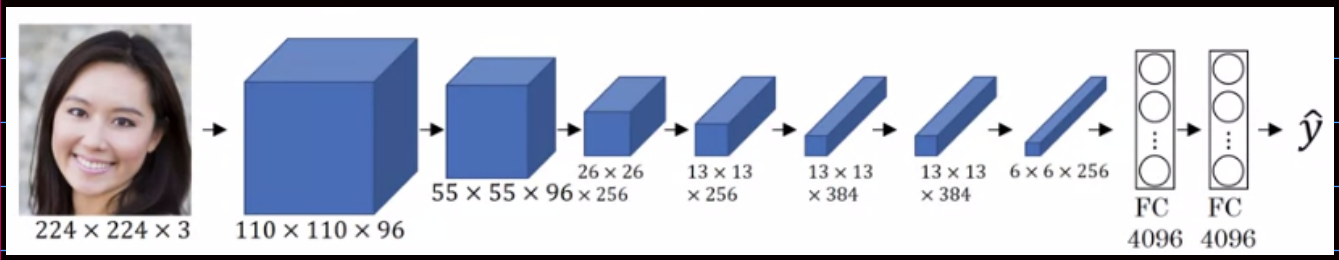
$$\frac{\left(f(x^{(i)})_k - f(x^{(j)})_k\right)^2}{f(x^{(i)})_k + f(x^{(j)})_k} = \chi^2$$

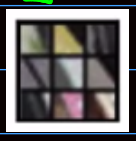
* If a new person is included in the dataset then instead of computing the entire Siamese Network, we can use the upper conv. net layers to directly get the encoding & then comparing it with the precomputed encodings

## Neural Style Transfer

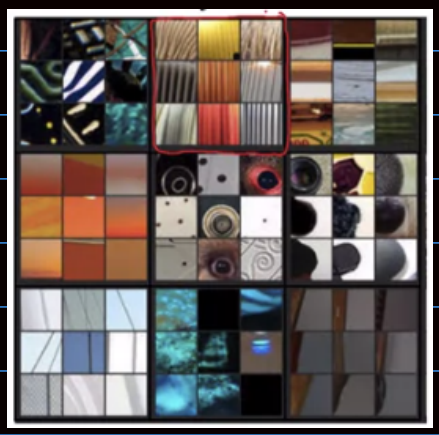





Pick a unit in layer 1. Find the nine images patches that maximizes the unit's activation → 
Repeat for other units.

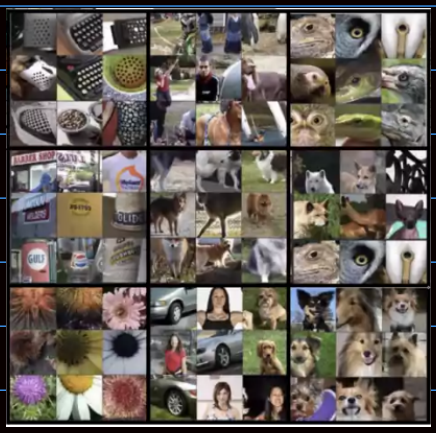
← Layer 1 (These are the 9 patches that causes layer 1 to be highly activated!)



Layer 2

Layer 3

Layer 4

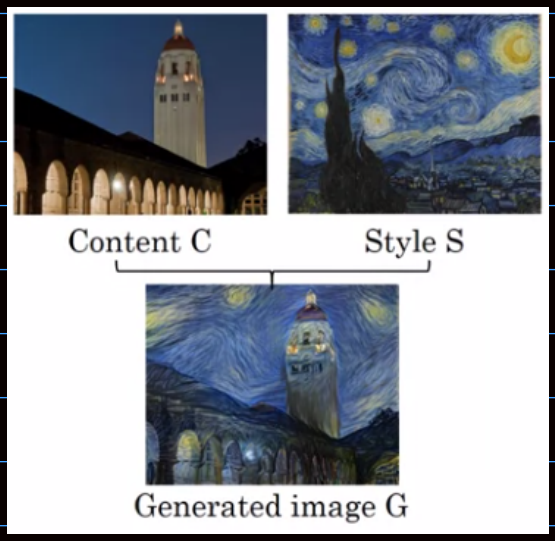


Layer 5

# Cost Function (for Neural Style Transfer)

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

(hyper parameters)



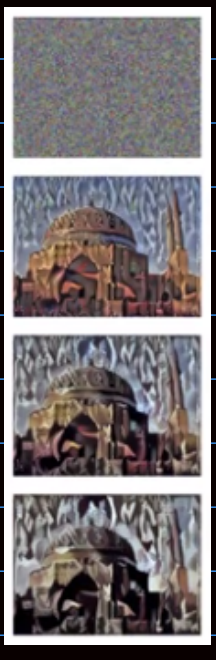

## Find the Generated Image G,

1. Initiate G randomly.

   G: 100 X 100 X 3

2. Use gradient descent to minimize $J(G)$

$$G = G - \frac{d}{dG} J(G)$$



→ Random Noise

→

## Content cost Function

$$J(G) = \alpha J_{content}(C,G) + \beta J_{style}(S,G)$$

① Say we use hidden layer $l$ to compute content cost.

{ $l$ → neighter too shallow nor too deep }

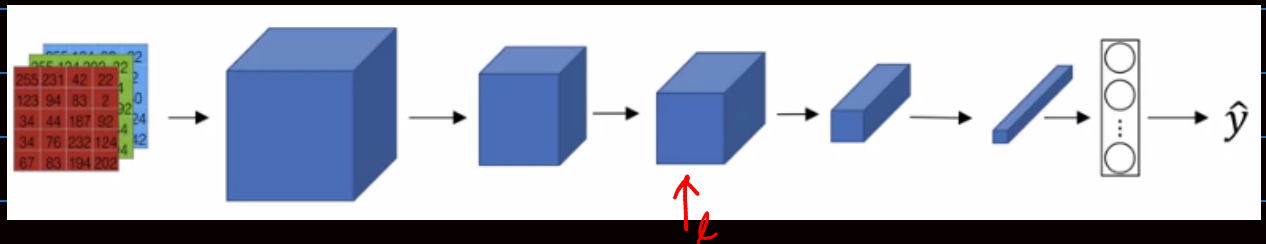
② Use Pretrained Conv. Net ( Eg VGG Net)

③ Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer $l$ on the images

④ If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content.

$$J_{content}(C,G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|^2$$

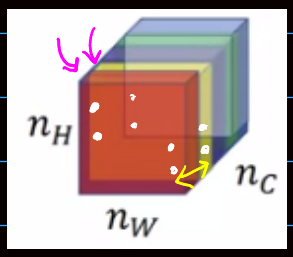
## Meaning of 'style' of an image



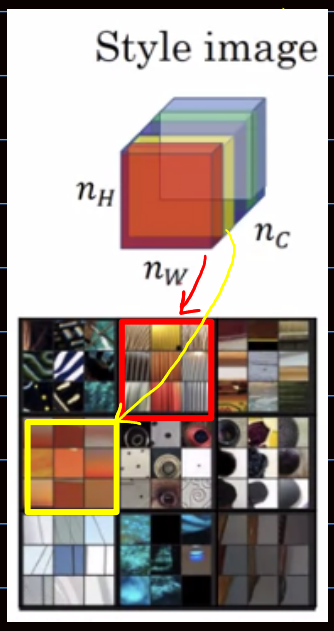

→ Say you are using layer $l$'s activation to measure 'style'

→ Define style as correlation between activations across channels

$n_c = 5$ (say)




How correlated are the activation across different channels?




If these two channels are correlated = Wherever part of the image will have texture ☐ will have have texture ☐ also Otherwise not correlated

* This degree of correlation gives us an idea about how often this features occur together or don't occur.

* Used to compare the style of the generated image.

## STYLE MATRIX

Let $a^{[l]}_{i,j,k}$ = activation at $(i, j, k)$. ($H$, $W$, $C$) $G^{[l]}$ is $n_c^{[l]} \times n_c^{[l]}$ (no. of channels)

$G^{[l]}_{kk'}$, $(k = 1, 2, \ldots n_c^{[l]})$ = Will measure how correlated are activations in channel $k$ to activations in channel $k'$

Style.

$$G^{[l](S)}_{kk'} = \sum_{i}^{n_H^{[l]}} \sum_{j}^{n_W^{[l]}} a^{[l](S)}_{ijk}\, a^{[l](S)}_{ijk'}$$

Generated.

$$G^{[l](G)}_{kk'} = \sum_{i}^{n_H^{[l]}} \sum_{j}^{n_W^{[l]}} a^{[l](G)}_{ijk}\, a^{[l](G)}_{ijk'}$$

If correlated $G^{[l]}_{kk'}$ = large.

else, $G^{[l]}_{kk'}$ = small

* 'G' in linear algebra is 'gram matrix'

$$J^{[l]}_{style}(S,G) = \left\| G^{[l](S)} - G^{[l](G)} \right\|^2_F$$

$$= \frac{1}{4 n_H^{[l]2} n_W^{[l]2} n_C^{[l]2}} \sum_{k} \sum_{k'} \left( G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'} \right)^2$$

Normalization constant

If we use Style cost fun^ from multiple layers

Therefore,

Overall Cost Function

$$J_{style}(S,G) = \sum_{l} \lambda^{[l]} J^{[l]}_{style}(S,G)$$

* Take both high level & low level layers into a/c

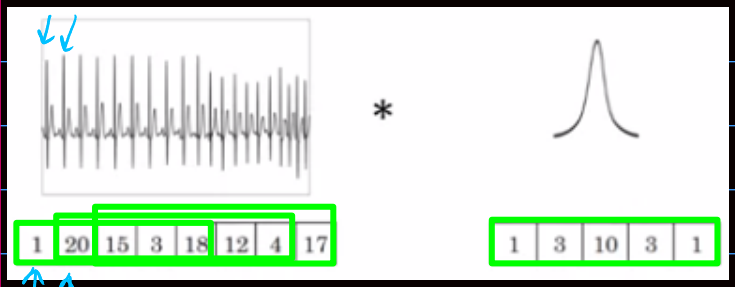
$$J(G) = \alpha J_{content}(C,G) + \beta J_{style}(S,G)$$

G to minimize J(G)

# 1D & 3D Generalization

## Convolutions

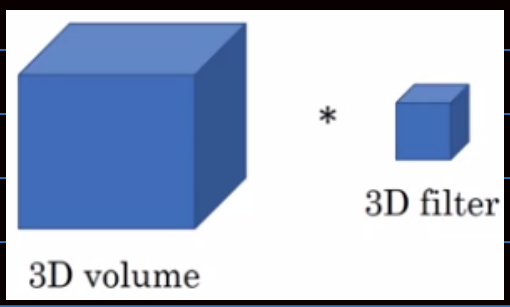
2D Input Image $14 \times 14 \times 3$ * 2D Filter $5 \times 5 \times 3$ → Result $10 \times 10 \times 16$




1D Time Series

$14 \times \underline{1}$ * $5 \times \underline{1}$ → $10 \times \underline{16}$ * $5 \times \underline{16}$ → $6 * 32$ (filters)

3D Data. → Eg. CT Scan, Movie Data




$14 \times 14 \times 14 \times 1$ $5 \times 5 \times 3 \times 1$

Layer 1 (no. of filters) → $10 \times 10 \times 10 \times 16$ * $5 \times 5 \times 5 \times 16 = 6 \times 6 \times 6 \times 32$ Layer 2