The Convolution Operation

- Suppose we are tracking the position of an airplane using a laser sensor at discrete time intervals.
- Suppose the laser sensor is noisy.
- To obtain a less noisy estimate, we take the average of several measurements.
- More recent measurements are more important so we take a weighted average.

$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} = (x * w)_t$$

weightage (w_{-a}); s_t: revised measurement; x_{t-a}: reading at time t; x: input, w: filter, $*$: convolution

- In practice we would rely on a window for measurements.
  eg- weighted average of last six measurements; upto $t-6$ seconds
- The weight average (w) is known as the filter
- We slide the filter over the input & compute the value of s_t based on window around x_t.

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5

window (a) is 6

X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S	=	1.80										

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_4 w_{-4} + x_5 w_{-5} + x_6 w_{-6}$$

- Input & kernel is one dimension.

- Input & kernel is one dimension.

Let's change story to image now.

- We will use a 2D filter (m x n)

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a, j+b} K_{a,b}$$

- We look at the preceeding neighbours $(i-a, j-b)$

- In practice we use the formula which looks at the succeeding neighbours.

I_{ij} → Particular pixel.

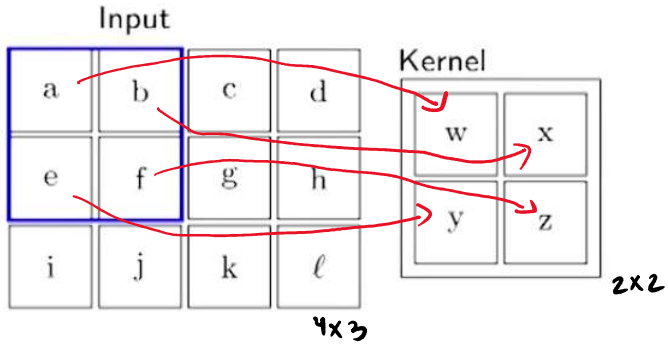
I^{th} i,jth entry

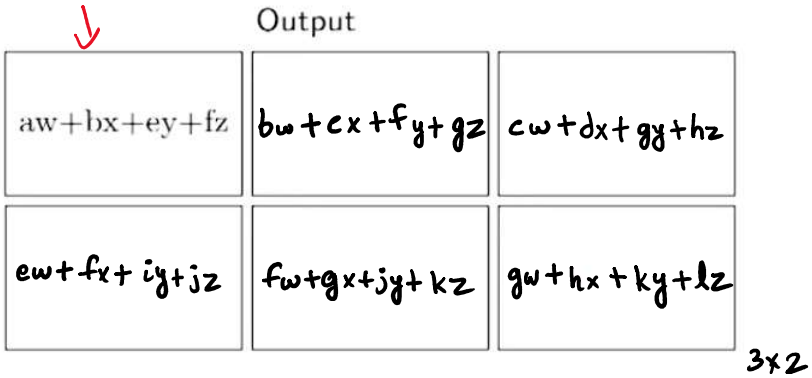
m → columns

n → rows

K → weight

Toy Example



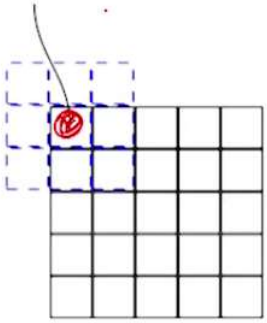


3x2

We will assume that the kernel is centered on the pixel of interest.

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a, \frac{n}{2}+b}$$

pixel of interest



Examples

$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

blurs the image

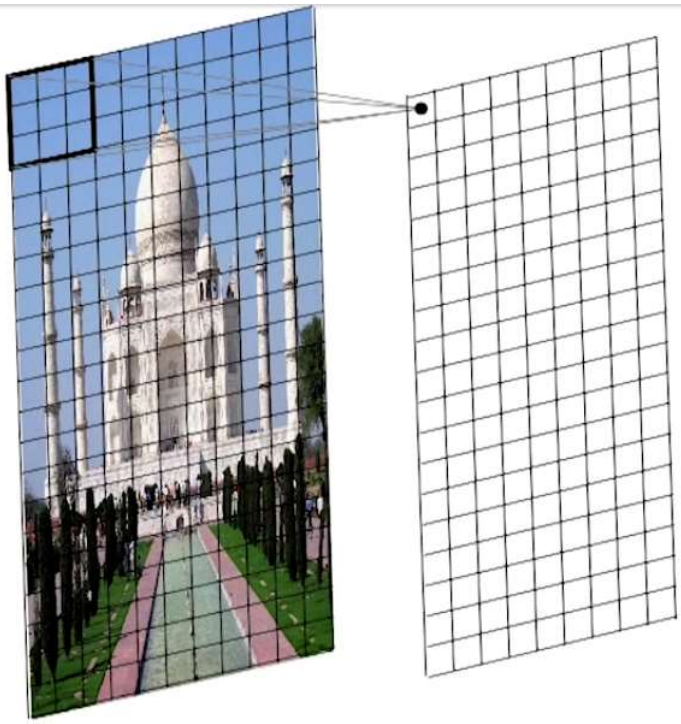
$$* \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix} =$$

sharpens the image

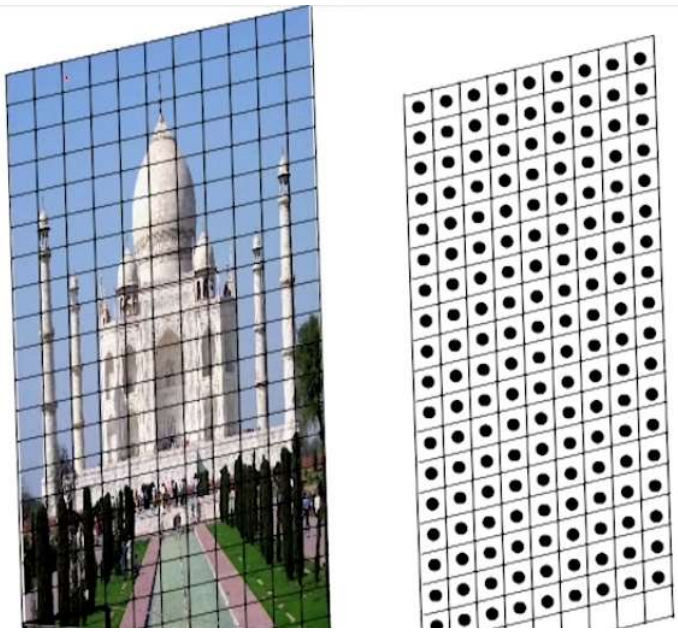
$$* \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

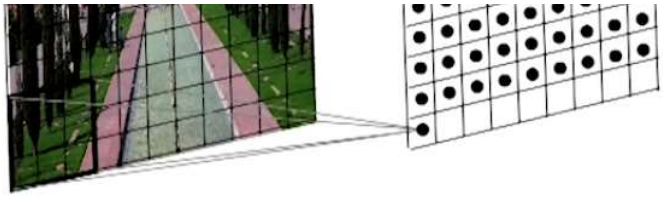
detects the edges

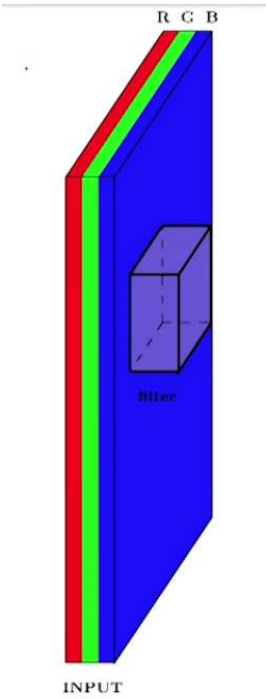
Working Example



- We slide this kernel over the input image.
- Each time we slide, we get one value as output
- The output we get is called a feature map.
- We apply k filter maps & we get k feature maps.
- Generally we apply a 3D filter





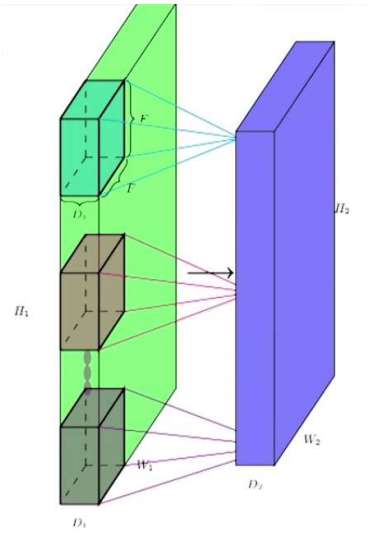


- 3D filters will be referred to as volume
- We will do a 2D convolution operation on 3D point & output will be 2D.

Relation b/w input output & filter size

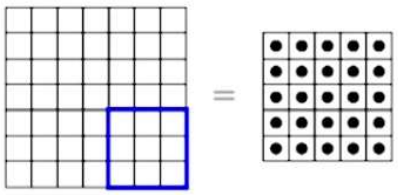
We'll start by defining the following quantities in the original input:

- Width (W_1), Height (H_1) & Depth (D_1) ← generally 3 (RGB)
- Stride (S)
- No. of filters (K) ← if we apply k filters we get k feature maps in 2D.
- Spatial extent (F) of each filter ← depth of each filter, same as depth of each input
- Output is $(W_2 \times H_2 \times D_2)$

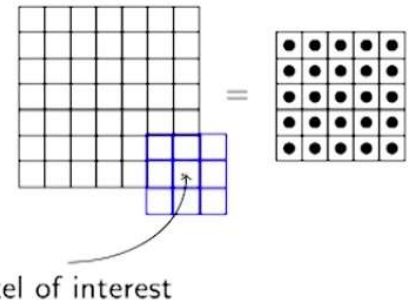


Toy Example

For a 7×7 image, let us apply one 3×3 filter & see output

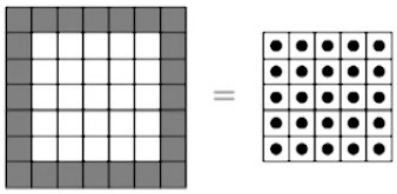


We can't place the kernel at boundaries as it'll cross input boundary



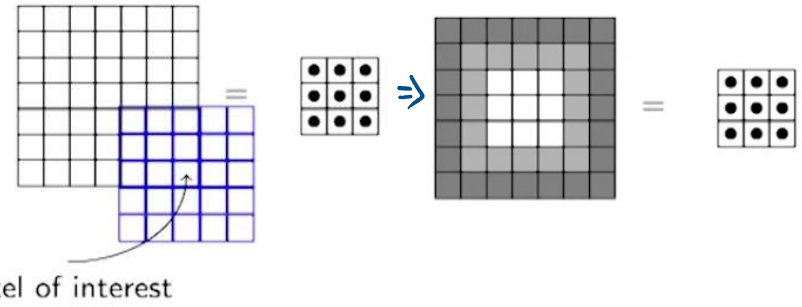
pixel of interest

For a 3x3 filter the reduction is: Height & width decreases by 2.



For a 5x5 kernel:

Width & Height reduced by 4



Conclusion: The reduction is $(F-1)$

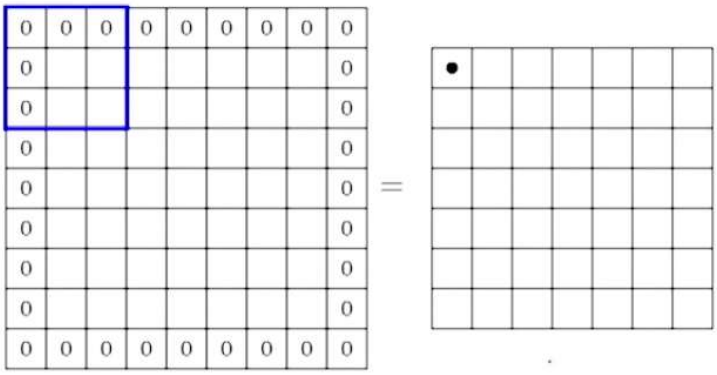
$$\therefore W_2 = W_1 - F + 1$$
$$H_2 = H_1 - F + 1$$

What if we want the output to be the same size as input?

We use something known as padding.

Pad the inputs w/ appropriate number of 0 inputs so that we can apply the kernel at the corners.

Let us use pad $P=1$ with a 3x3 kernel



We added a boundary of of 1 pixel each of black colour

So now for an input of 7x7 image, we made it a 8x8 image, applied a 3x3 kernel & got output of 7x7 ∴

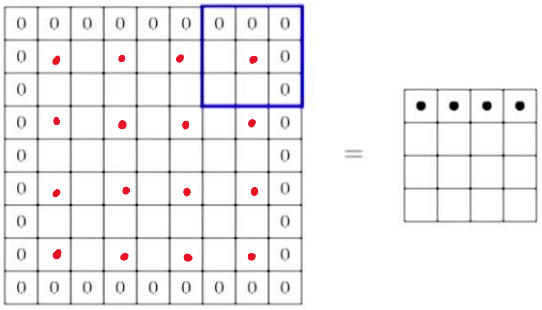
$$\therefore W_2 = W_1 - F + 2P + 1$$

$$H_2 = H_1 - F + 2P + 1$$

Stride

It is the interval at which filter is applied. (aka step)

If stride = 2 then we are skipping every 2nd pixel to apply the filter at. Result: Output with smaller dimensions



Final formula:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

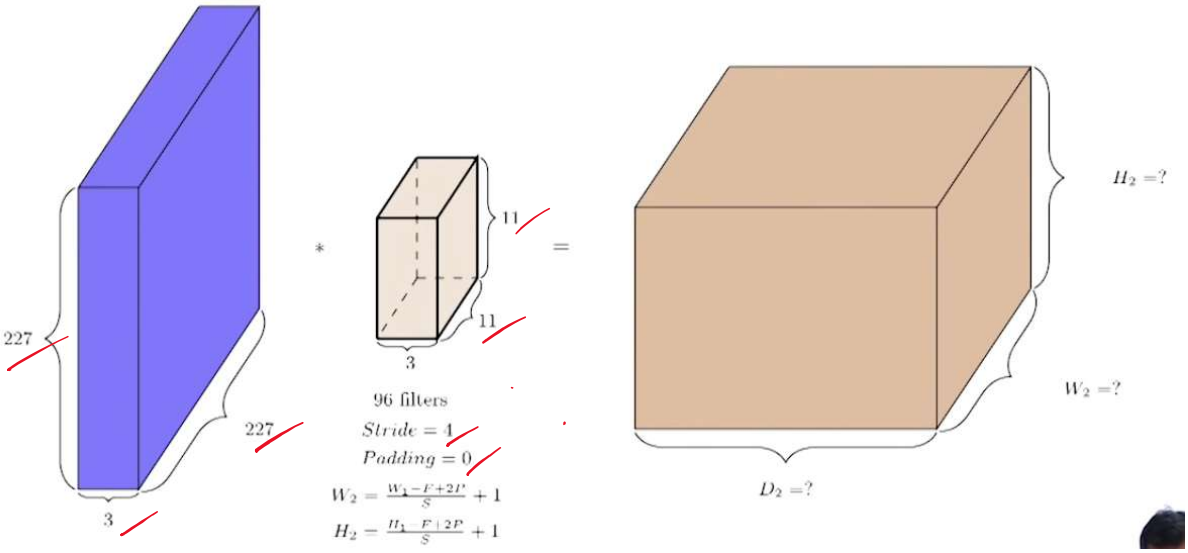
What would be the depth of output?

Each filter gives a 2D output

k filters will give k 2D outputs.

$$\therefore D_2 = k$$

Exercises

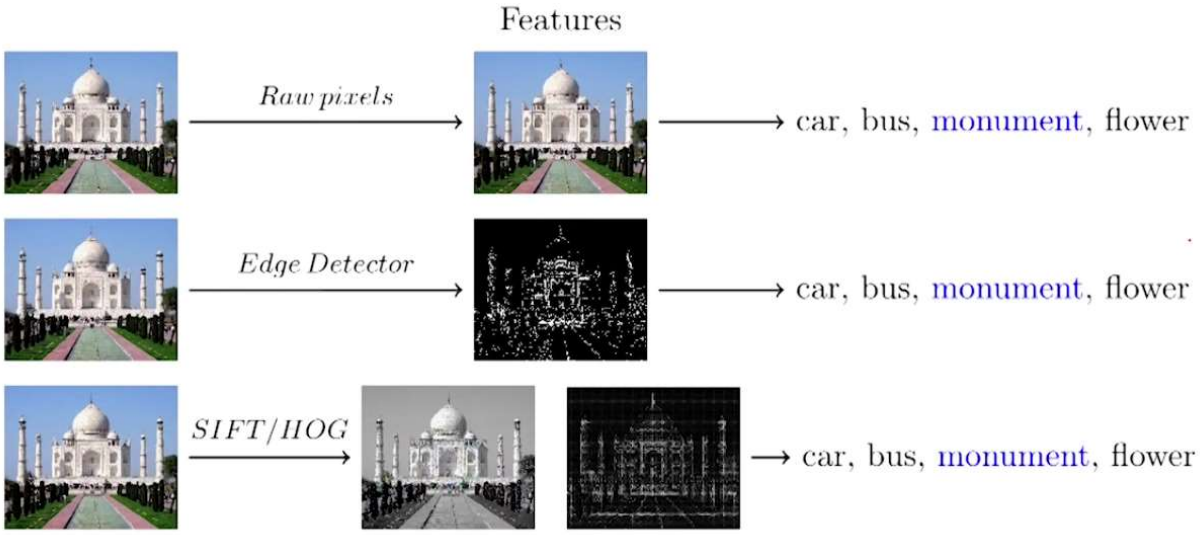


$$H_2 = \frac{H_1 - F + 2P}{S} + 1 = \frac{227 - 11 + 0}{4} + 1 = 55$$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 = \frac{227 - 11 + 0}{4} + 1 = 55$$

$$D_2 = 3$$

Convolutional Neural Networks



The kernels we used above are handcrafted. The learning is happening only on the weights of the classifier. Can we have learning for the weights of the kernels as well?

Can we learn weights of multiple kernels instead of always handcrafting them?

Can we learn multiple layers of meaningful kernels in addition to the weights of the classifier?

Yes. Yes. A thousand times yes!

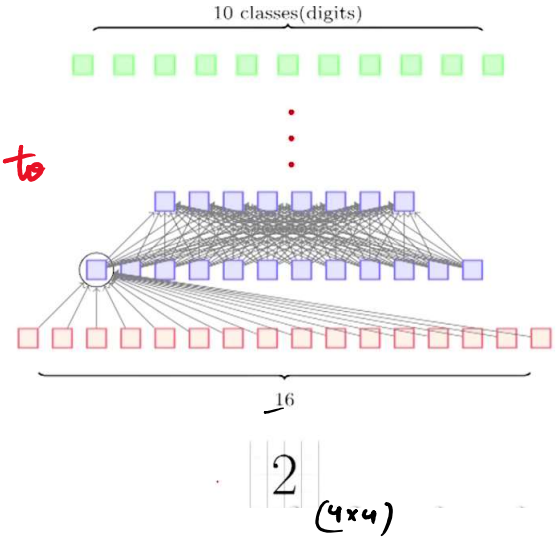
We can do that by treating the kernels as parameters & learning them in addition to the weights of the classifier. (using backpropagation)

Such a network is called Convolutional Network.

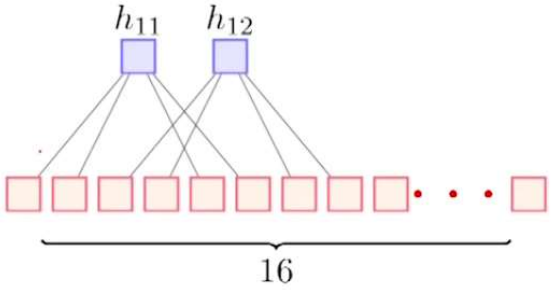
Such a network is called Convolutional Network.

Difference b/w CNN & NN?

For a given input : MNSIT database : 2 we want to predict what number it is:



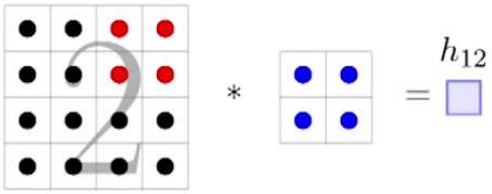
The next neuron is connected to every single input neuron.



In CNN, only a few neurons are connected to the next layer.

Using a stride of 2.

The connections are sparser here compared to feed forward neural network.



The intuition is that in an image we do not care about the interactions b/w the top left pixel & right bottom corner. We wanna capture only a small neighbours of a particular pixel & not the entire image.

This is the first property of a CNN.

Sparse Connectivity.

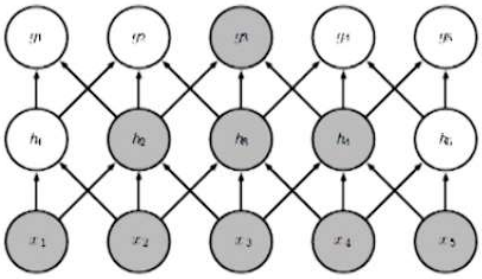
It reduces the no. of parameters in the model.

It reduces the no. of parameters in the model.

However, we are losing information by losing the interactions b/w the pixels aren't we?

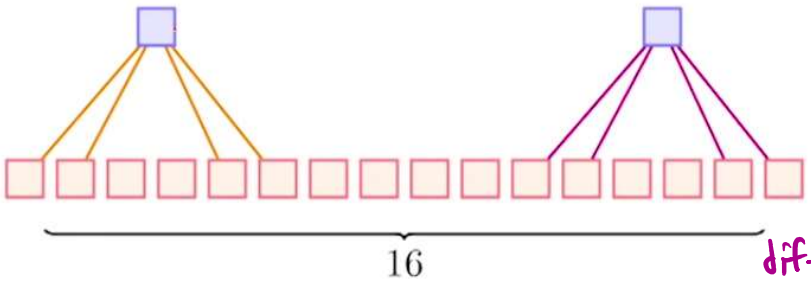
Nope! Because of multiple layers.

In a deep layered network, all pixels get to interact at deeper levels. At the more immediate layers, we just wanna capture the interactions b/w a neighbourhood.



This is what sparse connectivity looks like.

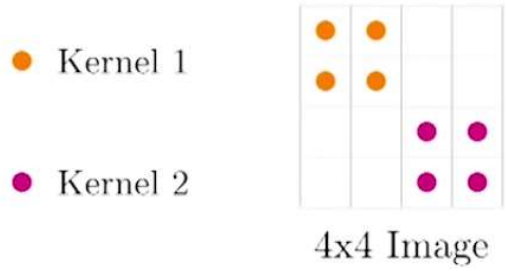
Another characteristic is something known as weight sharing.



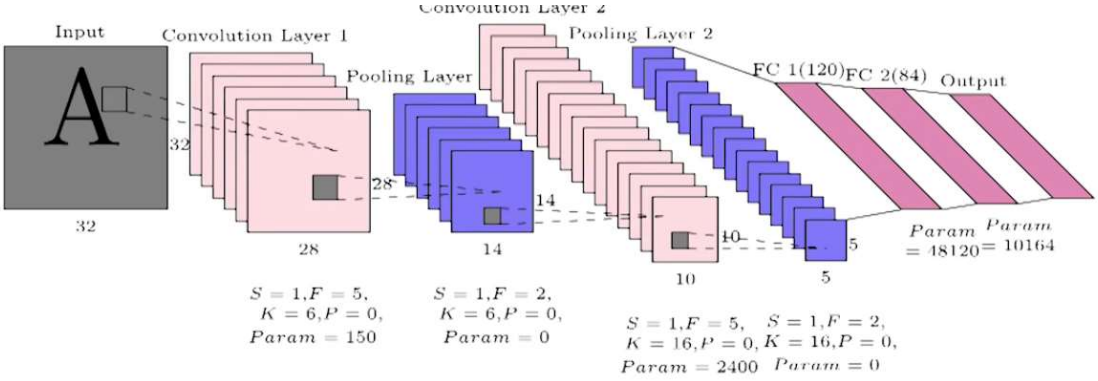
Do we want the kernel weights to be different for different portions of the image.

Imagine that we are trying to learn a kernel that detects edges

Shouldn't we apply the same kernels at all portions of the image?



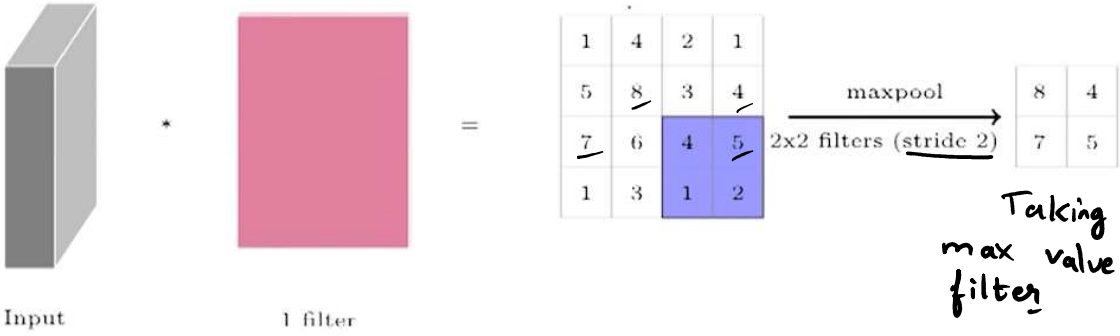
In other words, the pink & orage kernels be the same.



This is how a convolutional network looks like.

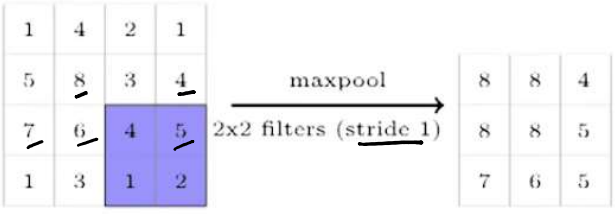
Pooling

It results in the reduction of the size of input.



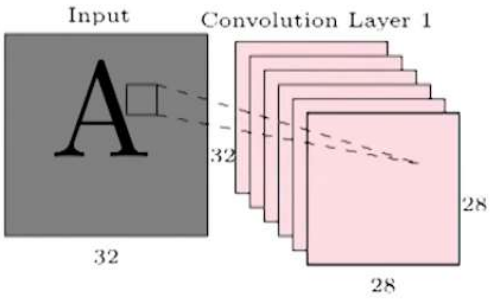
Taking only max value from the filter

we can also do average pooling instead of max pooling



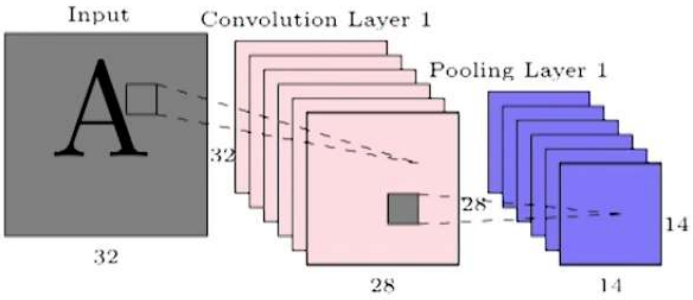
CASE STUDIES

LeNet-5 for handwritten character recognition



$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = ?$

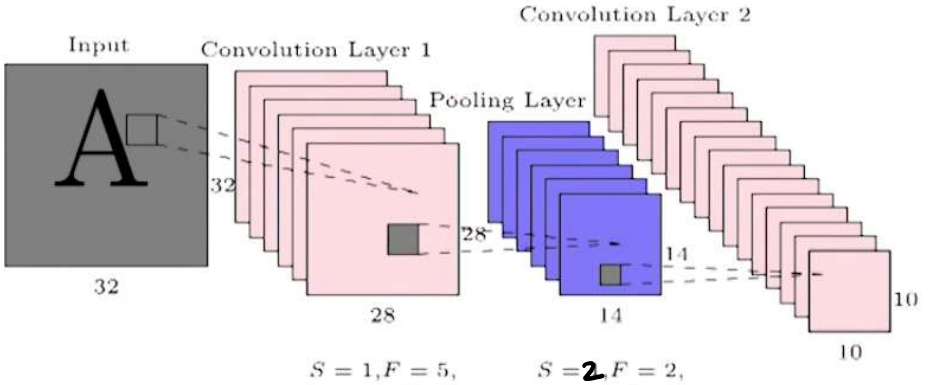
No. of parameters = 5×5×6
= 150



$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = 150$

$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = ?$

No of parameters = 0

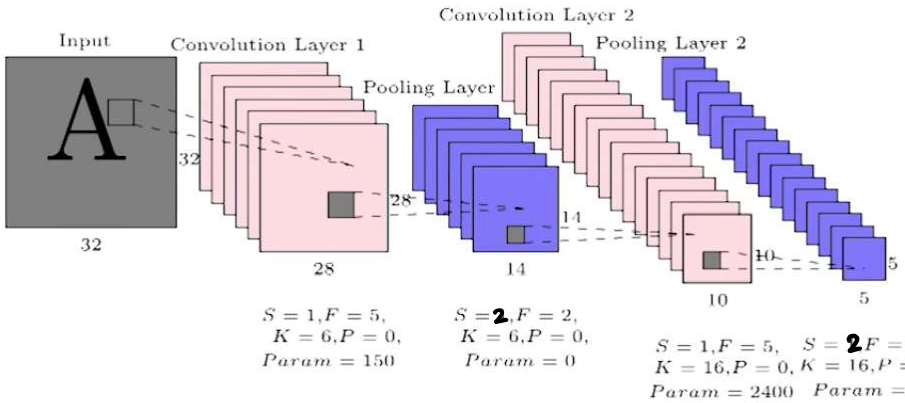


$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = 150$

$S = 2, F = 2,$
$K = 6, P = 0,$
$Param = 0$

$S = 1, F = 5,$
$K = 16, P = 0,$
$Param = ?$

No of parameters
= (5×5×6)×16
= 2400
↑ Depth / no. of filters



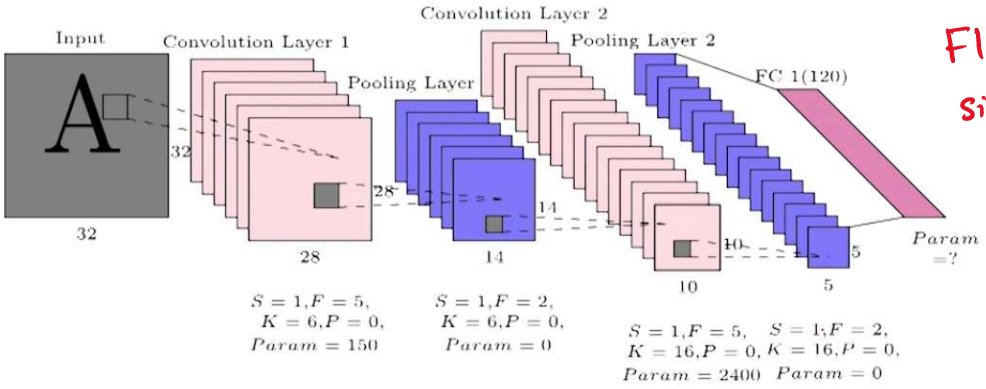
$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = 150$

$S = 2, F = 2,$
$K = 6, P = 0,$
$Param = 0$

$S = 1, F = 5,$
$K = 16, P = 0,$
$Param = 2400$

$S = 2, F = 2,$
$K = 16, P = 0,$
$Param = ?$

No. of parameters = 0

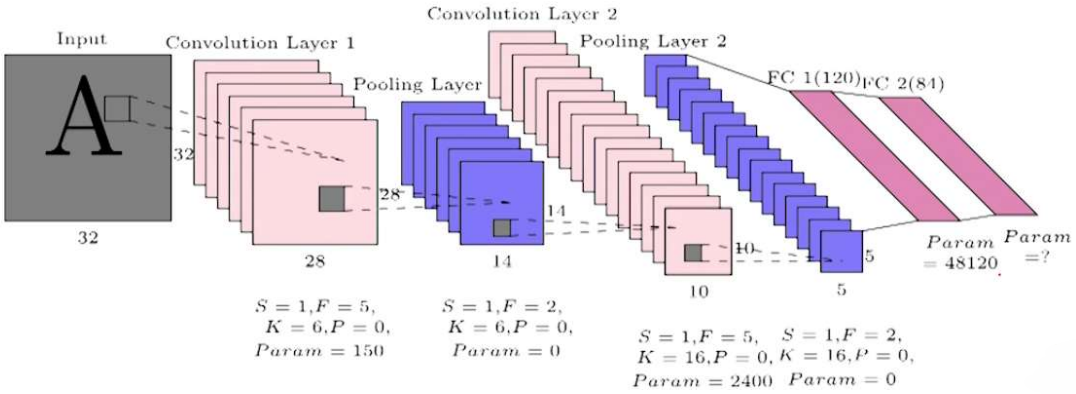


Flattening & getting a single vector of size 400 & fully connect to next layer

No. of parameters

$= 400 \times 120 + 120$ (↑ bias)

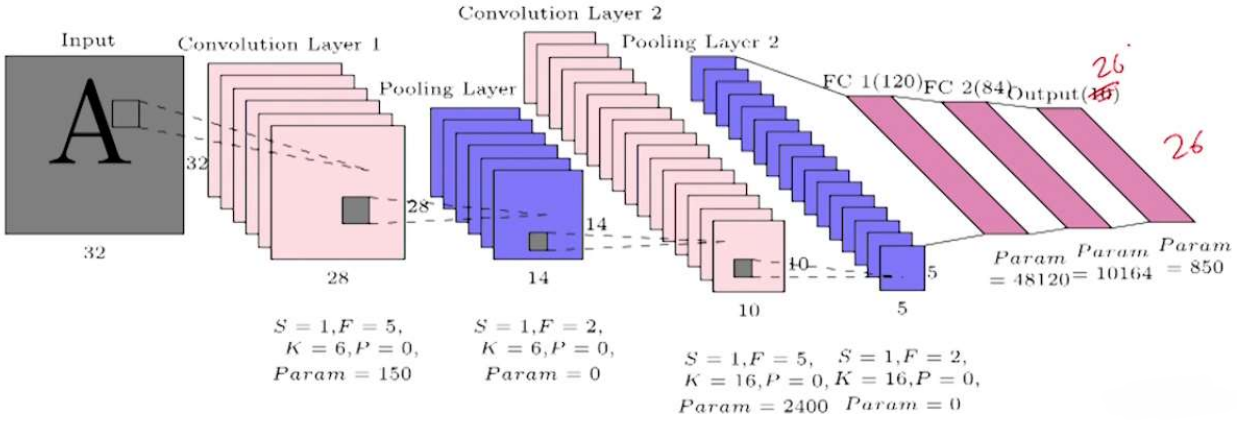
$= 48120$



No. of parameters

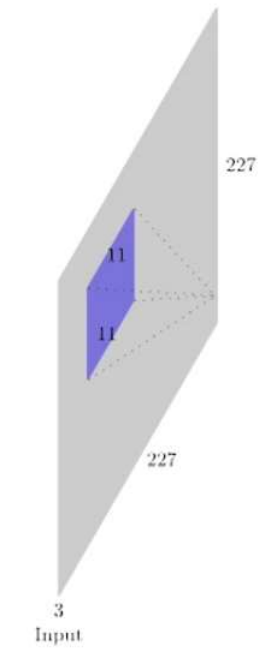
$= 120 \times 84 + 84$

$= 10164$



Success Story of ImageNet

AlexNet



Input = $227 \times 227 \times 3$

$K = 96, \quad F = 11$

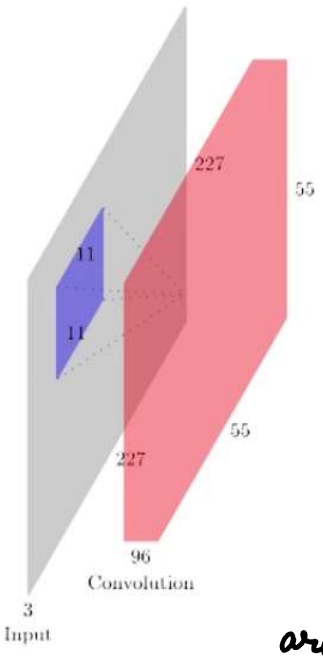
$S = 4, \quad P = 0$

Output, $W_2 = \frac{227 - 11}{4} + 1 = 55$

$H_2 = \frac{227 - 11}{4} + 1 = 55$

$D_2 = 3$

Parameters = $(11 \times 11 \times 3) \times 96 \approx 34000$



Max Pooling layer Input : $55 \times 55 \times 96$

$F = 3, \quad S = 2$

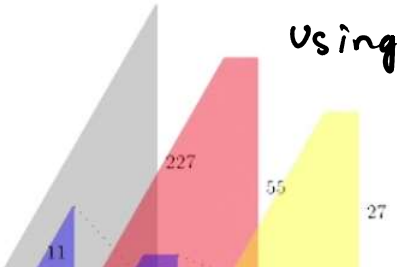
Output $W_2 = \frac{55 - 3}{2} + 1 = 27$

$H_2 = \frac{55 - 3}{2} + 1 = 27$

$D_2 = 96$

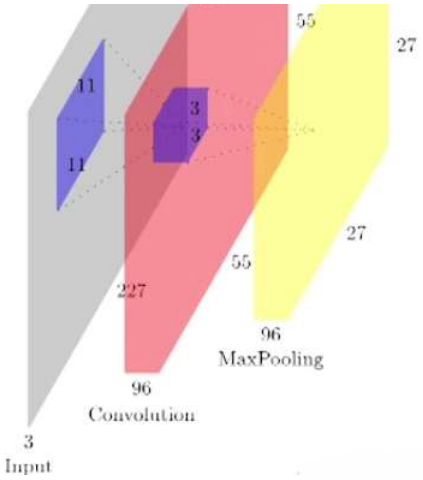
Parameters = 0

As we keep on proceeding, width & height are decreasing & depth is increasing as we are using more & more filters to capture



Input = $27 \times 27 \times 96$

$K = 256, \quad F = 5, \quad S = 1, \quad P = 0$



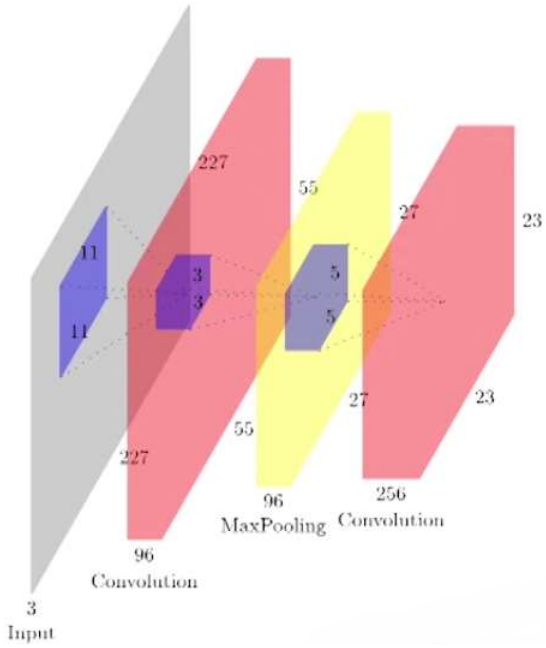
$K = 256$, $F = 5$, $S = 1$, $P = 0$

Output $W_2 = \frac{27-5}{1} + 1 = 23$

$H_2 = \frac{27-5}{1} + 1 = 23$

$D_2 = 256$

Parameters $= (5 \times 5 \times 96) \times 256 \approx 0.6$ Million



Max Pool Input: $23 \times 23 \times 256$

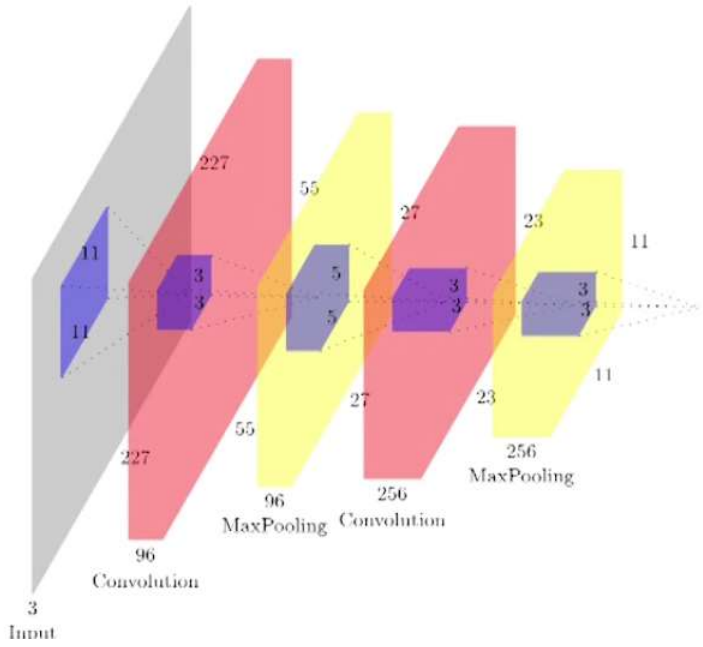
$F = 3$, $S = 2$

Output $W_2 = \frac{23-3}{2} + 1 = 11$

$H_2 = \frac{23-3}{2} + 1 = 11$

$D_2 = 256$

Parameters $= 0$



Input $= 11 \times 11 \times 256$

$K = 384$, $F = 3$, $S = 1$, $P = 0$

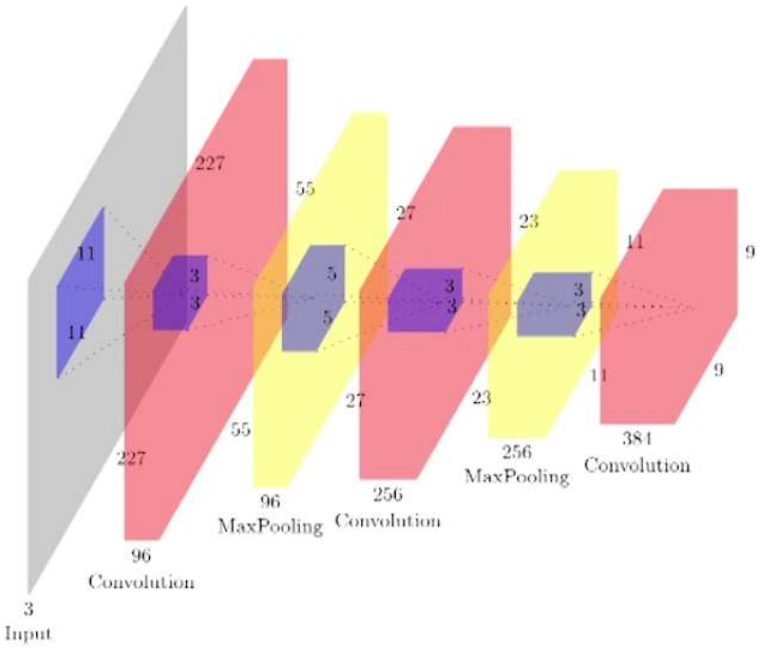
Output $W_2 = \frac{11-3}{1} + 1 = 9$

$H_2 = \frac{11-3}{1} + 1 = 9$

$D_2 = 384$

Parameters $= (3 \times 3 \times 256) \times 384$

≈ 0.8 Million



Input = 9 x 9 x 384

k = 384, F = 3, S = 1, P = 0

Output $H_2 = \frac{9-3}{1} + 1 = 7$

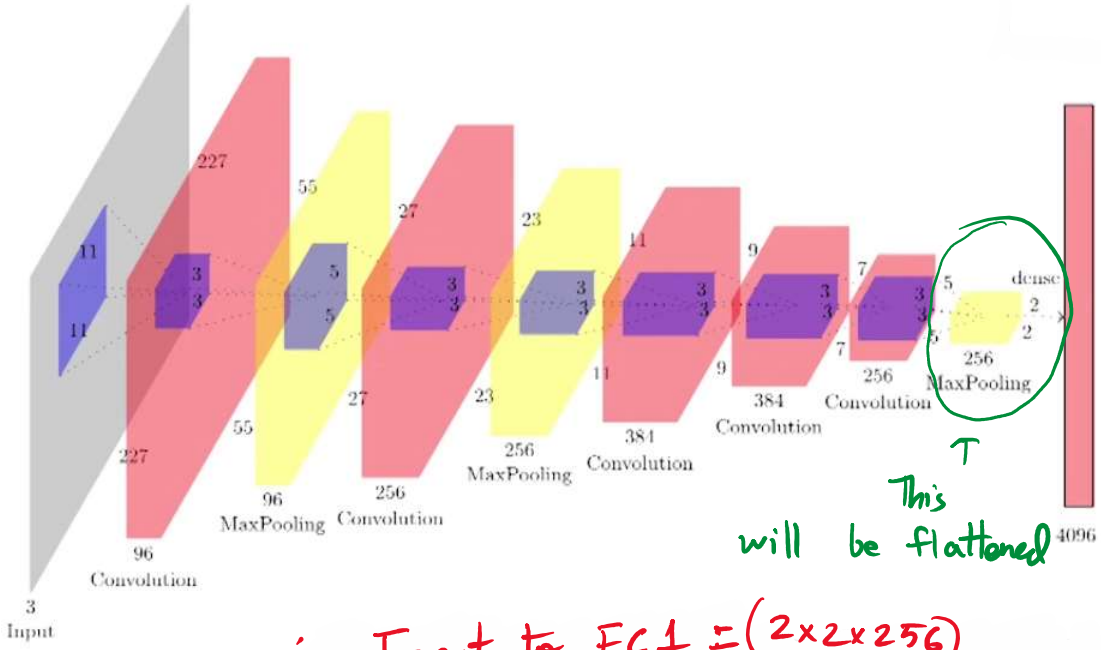
$W_2 = \frac{9-3}{1} + 1 = 7$

$D_2 = 384$

Parameters = $(3 \times 3 \times 384) \times 384$

≈ 1.372 Million

After applying a few more layers we get:

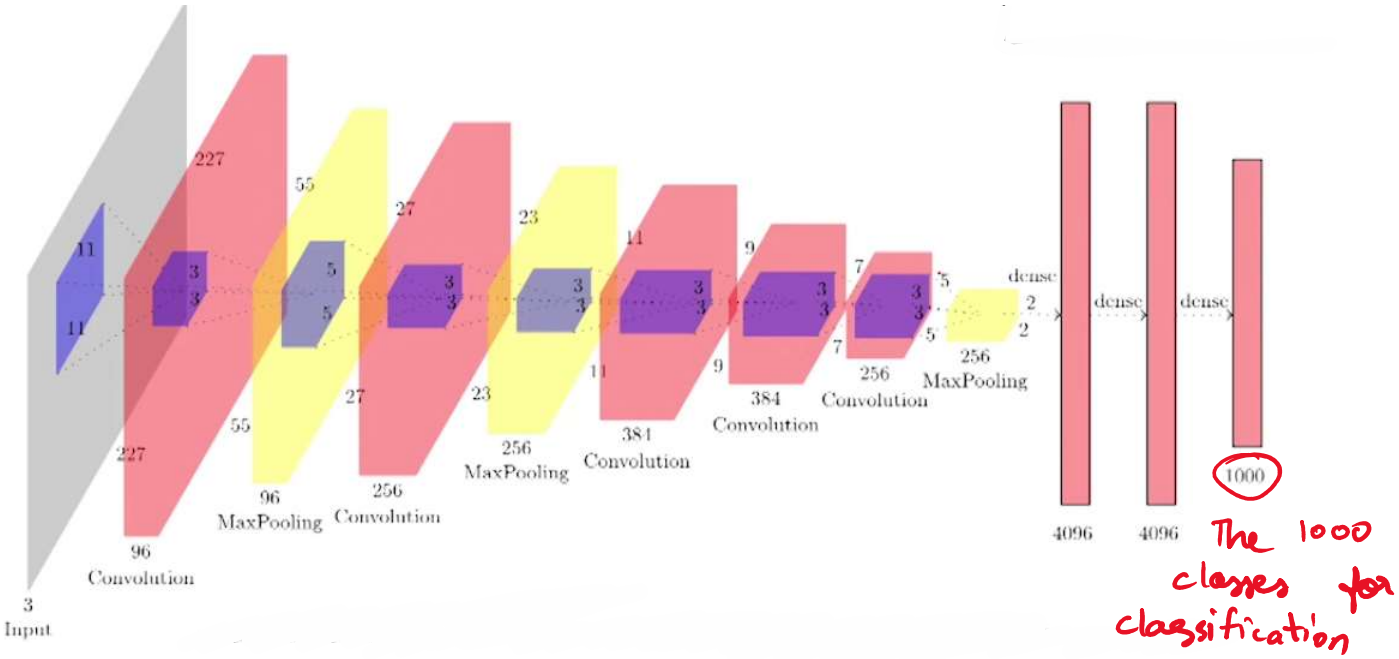


This will be flattened

∴ Input to FC1 = $(2 \times 2 \times 256)$

Parameters = $(2 * 2 \times 256) \times 4096 \approx 4$ Million

After 2 more Fully connected layers:



Hyperparameter : Kernel size
No. of filters

When we count the no. of layers, we only count the ones w/ parameter. So pooling layers are not counted.

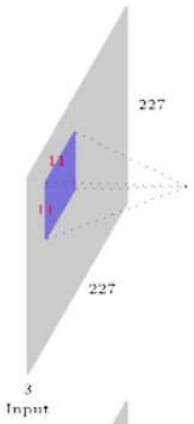
∴ No. of layers = 8
5 convolutions & 3 connected

Total no. of parameters = 27.55 Million

ZFNet

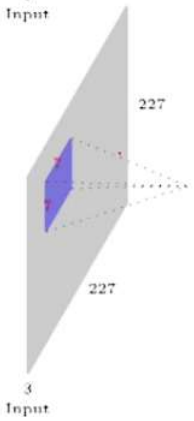
Comparing AlexNet w/ ZFNet.

AlexNet

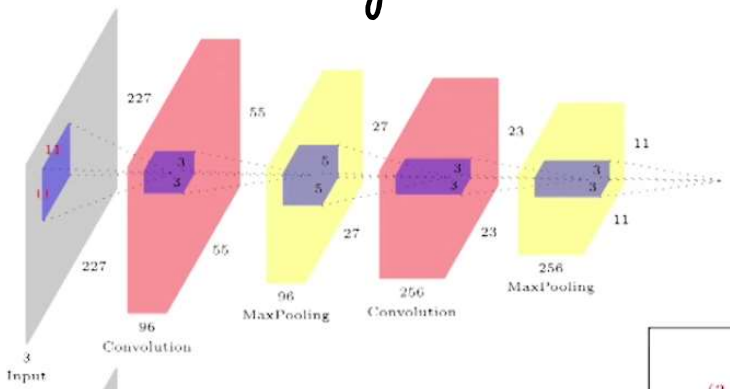


Layer1: $F = 11 \rightarrow 7$
Difference in Parameters
$((11^2 - 7^2) \times 3) \times 96 = 20.7K$

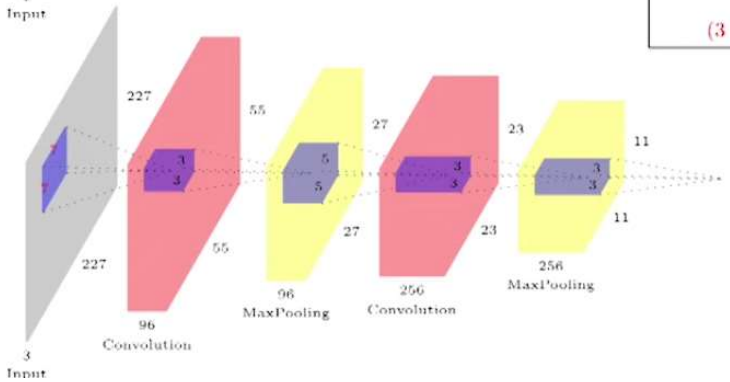
ZFNet

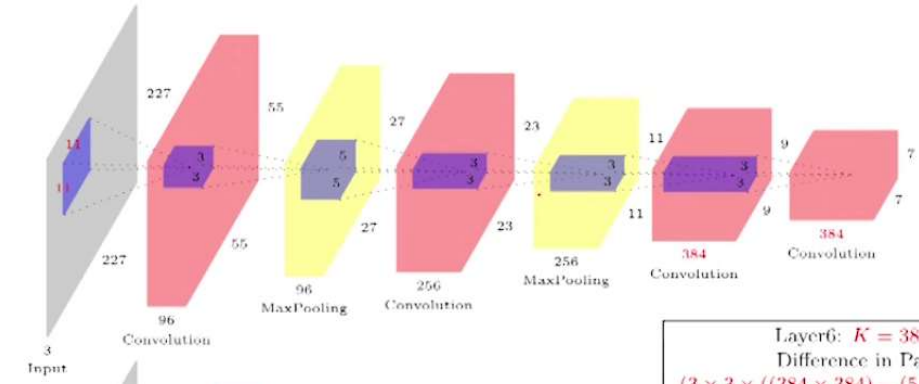


No changes till layer 5

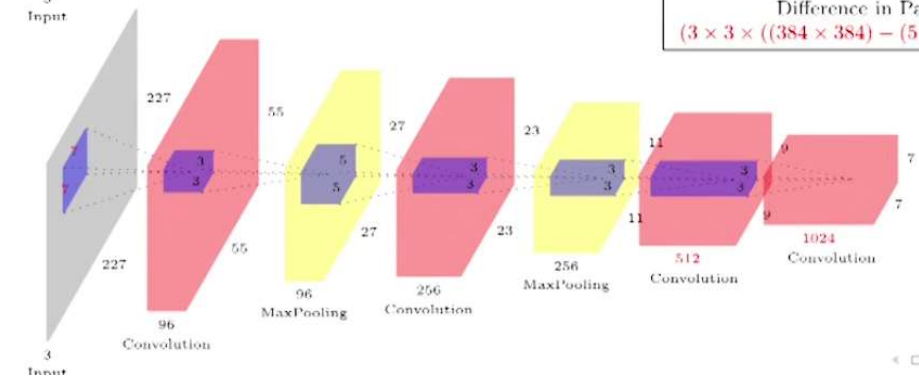


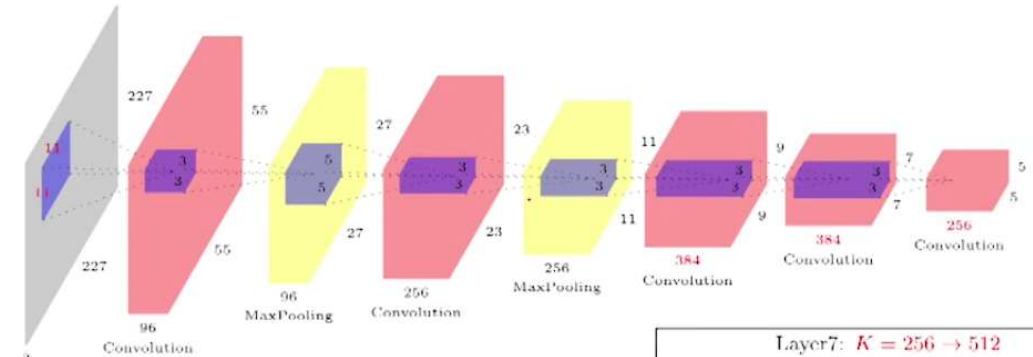
Layer5: $K = 384 \rightarrow 512$
Difference in Parameters
$(3 \times 3 \times 256) \times (512 - 384) = 0.29M$



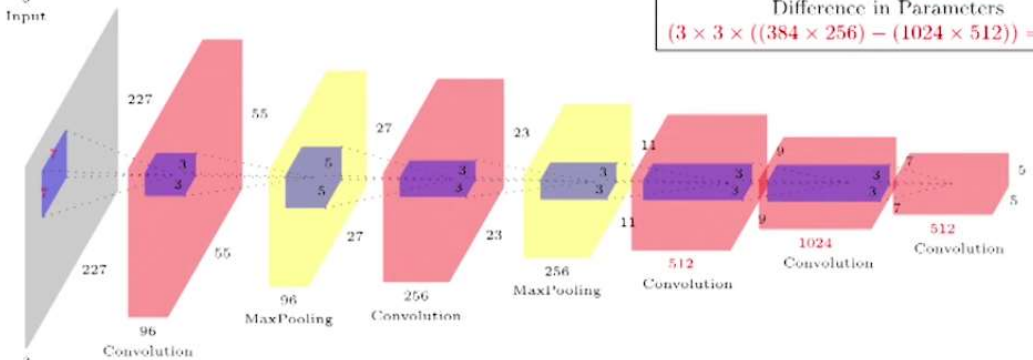


Layer6: $K = 384 \rightarrow 1024$
Difference in Parameters
$(3 \times 3 \times ((384 \times 384) - (512 \times 1024)) = 0.8M$

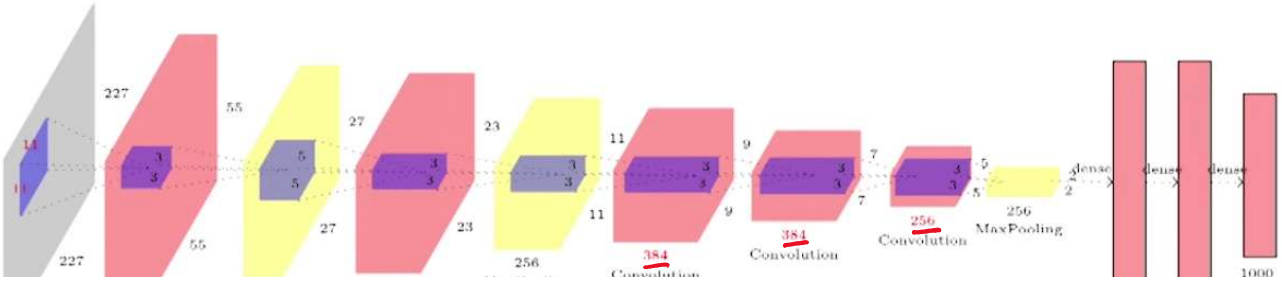


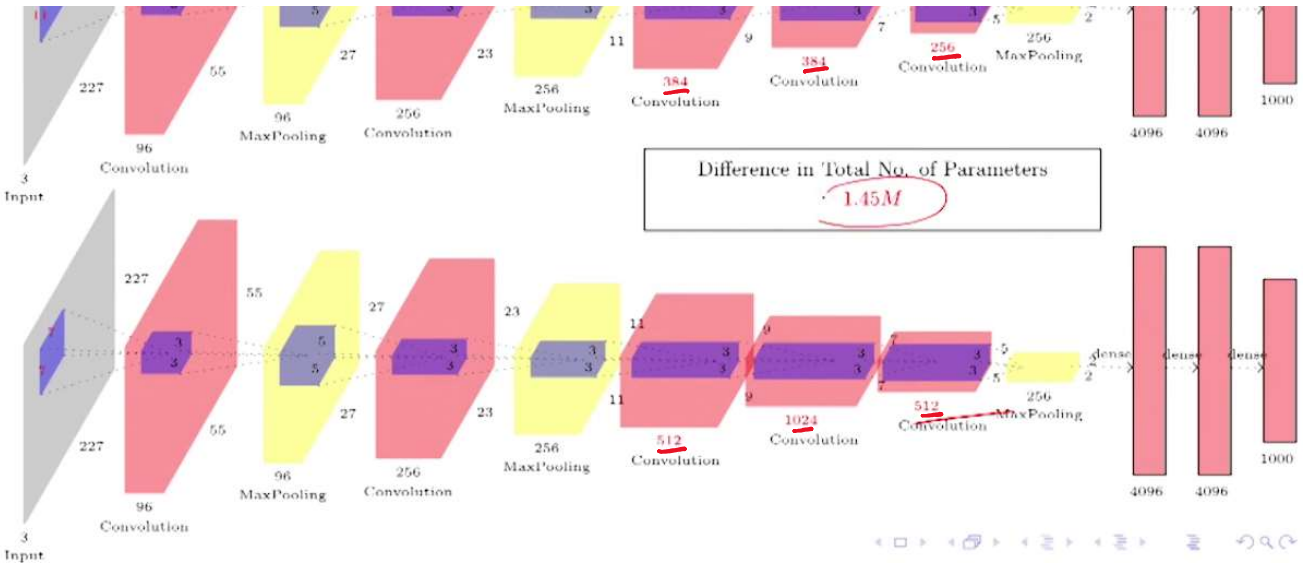


Layer7: $K = 256 \rightarrow 512$
Difference in Parameters
$(3 \times 3 \times ((384 \times 256) - (1024 \times 512)) = 0.36M$

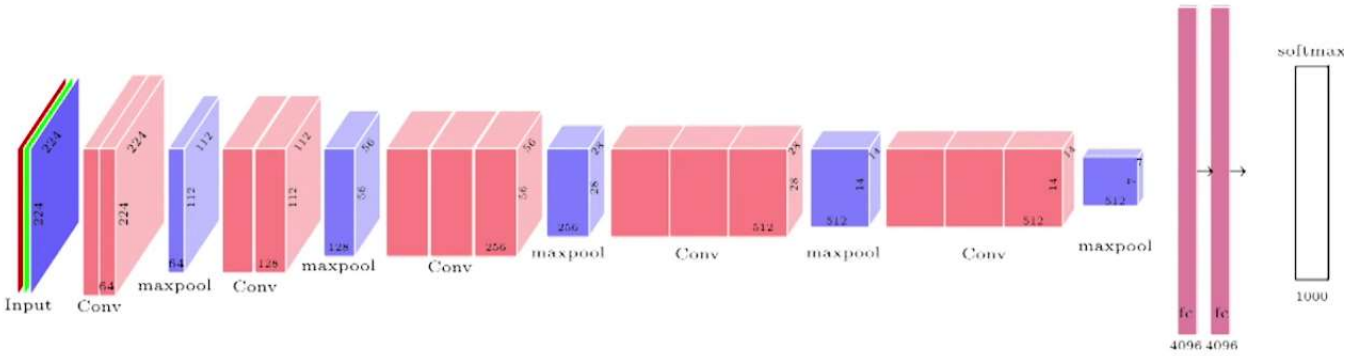


No differences in layer 8, 9, 10 then maxpooling then fully connected layer





VGGNet



Kernel size : 3x3 throughout

Total Parameters in non FC layers ~ 16 Million

Total no. of parameters in FC layers =

$(512 \times 7 \times 7 \times 4096) + (4096 \times 4096) + (4096 * 1024)$

$\sim$ 122 Million

Most no. of parameters are in first FC layer (~102 M)

GoogleNet & RESNet

Idea: Apply max pooling, & different convolution layers (1×1), (3×3) & (5×5) at the same time & concatenate the feature maps.

Problem: Large no. of computation

If P=0 & S=1 then convolving a W×H×D input with a F×F×D filter results in (W−F+1)(H−F+1) output

∴ Each element of the output requires O(F×F×D) computations

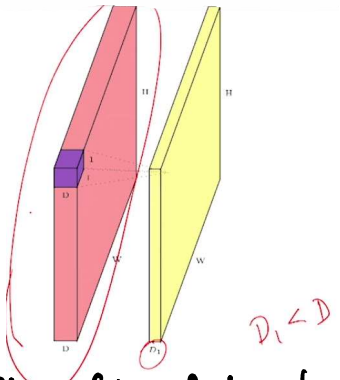
Solution: We can reduce no. of computations by using 1×1 computations

1×1 convolution aggregates along the depth

Convolving a D×W×H input with D_1 ($D_1 < D$) 1×1 filters will result in a D_1×W×H output (S=1, P=0)

$D_1 < D$ ∴ dimensions of input are reduced ⇒ less computations

Now we'll have O(F×F×D_1) complexity

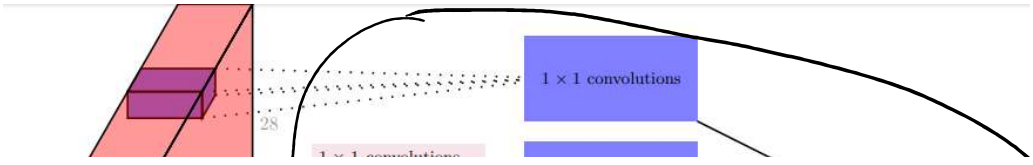


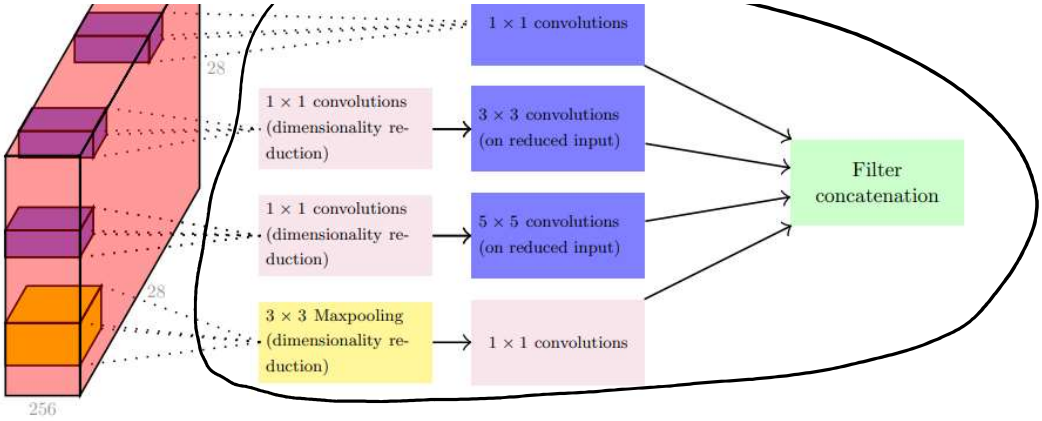
We can also use D_1 & D_2 1×1 filters before 3×3 & 5×5 filters respectively.

We can then add max pooling layer followed by dimensionality reduction.

And a new set of 1×1 convolution

Then we finally concaterate all these layers.

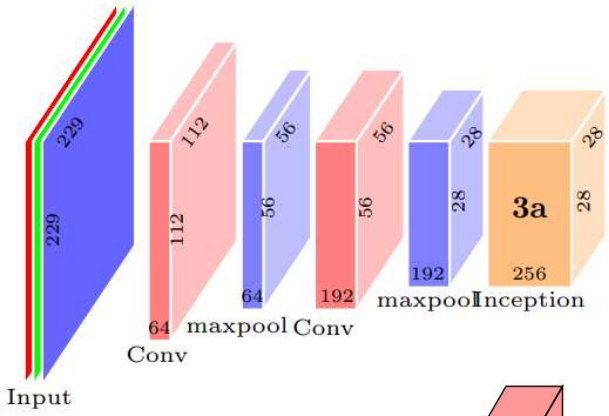


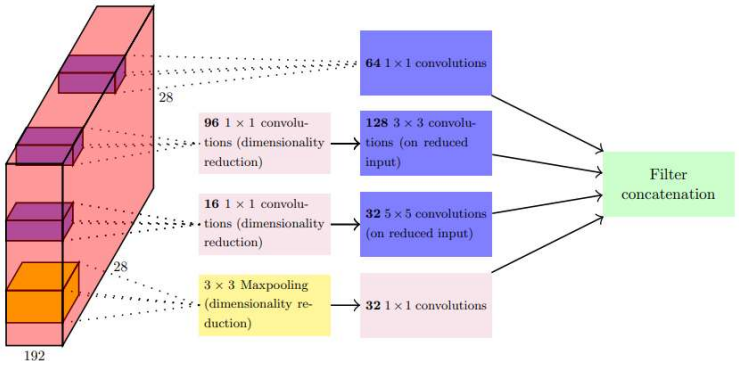


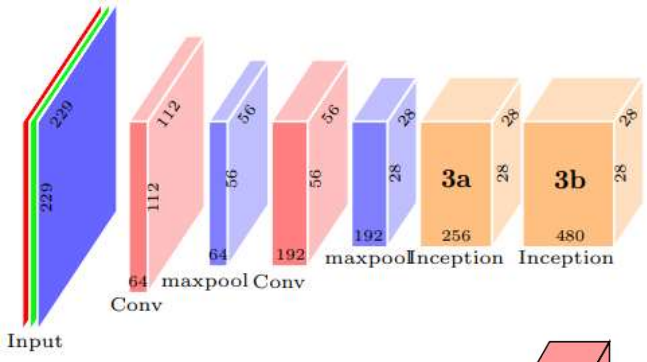
This is called the Inception Module

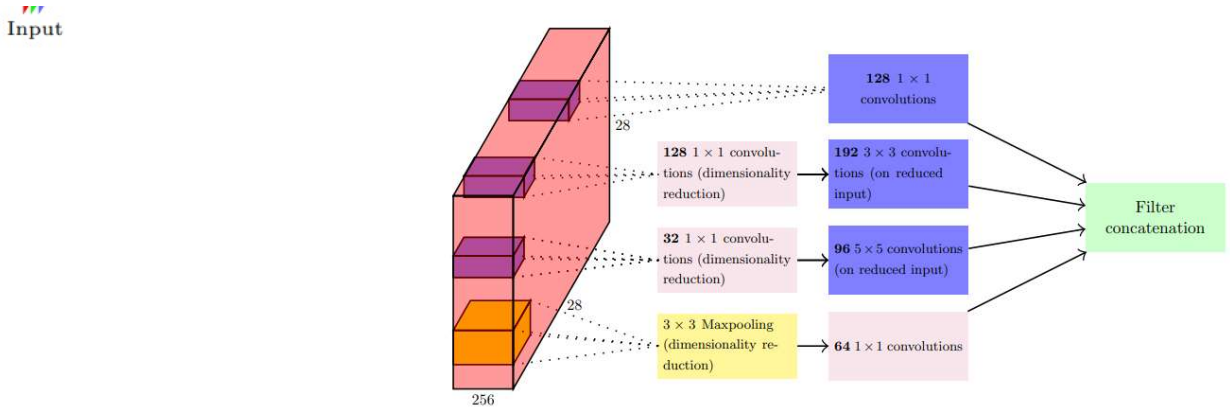
GoogleNet contains many Inception Modules

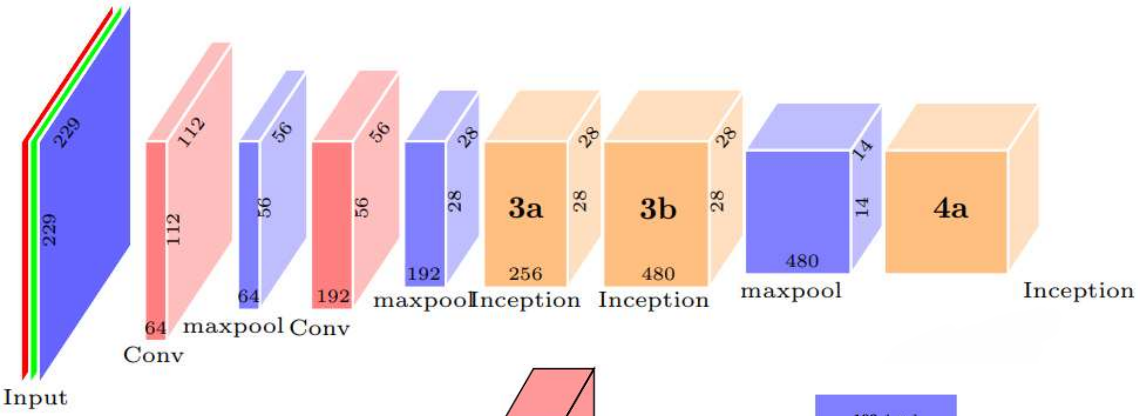
GoogleNet

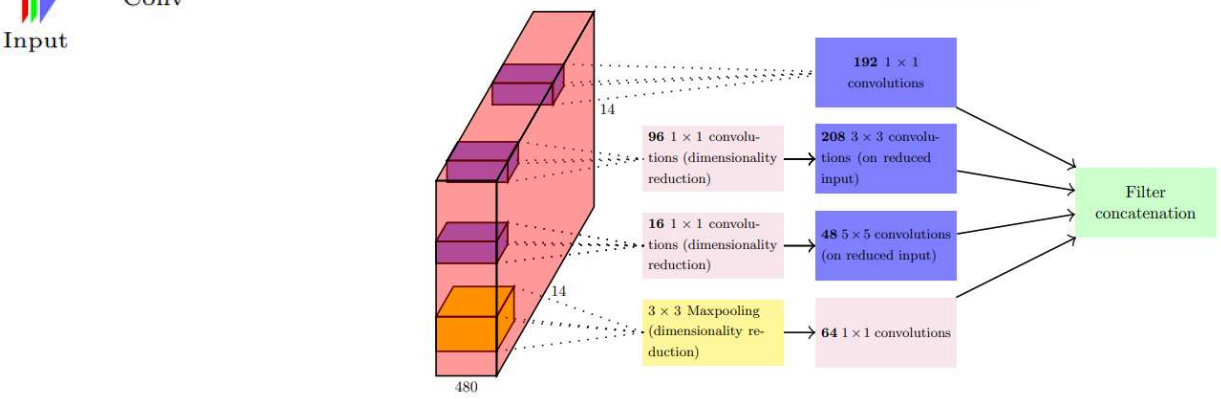


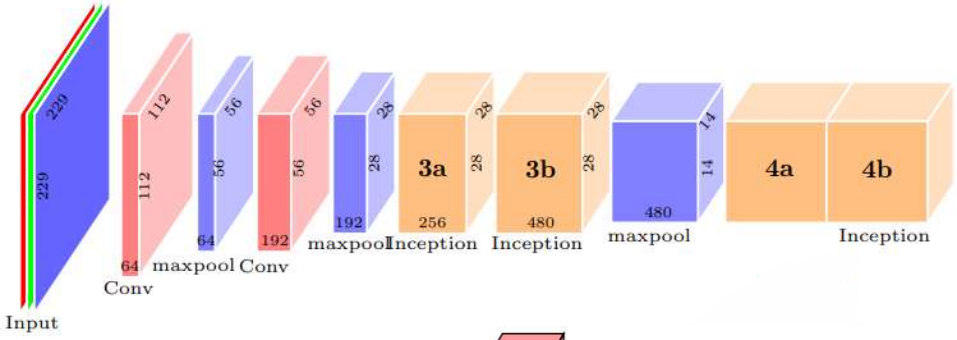


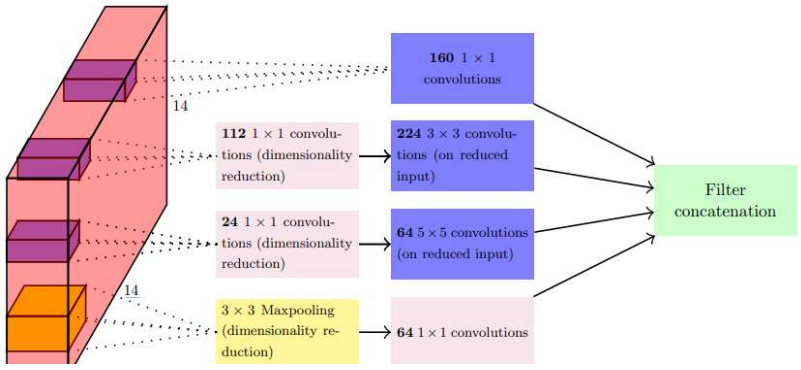


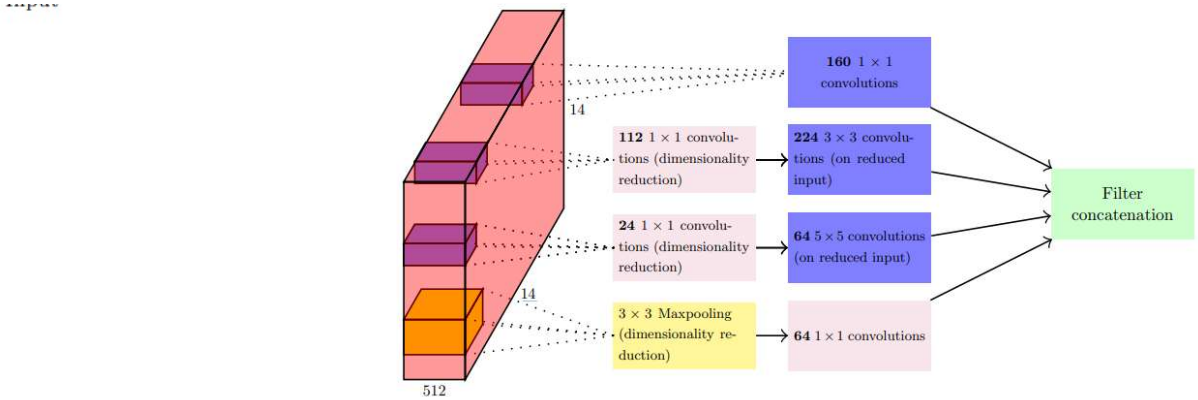

Input
28
256
28
128 1 × 1 convolutions
128 1 × 1 convolutions (dimensionality reduction)
192 3 × 3 convolutions (on reduced input)
32 1 × 1 convolutions (dimensionality reduction)
96 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
64 1 × 1 convolutions
Filter concatenation

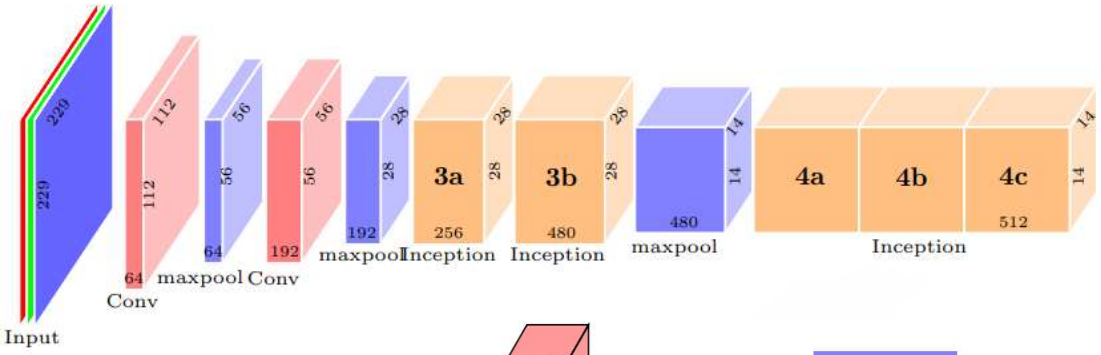

Input
229
229
Conv
64
112
112
maxpool
64
56
56
Conv
192
56
56
maxpool
192
28
28
Inception
3a
256
28
28
Inception
3b
480
28
28
maxpool
480
14
14
Inception
4a

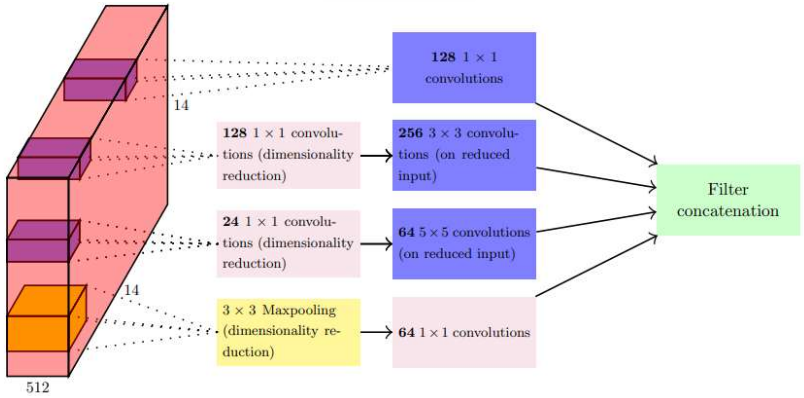

14
480
14
192 1 × 1 convolutions
96 1 × 1 convolutions (dimensionality reduction)
208 3 × 3 convolutions (on reduced input)
16 1 × 1 convolutions (dimensionality reduction)
48 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
64 1 × 1 convolutions
Filter concatenation

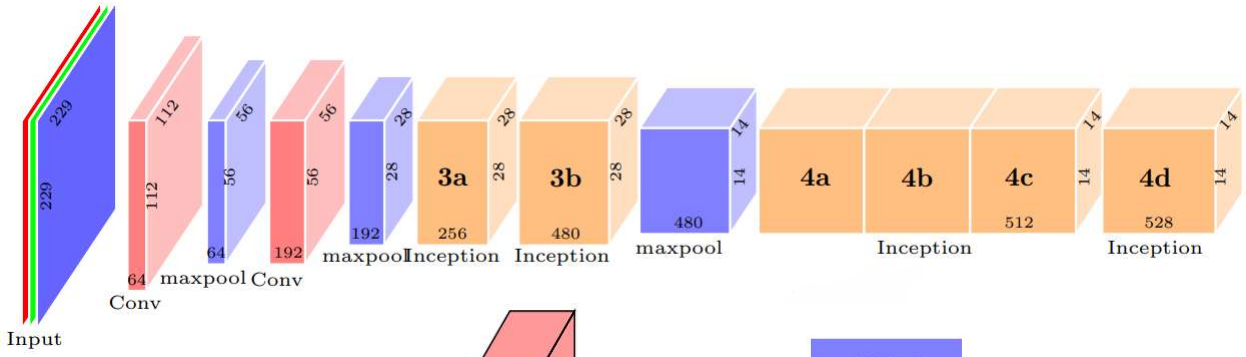

Input
229
229
Conv
64
112
112
maxpool
64
56
56
Conv
192
56
56
maxpool
192
28
28
Inception
3a
256
28
28
Inception
3b
480
28
28
maxpool
480
14
14
Inception
4a
4b

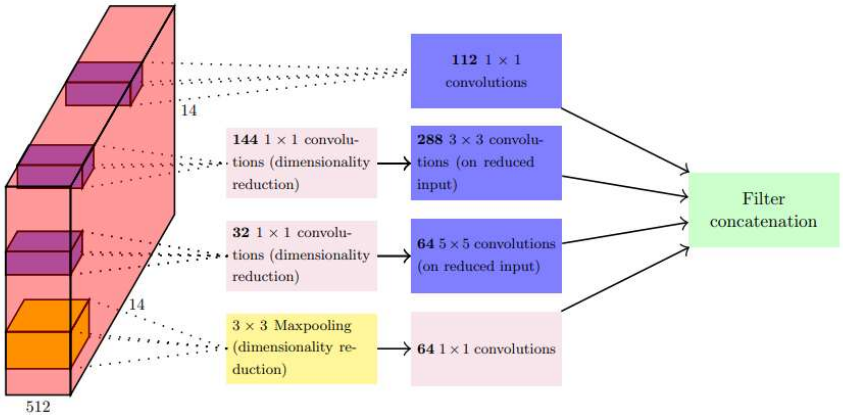

14
14
14
160 1 × 1 convolutions
112 1 × 1 convolutions (dimensionality reduction)
224 3 × 3 convolutions (on reduced input)
24 1 × 1 convolutions (dimensionality reduction)
64 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
64 1 × 1 convolutions
Filter concatenation

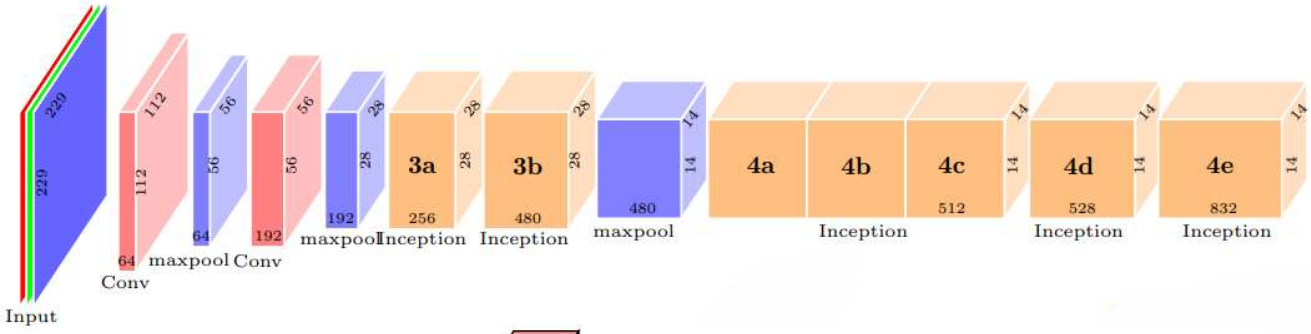

160 1 × 1 convolutions
112 1 × 1 convolutions (dimensionality reduction)
224 3 × 3 convolutions (on reduced input)
24 1 × 1 convolutions (dimensionality reduction)
64 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
64 1 × 1 convolutions
Filter concatenation
14
14
512

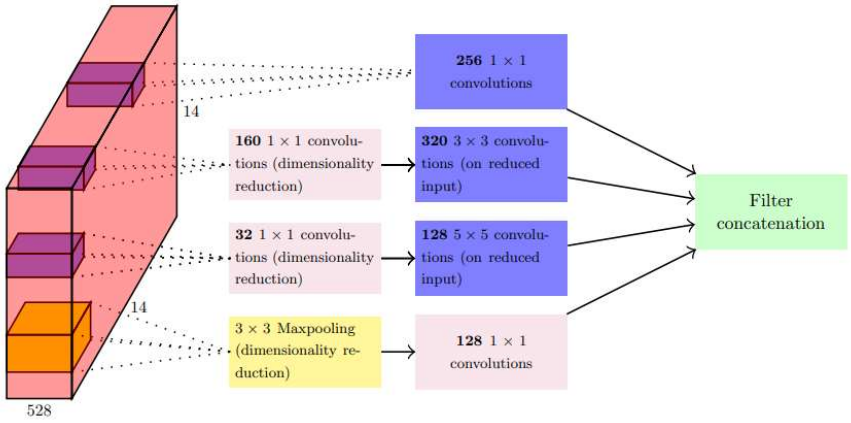

Input
Conv
maxpool
Conv
maxpool
Inception
Inception
maxpool
Inception
3a
3b
4a
4b
4c

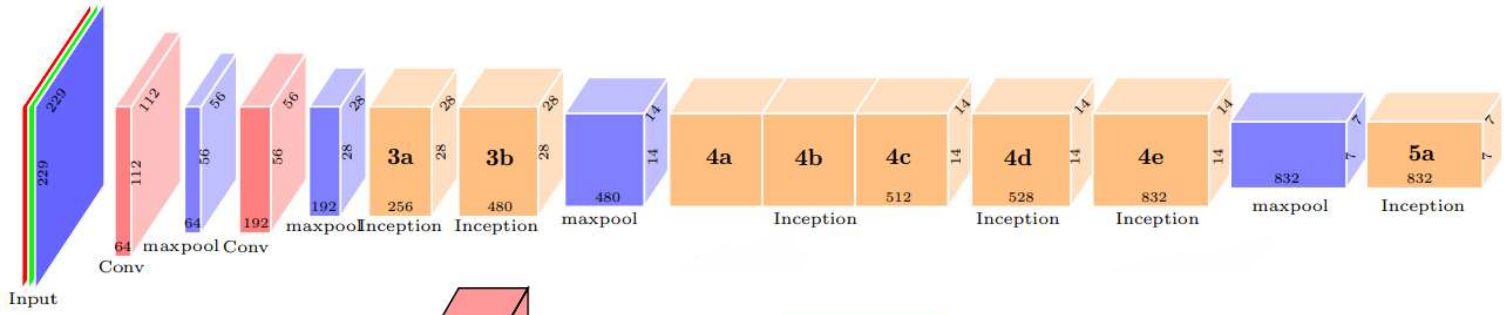

128 1 × 1 convolutions
128 1 × 1 convolutions (dimensionality reduction)
256 3 × 3 convolutions (on reduced input)
24 1 × 1 convolutions (dimensionality reduction)
64 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
64 1 × 1 convolutions
Filter concatenation
14
14
512

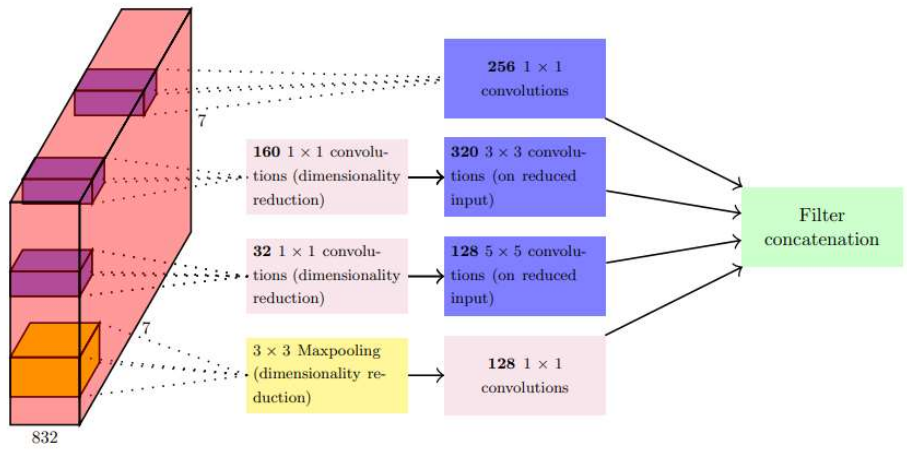

Input
Conv
maxpool
Conv
maxpool
Inception
Inception
maxpool
Inception
Inception
3a
3b
4a
4b
4c
4d

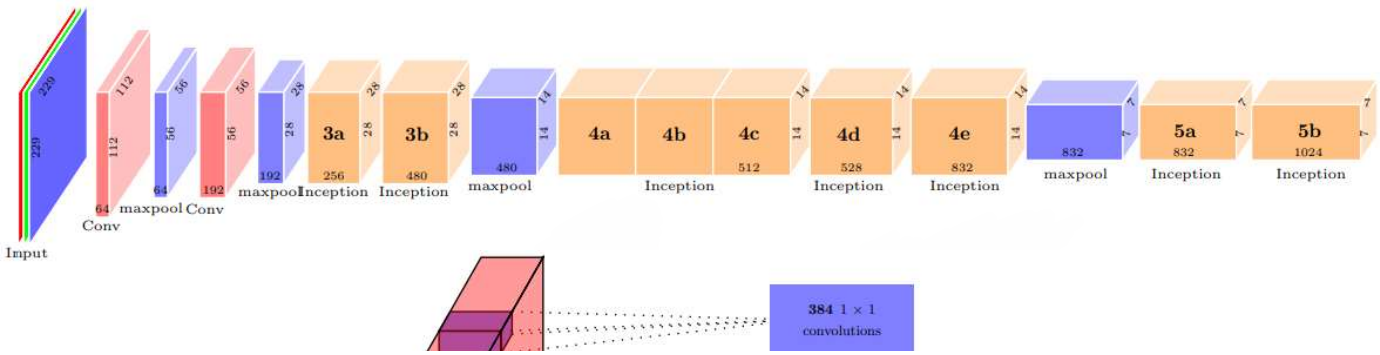

112 1 × 1 convolutions
144 1 × 1 convolutions (dimensionality reduction)
288 3 × 3 convolutions (on reduced input)
32 1 × 1 convolutions (dimensionality reduction)
64 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
64 1 × 1 convolutions
Filter concatenation
14
14
512

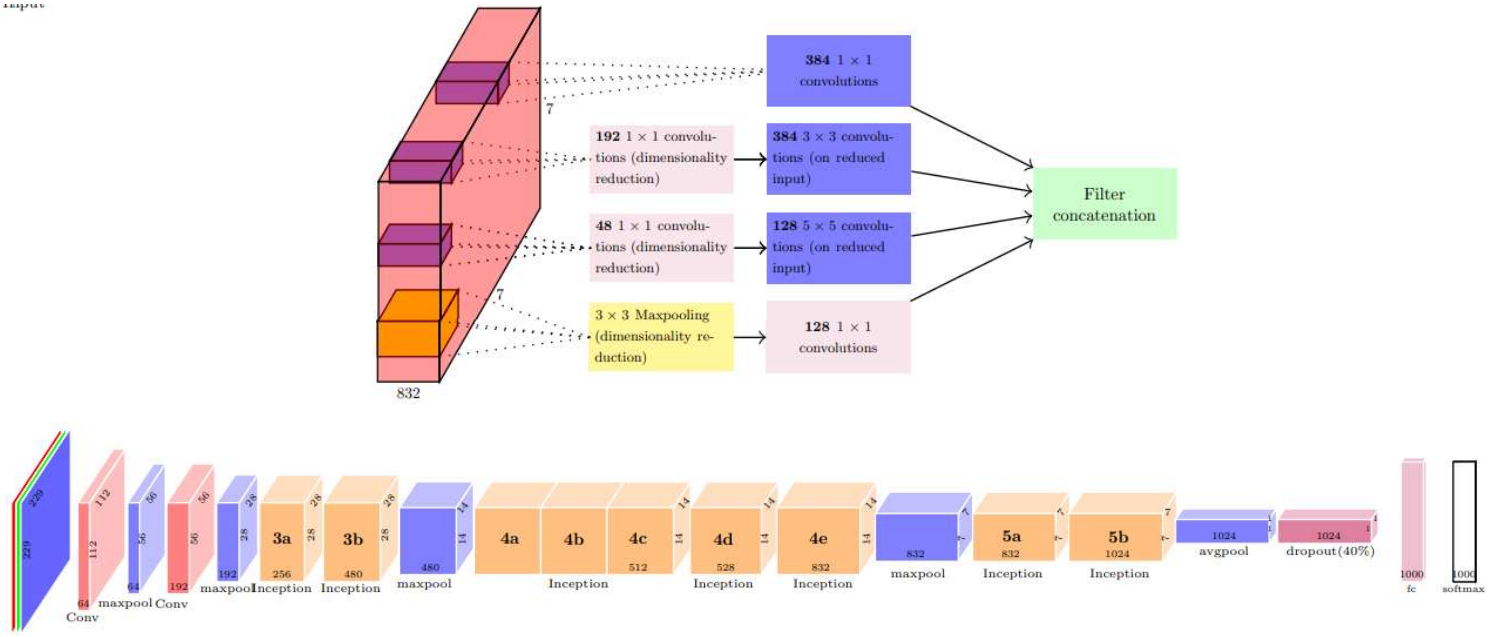

Input
Conv
maxpool
Conv
maxpool
Inception
Inception
maxpool
Inception
Inception
Inception
3a
3b
4a
4b
4c
4d
4e


256 1 × 1 convolutions
160 1 × 1 convolutions (dimensionality reduction)
320 3 × 3 convolutions (on reduced input)
32 1 × 1 convolutions (dimensionality reduction)
128 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
128 1 × 1 convolutions
Filter concatenation
14
14
528


Input
Conv
maxpool
Conv
maxpool
Inception
Inception
maxpool
Inception
Inception
Inception
maxpool
Inception
3a
3b
4a
4b
4c
4d
4e
5a


256 1 × 1 convolutions
160 1 × 1 convolutions (dimensionality reduction)
320 3 × 3 convolutions (on reduced input)
32 1 × 1 convolutions (dimensionality reduction)
128 5 × 5 convolutions (on reduced input)
3 × 3 Maxpooling (dimensionality reduction)
128 1 × 1 convolutions
Filter concatenation
7
7
832

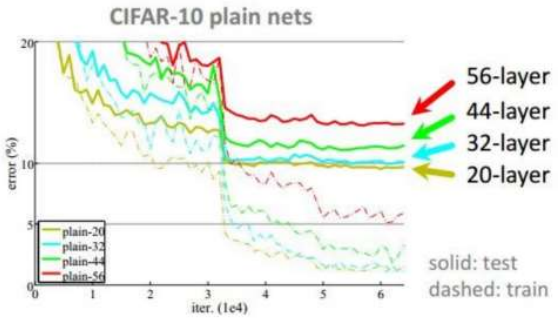

Input
Conv
maxpool
Conv
maxpool
Inception
Inception
maxpool
Inception
Inception
Inception
maxpool
Inception
Inception
3a
3b
4a
4b
4c
4d
4e
5a
5b
384 1 × 1 convolutions



This has 12 times less parameters than AlexNet
2 times more computations

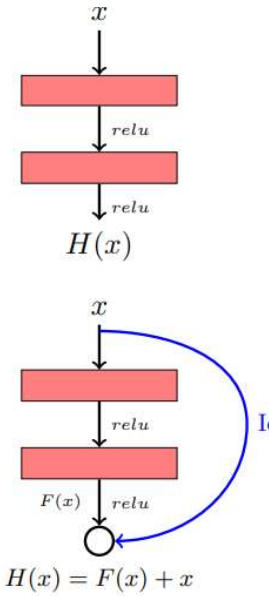
ResNet

- Suppose we have been able to train a shallow neural network well
- Now suppose we construct a deeper neural network which has a few more layers
  - Now if the shallow network works well then the deep network should also work well by simply learning to compute identity functions in the new layers
  - ∴ the solution space of shallow neural network is a subset of the solution space of deep neural network.
  - However this isn't what was observed in practice.



More layers → more error

- Consider any two stacked layers in CNN
- The layers are learning some function of the input
- What if we enable it to learn only a residual function of the input



This idea was highly successful.

We have ensured that the transformations have been learned & x the original input is also fed at every reasonable stage.

Maybe after 2/3/5 layers the x is fed. This is known as skip connections.

ResNet, 152 layers

Bag of tricks

- Batch Normalizaton after every CONV layer
- Xavier/2 initialization from [He et al]
- SGD + Momentum(0.9)
- Learning rate:0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

Visualising Patches which maximally activate a neuron

- Consider some neurons in a given layer of a CNN.
- We wanna find out, for which given input, will the neuron fire. eg- if many classes of images are fed as input, which class will the neuron fire for.
- Also, we can trace back to the patch in the image which causes the neurons to fire.
- We will now look at the results of one such experiment by Grishick et al, 2014

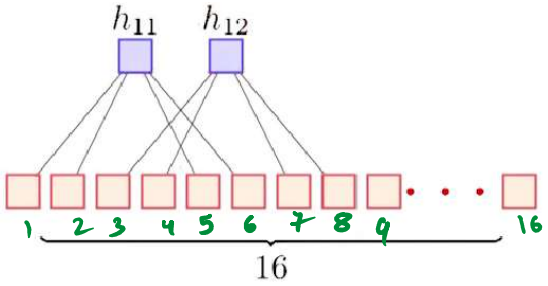
Experiment

- 6 neurons in pool5 layer are considered & they find the image patches which causes these neurons to fire.
- They found a set of neurons which fire for:
  - people's faces
  - dog faces
  - flowers
  - numbers
  - houses
  - shiny surface

Visualising Filters of a CNN

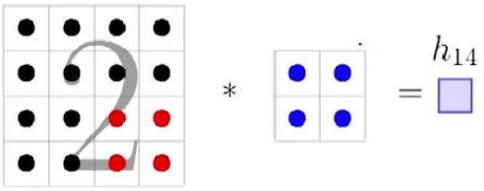
We are interested in finding an input which maximally excites a neuron & causes it to fire

We can see CNN as a fed forward neural network w/ sparse connections & weight sharing



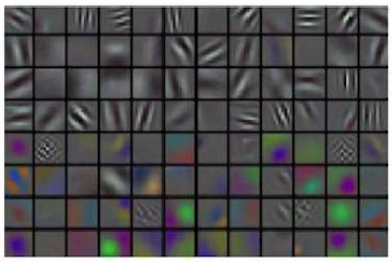
We are asking what should be patches 1,2, 5,6, for h_{11} to fire and 3,4, 7,8 for h_{12} to fire

The solution would be the $\frac{w}{\|w\|}$ where w is the filter (2x2) in this case



We can simply plot the K×K weights (filters) as images & visualize them as patterns.

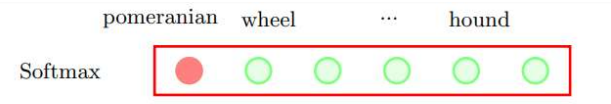
This is only interpretable for the first convolutional layer



$$\max_x \{w^T x\}$$
$$s.t. \quad ||x||^2 = x^T x = 1$$
$$\text{Solution:} \quad x = \frac{w_1}{\sqrt{w_1^T w_1}}$$

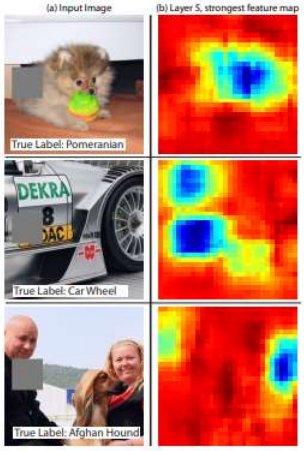
Occlusion Experiments

- We are interested in understanding which portions of the image are responsible for maximizing the prob. of a certain class.
- We have an image of a dog & I wanna know what patch in the image resulted in the class being predicted as pomerian. Basically, what influenced the output.



- We do this by occluding (graying out) certain patches of image & seeing if there's a drop in the probability of predicted class.

- We will now create a heatmap. The red portions are patches which do not cause a relatively large drop in predicted probability. Blue portions do cause a large drop.

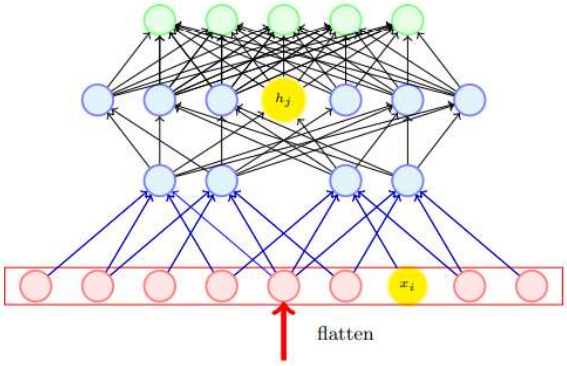


Finding influence of input pixels using backpropagation

We can think of an image as $m \times n$ inputs

$x_0, x_1, \cdots, x_{m \times n}$

We are interested in finding the influence of each of these inputs (x_i) on a given neuron (h_j)



How?

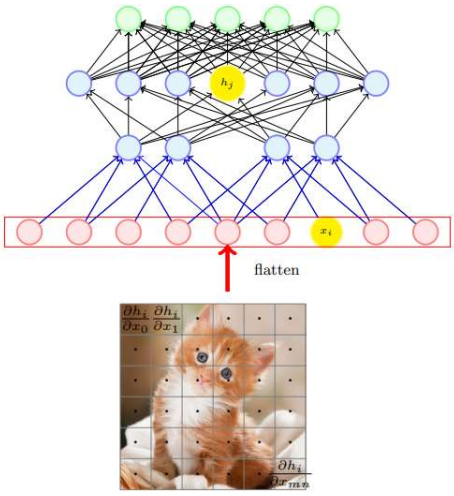
Using a superhero called

GRADIENTS

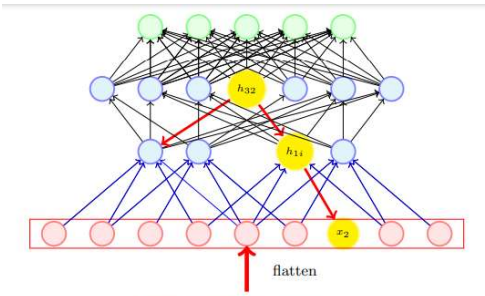
If a small change in x_i causes a large change in h_j then we can say x_i has a lot of influence on h_j.

$$\frac{\partial h_j}{\partial x_i}$$

- We can compute these partial derivatives wrt all inputs & then visualize this gradient matrix as an image itself.



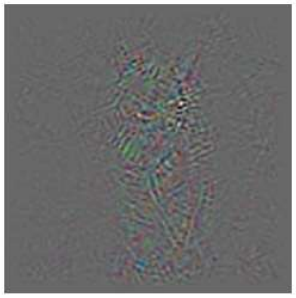
- We can compute these CNNs as a feedforward neural network
- Then do back propagation
- We already know how to do backpropagation till first hidden layer



$$\frac{\partial h_{32}}{\partial x_2} = \sum_{i=1}^{3} \frac{\partial h_{32}}{\partial h_{1i}} \frac{\partial h_{1i}}{\partial x_2}$$

$$h_{1i} = \sum_{j=1}^{4} w_{ji} x_j$$

$$\frac{\partial h_{1i}}{\partial x_2} = w_{12}$$



This will be our output if we plot the gradients as image.

Reason ?

Most pixels have low influence

∴ their gradients → 0

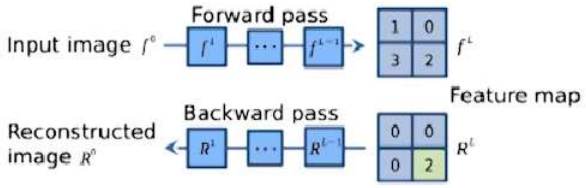
& 0 → grey

Although, it seems like there isn't much which has any good influence. Features like ears, nose, eyes etc. should've had big influence ∴ larger gradients

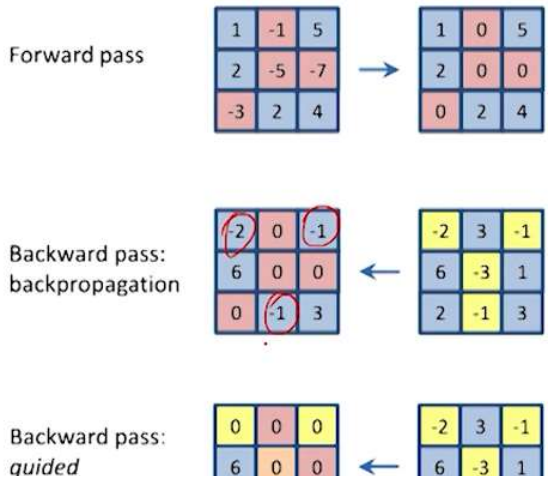
We can try something called 'Guided Backpropagation' which gives a better idea about the influences.

Guided Backpropagation

- We feed an input to the CNN and do a forward pass
- We consider one neuron in some feature map at some layer.
- We are interested in finding the influence of input on this neuron
- We retain this neuron & set all other neurons in the layer to zero.
- We will now backpropagate all the way to the inputs.



- During forward pass ReLU activation allows only positive values to pass & clamps the negative values to zero.
- Similarly during backward pass no gradient passes through the dead ReLU neurons
- In guided back propagation, any negative gradients flowing from the upper layer are set to 0.



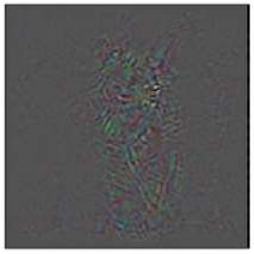
Backward pass: *guided backpropagation*

$$\begin{bmatrix} 0 & 0 & 0 \\ 6 & 0 & 0 \\ 0 & 0 & 3 \end{bmatrix} \leftarrow \begin{bmatrix} -2 & 3 & -1 \\ 6 & -3 & 1 \\ 2 & -1 & 3 \end{bmatrix}$$

Coming back to our cat example,

Intuition: Neglect all negative influences (gradients) & focus only on the positive ones.

This gives a better picture of the true influence of input.



Backpropagation

Guided Backpropagation

Optimization over Images

- Suppose we have a blank slate containing some pixels & we wanna modify the pixels such that in the CNN, dumbell class gets fired (if we pass it through a CNN, the prob. of the class dumbbell should be maximum)
- We could pose this as an optimization problem wrt $I(i_0, i_1, \ldots i_{mn})$

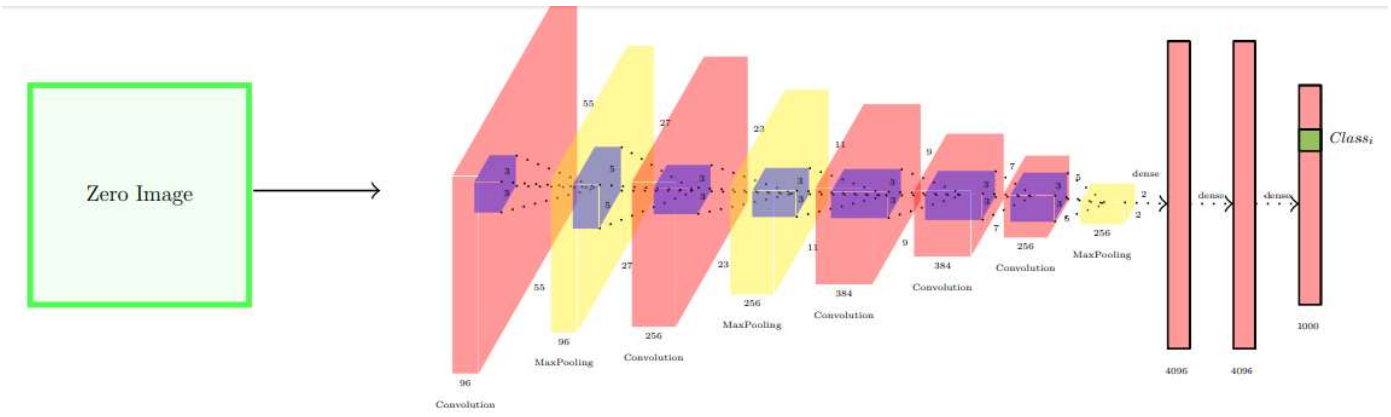
$$\arg\max_{I} \left(S_c(I) - \lambda \Omega(I)\right)$$

$S_c(I)$ = Score for class I before softmax

$\Omega(I)$ = Some regularizer to ensure I looks like an image

I = input pixels

- We will now essentially think of all images as parameters
- Keep the weights of neural network fixed
- And adjust the parameters (pixels) of the images so that the score of our desired class is maximised.



Algorithm:

- Start with zero image
- Set score vector to $[0, 0, \cdots, 1, 0, 0]$
- Compute gradient $\frac{\partial S_c(I)}{\partial i_k}$
- Now update the pixel $i_k = i_k - \eta \frac{\partial S_c(I)}{\partial i_k}$
- Again do a forward pass through the network
- Go to step 2

Create Images from Embeddings

- We can think of fc7 layer as some kind of embedding for the image.
- A good embedding is one from which we can reconstruct the original image.
- Given this embedding, can we reconstruct the original image?
- Problem statement:
  Given an image, we want to create embeddings of that image. Then, we modify a blank image such that it's embeddings will look same as the original embedding.

Let ϕ_0 : embedding of original image

X: random image (say zero image)

Repeat: Forward pass using X & compute $\phi(x)$

Compute: $L(i) = \|\phi(x) - \phi_0\|^2 + \lambda \|\phi(x)\|_6^6$

Update rule: $i_k = i_k - \eta \frac{L(i)}{\partial i_k}$

Original Image

Conv-1

Relu-1

Mpool-1

Norm-1

Norm-1

Relu-2

Relu-2

Norm-2

Conv-3

Relu-3

Conv-4

Relu-4

Conv-5

Relu-5

Mpool-5

Moral of the story, as we progress through layers, the embeddings will keep on losing features & will be far from the original layer!

Deep Dream

So far we have seen that if we start from a blank image, then we could suitably modify it by constructing an optimization problem whose parameters are the pixels of the image & we can modify the image so that it starts looking like a certain class of image.

Now suppose we start with a natural image instead of a blank image. Then we focus on some layer & check the activations of the neurons. Then we wanna change the image such that these neurons fire even more.

Suppose we want to boost the activation of neuron h_{ij}

$$\max_{I} L(I) = h_{ij}^2$$

parameters -> I -> image pixels.

Consider a pixel i_{mn} in the image,

$$\frac{\partial L(I)}{\partial i_{mn}} = \frac{\partial L(I)}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial i_{mn}}$$

Once the image is updated $\left(i_{mn} = i_{mn} + \frac{\partial L(I)}{\partial i_{mn}} \right)$ we feed it back to the network.

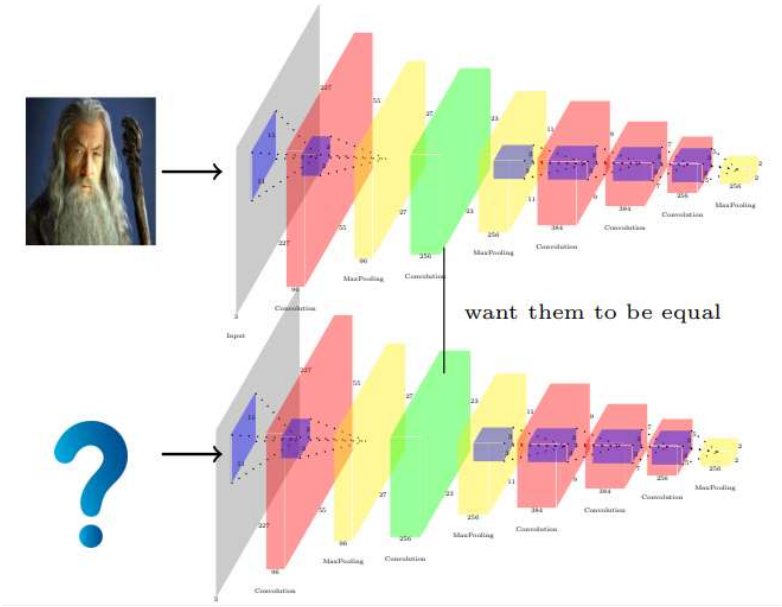
This will result in the target neuron fire even more & make the image more & more like the patterns which caused the neuron to fire.

Deep Art

To design a network which can do these, we first define two quantities:

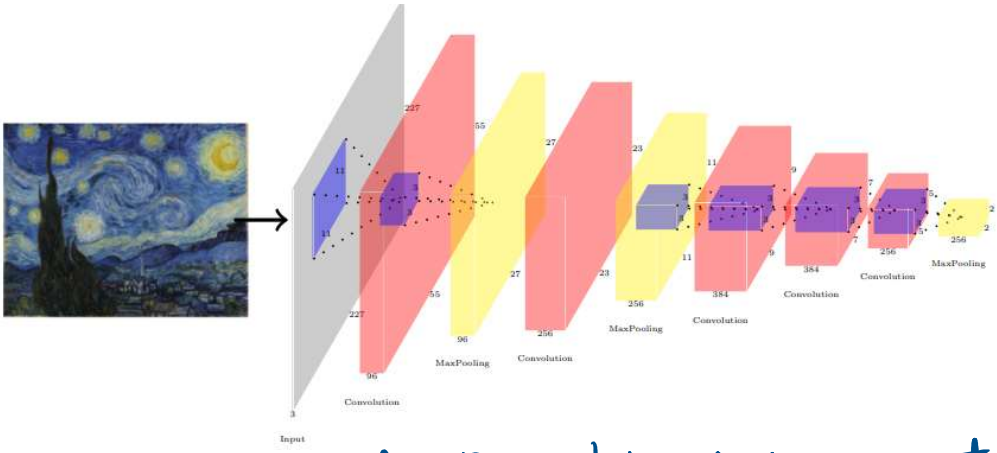
Conten targets: The activations of all layers for the given content image.

Ideally, we would want the new image to be such that its activations are also close to those of the originel image.



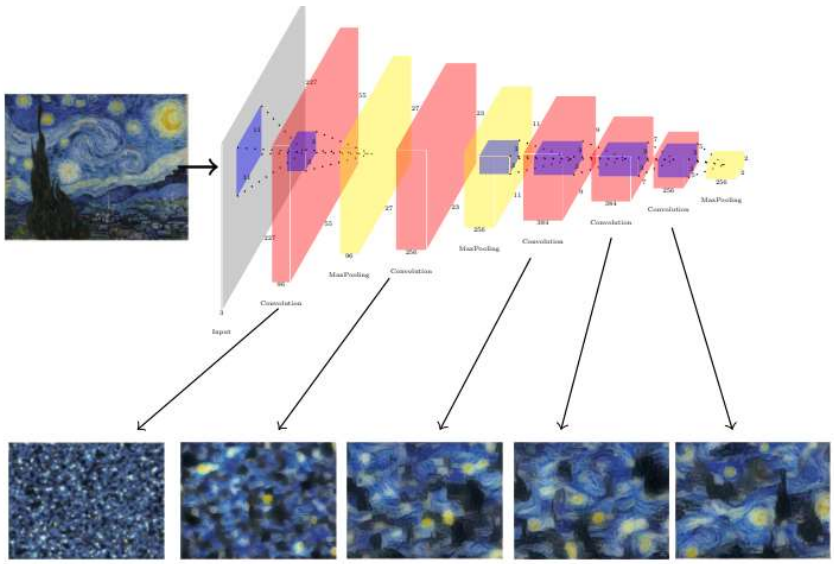
Let $\vec{p}, \vec{x}$ be the activations of the content image & the new image (to be generated) respectively.

$$L_{content}(\vec{p}, \vec{x}) = \sum_{ijk} (\vec{p}_{ijk} - \vec{x}_{ijk})^2$$



Next, we want the style of the generated image be of the same style as the style image (above)

How to capture style? if $V \in \mathbb{R}^{64 \times (256 \times 256)}$ is the activation at a layer then $V^T V \in \mathbb{R}^{64 \times 64}$ captures the style of image. The deeper layers capture more of this style information



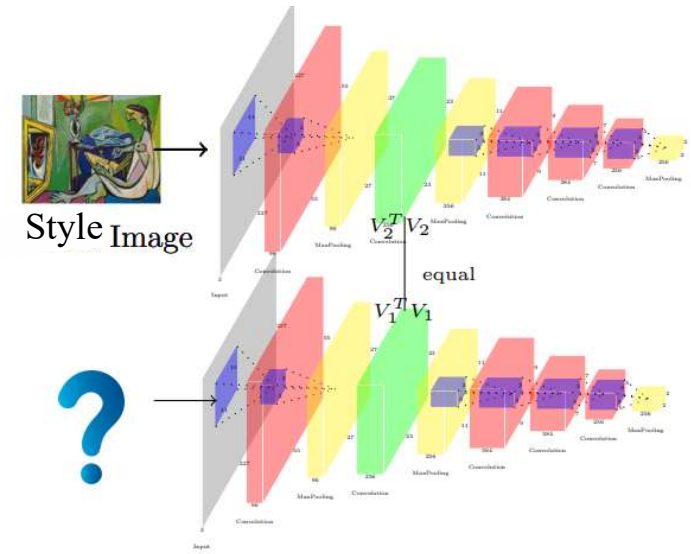
Objective function to ensure style of new image captured by layer l matches the style of the style image:

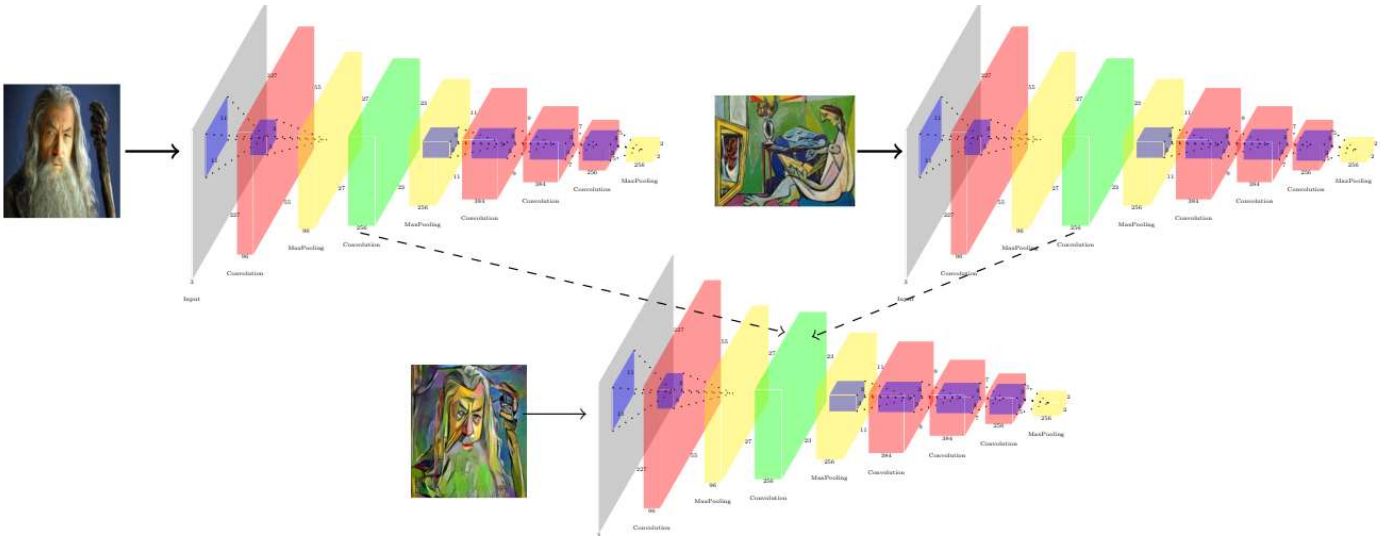
$$E_l = \sum_{ij} \left(G^l_{ij} - A^l_{ij}\right)^2$$

G^l & A^l → style gram matrices computed at layer l for style & new image respectively

G & H → style gram matrices ... l for style & new image respectively

$$L_{style}(\vec{a},\vec{x}) = \sum_{l=0}^{L} w_l E_l$$





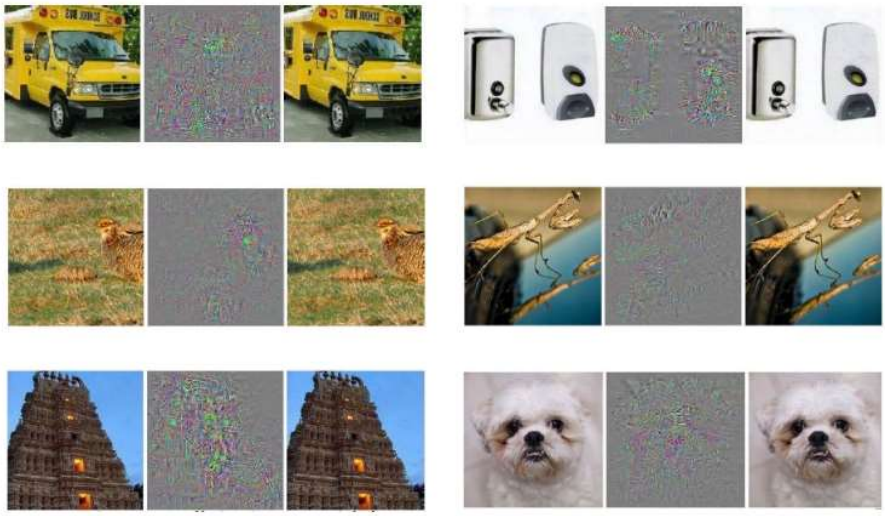
Total loss is given by:

$$L_{total}(\vec{p},\vec{a},\vec{x}) = \alpha L_{content}(\vec{p},\vec{x}) + \beta L_{style}(\vec{a},\vec{x})$$

Fooling Deep Convolutional Neural Networks

Suppose we fed an image to Convnet. Now, instead of maximizing the log likelihood of the correct class, we set the objective to maximize the log likelihood of the incorrect class.

We can do this w/ minimal changes to the image (using backdrop)



The changes are not visible to human.

The convnet that the third image is different & belongs to a different class.

Why does this happen?

Images are extremely high dimensional objects $\mathbb{R}^{227 \times 227}$

There are many many points in this high dimensional space.

Of these, only a few are images (of which we see during training). Using these training images we fit some decision boundaries. While doing so, we also end up taking decisions about the many many unseen points in this high dimensional space (notice the large green & red areas which

dimensional space (notice the large green & red areas which do not contain any training points)