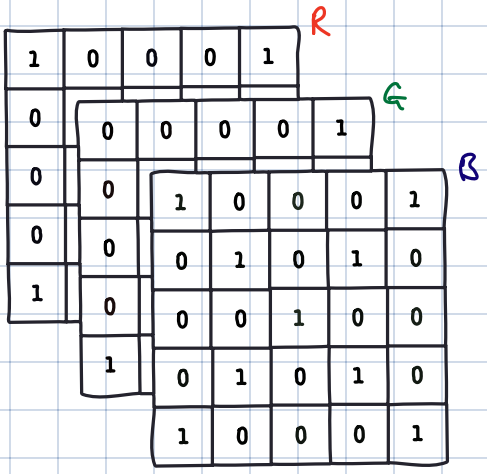
CONVOLUTIONAL NEURAL NETWORKS

A convolutional neural network (CNN) is a network architecture for deep learning specifically designed for finding patterns in images to recognise objects, classes and categories.

An image, made of pixels, is just a matrix of numbers (just luminescence) or multiple matrices (RGB)



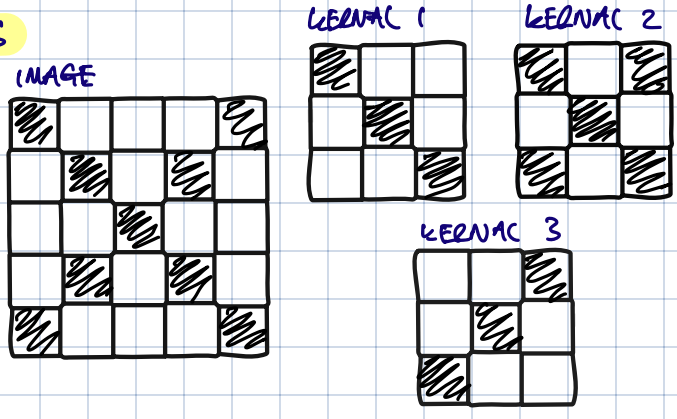
Why not use a CNN?:

① E.g. 1024 x 1024 image with 64 neurons in first layer:

$1024^2 \times 64 = 67,108,864$ weights in first layer

② - Images have special features therefore the order in which data is presented matters.
- With CNN's we 'flatten' our input into a vector and lose spatial information.

- The solution is to use CONVOLUTIONAL KERNELS
- We 'drag' these 'filters' across the image and perform a convolution operation at each instance.
- This operation tells us how similar that section of the image is to the filter (kernal).



IMAGE

1	-1	-1	-1	1
-1	1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	1	-1
1	-1	-1	-1	1

$*$

KERNAL 1

1	-1	-1
-1	1	-1
-1	-1	1

$=$

9	-3	1
-3	3	-3
1	-3	9

$=$

y_{111}	y_{121}	y_{131}
y_{211}	y_{221}	y_{231}
y_{311}	y_{321}	y_{331}

9 neurons outputs

$(1 \cdot 1) + (-1 \cdot -1) + (-1 \cdot -1) + (-1 \cdot -1) + (1 \cdot 1) + (-1 \cdot -1) + (-1 \cdot -1) + (-1 \cdot -1) + (1 \cdot 1) = 9$

Each neuron is connected to a limited amount of inputs (limited amount of image), called the RECEPTIVE FIELD of that specific neuron

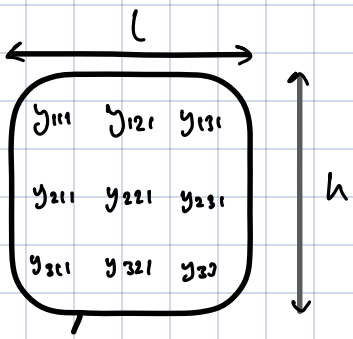
$$y_{ij1} = h(W_1 * X_{ij} + b_1)$$

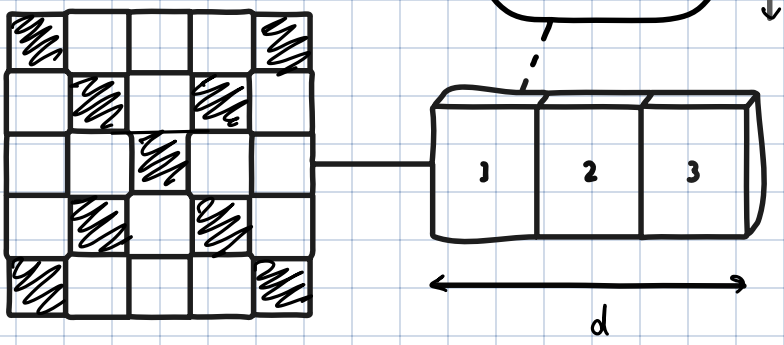
↑ Relu ↑ kernal ↑ patch ↑ bias

where $W_1 = \begin{Bmatrix} w_{111} & w_{121} & w_{131} \\ w_{211} & w_{221} & w_{231} \\ w_{311} & w_{321} & w_{331} \end{Bmatrix}$ = kernal 1

- Convolutional layer is composed of $h \times l \times d$ neurons
- The resulting output of a convolutional layer is $h \times l \times d$ array (i.e. a tensor)

h = height of output (e.g. 3)
l = length of output (e.g. 3)
d = number of kernal (e.g. 3)

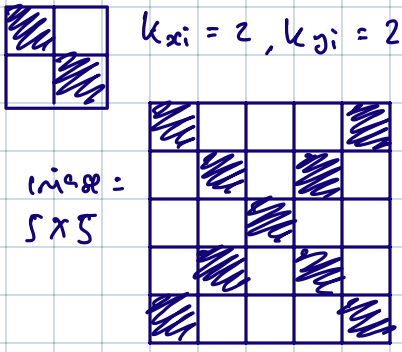




FEATURE MAP - set of neurons that share the same feature (kernal).

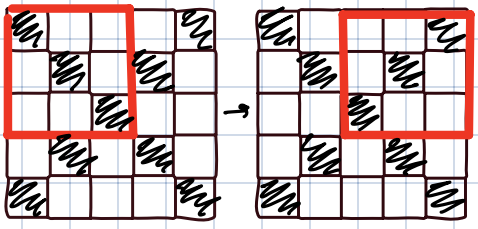
- You can see the size of the output depends on a series of hyperparameters:

(1) kernal and image dimensions

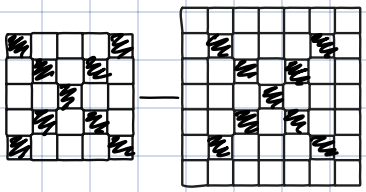


(2) STRIDE(S) - distance between centre of current and next:

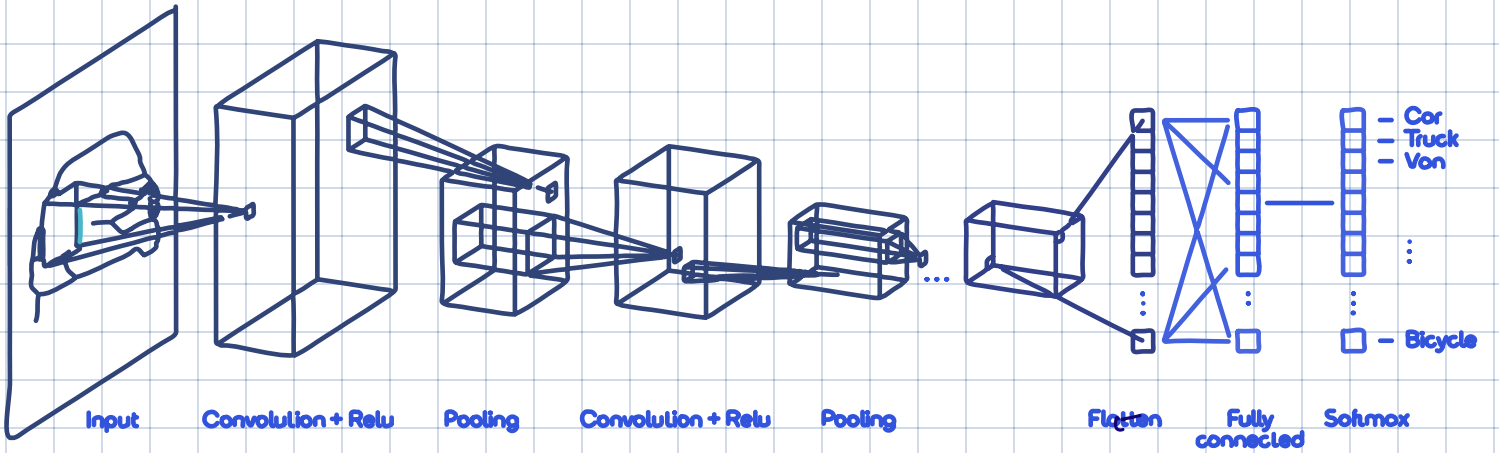
e.g. $S = 3$



(3) PADDING (P) - Fills in the image with 0 on the edges to include edge pixels, avoid artifacts.



CNN ARCHITECTURE



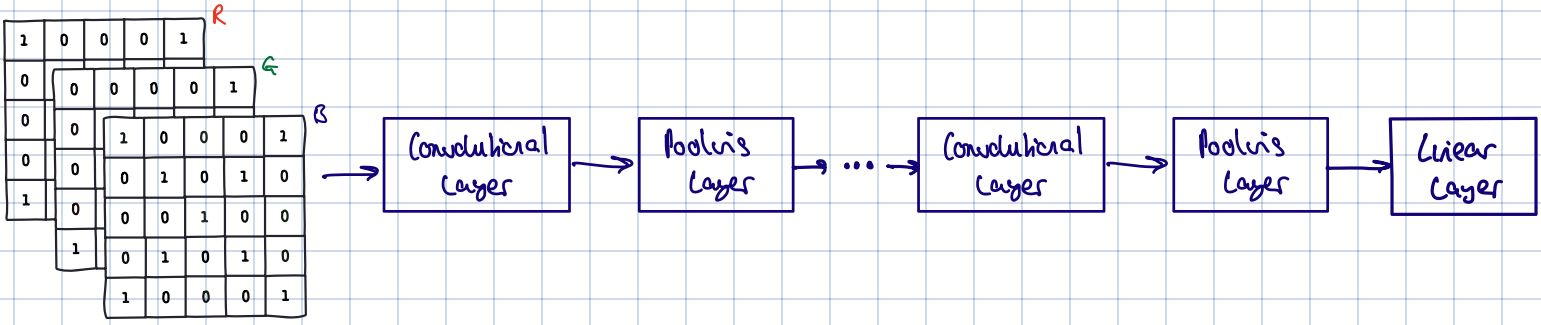
- Initialise weights, set up training set with labels and then backpropagation to teach our network the desired outputs.

$$w_{ij}^{lin} = w_{ij}^{lin} - \alpha \frac{\partial H}{\partial w_{ij}^{lin}} \quad \text{and} \quad w_{ijz}^{conv} = w_{ijz}^{conv} - \alpha \frac{\partial H}{\partial w_{ijz}^{conv}}$$

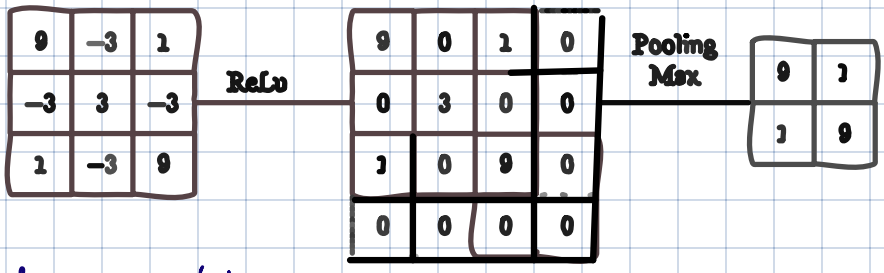
← where it learns the features (i.e. kernals not given).

POOLING LAYER

- This layer is used to reduce the dimension of the input into the next layer.
- To focus on larger areas of the image
- Capture combination of the features (the dimension of the kernal in successive layers should increase).



- Max Pool : Maximum value in pooling filter
- Avg Pool : Average over values in pooling filter.
- Dimension of pooling filter is design choice
- The stride of the pooling filter is almost always equal to its x-size.



LINEAR LAYER

- A fully connected neural network is used to provide a probability distribution for our input image.
- We must flatten the output of the last pooling layer in order to input it into the LNN.
- For the output of the LNN, we will use the SOFTMAX ACTIVATION FUNCTION which will rescales the outputs in the range $[0,1]$ and sets their sum equal to 1 so the output represents the probability:

$$y_i = \text{softmax}(\bar{y}_i) = \frac{e^{\bar{y}_i}}{\sum_{j=1}^{N} e^{\bar{y}_j}} \quad \text{where } N = \text{number of outputs}$$

- We can compare this probability output to the "ONE-HOT VECTOR" of the actual label which is the label as a 1 and everything else 0.

e.g. $y = \begin{bmatrix} 0.2 \\ 0.2 \\ 0.6 \end{bmatrix} \longrightarrow z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ e.g. $z_{dog} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

- Moreover, the softmax matches perfectly with the CROSS-ENTROPY LOSS FUNCTION which measures the performance of classification model with probability output:

$$H = -\sum_{j=1}^{N} z_j \log(y_j) \quad \text{where} \quad \frac{dH}{d\bar{y}_i} = y_i - z_i$$

(z_j: actual, y_j: predicted)