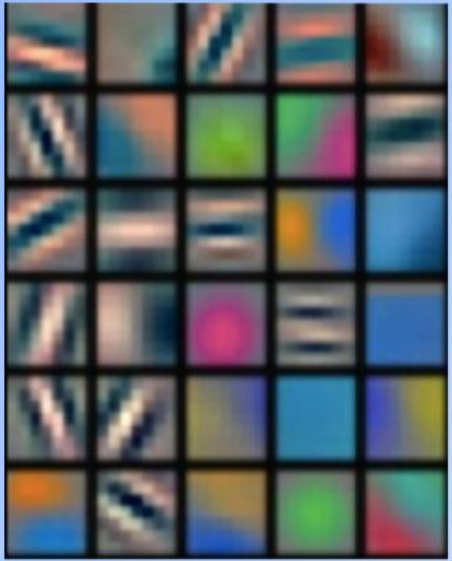
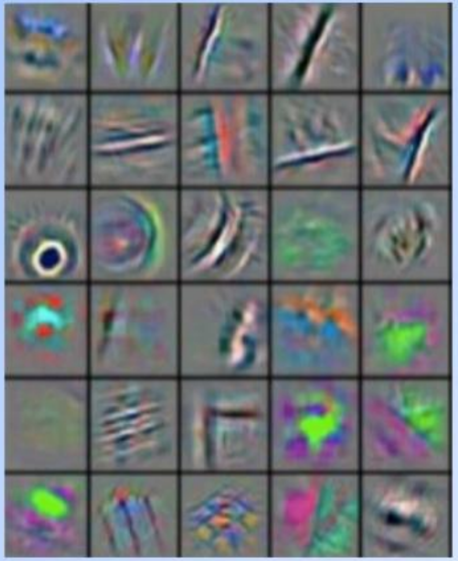
Learning Hierarchical Abstractions with CNNs

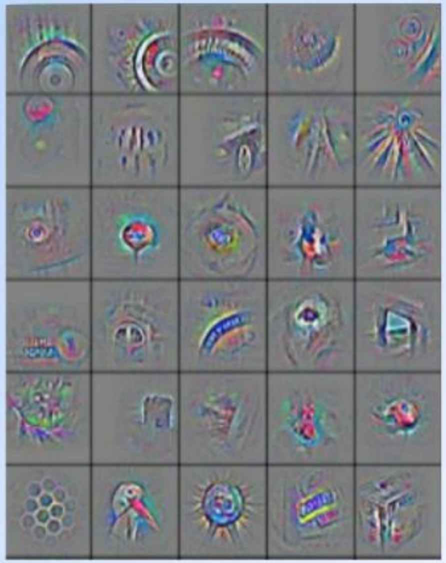
Low-level features



Mid-level features



High-level features

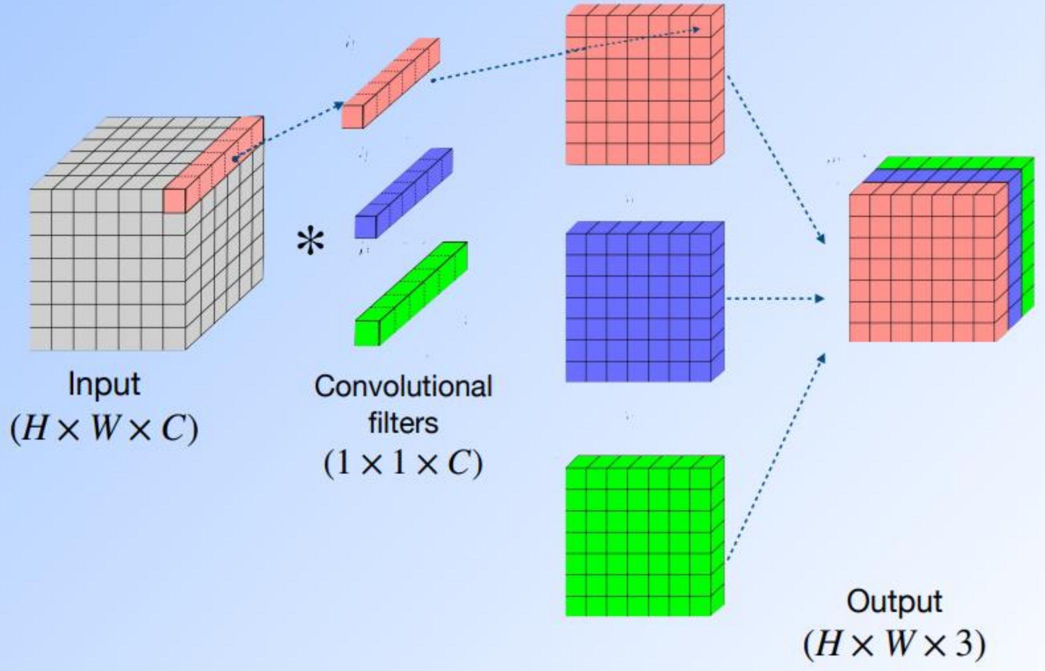


Visualization of learned CNN filters at different depths

Types of Convolutional Filters

1×1 Convolutions

- 1×1 convolutions, also known as pointwise convolutions operates on a single element spatially, but across all the channels of the input.
- The primary purpose of pointwise convolutions is channel mixing, i.e. it essentially performs a weighted sum of its input channels.
- It is also used for controlling or changing the channel dimension of the feature map, without altering its spatial dimensions.
- Common usages include:
  - Replacing FC/dense layers with 1×1 convolution and global average pooling.
  - Creating bottleneck layers
  - Designing other modules (e.g. inception module, skip connections etc.)

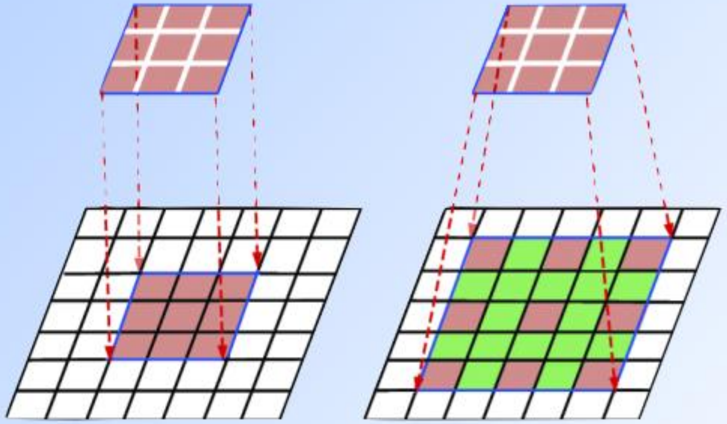


1×1 convolution with a layer having 3 convolutional filters

Types of Convolutional Filters

Dilated Convolutions

- Dilated convolutions or atrous convolutions serve to increase the RF size without increasing the number of parameters or the number of computations.
- The RF field size is increased by adding gaps or holes between the filter elements, or equivalently by skipping input elements while performing the convolution operation.
- The number of pixels skipped is determined by the dilation rate.
- If the dilation rate is d, elements of the kernel will be spaced apart by $d - 1$ along both dimensions while performing convolution.



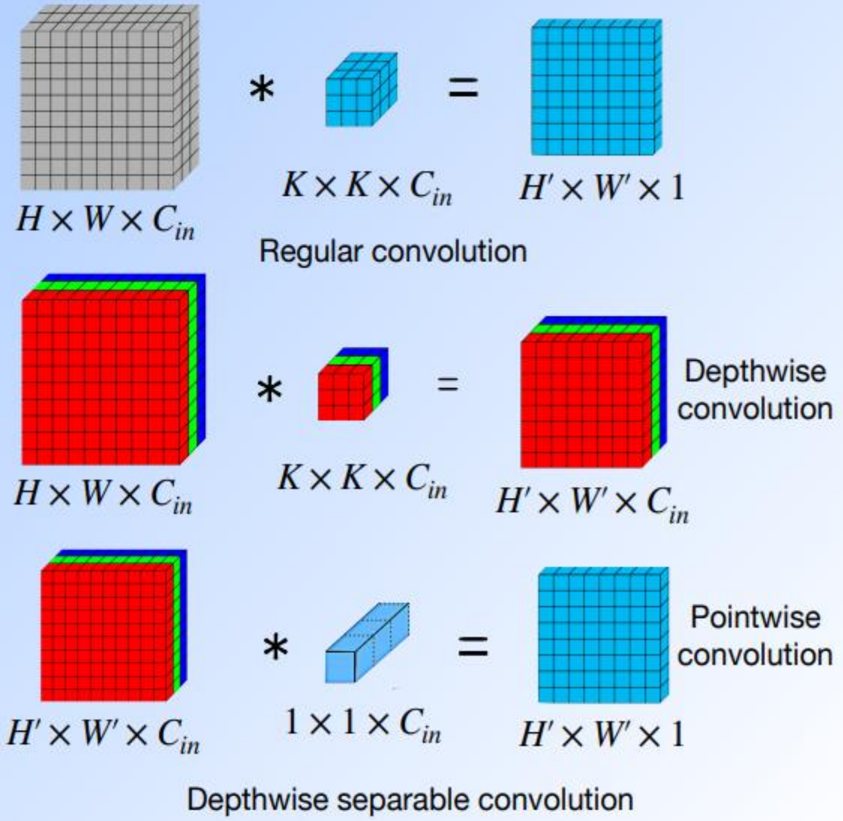
Regular convolution

Dilated convolutions with $d = 2$

Types of Convolutional Filters

Depthwise Separable Convolutions

- Let us consider an input feature map having dimensions $H \times W \times C_{in}$.
- In case of normal convolutions, each convolutional filter having spatial dimension $K \times K \times C_{in}$ produces an output feature map of size $H' \times W' \times 1$.
- C_{out} such convolutional filters are used in the convolutional layer to get a feature map of size $W' \times H' \times C_{out}$.
- In case of depthwise separable convolutions the convolutional filter of size $K \times K \times C_{in}$ act on each channel separately to produce an output feature map of dimension $W' \times H' \times C_{in}$.
- C_{out} number of $1 \times 1 \times C_{in}$ convolutional filters are then applied to get the final feature map of size $W' \times H' \times C_{out}$.



$*$
$=$
$H \times W \times C_{in}$
$K \times K \times C_{in}$
$H' \times W' \times 1$
Regular convolution
$*$
$=$
Depthwise convolution
$H \times W \times C_{in}$
$K \times K \times C_{in}$
$H' \times W' \times C_{in}$
$*$
$=$
Pointwise convolution
$H' \times W' \times C_{in}$
$1 \times 1 \times C_{in}$
$H' \times W' \times 1$
Depthwise separable convolution

Types of Convolutional Filters

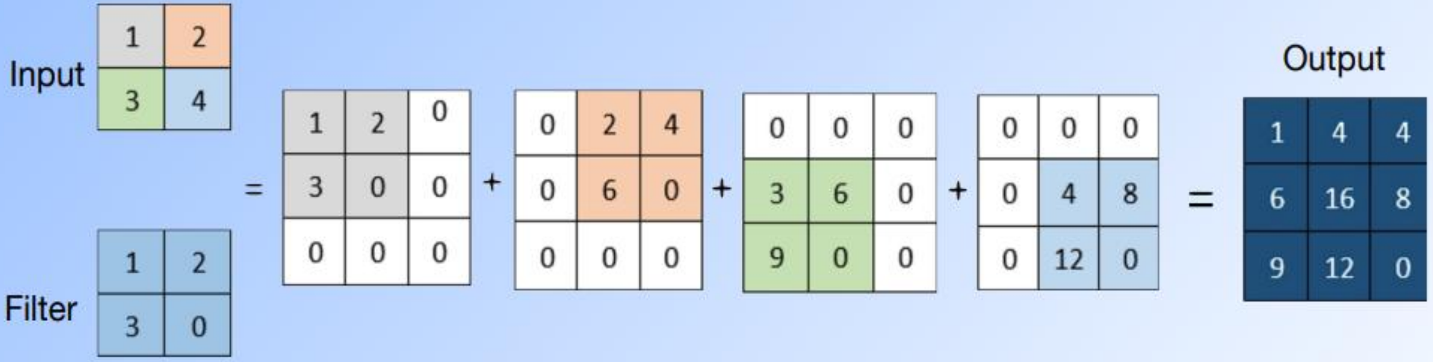
Depthwise Separable Convolutions

- Computational complexity of regular convolutional layer:
  $O(C_{out} \times C_{in} \times K \times K \times W \times H)$
- Computational complexity of depthwise separable convolutional layer:
  $O(C_{in} \times K \times K \times W \times H + C_{out} \times C_{in} \times W \times H)$
- Thus, depthwise separable convolutions are a popular choice for architectures that are designed for computational efficiency.
- Depthwise separable convolution also reduces the number of trainable parameters, which can help to reduce overfitting as well as memory overheads.
- Despite the significant reduction in the number of parameters and computations, depth wise separable convolutions can still achieve competitive performance as demonstrated by popular architectures such as MobileNet, Xception, EfficientNet etc.

Types of Convolutional Filters

Transposed Convolutions

- While regular convolution reduce the spatial dimensions of the input feature map (for strides > 1), transposed convolution or deconvolution serves to increase the spatial dimensions of the input feature map.
- The transposed convolution is thus a learned upsampling layer.
- Instead of computing the sum of products of the kernel as in regular convolution, the products of each input element with the kernel elements is distributed spatially and overlapping regions are summed to generate the output.



$$\text{Input} = \begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}, \quad \text{Filter} = \begin{bmatrix}1 & 2\\ 3 & 0\end{bmatrix}$$

$$= \begin{bmatrix}1 & 2 & 0\\ 3 & 0 & 0\\ 0 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 2 & 4\\ 0 & 6 & 0\\ 0 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0 & 0\\ 3 & 6 & 0\\ 9 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0 & 0\\ 0 & 4 & 8\\ 0 & 12 & 0\end{bmatrix} = \begin{bmatrix}1 & 4 & 4\\ 6 & 16 & 8\\ 9 & 12 & 0\end{bmatrix}$$

Transposed Convolution with stride of 1.

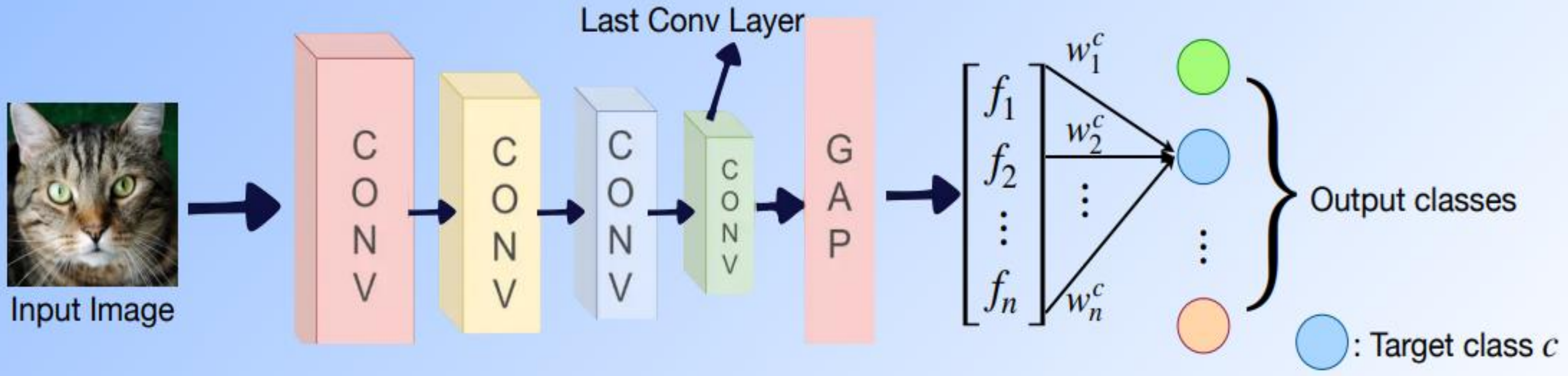
Types of Convolutional Filters

Transposed Convolutions

- The kernel size and stride of the transposed convolutions determines the extent to which the input is upsampled.
- The size of the output feature map for transposed convolution is given as:
  $W_0 = K + (W_i - 1) \times S - 2P$
  K : kernel size, S : stride, P : padding width, W_i : dimension of input feature map.
- Transposed convolutions are used to implement learned upscaling of feature maps in many popular CNN architectures such as U-Net.

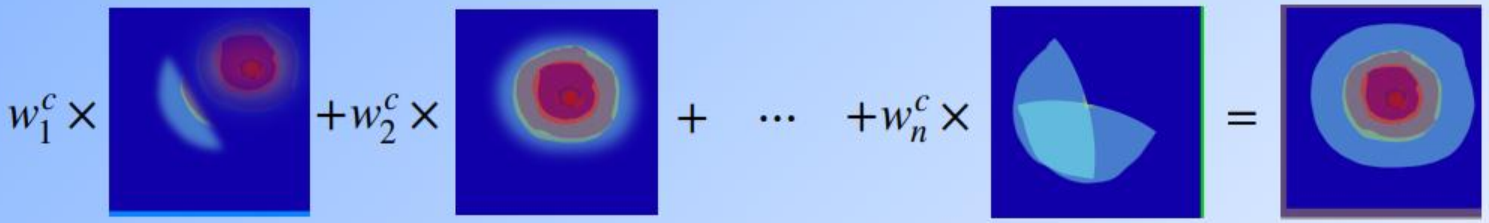
Class Activation Maps

- Class Activation Maps or CAMs highlight which regions of the image contribute the most to the model's prediction by utilizing the weights and feature maps of the final convolutional layer (Zhou et al., 2016).
- While gradient based regions highlight pixel level importance, CAMs are designed to highlight regional importances which make them more interpretable.
- CAM uses global average pooling (GAP) after the last convolutional layer to reduce the spatial dimensions of the feature map after the last convolutional layer to a single element for each channel.



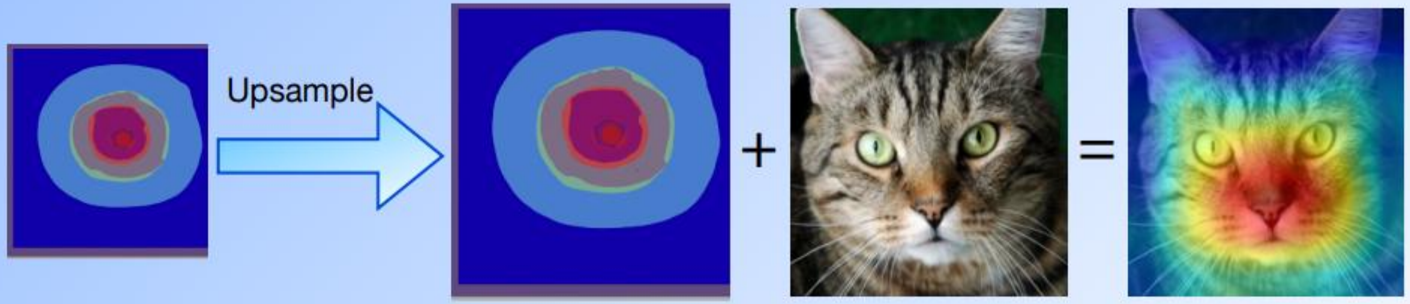
Class Activation Maps

- To generate the CAM for a particular class, the weights learned by the linear classification layer are used to weight the feature maps learned by the last convolutional layer.



$$w_1^c \times \square + w_2^c \times \square + \cdots + w_n^c \times \square = \square$$

- The weighted feature map is finally upsampled to the size of the input image to get the CAM result.



Gradient-Weighted CAMs

- CAMs require the CNN architecture to have a GAP layer before the final classification layer, which may not be suitable for all kinds of vision tasks.
- Gradient-weighted CAMs (Grad-CAMs) overcame the limitation of having a GAP layer by computing the gradient of the class logits with respect to the feature map from any convolutional layer (Selvaraju et al., 2017), i.e. $w_k^c = \frac{1}{Z}\sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$, where A^k is the kth channel of a convolutional layer's feature map, Z is the total number of pixels in A^k, and y^c is the logit corresponding to class c.
- A weighted average of the feature maps are taken as before, which is then subjected to a ReLU activation. $S^c = \text{ReLU}\left(\sum_k w_k^c A^k\right)$.
- The class specific Grad-CAMs are then upsampled to the input resolution.

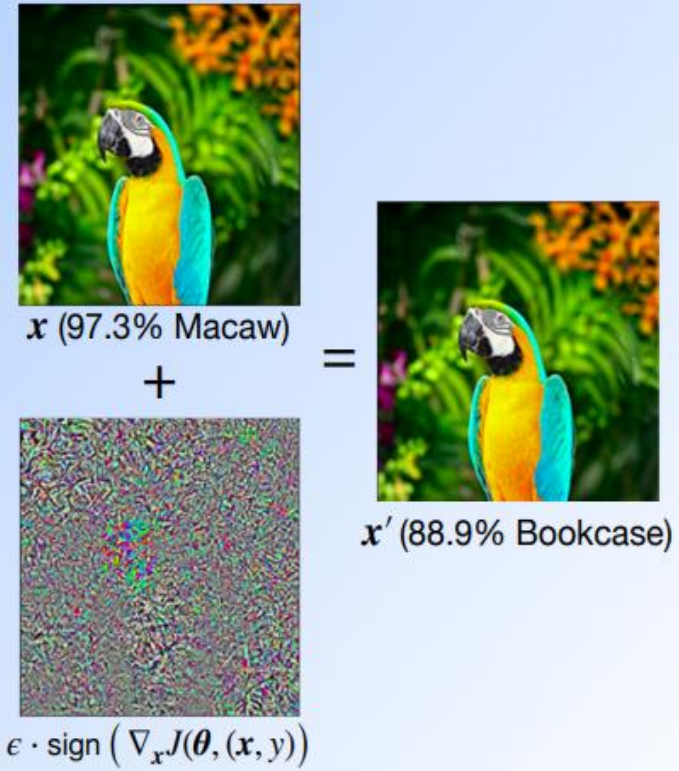
Adversarial Attacks on CNNs

- Adversarial attacks seek to fool a trained deep neural network such as a CNN by changing the input in a subtle and imperceptible manner.
- An adversarial image is an image that has been modified to be misclassified by the trained classifier, but is visually indistinguishable from the correct class.
- Adversarial attacks could be of the following types:
  - Poisoning vs. evasion attacks: poisoning attacks take place during training by altering training samples, while evasion attacks target a trained model.
  - Black box attacks vs. white box attacks: in case of white box adversarial attacks, the attacker possesses complete knowledge of the model architecture and its trained weights, while in case of black box attacks, the attacker has no access to the internal specifications and model weights, and can only observe the model's output in relation to specific inputs.
  - Untargeted vs. targeted attacks: the goal of untargeted attacks is to make the model misclassify the adversarial input, while the goal of targeted attacks is to make the model misclassify the adversarial input as a specific target class.

Common Adversarial Attacks

White box Adversarial attacks commonly used gradient based methods to modify inputs:

- **Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2015):** FGSM is a computationally efficient technique that introduces pixel-level perturbations in the direction of the gradient of the cost with respect to the input as follows: $\boldsymbol{x}' = \boldsymbol{x} + \epsilon \cdot \text{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)\right)$, where $\boldsymbol{x}$ is the original input, y is the input label, $\boldsymbol{\theta}$ represents the model parameters, and $\boldsymbol{x}'$ is the adversarial input.
- **Targeted FGSM**: In targeted FGSM, the input is perturbed in the direction of the negative gradient of the target class as follows: $\boldsymbol{x}' = \boldsymbol{x} - \epsilon \cdot \text{sign}\left(\nabla_{X} J(\boldsymbol{\theta}, \boldsymbol{x}, y^c)\right)$, where y^c is the label corresponding to the target class.


$\boldsymbol{x}$ (97.3% Macaw)
+
=
$\epsilon \cdot \text{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, (\boldsymbol{x}, y))\right)$
$\boldsymbol{x}'$ (88.9% Bookcase)