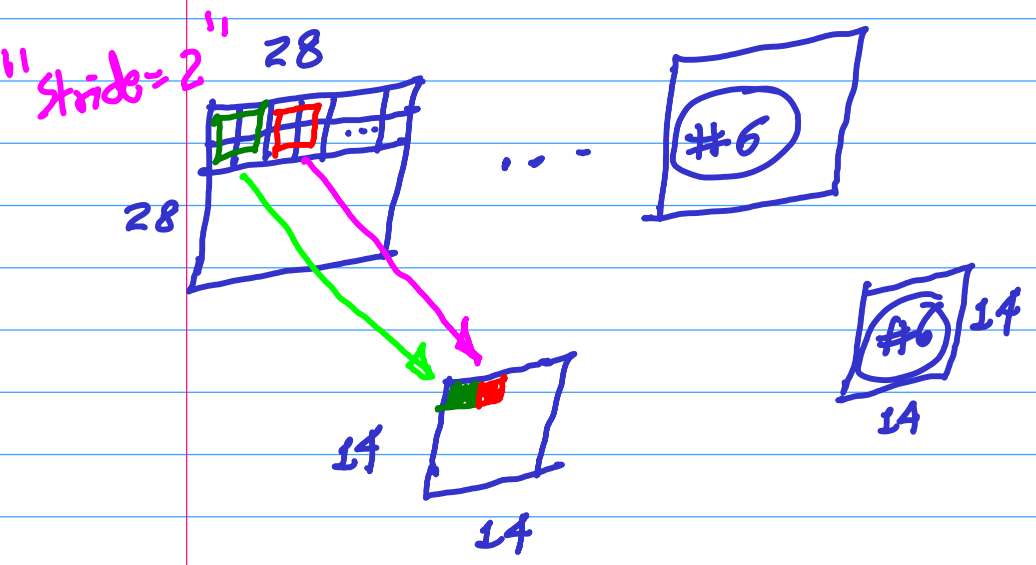
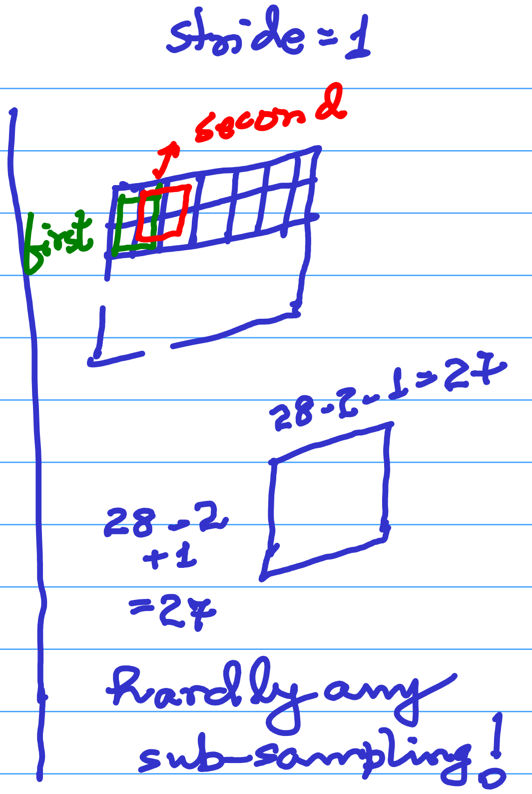
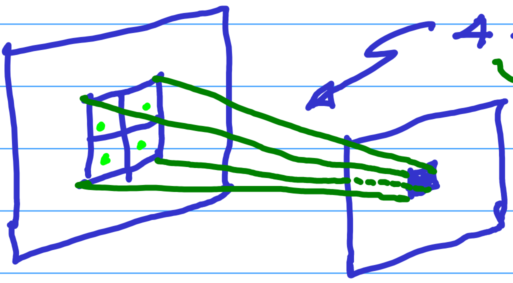
Layer #2 (S2) Subsampling





parameters & connections?



Parameters $6 \times (1 + 1 \text{ (bias)}) = 12$ parameters

Connections $6 \times (14 \times 14) \times (2 \times 2 + 1)$

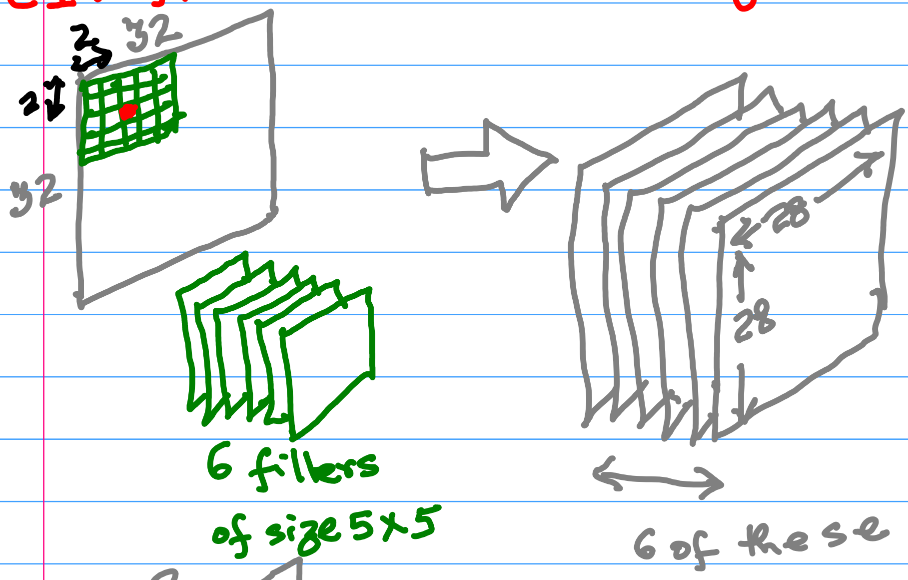
$= 6 \times 14 \times 14 \times 5 = 5,880$ connections

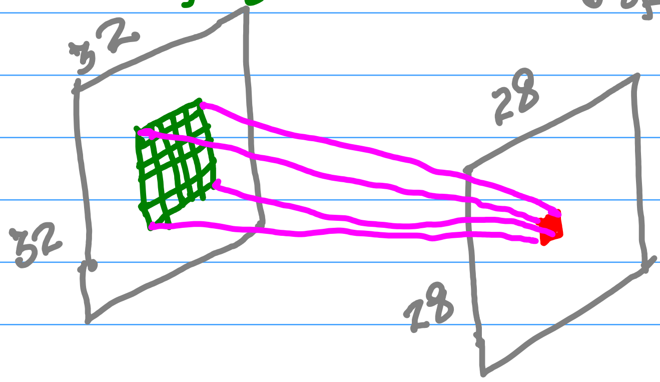
I/p ⇒ C1 ⇒ S2 ⇒ C3 ⇒ S4 ⇒ C5 ⇒ F6 ⇒ o/p

[tanh] [softmax]

LeNet-5 DETAILS

C1: first convolutional layer





$$a_j = \sum_{i=1}^{25} w_{ji}^{(1)} x_i + w_{j0}$$

(bias)

parameters

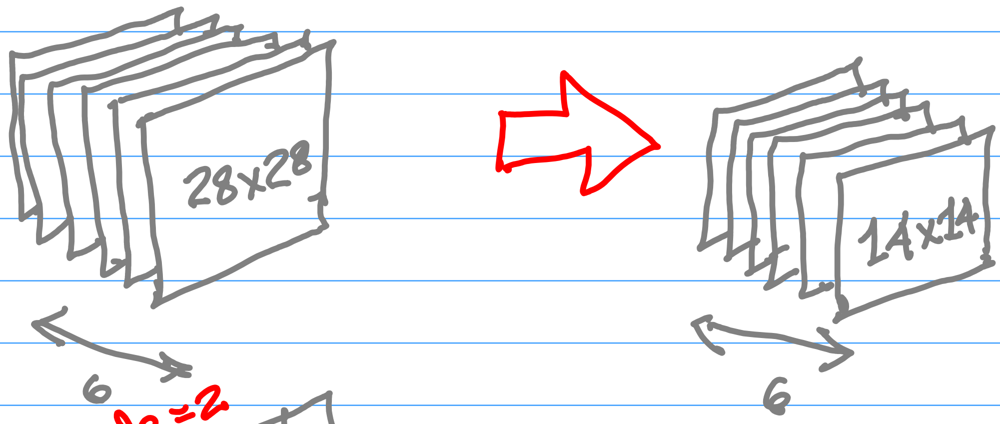
$$6 \times [5 \times 5 + 1]$$

$$= 6 \times 26 = 156$$

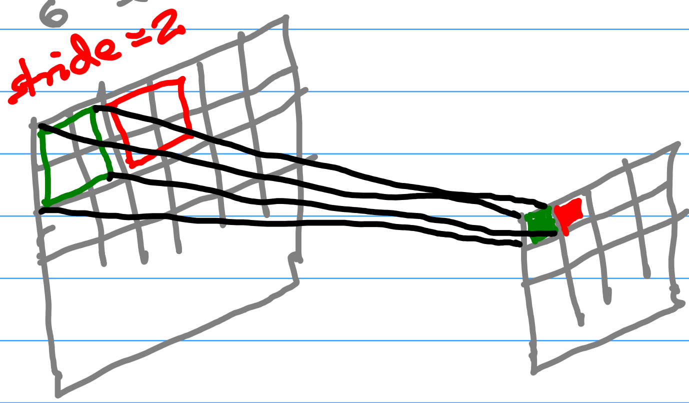
connections: for each of the 6 filters, for each pixel in the resultant 28×28 image

$$28 \times 28 \times 6 \times [\underbrace{5 \times 5 + 1}_{26}] = 1{,}22{,}304$$

Second layer (S2) "Sub-sampling" or Average Pooling layer



Stride = 2



Average pooling
same weight on each

$$a_j = \sum^{4} w_{ji} x_i + \begin{matrix} w_{j0} \\ \text{or} \\ b_j \end{matrix}$$

resultant: $\frac{w}{4}$ → 1 parameter

bias → 1 parameter

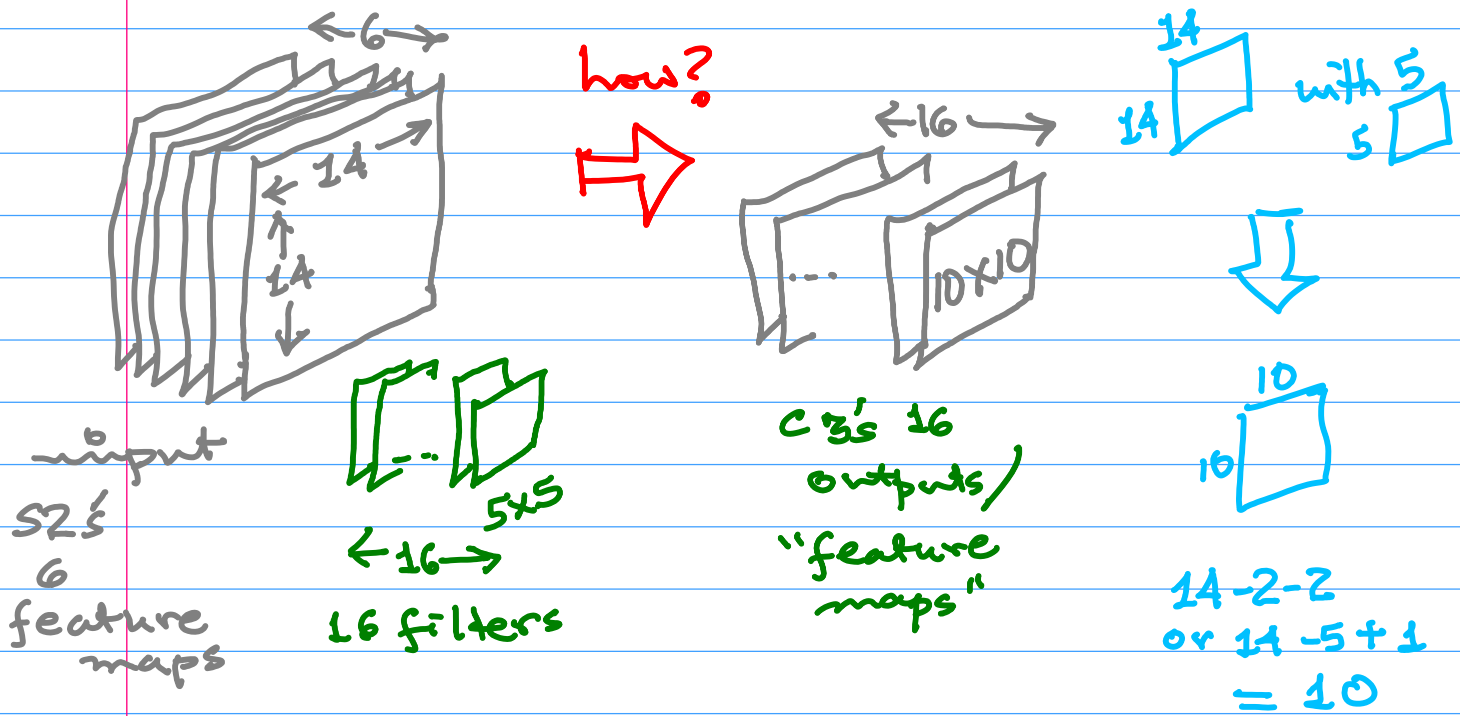
this is throughout the image,
independent of the image size (14 x 14)

$2 \times 6 = 12$ parameters

How many connections:

$$\underbrace{6}_{\text{for each filter/channel}} \times \underbrace{(4+1)}_{\text{connections}} \times \underbrace{14 \times 14}_{\text{image}} = 5{,}880 \text{ connections}$$

C3: 3rd Convolutional layer



How do 6 images of size 14 x 14 map onto 16 images of size 10 x 10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	x				x	x	x			x	x	x	x		x	x
1	x	x				x	x	x			x	x	x	x		x
2	x	x	x				x	x	x			x		x	x	x
3		x	x	x			x	x	x	x			x		x	x
4			x	x	x			x	x	x	x		x	x		x
5				x	x	x			x	x	x	x		x	x	x

"3 at a time" ← "4 at a time" → "all 6"

Each column indicates which feature maps in S2 are combined by the units in a particular feature map of C3

→ To break the symmetry in the network

→ to keep the number of connections within reasonable bounds.

16 filters of size 5x5 give us 16 output feature maps of size 10x10

However, which ones of the 6 input feature maps of the previous layer do we consider to take inputs from? Any one / some / all ?

Answer: 3 at a time + 4 at a time + 6 at a time.

How many parameters ?

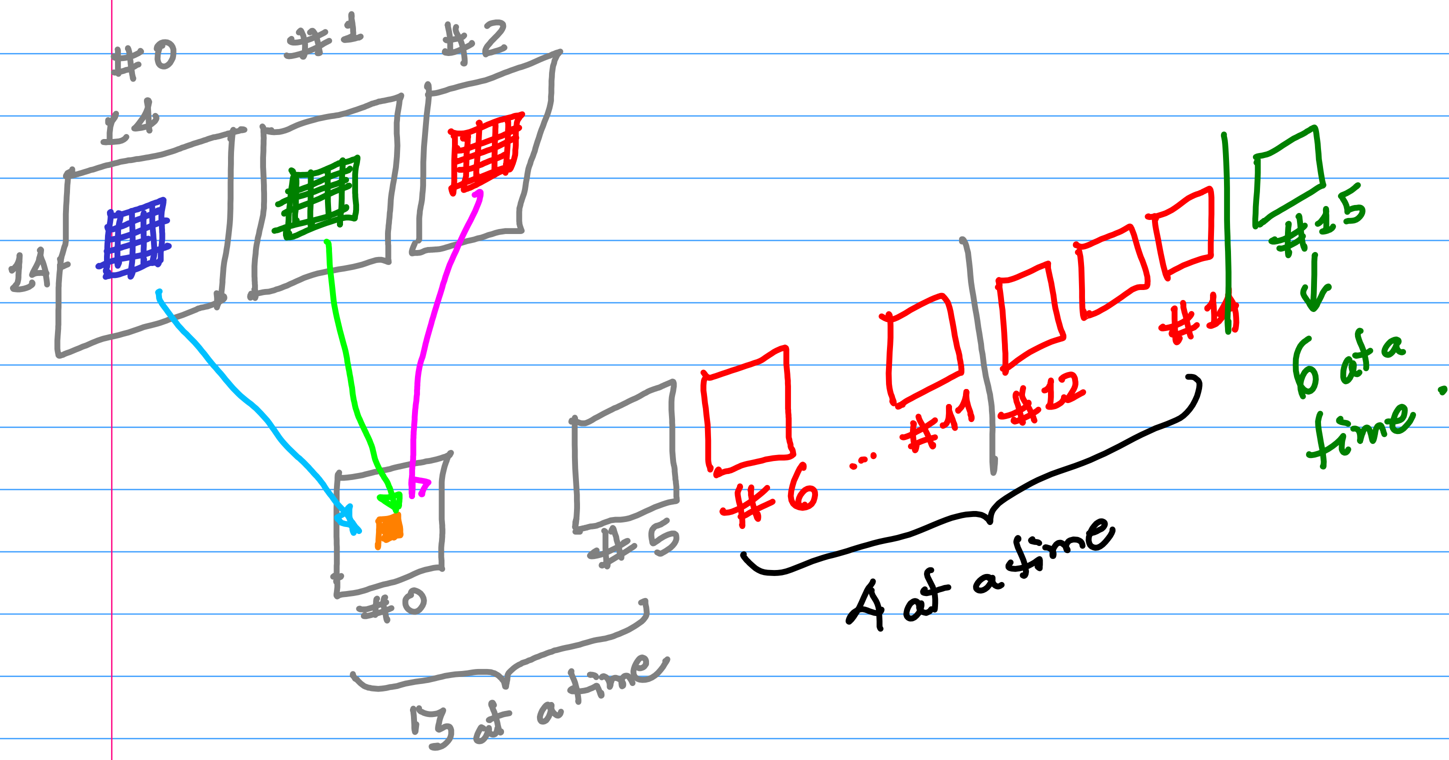
"3 at a time" (#0 to #5: 6 of these) take input from 3

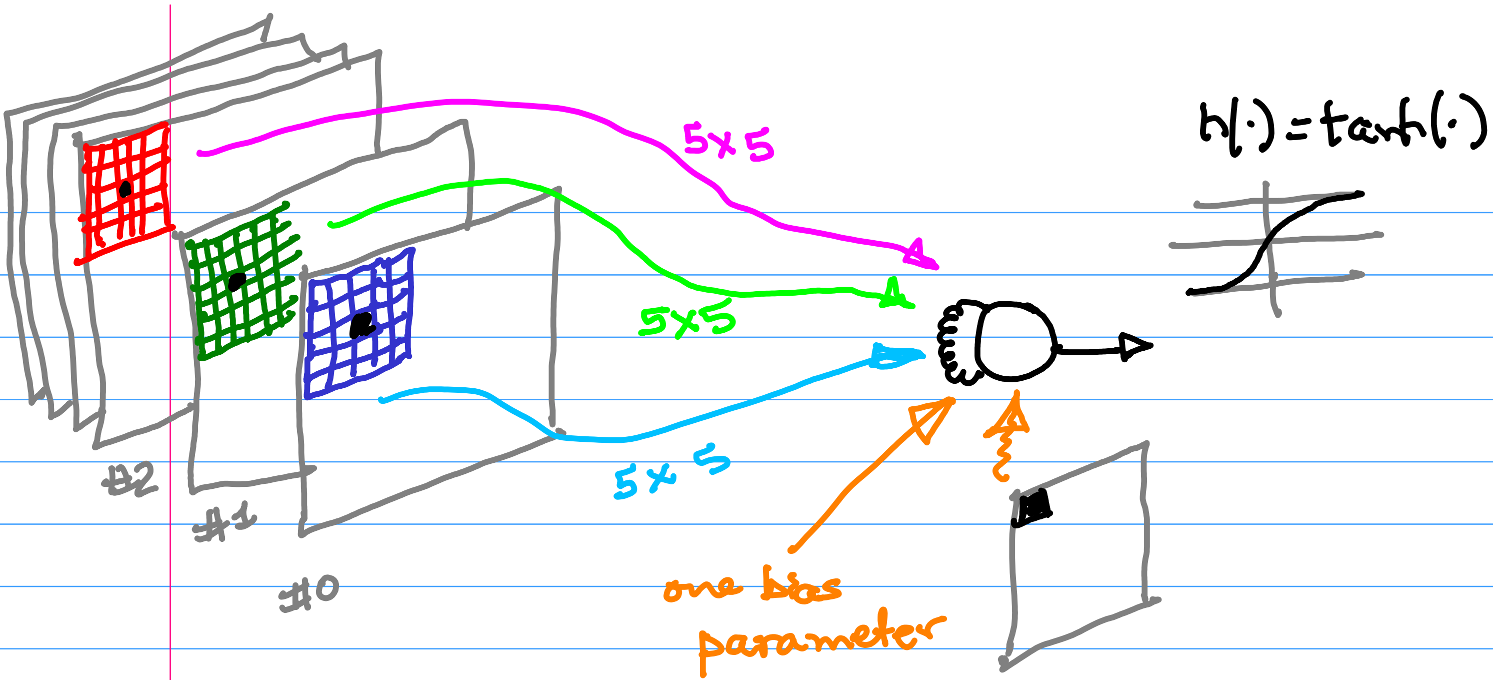
"4 at a time" (#6 to #14: 9 of these) take input from 4

"all 6" (#15: 1 of this) takes input from 6

$$6\times(3\times5\times5+1)+9\times(4\times5\times5+1)+1\times(6\times5\times5+1)$$
$$=6\times(75+1)+9\times(100+1)+1(150+1)=1516.$$

How many connections? for each pixel in the 10x10 image = 1516 x 100 = 1,51,600 connections.

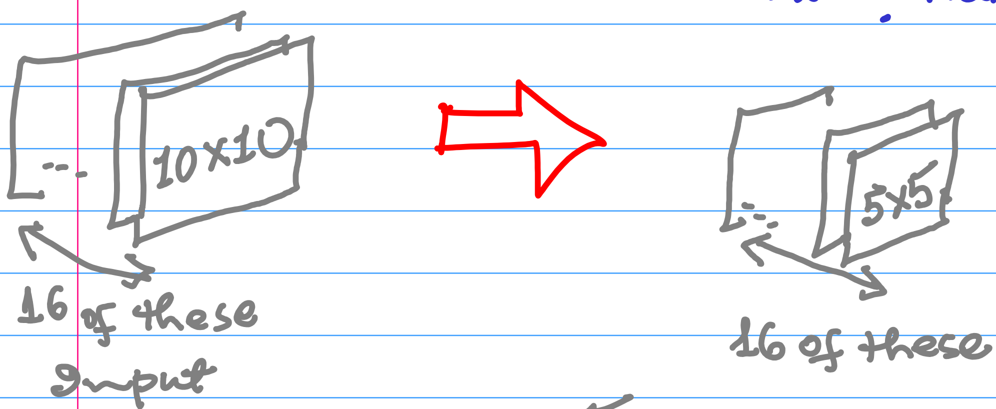


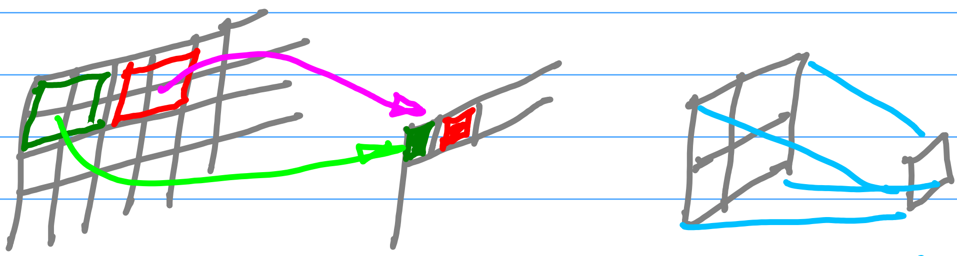


Example: for the first out of the "3 at a time" cases.

(4) "S4" The fourth (subsampling) layer

how? half the dimensions
$\Rightarrow$ stride = 2





Same weight for all 4 connections + 1 bias = 2 parameters, across the entire image, for each of the 16 output 5x5 images

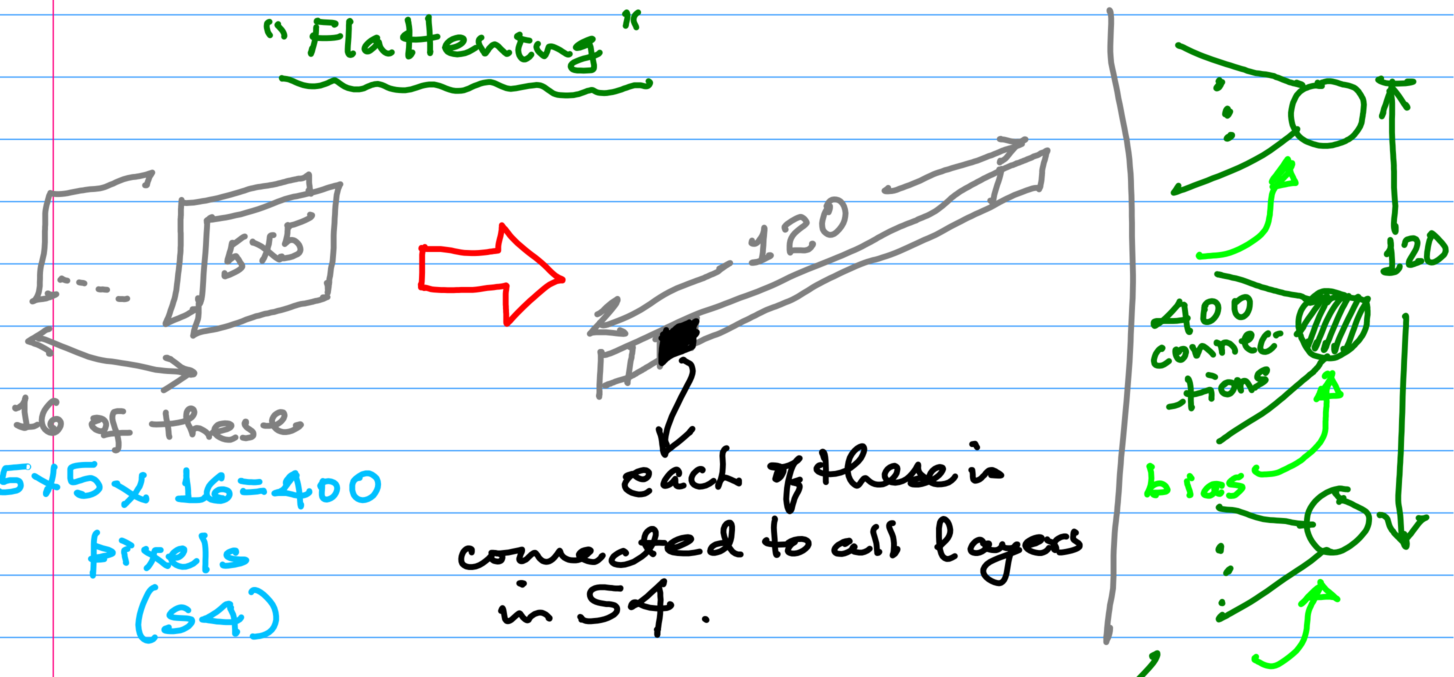
$= (1+1) \times 16 = 32.$

number of connections: The above structure repeats for each pixel in the 5x5 output image: 16 of them:

$$\Rightarrow \underbrace{(5\times5)}_{\text{size}}\times \underbrace{16}_{\text{16 images}} \times (\underbrace{4}_{\text{square: 2x2}}+\underbrace{1}_{\text{bias}}) = 5\times5\times80 = 2000$$

(5) "C5" 5th layer, C ⇒ convolution

"Flattening"



A fully connected layer has no local receptive fields

Parameters: For each of these 120 neurons e.g., the j-th one: there are 400 weights + 1 bias term

$$\Rightarrow 120\times(400+1) = 120\times401 = 48,120$$
parameters

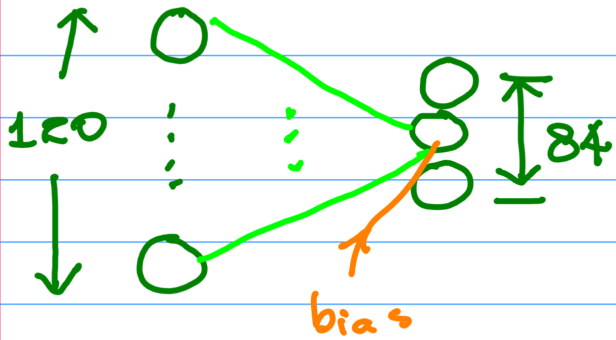
connections: The number of connections is the same as the number of parameters: this is a fully connected layer.

Why is this called a Convolutional layer

5x5 input with a 5x5 mask ⟶ results in a 1x1 image; there are 120 of these.

Why 120 & not any other number? (arbitrary!)

(6) "F6" Fully connected layer



Parameters:

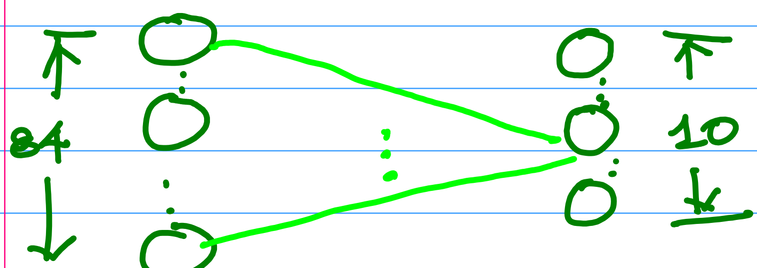
$$\underbrace{84}_{\#\text{ of outputs}} \times [\underbrace{120}_{\text{weights}} + \underbrace{1}_{\text{bias}}]$$

$= 84 \times 121$

$= 10{,}164$

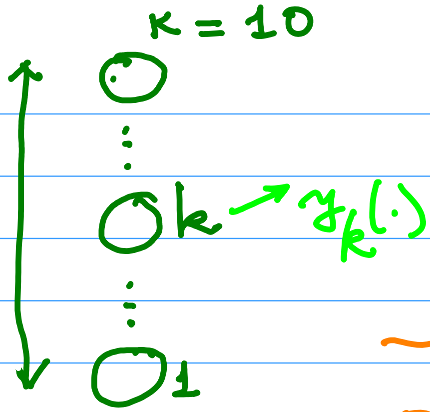
Connections: (same)

(7) Output layer



The activation fn's at all other layers were the tanh(.) function (smooth version of the signum function of the Perceptron)

For this output layer, it is the <u>SOFTMAX</u>



$$\sigma_k(y) \triangleq \frac{e^{y_k}}{\sum_{\ell} e^{y_\ell}}$$

why?

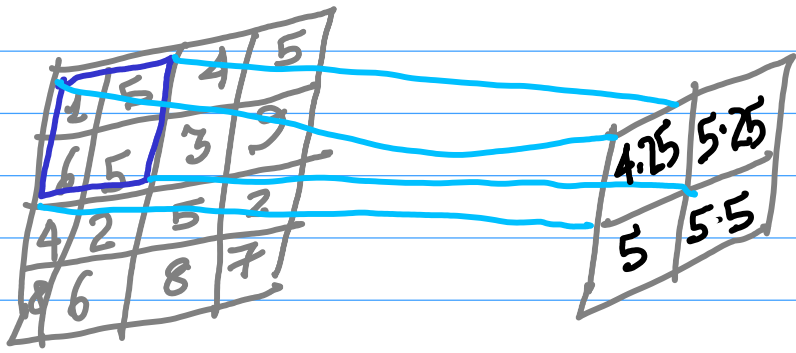
normalisation: getting a probability. → like a predictor.

why did we raise a number to an exponential? → large #, & not a small ratio.

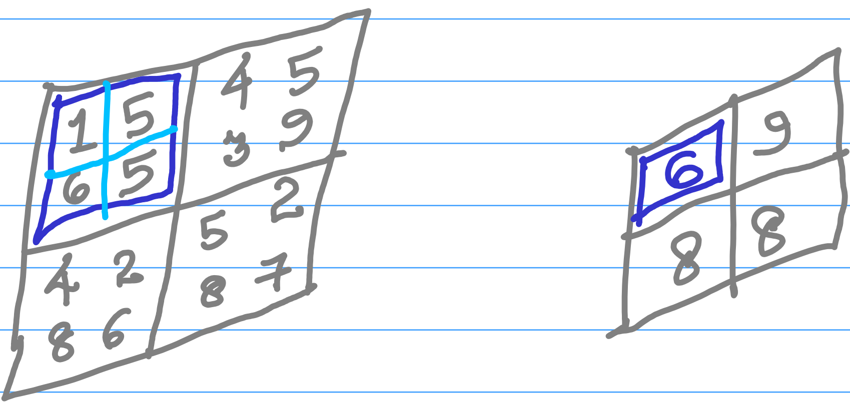
SOME IMPORTANT CONCEPTS

POOLING:

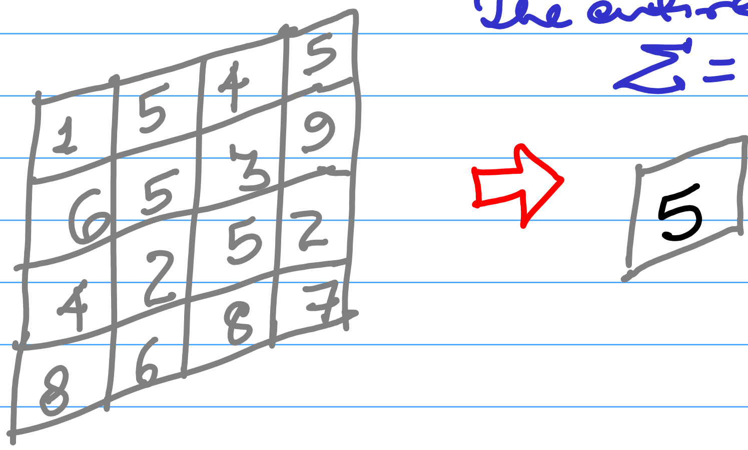
1) Average Pooling (LeNet)



2) Max Pooling (AlexNet)



5) Global Average Pooling



The entire frame into one

$\Sigma = 80,\ 80/4\times4 = 5$

(16 pixels: 4x4 grid)

Why Pooling? — contraction

— Invariance. Invariance to small transformations, distortions, translation. A small distortion in the input will not change the outcome of pooling drastically since we take the max/average value in a local neighbourhood

LOCAL RESPONSE NORMALISATION (LRN)

from neuro-biology (AlexNet)

→ 'lateral inhibition' → an excited neuron inhibits its neighbours → creates contrast in the area and increases sensory perception

→ No solid mathematical background

→ motivated by biology (evolution), shown statistically.

Batch Normalisation (2015) ["BN"]

[Ioffe & Szegedy, 2015]

→ Addresses the problem of Internal Covariate Shift (initially believed) current opinion: works because it smooths the objective function.

At initialisation: actually induces severe gradient explosion, which is only alleviated by skip connections in residual networks

The basic issue (historically) Each layer of a neural network has inputs with a corresponding distribution. This is affected during the training process by the randomisation

- the parameter initialisation
- the input

- During training, as parameters of the preceding layers change, the distribution of inputs to the current layer changes accordingly
- The current layer needs to constantly adjust to new distributions.
- small changes in the initial layers amplify, and result in a significant shift in deeper hidden layers

BENEFITS:

- Reduce unwanted shifts to speed up training
- permits a higher learning rate without vanishing/exploding gradients
- Regularisation effect: unnecessary to use 'dropout' to mitigate overfitting
- Robustness to different initialisation schemes and learning rates

Stochastic Optimisation → not all training data is taken together → normalisation is done in batches. Batch B has size m

$$\mu_B = \frac{1}{m}\sum^{m} x^{(n)} \; ; \; \sigma_B^2 = \frac{1}{m}\sum^{m}\left[x^{(n)} - \mu_B\right]^2$$

x : x is a scalar → one of the input dimensions; call it x_i, say.

$$\hat{x}_i \triangleq \frac{x_i - \mu_{B_i}}{\sqrt{\sigma_{B_i}^2 + \epsilon}}$$

ϵ → small positive constant used for numerical stability

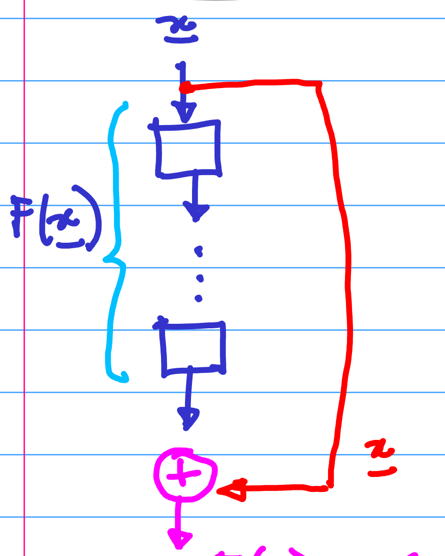
$$y_i \triangleq \gamma_i \hat{x}_i + \beta_i$$

γ_i, β_i → tunable parameters

Residual Connections / Skip Connections / Highway networks

Trend towards deeper neural networks → better accuracy or performance

trade-off: harder for training to converge



$H(\underline{x}) = F(\underline{x}) + \underline{x}$

$F(\underline{x}) = H(\underline{x}) - \underline{x}$

Used widely →

ResNet (Computer Vision),
Transformer (NLP, computer vision)
AlphaZero (RL),
AlphaFold (Protein Structure prediction)

Issues:

- Difficult to learn an identity mapping across layers
- Training a deep network is difficult because of exploding and vanishing gradients
- Observation: convolutional layers are often better at learning the residual rather than learning the feature map directly.

(*) Residual connections / skip connections / highway connections (contd.)

(*) The magnitude of the problem: →

- AlexNet (2012) had 5 convolutional layers
- 2014 : VGG , GoogleNet

19 layers   22 layers

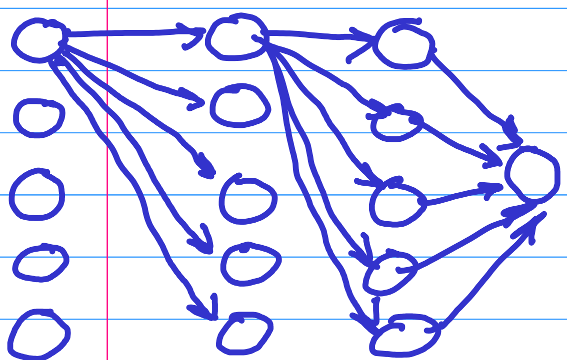
- ResNet (34 → 50 layers deep architectures)

deeper but had overall lower complexity.

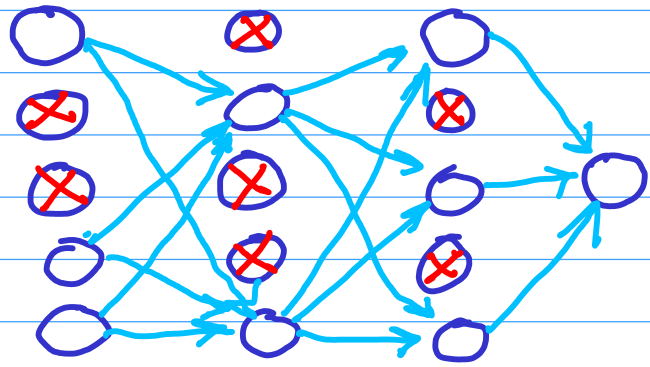
(*) 'DROPOUT' (AlexNet 2012)

Heuristic : applied at the training phase (FC layers)

Dropout is a kind of regularisation technique to reduce overfitting.



Usual FC scenario



Scenario with dropout

AlexNet : $p = 0.5$ at the first two fully connected layers

Neuron : has a probability not to contribute to the feedforward phase & participate in the backpropagation. ⇒ Each neuron can have a larger chance to be trained, and not depend on some 'strong' neuron. No dropout at the test time

(*) Residual connections/skip connections/ highway connections (contd.)

(*) The magnitude of the problem: →

- AlexNet (2012) had 5 convolutional layers
- 2014 : VGG, GoogleNet

19 layers — 22 layers

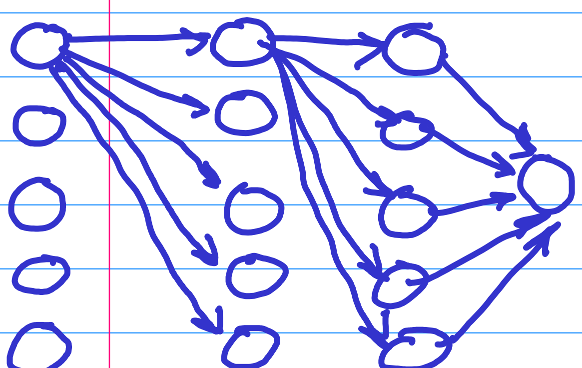
- ResNet (34 → 50 layers deep architectures)

deeper but had overall lower complexity.

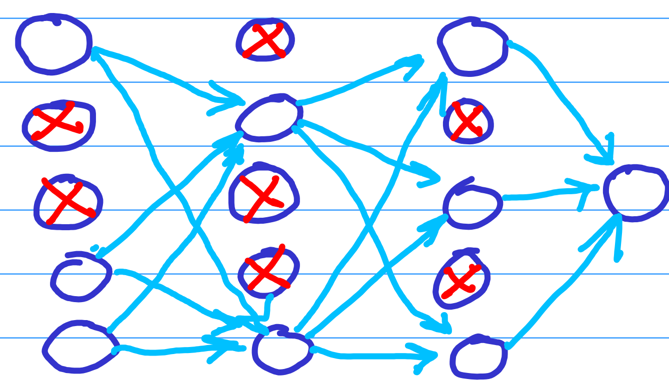
(*) 'DROPOUT' (AlexNet 2012)

Heuristic : applied at the training phase (FC layers)

Dropout is a kind of regularisation technique to reduce overfitting.



Usual FC scenario



Scenario with dropout

AlexNet : $p = 0.5$ at the first two fully connected layers

Neuron : has a probability not to contribute to the feedforward phase & participate in the backpropagation. ⇒ Each neuron can have a larger chance to be trained, and not depend on some 'strong' neuron. No dropout at the test time

Second class of successful deep Architectures:

AlexNet (2012) and CaffeNet

↳ used 2 GPUs (two parallel paths)

↳ used 1 GPU

What & why? ImageNet: 15-million labelled high-resolution images, with 22K categories (22,000)

2012: ImageNet Large-Scale Visual Recognition Competition

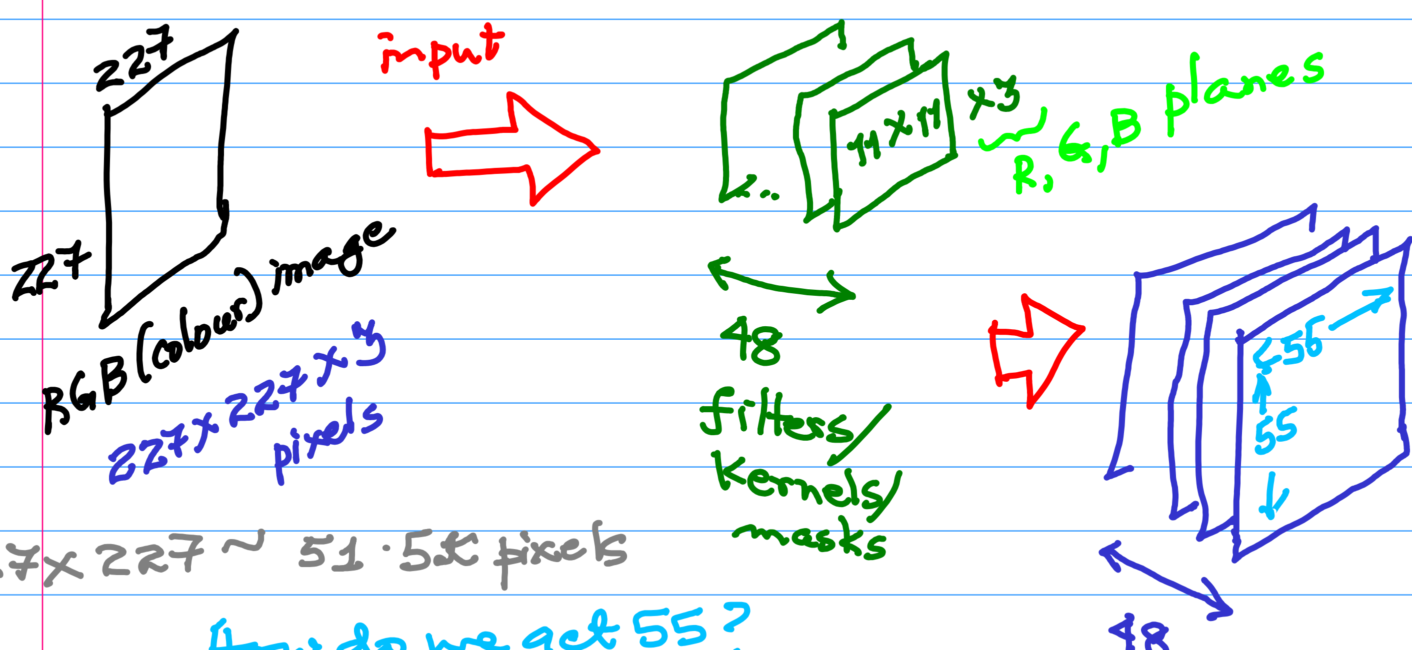
dataset: subset of ImageNet

* 1K images in each of the 1K categories/classes.

- 1.2 million training images
- 50K validation images
- 1.5L testing images

AlexNet: 8 layers (5 Conv layers)

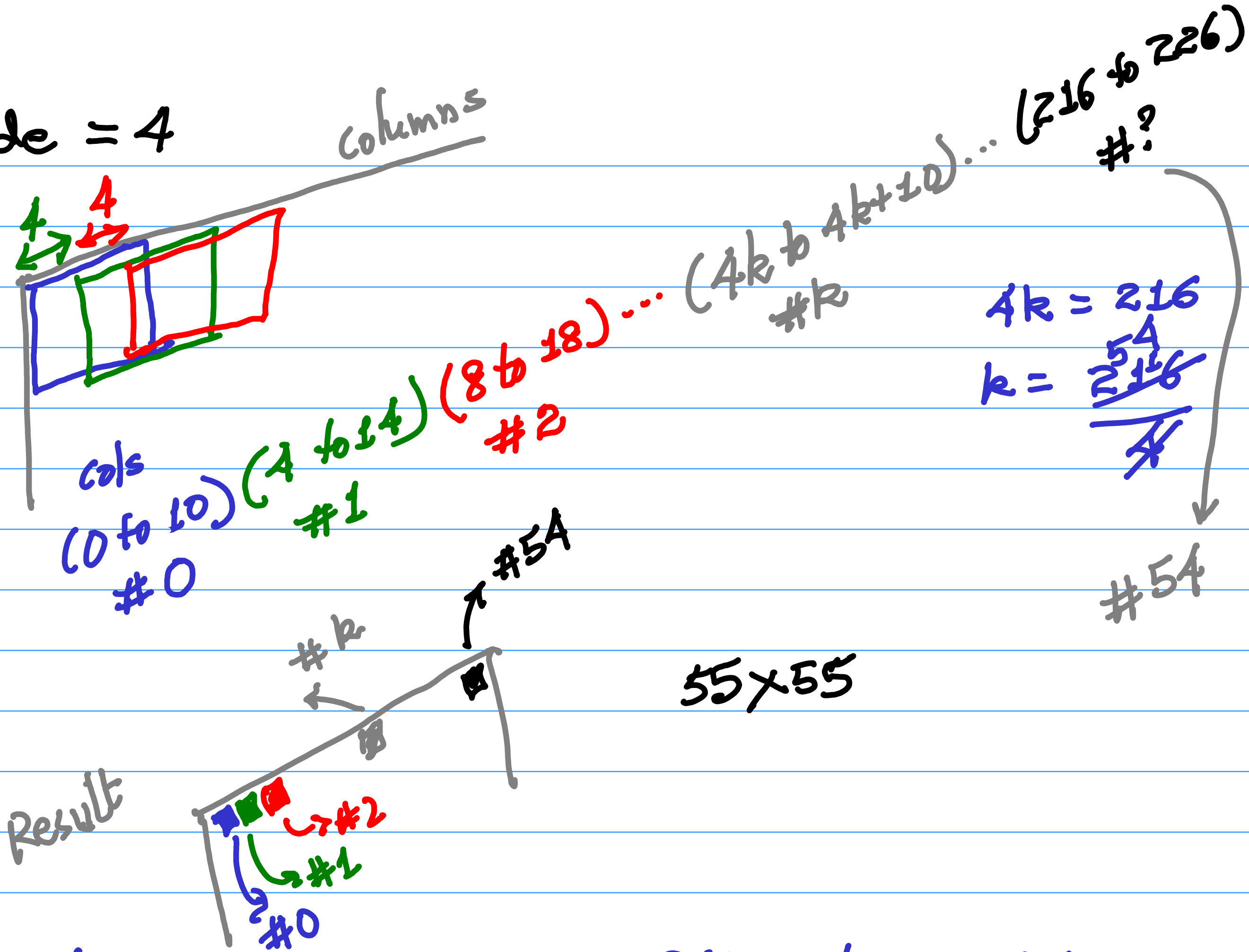
(1) 1st Convolutional layer



227×227 ~ 51.5K pixels

How do we get 55?

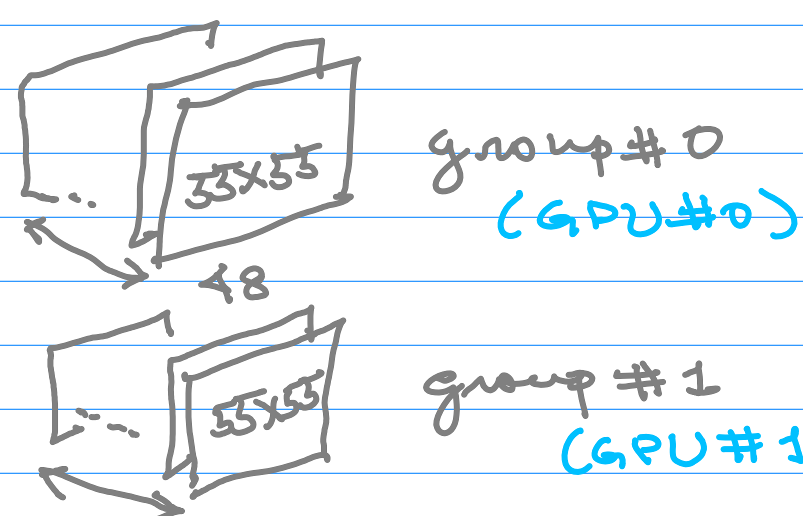
Stride = 4



AlexNet has 2 groups of 48 filters/kernels/masks each, each group given to one of the two GPUs used.

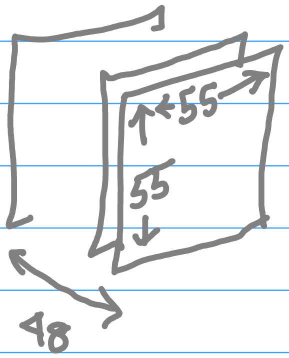
[CaffeNet is a 1 GPU version, 96 filters / kernels / masks]

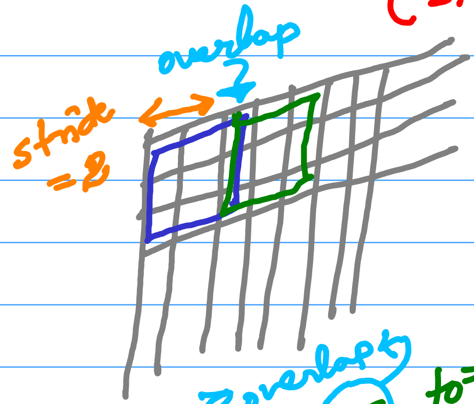
→ Result: 2 groups of 48 outputs, each of size 55x55
(AlexNet)



AlexNet (contd.)

② Second layer: 3x3 overlapping Max Pooling (stride = 2)





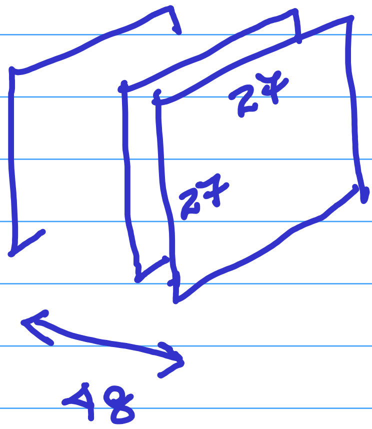
overlaps

(#0 to #2) (#2 to #4) (#4 to #6) ... (#2k to #2k+2) ... (#52 to #54)

#0 #1 #2 #k

$2k = 52 \Rightarrow k = 26$

→ There are 27x27 outputs
48 of these, 2 groups

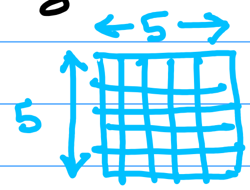


③ 3rd layer: Local Response Normalisation

→ only the values change, the size does not change.

④ Second Convolutional Layer

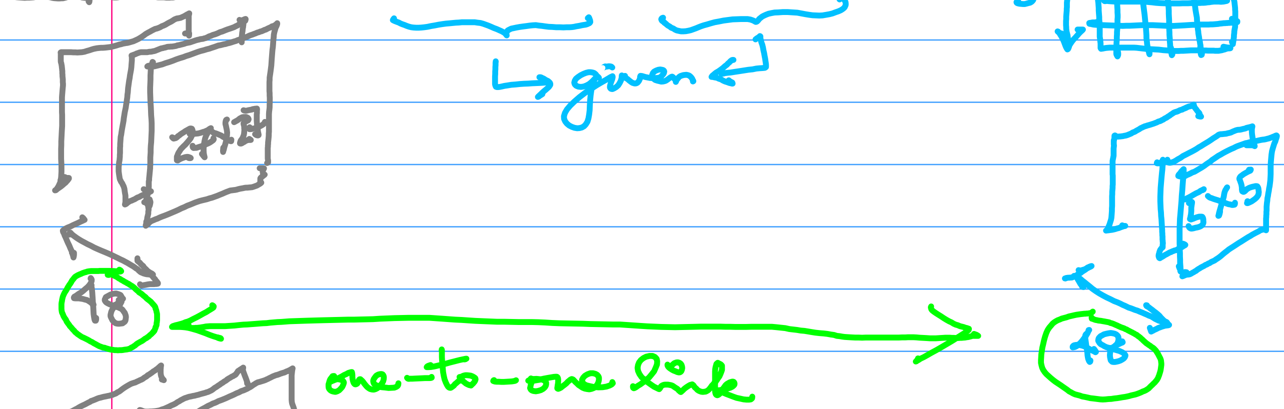
2 groups of 128 kernels of size 5×5×48

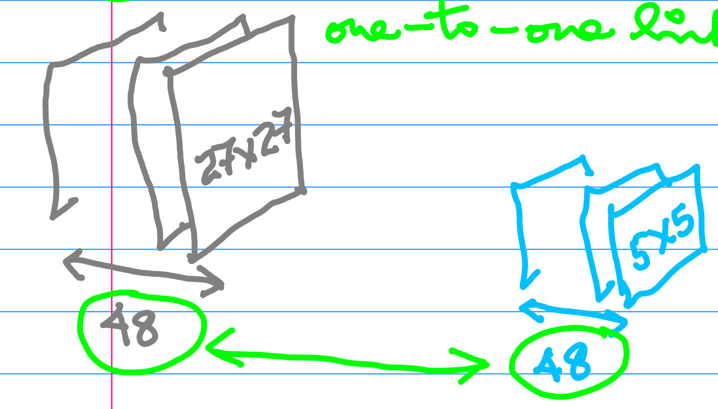


Input:

$\underbrace{\text{stride = 1}}, \underbrace{\text{pad = 2}}$

↳ given ↲

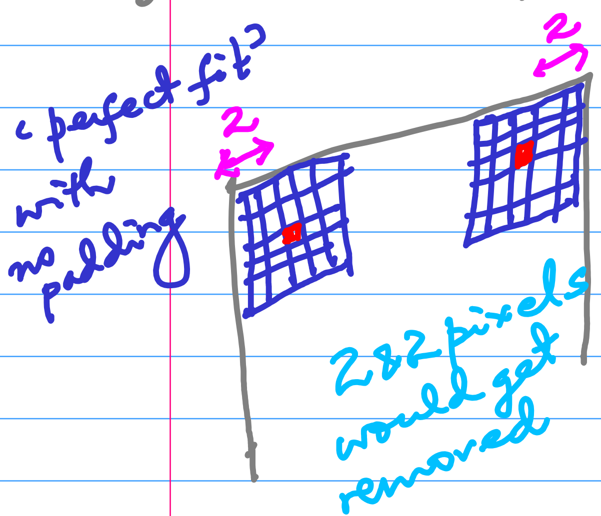


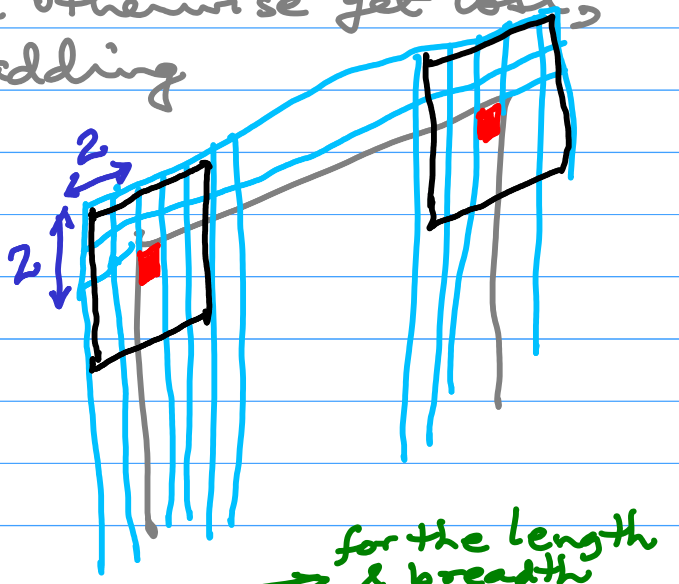


stride = 1

⇒ Result can get smaller with a 'perfect fit' convolution, but here

pad = 2 → doesn't in this case, since the number of pixels which would otherwise get lost, get made up with the padding

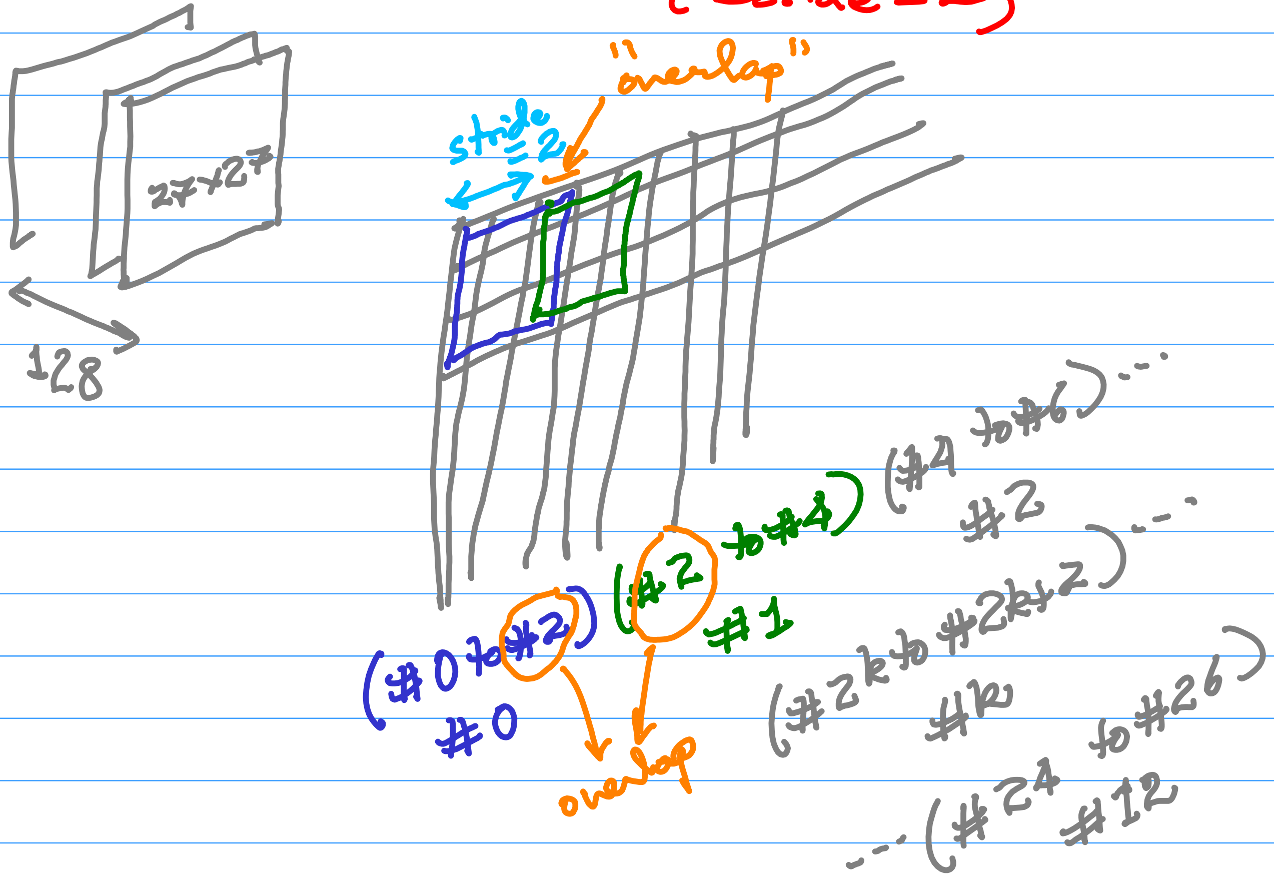




padding is an extra 2 pixel layer for the length & breadth

→ The resultant image size remains 27 × 27
128 filters / kernels in 2 groups
→ Output 27 × 27 × 128 × 2 groups.

⑤ **Next layer: 3 × 3 overlapping Max Pooling**
(stride = 2)



→ there are 13 × 13 × 128 outputs in 2 groups.

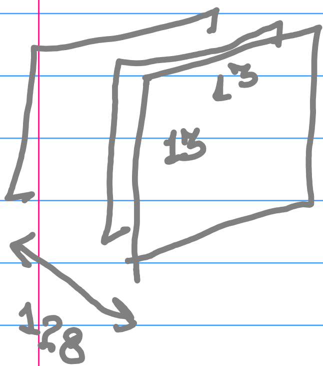
⑥ **Local Response Normalisation**
(This does not change the output size: size-preserving, but changes the values)

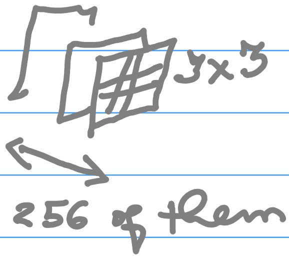
AlexNet (contd).

(7) Third Convolutional layer:

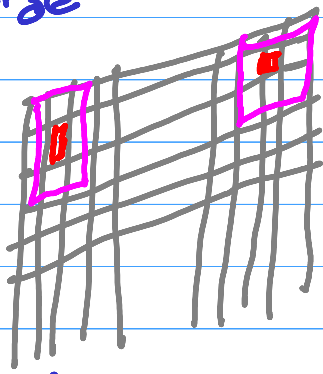
2 groups of 192 kernels of size $3 \times 3 \times 256$
stride = 1, pad = 1

Input: 13×13 images $\times 128$ in 2 groups.

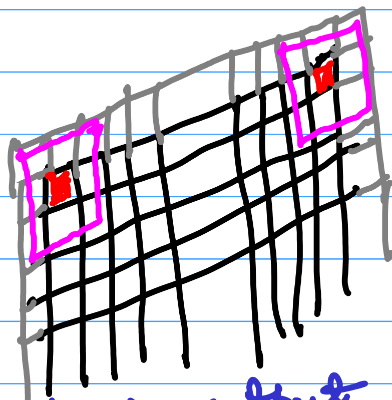




first, let us consider the size



'just fit' $\rightarrow$ 12×12



pad = 1 $\Rightarrow$ output 13×13

Now, how do we account for the number? $\rightarrow$ heuristic!

To resolve 128, 256 and 192

$$128 \xrightarrow{+128/2} 192 \xrightarrow{+128/2} 256$$

nothing mentioned about pooling, so possibly 2 kernels each for the 128 to give 256 and then some selection and pooling to give 192.

now that we have an understanding of padding, convolutions, stride & pooling, we will recognise that there are many heuristics to get actual numbers.

→ Try to look for conceptual ideas from different families of successful architectures.

VGG-16/-19

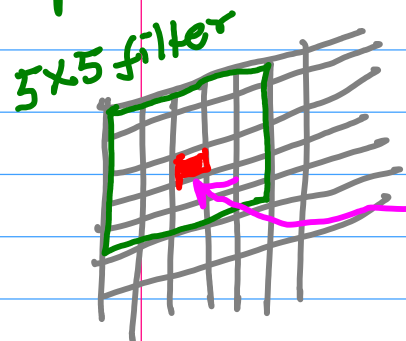
"Visual Geometry Group" at the University of Oxford

Karen Simonyan, Andrew Zisserman (2014)

Basic Concept:

The use of 3×3 filters/kernels in place of larger 11×11 or 7×7 filters.

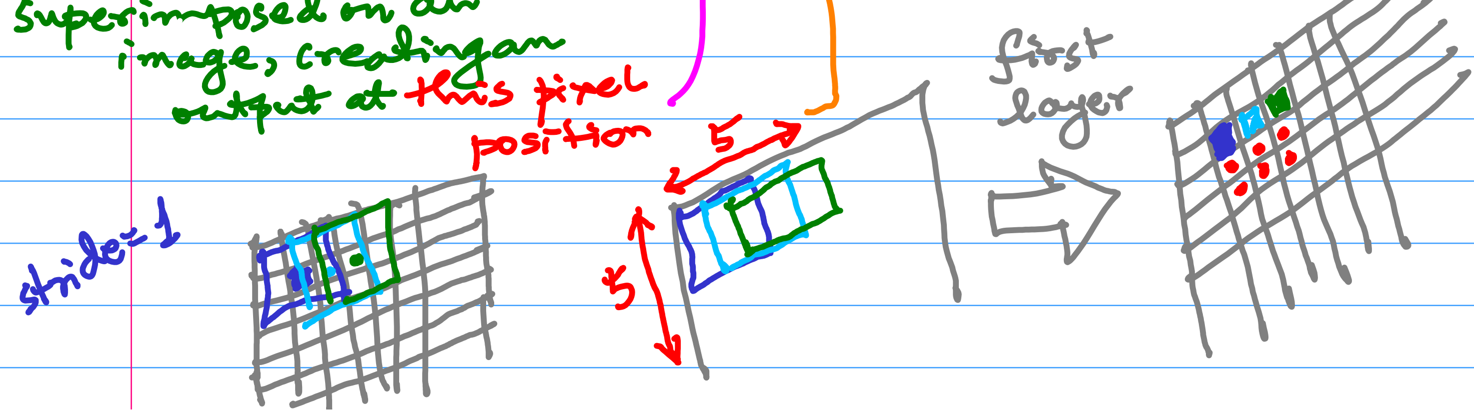
Result: Simpler architecture with a smaller no. of parameters, but an increased depth: (16–19 layers)

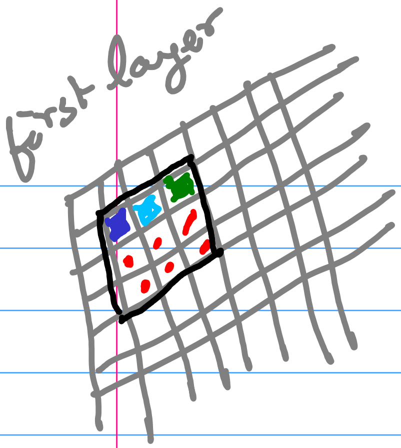


large 5x5 filter superimposed on an image, creating an output at this pixel position

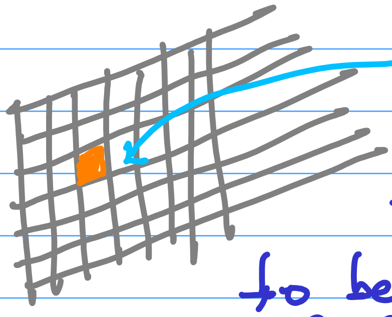
instead of this

2 layers of 3×3 filters: cover the same effective pixel positions as one larger 5×5 mask





we note that the pixel positions in the first layer allow a 3x3 mask/ filter/ kernel at another layer to be put right here



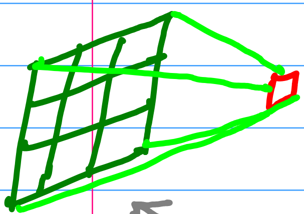
position of the output which we note (happily :-)) to be the position at which we calculated the output of the layer 5x5 mask.

→ Advantage: there is a smaller # of parameters

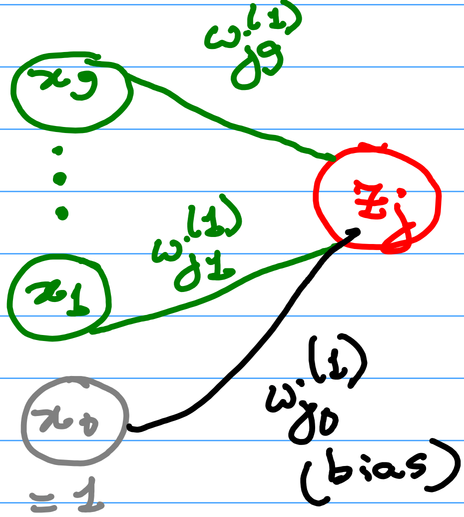
(*) 5x5 filter : $5\times5+1$ (bias) $= 26$ parameters

2 layers of 3x3 : $2\times(3\times3+\underbrace{1}_{\text{bias}}) = 2\times10 = 20$ parameters

These weights (parameters): they are the relative-position-invariant weights of the local receptive field



3x3 inputs



$$z_j = \sum_{i=1}^{9} w_{ji}^{(1)} x_i + w_{j0}$$

(activation)