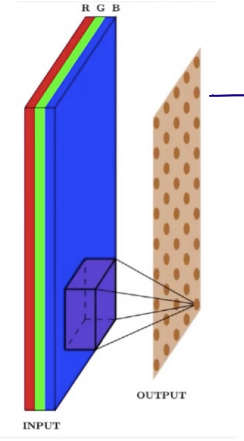
Module 31: Convolution to Neural Networks

→ Difference b/w fully connected neural networks and Convnets.

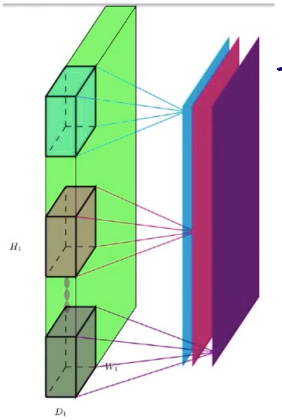
Rather than taking a weighted sum of all the pixels in an image, we take the weighted sum of the pixels in a given neighbourhood of a pixel. This aggregation is again fed to an activation and that gives us an output set of values.

↓

Also the weights are not different for different i/p features in an image as is the case in fully connected networks. Rather they are shared across the image, in the filter / Kernel.



→ This is similar to the next layer for the given i/p layer in a fully connected network



→ each kernel leads to a different 2D o/p. These o/ps can be stacked over each other to get the next layered o/p.

→ Conventional machine learning vs deep representation learning

Neural networks are also used for feature extraction in unstructured data, also called representation learning.

↓

Rather than relying on experts for feature engineering, we allow the neural networks to find the important features in the data. Before, we had allowed them to learn weights for model representation to be used in the task.

↓

for example, classification on image after applying certain feature engineering methods like edge detection or a combination of SIFT/HOG, feeding into a classification model like SVM or naive bayes. This requires static feature extraction and these methods require domain knowledge, and also an underlying study of the data, and the task at hand. Also sometimes, a kernel with non conventional values might give a better result.

↓

Rather we even allow the model to learn the representation that minimizes the loss by deciding the weights of the kernel. This tunes the process to the data, hence allowing it to be much more effective.

↓

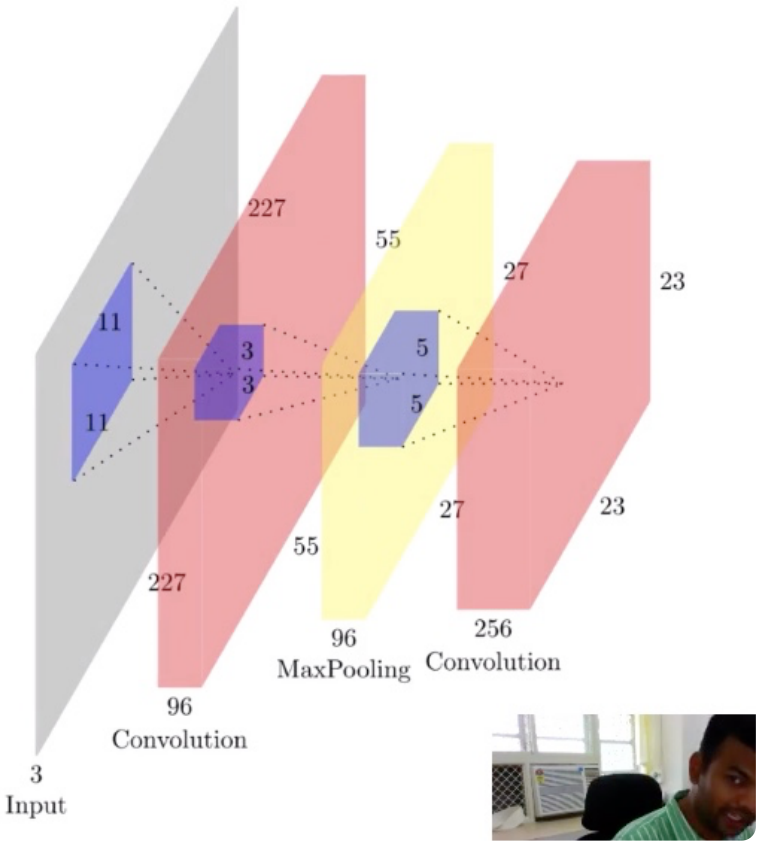
We move further by taking multiple such filters, and then tune them as per the loss which could give different representation, which themselves could be weighted (one representation being more important than the other), and

these representations being used in the model approximation for the task.

↓

Extending to multiple layers of feature representations to get more complex representations.

→ Dimensionality for layers of Convnets



» W_I H_I D_I — W_o H_o D_o

227 227 3 — 55 55 96

» filter : 11 * 11 * 3

» stride : 4

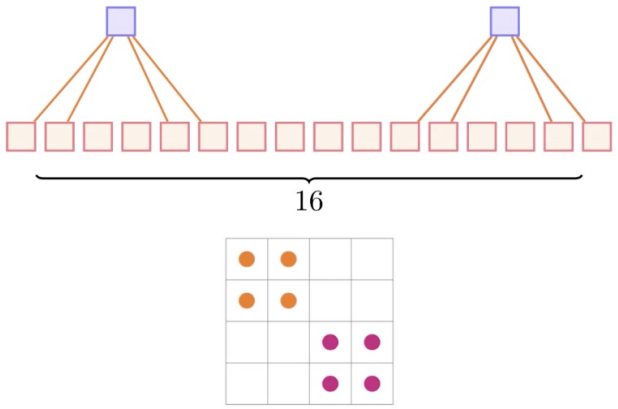
» padding : 0

» no. of filter : 96

$$W_o = \frac{W_I - f + 2P}{S} + 1$$

→ Sparse connectivity and weight sharing

① Rather than taking into account all pixels for the computation of an input for next layer, we only take pixels in the neighbourhood of a given pixel in the o/p image. This is referred to as **sparse connectivity**. This can also be understood as the fact that weights of pixels not in the neighbourhood of a pixel are taken to be zero.



Also the weights are not different for different i/p features in an image as is the case in fully connected networks. Rather they are shared across the image, in the filter / kernel. This is referred to as weight sharing.

↓

This drastically reduces the no. of parameters to be trained, maintaining the sense of complexity.

*) max-pooling operation : The 2D o/p after applying a kernel to an image is called a **feature map**. Maxpooling involves moving the mask over the image applying the max fn.

Also called sub-sampling. Sometimes even average pooling is done.

*) Applying non-linearities: Non linearities are applied elementwise to get final o/p.

→ Training a ConvNet : Backpropogation for convnet

① At any point of a backward pass, we would require two derivatives : a) wrt kernel weights of that layer
b) wrt i/ps of that layer to allow further backpropogation

② To update weights, we need the derivative of loss wrt the weights. We use the chain rule, by finding out the outputs for that layer which we contributed to, and then using the fact as to how they contributed to the loss (abstracted from this layer).

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial w}$$

③ Due to weight sharing, the weight contributes to multiple o/p features in the feature map. The contribution is added to its own.

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial o_{11}} * \frac{\partial o_{11}}{\partial w_{11}} + \frac{\partial L}{\partial o_{12}} * \frac{\partial o_{12}}{\partial w_{11}} + \frac{\partial L}{\partial o_{13}} * \frac{\partial o_{13}}{\partial w_{11}}$$

$$+ \frac{\partial L}{\partial o_{21}} * \frac{\partial o_{21}}{\partial w_{11}} + \frac{\partial L}{\partial o_{22}} * \frac{\partial o_{22}}{\partial w_{11}} + \frac{\partial L}{\partial o_{23}} * \frac{\partial o_{23}}{\partial w_{11}}$$

$$+ \frac{\partial L}{\partial o_{31}} * \frac{\partial o_{31}}{\partial w_{11}} + \frac{\partial L}{\partial o_{32}} * \frac{\partial o_{32}}{\partial w_{11}} + \frac{\partial L}{\partial o_{33}} * \frac{\partial o_{33}}{\partial w_{11}}$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial o_{11}} * i_{11} + \frac{\partial L}{\partial o_{12}} * i_{12} + \frac{\partial L}{\partial o_{13}} * i_{13}$$

$$+ \frac{\partial L}{\partial o_{21}} * i_{21} + \frac{\partial L}{\partial o_{22}} * i_{22} + \frac{\partial L}{\partial o_{23}} * i_{23}$$

$$+ \frac{\partial L}{\partial o_{31}} * i_{31} + \frac{\partial L}{\partial o_{32}} * i_{32} + \frac{\partial L}{\partial o_{33}} * i_{33}$$

↓

This can be seen as a convolution between the input of the layer and the loss wrt output feature map, which gives the derivatives to tune the kernel.

④ Similarly, the derivative wrt the input can be generated.

$$\frac{\partial L}{\partial i_{11}} = \frac{\partial L}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial i_{11}} = \frac{\partial L}{\partial o_{11}} \cdot w_{11}$$

$$\frac{\partial L}{\partial i_{12}} = \frac{\partial L}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial i_{12}} + \frac{\partial L}{\partial o_{12}} \cdot \frac{\partial o_{12}}{\partial i_{12}}$$

⋮

similarly for all other inputs.

↓

These will be used by the next layer in the backward pass.

(5) The initial loss for the convolutional layers comes from the units that were flattened and used in the dense layers. Using conventional backpropogation those weights can be figured, and then those can be backpropogated

(6) In case maxpooling is employed, then it is picking one of many features from the feature maps, that get to contribute to the o/p, hence the weights get updated wrt only those features.