AI61201: Visual Computing With AI/ML

Module 7: CNNs for Visual Recognition

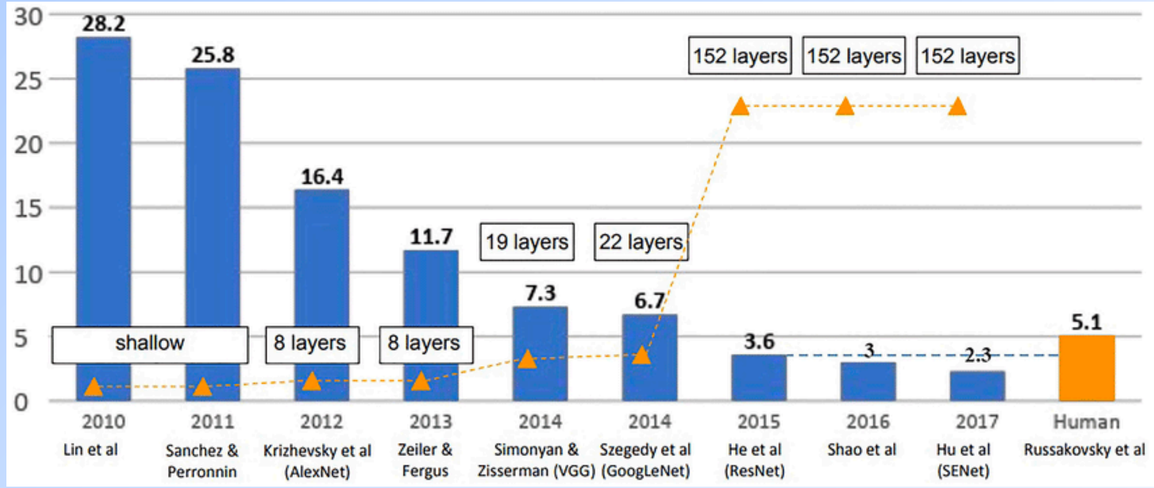
Dr. Somdyuti Paul

Feature Design Vs. Feature Learning

- In the last module we looked at some low and mid level features that could be designed to train ML models for visual recognition tasks.
- While feature design yielded promising results on early vision tasks, they also suffered from prominent drawbacks:
  - Requires domain specific knowledge to manually engineer effective features
  - Does not generalize well on diverse data
- In contrast, feature learning involves automatically discovering representations or features from raw data, often using deep neural networks.
- Feature learning is generally more computationally intensive than feature design, and often require larger volumes of data to learn effective representations.

The Deep Learning Era

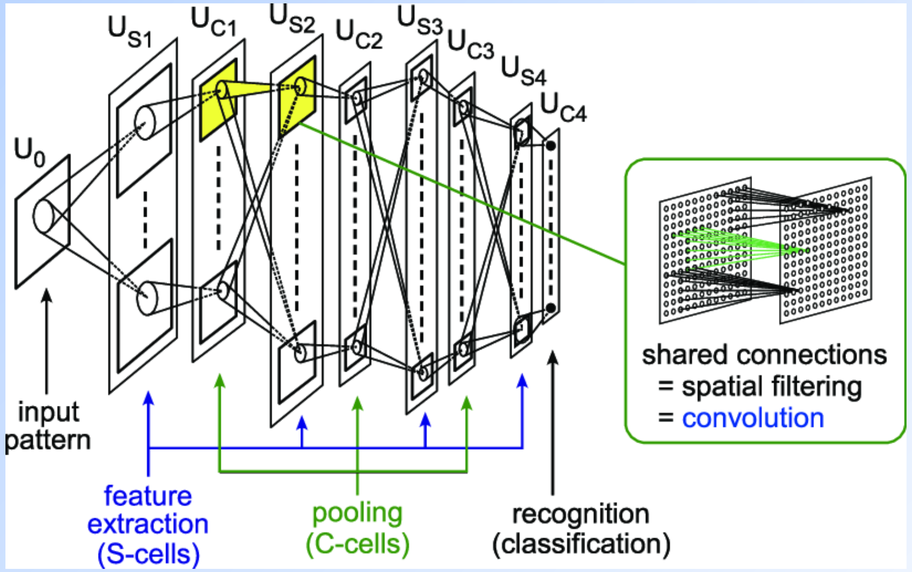
- In 2012, a large scale image classification challenge called ILSRVC was won by a model called AlexNet (Krizhevsky et al., 2012) that learned visual features using CNNs.
- This major breakthrough marked the end of the AI winter
- Since then, CNNs based feature learning have pioneered rapid advancement of the field, attaining unprecedented performance levels on key vision tasks, with models progressively becoming deeper and more complex.
- Although neural networks have been in use since early 1980s, the following factors propelled their resurgence in the form of deep learning:
  - Better hardware availability
  - Availability of large scale datasets
  - Algorithmic advances
  - Large scale investments



Performance of ILSRVC challenge winners

Biological Inspirations for the Modern CNNs

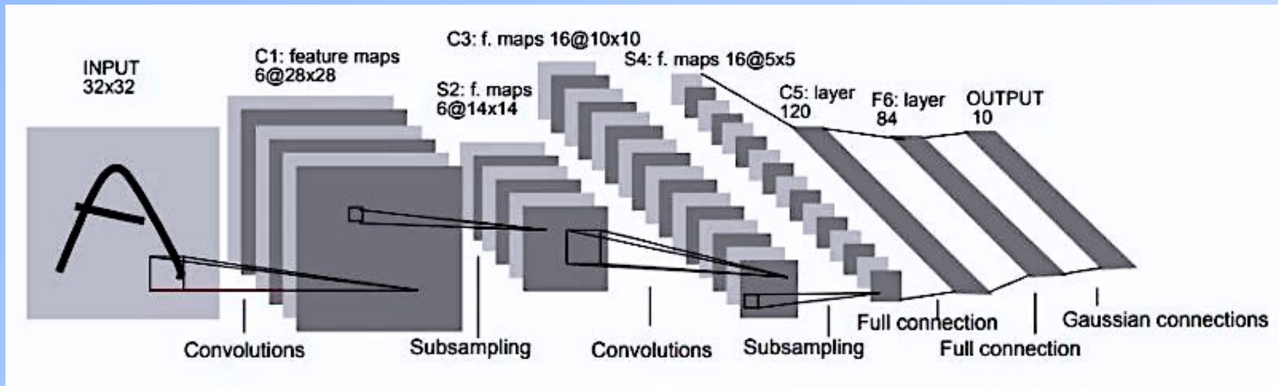
- The earliest version of the modern CNN was proposed for the recognition of handwritten digits in 1980.
- The corresponding architecture, known as Neocognitron (Fukushima, 1980) was inspired by Hubel and Wiesel's model of the primary visual cortex, consisting of simple and complex cells.
- The neocognitron architecture consisted of alternating stacks of S-cells and C-cells
  - S - cells: feature extractors with localized receptive fields.
  - C-cells: processes S-cell outputs to achieve position invariance.
- The neocognitron model was trained using unsupervised learning for pattern recognition.



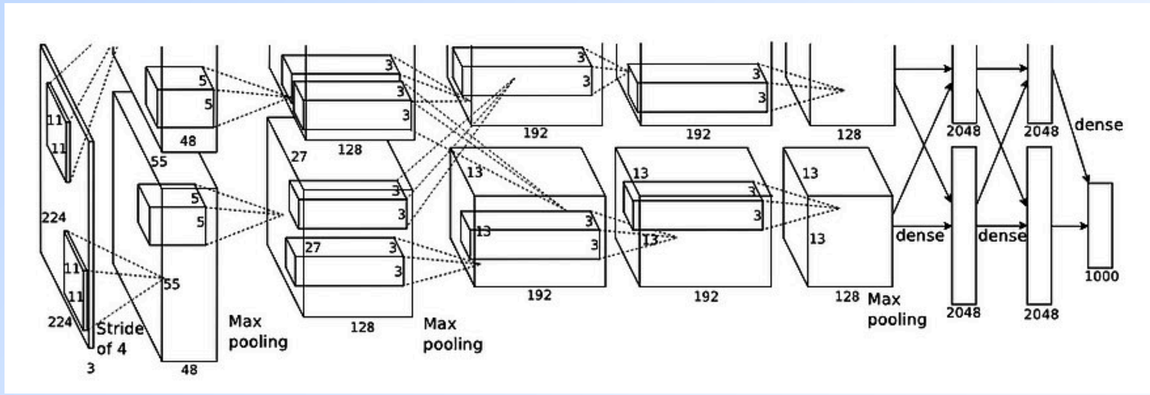
Architecture of Neocognitron

CNNs for Visual Recognition

- Subsequent advancements led to the LeNet-5 (LeCun et al.,1998), which was a CNN model consisting of 5 layers that was trained in a supervised manner using backpropagation.
- LeNet-5 did not become popular at that time due to hardware limitations.
- AlexNet retained the architectural features of LeNet-5, while designing a deeper model trained on a larger dataset.



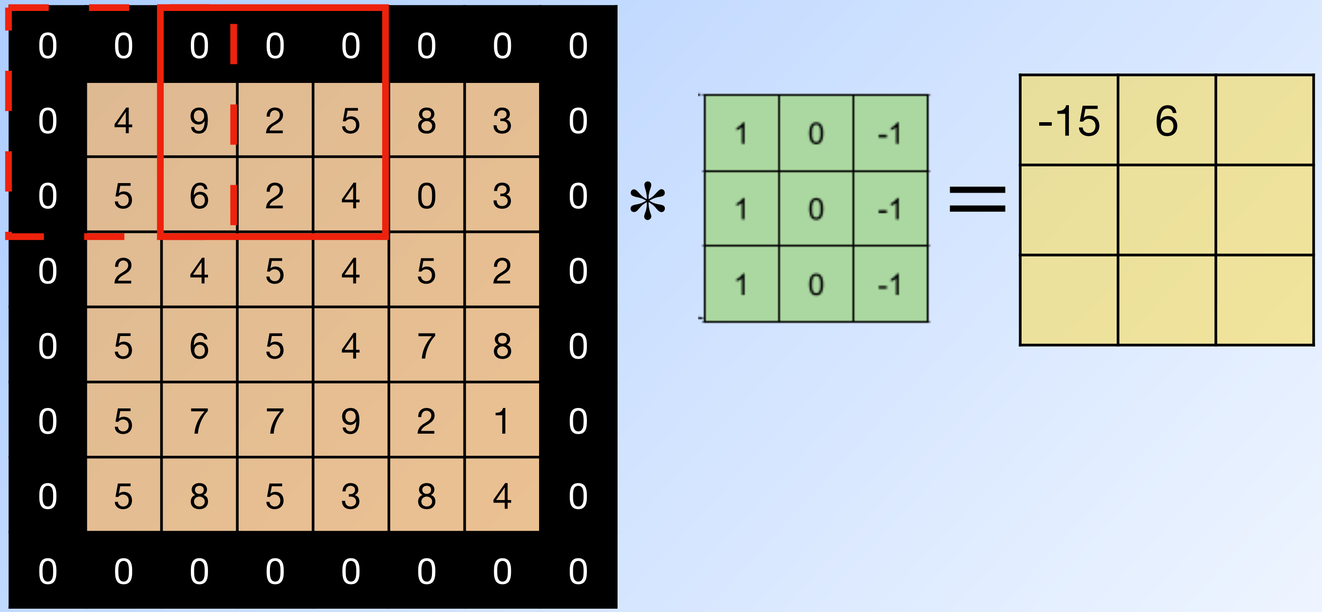
Architecture of LeNet-5



Architecture of AlexNet

Convolutional Filters

- The building blocks of a CNN are the convolutional filters, which are used to perform convolutions between the input image or feature map, and the filter coefficients, learned from the data through back propagation.
- A convolutional filter is defined by the following hyperparameters:
  - *Kernel size*: spatial dimensions of the filter
  - *Stride*: number of steps by which the filter slides over the input
  - *Padding width*: number of extra rows and columns added along the boundary of the input



$$\lfloor W_o = \frac{W_i + 2P - K}{S} + 1 \rfloor$$

W_o : size of output feature map, W_i : size of input image/feature map

P: padding size, K : filter/kernel size, S : stride

Convolutional Filters

- Padding the borders of the input image or feature map is used to control the size of the output feature map after convolution.

- Different types of padding could be applied, depending on the application.

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	5	3	8	1	6	0	0
0	0	4	7	2	9	0	0	0
0	0	3	5	7	8	1	0	0
0	0	2	6	0	4	3	0	0
0	0	9	1	5	2	7	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Zero/Constant Padding

5	5	5	3	8	1	6	6	6
5	5	5	3	8	1	6	6	6
5	5	5	3	8	1	6	6	6
4	4	4	7	2	9	0	0	0
3	3	3	5	7	8	1	1	1
2	2	2	6	0	4	3	3	3
9	9	9	1	5	2	7	7	7
9	9	9	1	5	2	7	7	7
9	9	9	1	5	2	7	7	7

Replicate/Edge Padding

7	4	4	7	2	9	0	0	9
3	5	5	3	8	1	6	6	1
3	5	5	3	8	1	6	6	1
7	4	4	7	2	9	0	0	9
5	3	3	5	7	8	1	1	8
6	2	2	6	0	4	3	3	4
1	9	9	1	5	2	7	7	2
1	9	9	1	5	2	7	7	2
6	2	2	6	0	4	3	3	4

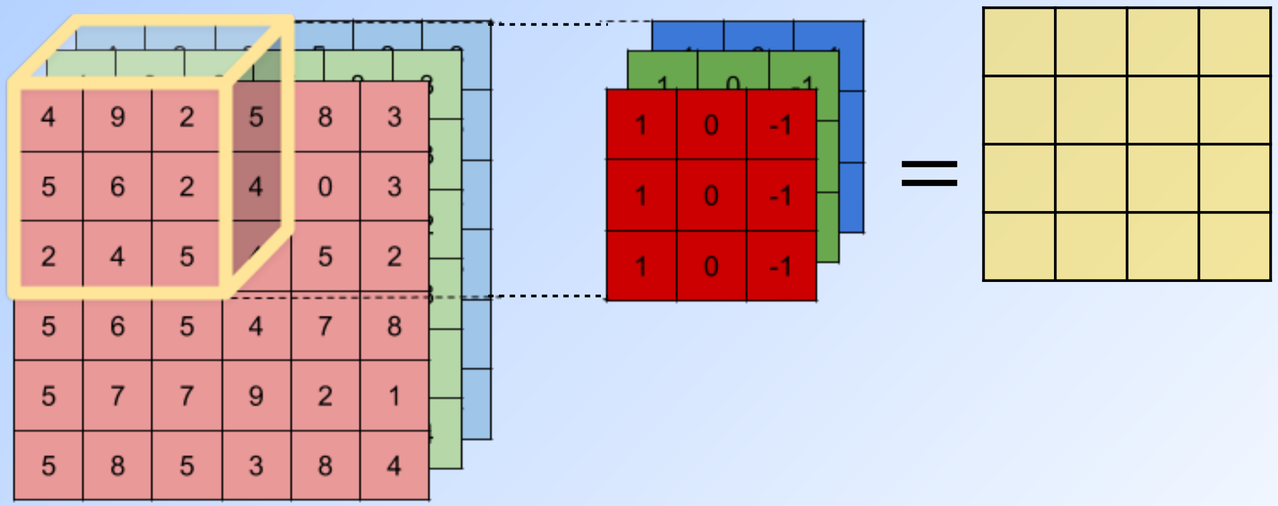
Reflective Padding

4	3	2	6	0	4	3	2	6
2	7	9	1	5	2	7	9	1
1	6	5	3	8	1	6	5	3
9	0	4	7	2	9	0	4	7
8	1	3	5	7	8	1	3	5
4	3	2	6	0	4	3	2	6
2	7	9	1	5	2	7	9	1
1	6	5	3	8	1	6	5	3
9	0	4	7	2	9	0	4	7

Circular Padding

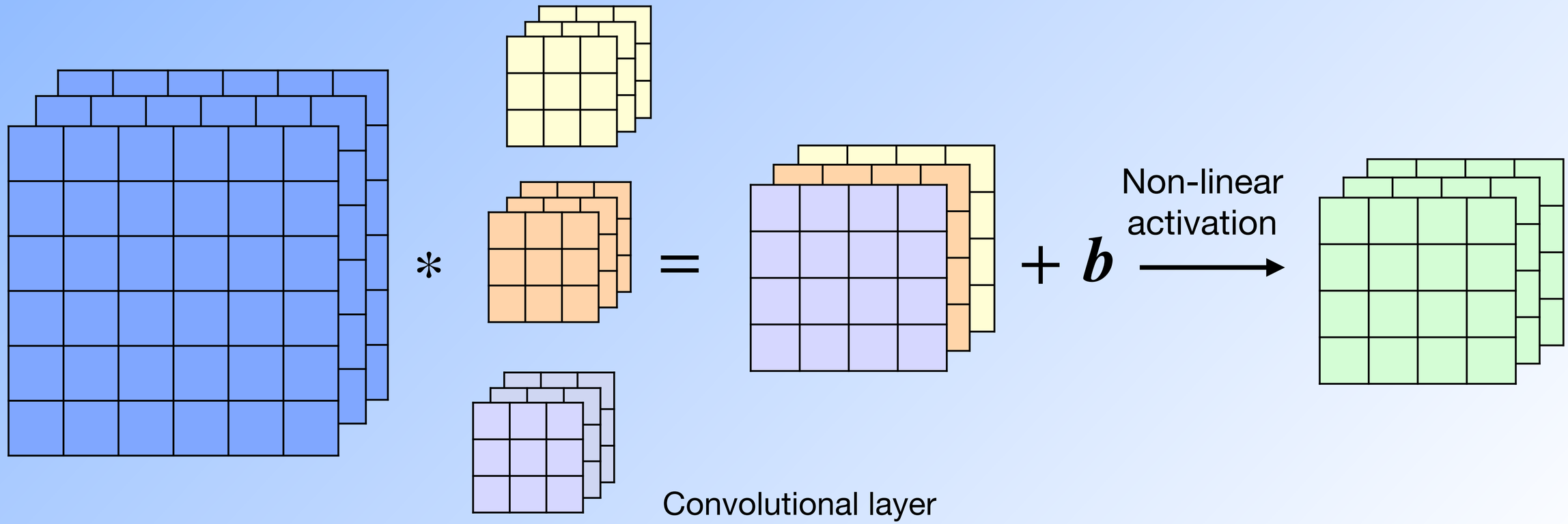
Convolutional Filters

- The number of channels of the input must match the number of channels of the convolutional filter.
- The filters are applied channel-wise and the result of the convolution operation for each channel are summed to constitute the final output.
- The use of convolutional filters achieves the following:
  - Spatial invariance
  - Preservation of spatial structure
  - Parameter sharing
  - Sparsity of connections



Convolutional Layer

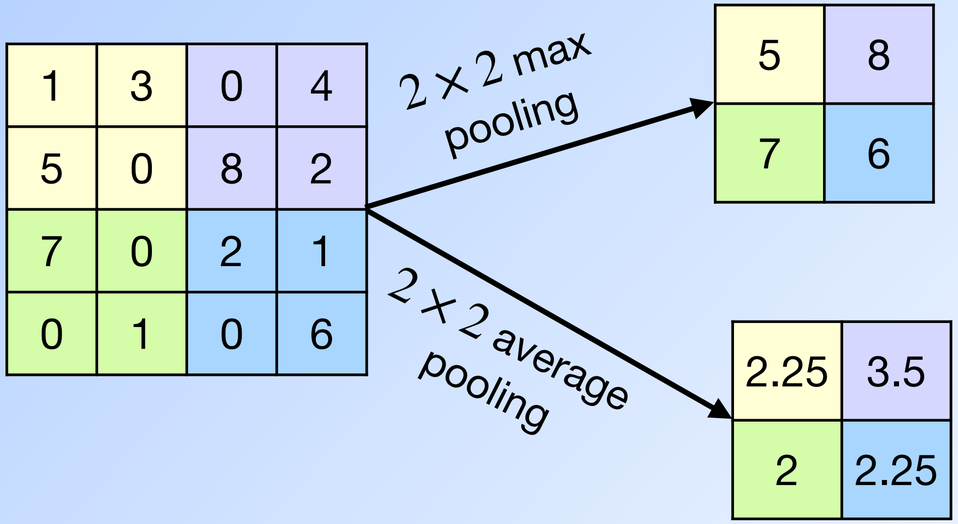
- The output has multiple channels when multiple convolutional filters are used.
- A learnable bias might be added to the result obtained after the convolution operation, followed by the element-wise application of a non-linear activation function (such as ReLU) to constitute a convolutional layer



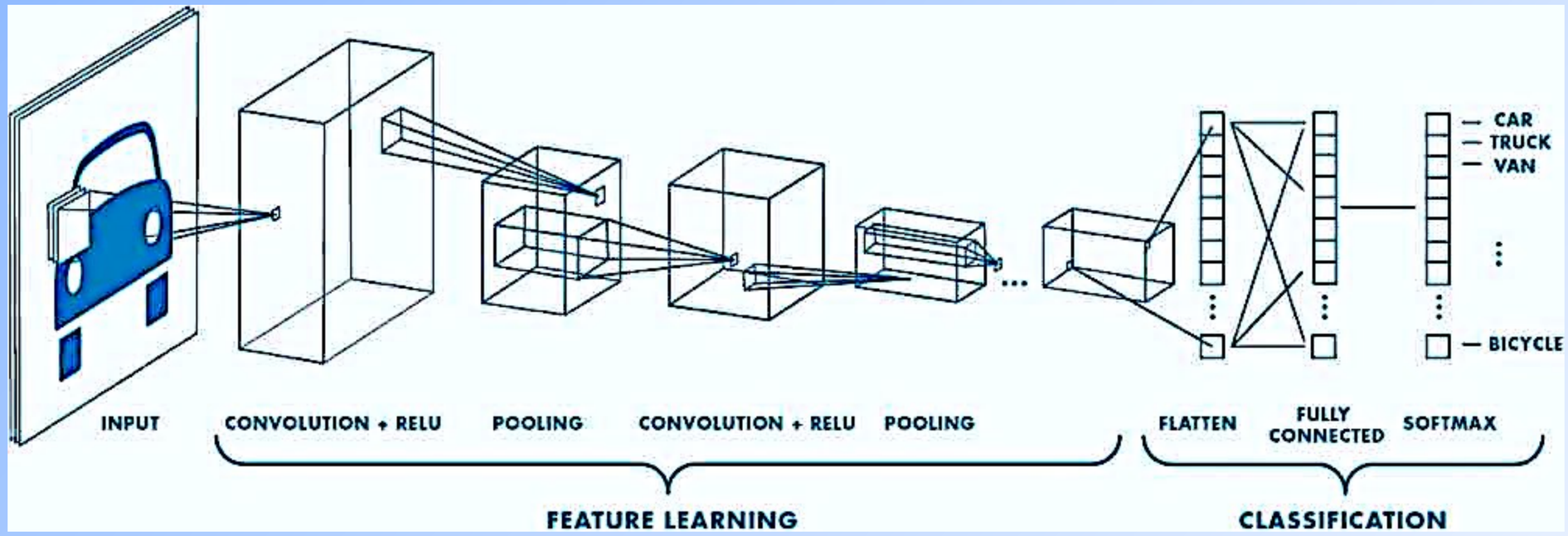
Pooling Layer

- Spatial pooling layers are often applied to the feature maps produced by the convolutional layers.
- Two types of pooling layers are most prevalent:
  - Max pooling
  - Average pooling

- Pooling layers accomplish the following:
  - Spatial invariance
  - Reduction of spatial dimensions of feature maps
  - Faster computations



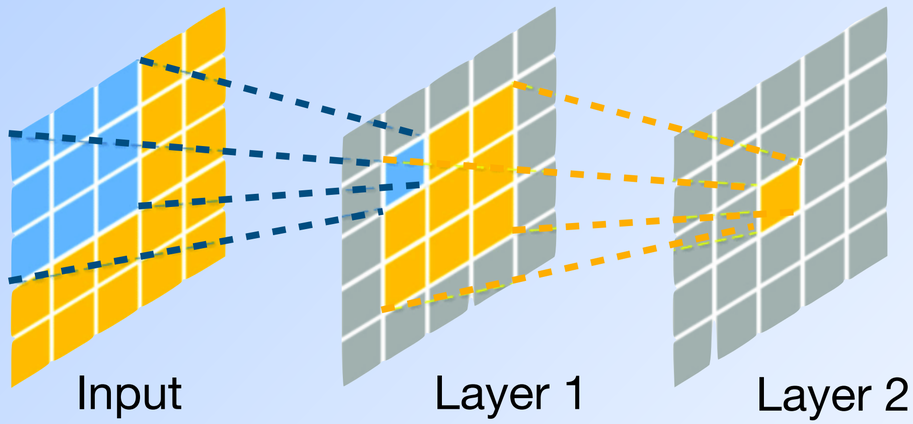
Convolutional Neural Network



- Deep feature map produced after several stages of convolutional layers is flattened and passed to a fully connected (dense) layer.
- The predictions are finally obtained by applying an appropriate activation function to the output of the fully connected layer.

Receptive Field of CNNs

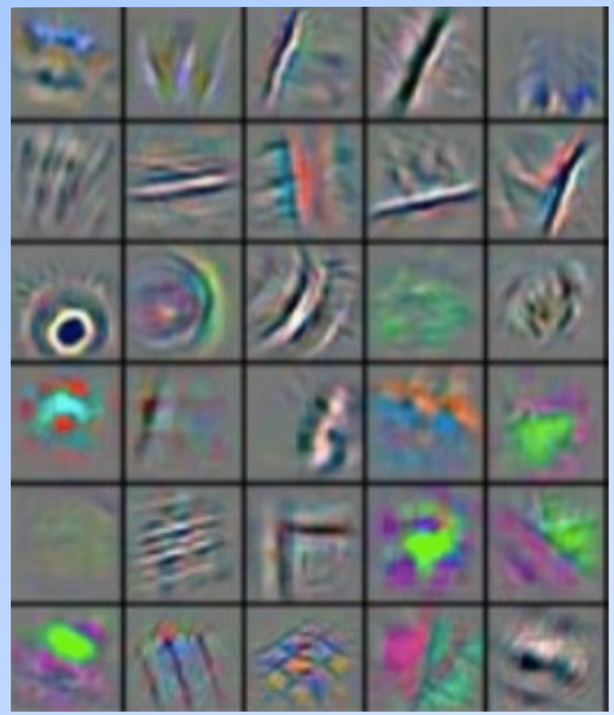
- The receptive field (RF) or field of view (FoV) of a convolutional layer is defined as the area of the input image that is “visible” to this layer.
- In other words, the RF is determined by the pixel area of the input that contributes to a single element of the output of the convolutional layer under consideration.
- The receptive field size for each successive layer of a CNN could be determined by the following recursive formula:
  - For first layer: $R_1 = K_1$, where K_1 is the size of the filter kernel for the first layer.
  - For layer l: $R_l = R_{l-1} + (K_l - 1)\prod_{i=1}^{l-1} S_i$
- Thus, the RF size increases with the depth of the CNN.



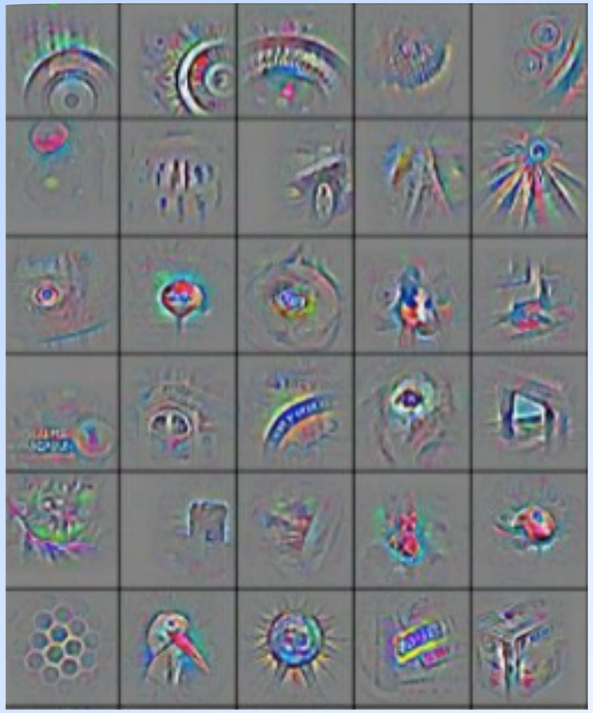
Learning Hierarchical Abstractions with CNNs

Low-level features

Mid-level features



High-level features

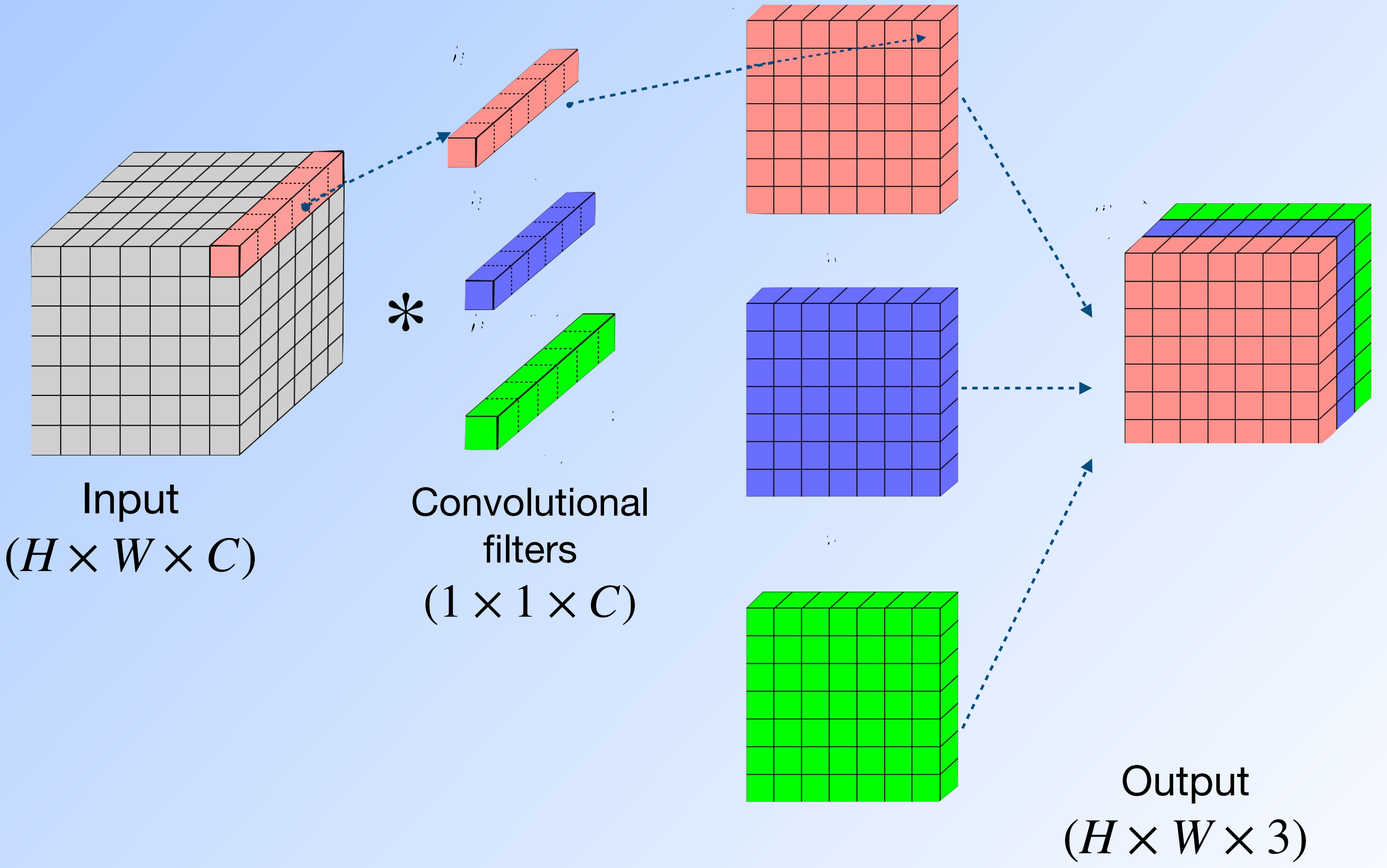


Visualization of learned CNN filters at different depths

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.)

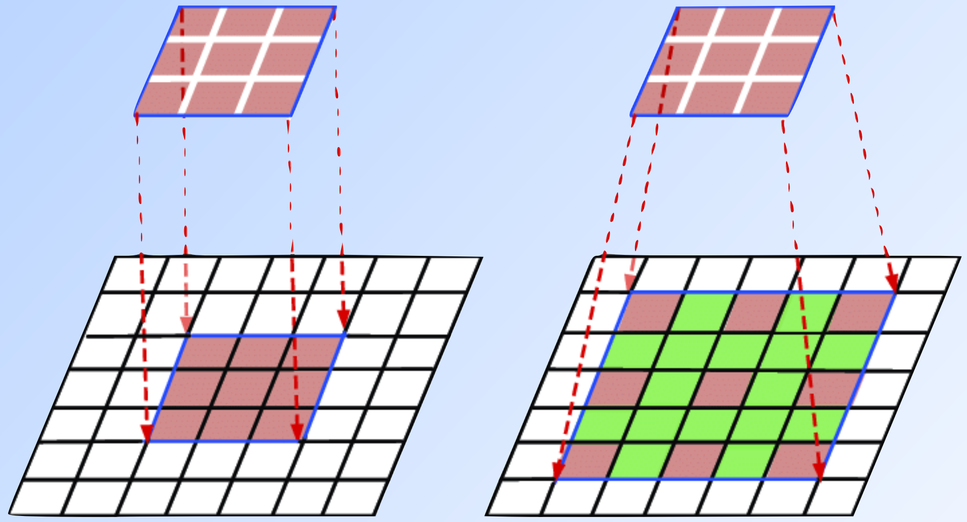


1×1 convolution with a layer having 3 convolutional filters

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

Dilated Convolutions

- Mathematically, for an input image or feature map f, and a filter kernel w, with a dilation factor of d, the dilated convolution operation is represented as:

$$y(i,j) = \sum_{m=-k}^{k} \sum_{n=-k}^{k} f(i + m \cdot d, j + n \cdot d) \cdot w(m,n)$$

- The RF size of a convolutional layer having a kernel size K and dilation factor of d is given by:

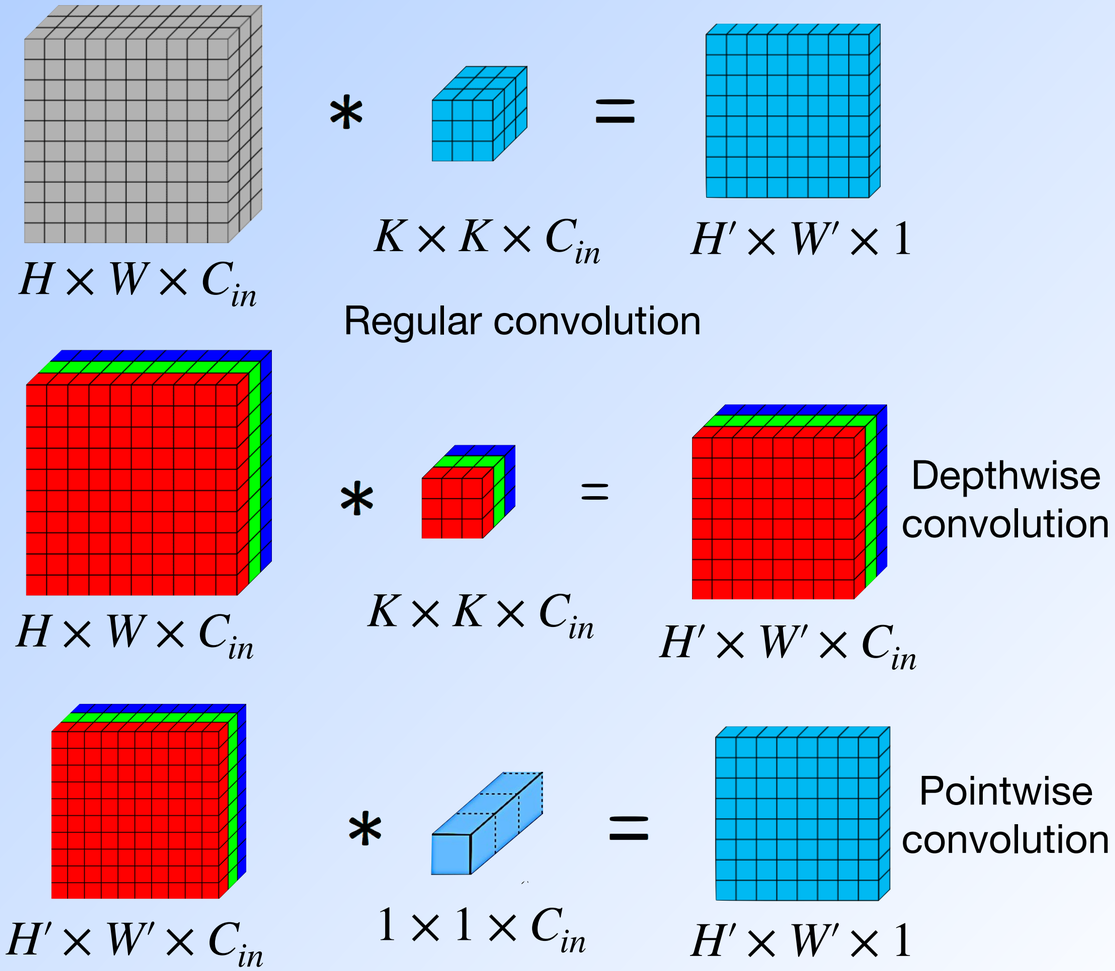
$$R = K + (K-1) \cdot (d-1)$$

- Effectively, dilated convolution subsamples the input image/feature map. So, a large dilation rate can introduce undesirable gridding artifacts in the output.

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 $W' \times H' \times C_{out}$.



Types of Convolutional Filters

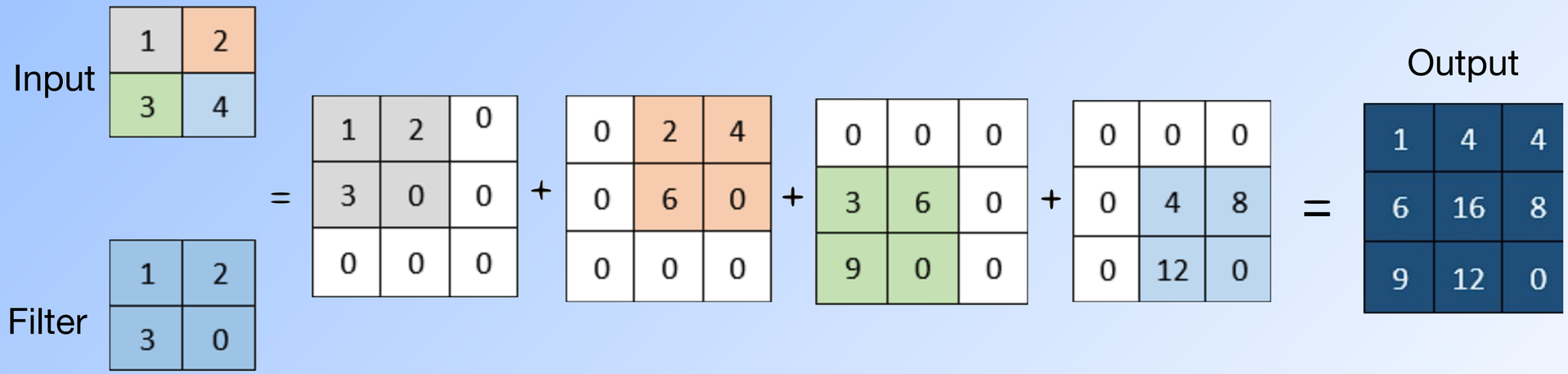
Depthwise Separable Convolutions

- 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.

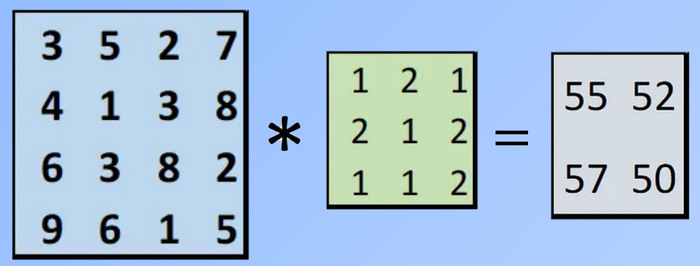


Transposed Convolution with stride of 1.

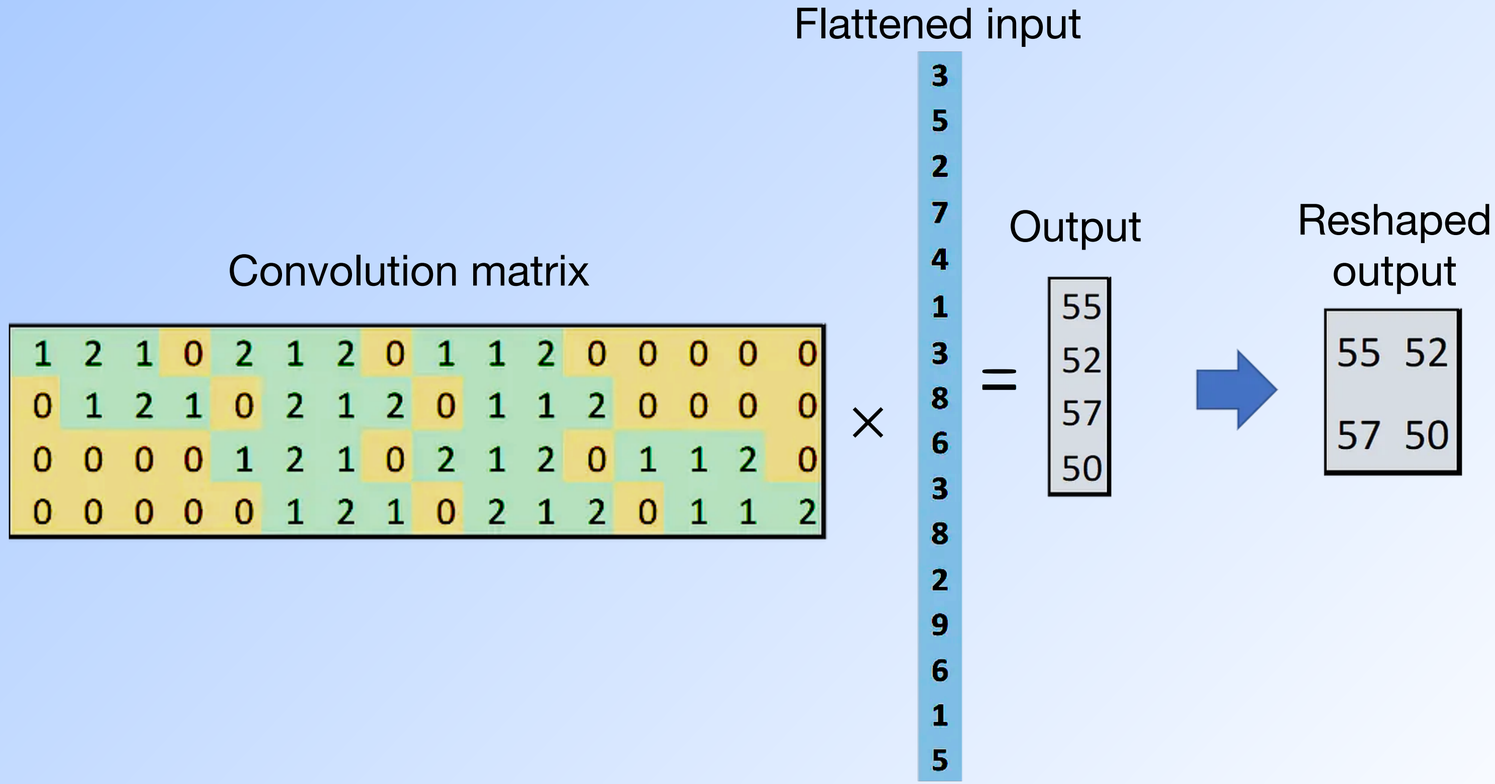
Types of Convolution Operations

Transposed Convolutions

- Regular convolutions could be expressed as a matrix multiplication between a convolution matrix and the flattened input.



Ordinary convolution (stride =1)

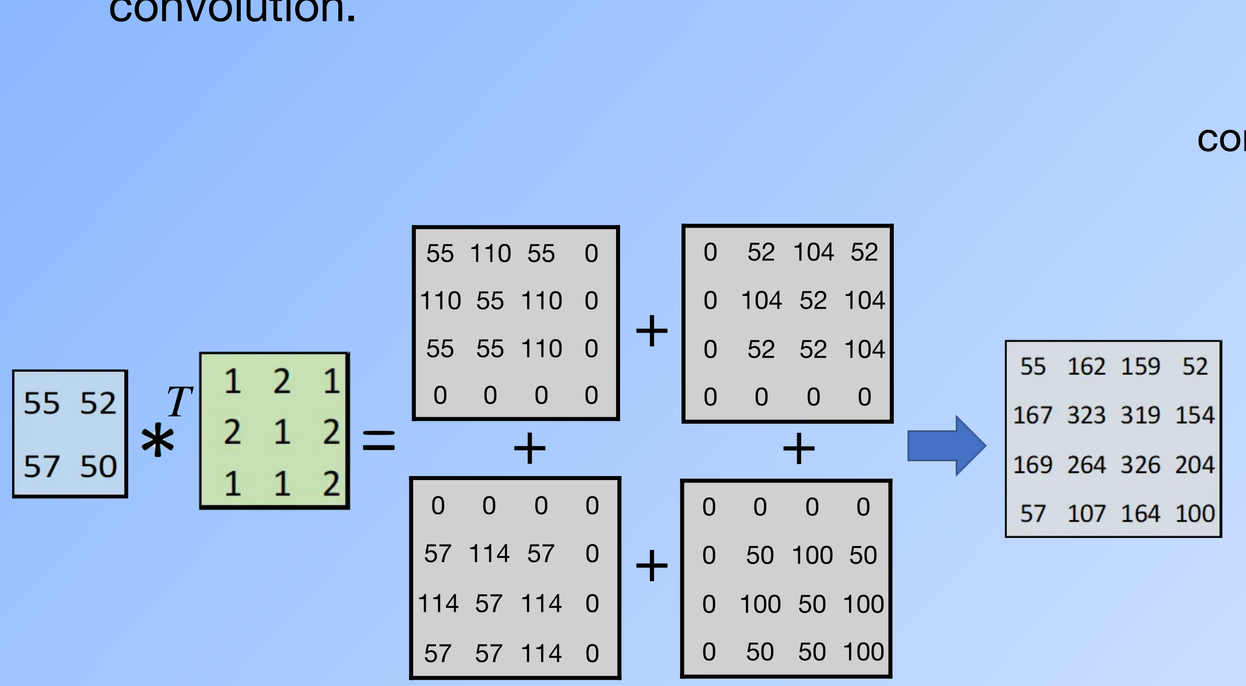


Ordinary convolution as matrix multiplication

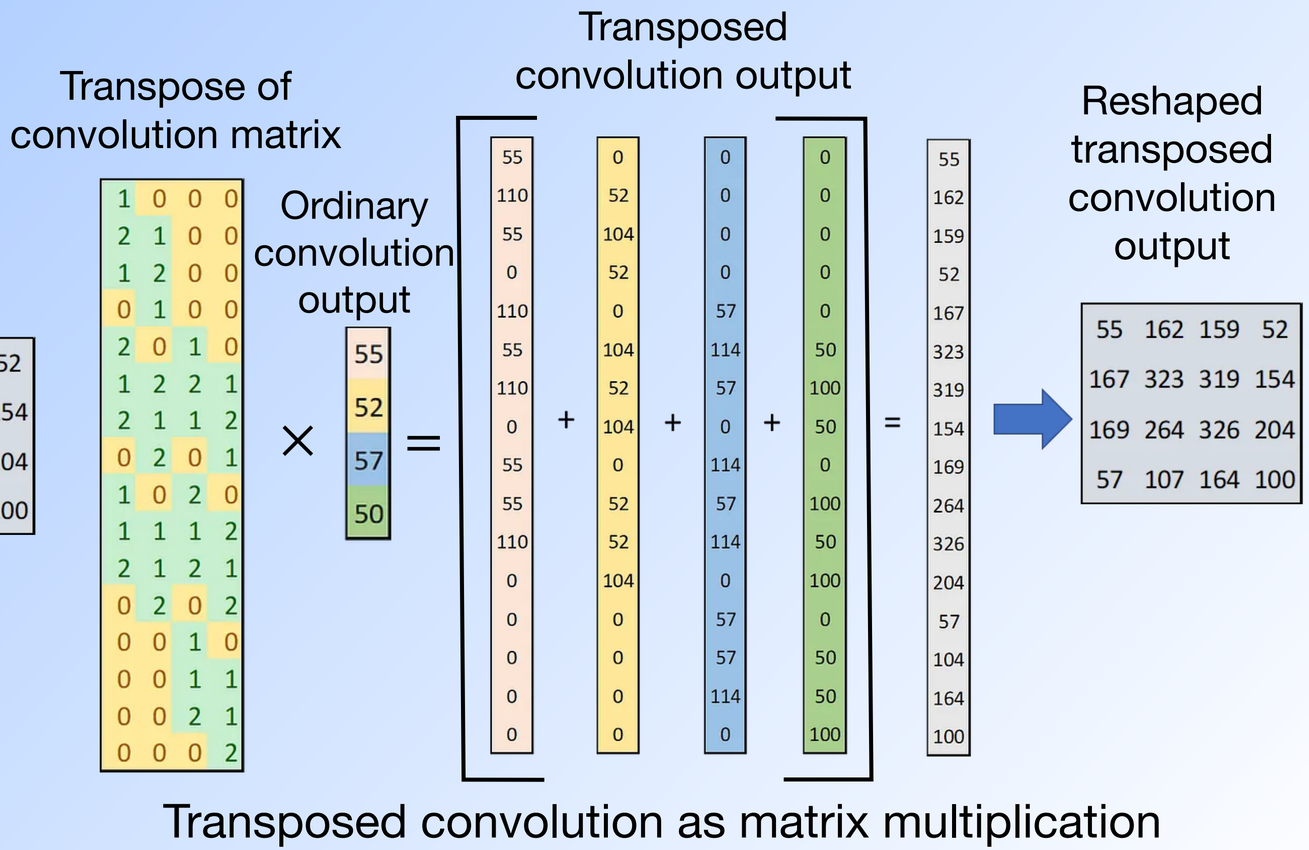
Types of Convolutional Filters

Transposed Convolutions

- The transpose of the convolution matrix multiplied by the regular convolution output gives the result of transposed convolution.



Transposed Convolution (stride=1)



Transposed convolution as matrix multiplication

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.

Popular CNN Architectures for Visual Recognition

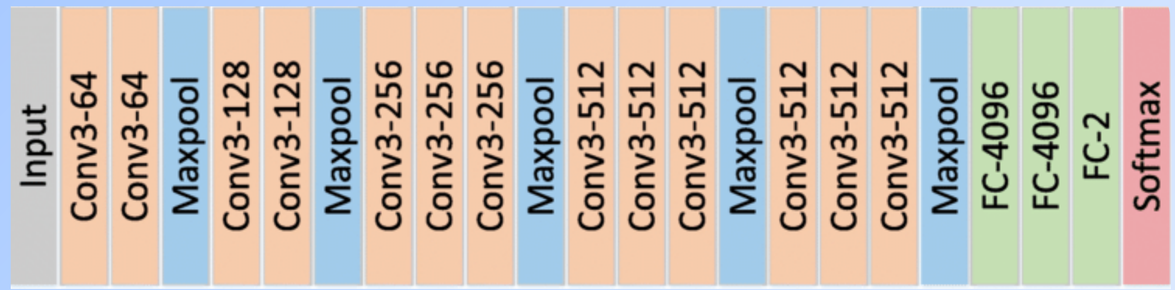
- The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) has driven advancements in visual recognition since 2010.
- The challenge was conducted on the ImageNet dataset, which provides over 1.2 million annotated images belonging to 1000 classes.
- From 2012 - 2015, the classification error rate reduced 10 times, and eventually surpassed human performance level!
- Many popular CNN architectures such as VGG, Inception-v1 and ResNet emerged from the ILSVRC challenge.
- These CNN models are still widely used as learned feature extractors (using transfer learning) for performing diverse vision tasks.

Model	ILSVRC Challenge result	# of convoluotional layers	# of parameters	Top 5 error rate (%)
AlexNet	2012 - winner	8	62 million	16.4
ZFNet	2013 - winner	8	_	11. 7
GoogLeNet/ Inception-v1	2014 - winner	22	5 million	6.67
VGG-16	2014 - 2nd place	16	138 million	7.4
ResNet-152	2015 - winner	152	115.6 million	3.57

The VGG Model

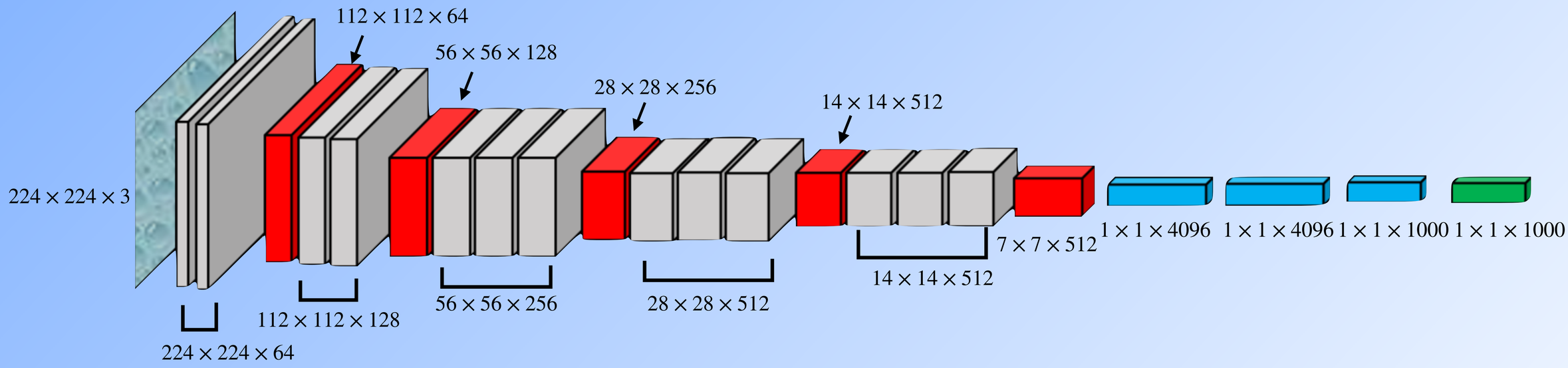
- The VGG model (Simonyan and Zisserman, 2014) consists of a series of convolutional layers interspersed with pooling layers.
- 3×3 convolutional filters and ReLU activation functions were used throughout the network.
- Employs dropout to control overfitting.
- Fully connected layers are employed at the end to combine the features learned by the convolutional layers.

VGG-16



VGG-19

The VGG Model

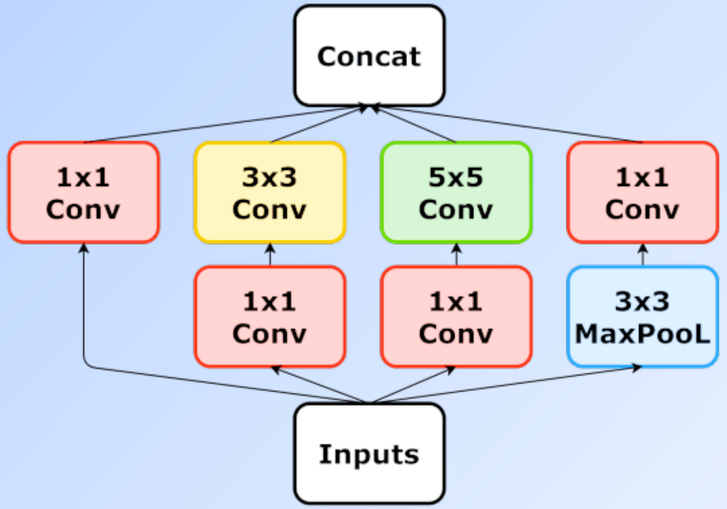


Feature maps of the VGG-16 model

- The VGG is a foundational vision model that is widely used in vision tasks due to its simplicity and effectiveness.

The Inception-v1 Model

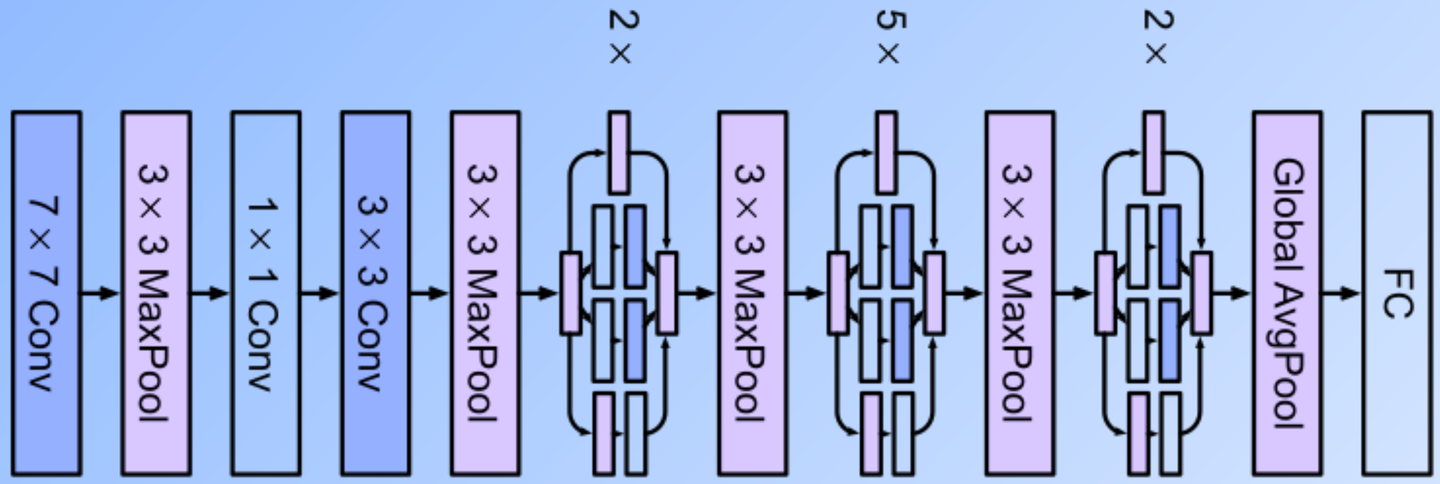
- The Inception-v1/GoogLeNet model introduced Inception modules which consist of parallel convolutional layers having convolutional filters of different sizes as well as pooling layers.
- 1×1 convolutions were used to reduce the channel dimensions of the inputs to layers having larger convolutional kernels like 3×3 and 5×5.
- The Inception modules allow the model to capture features at different scales and learn richer representations.
- Inception-v1 uses global average pooling (GAP) instead of FC layers at the end.
- The number of trainable parameters is considerably smaller as compared to other models having similar performance



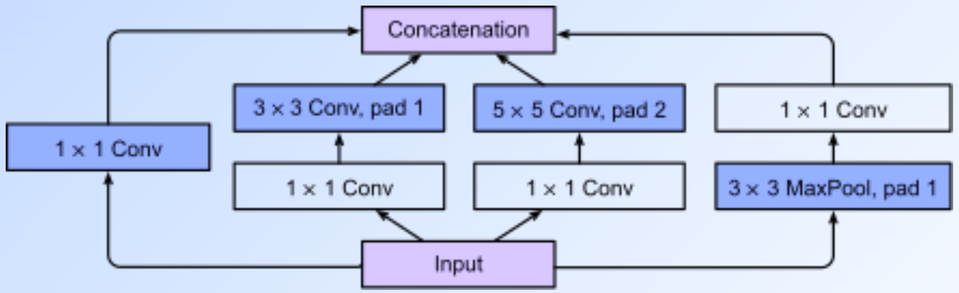
Inception module

The Inception-v1 Model

- The inception module was designed for multi-scale feature extraction.
- The inception-v1 model also employed auxiliary classifiers at intermediate layers to combat the problem of vanishing gradients.



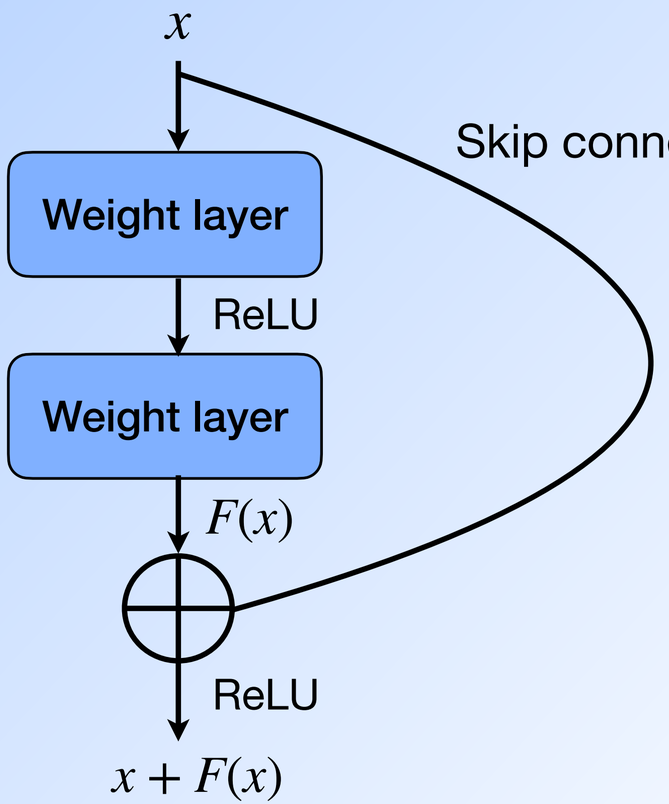
Inception-v1 architecture



Inception module

The ResNet Model

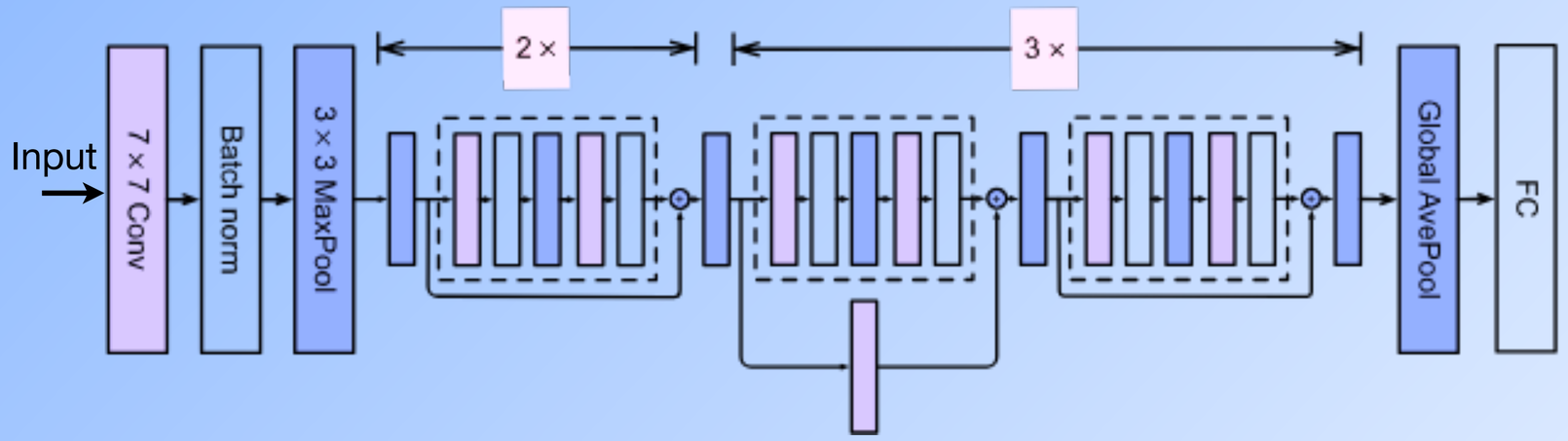
- The ResNet (He et al., 2015) models introduced residual blocks that employ skip connection that allow inputs to directly flow from shallower layers to the deeper layers, while bypassing intermediate layers.
- Each residual block typically consists of two or three convolutional layers, ReLU and batch normalization.
- The input to the residual block is added to the output of the last layer of the same block using skip connections.
- The idea of residual blocks is to enable the deeper layers learn the “residual” information.
- Residual blocks mitigate the vanishing gradient problem to a significant extent by providing multiple paths for the gradients to flow from the output to the input during backpropagation.



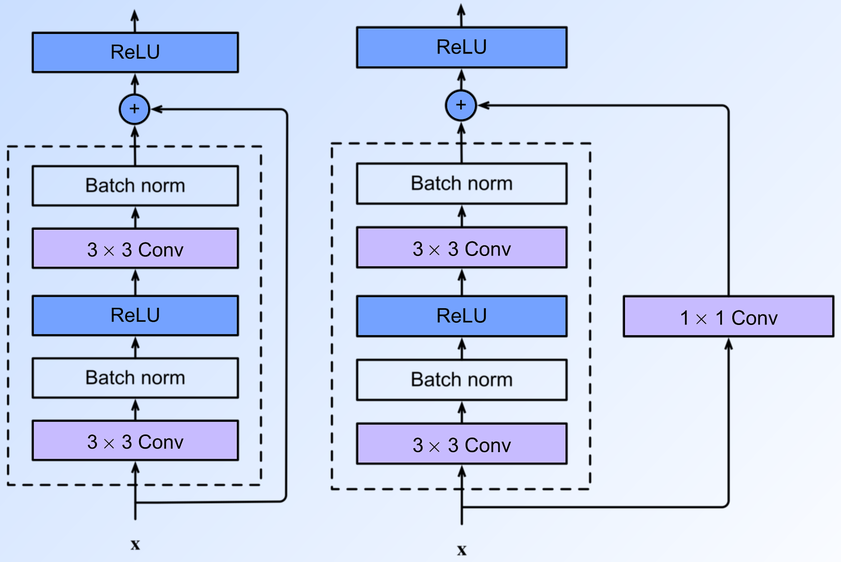
Residual block

The ResNet Model

- ResNet is another landmark model for vision tasks and comes in different variants such as ResNet-18, ResNet-34, ResNet-50 ResNet-101 and ResNet-152.
- 1×1 convolutions are used in some residual blocks to adjust the channel dimension to the required size.
- ResNet employs global average pooling at the end to reduce the dimension of the feature map to 1×1



ResNet-18 architecture



Residual blocks used

Other Popular CNN Architectures

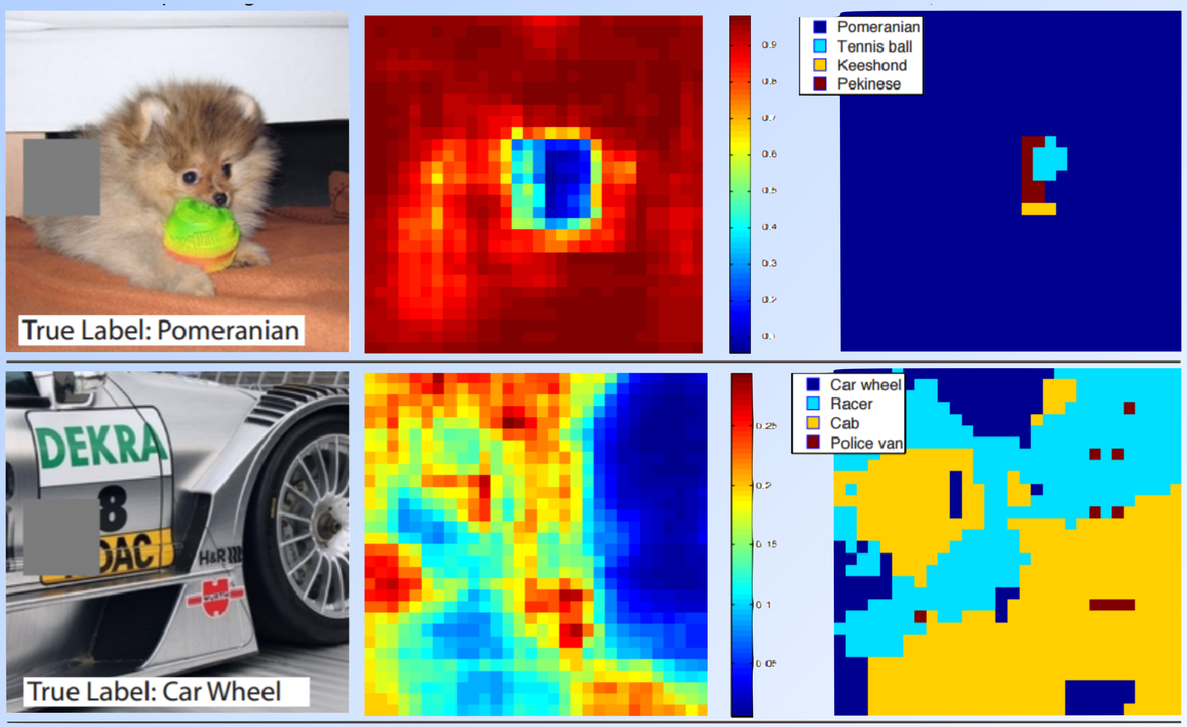
- **SqueezeNet** (Landola et al., 2016) - uses fire modules consisting of squeeze and expand layers. Uses GAP instead of FC layers and significantly reduces the number of parameters.
- **DenseNet** (Huang et al., 2017) - Each layer receives input from all previous layer outputs, which encourages feature reuse and reduces the number of parameters.
- **MobileNet** (Howard et al., 2017)- designed for mobile and embedded applications, it reduces the number of computations and parameters using depth wise separable convolutions.
- **ShuffleNet** (Zhang et al., 2018) - introduces channel shuffle operation to enable feature re-use and grouped convolutions to reduce computational complexity.
- **EfficientNet** (Tan et al, 2019): Uses neural architecture search to design a baseline model and scale its depth, width and resolution using compound scaling.

Transfer Learning with Foundational Vision Models

- Transfer learning is a powerful ML technique where a model trained for one task is used or adapted to perform a different task.
- It is particularly useful for scenarios where the data available to learn the target task is limited.
- Foundational vision models (VGG, ResNet, Inception, Xception, MobileNet, DenseNet etc.) that were trained on millions of images can extract useful visual features such as edges, corners, shapes, texture, and color effectively.
- These features could be leveraged for many other vision tasks through transfer learning.
- In transfer learning, the pretrained models could be used in two different ways:
  - Feature extraction: the features extracted from the pretrained models could be used to train another model designed for the specific task
  - Fine tuning: the entire pertained model or few of its last layers could be fine tuned for the specific task.
- Overall, transfer learning reduces data requirement as well as training time, while enhancing performance in general.

Visualizing Saliency Maps Using Occlusion

- The prediction of CNNs could be explained using saliency maps, which give a visual description of how much impact each pixel of the input has on the model's prediction.
- An early technique for visualizing the saliency map is the occlusion based method, where parts of the input image are occluded and the predicted class likelihood plotted as a function of the occluded patch (Zeiler and Fergus, 2014).
- Although the method is simple and intuitive it is also computationally intensive and may not capture global structures and contextual relationships.



Input Image

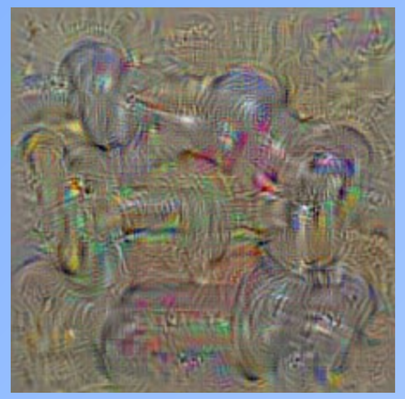
Predicted likelihood of the true class label

Predicted class

Class Model Visualization

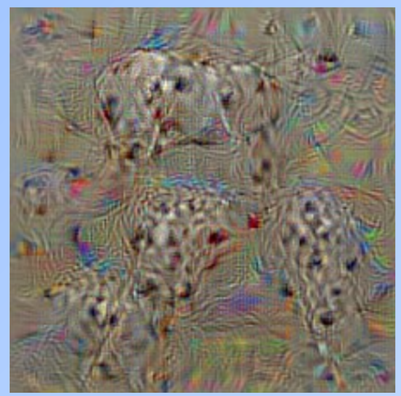
- Another interesting technique to visualize CNNs involves generating images that maximize specific class logits, i.e.
  arg $\max_I S_c(I) - \lambda \|I\|_2$, where $S_c(I)$ is the unnormalized score of class C for the image I, and λ is the regularization parameter (Simonyan et al., 2014).
- I is initialized with zeros, and instead of performing gradient descent to update the weights through backpropgation, gradient ascent is performed to update I through backpropagation.
- The weights of the CNN are held fixed to the weights found during training (on zero-centered training images).
- The training set mean image was added to the image obtained by gradient ascent.

Class Model Visualization

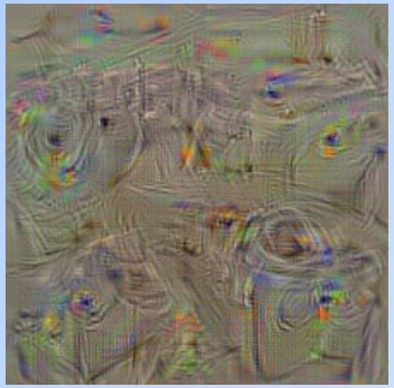


Dumbell

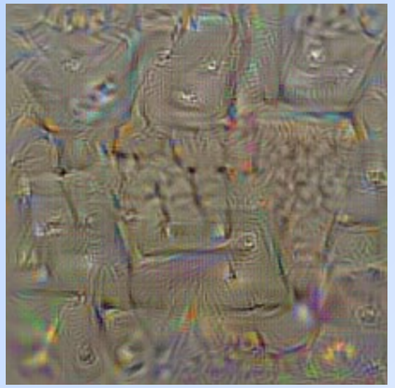
Cup



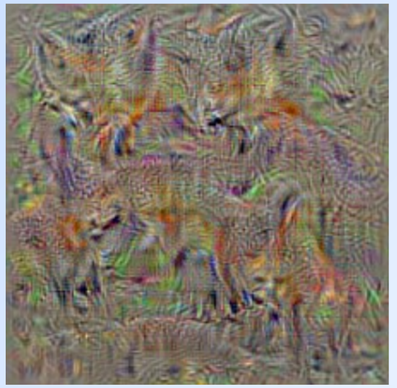
Dalmatian



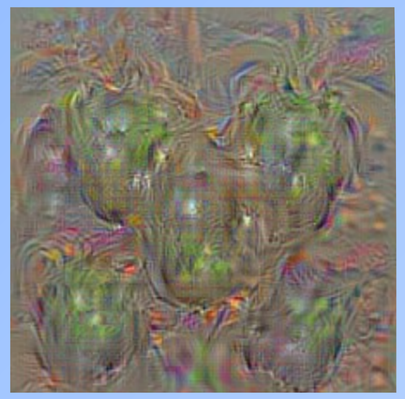
Washing machine



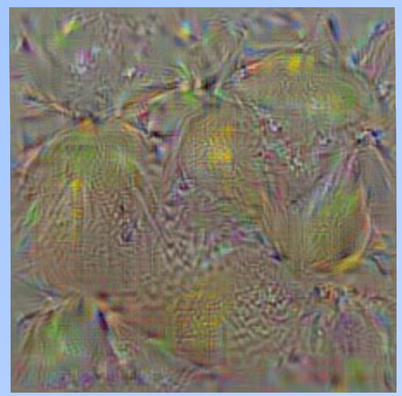
Computer keyboard



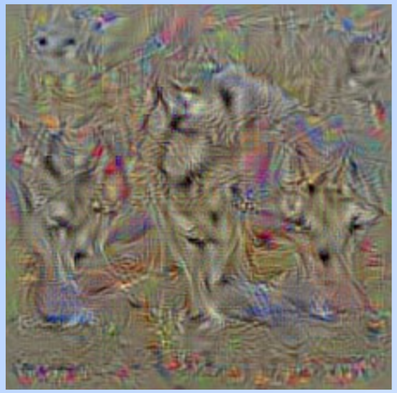
Kit fox



Bell pepper



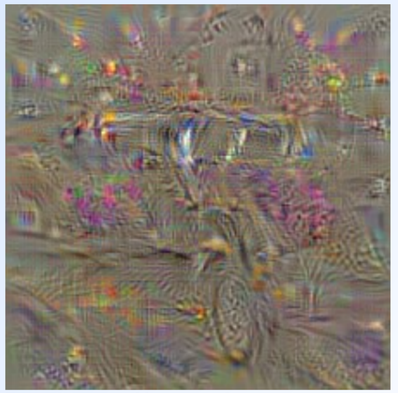
Lemon



Husky

Goose

Ostrich



Limousine

Saliency Map Visualization Using Backpropagation

- To compute the saliency maps using backpropagation, the class-score is approximated by a linear function using the first order Taylor series expansion as follows (Simonyan et al., 2014):

$$S_c(I) \approx w^T I + b, \text{ where } w = \left.\frac{\partial S_c(I)}{\partial I}\right|_{I=I_0}.$$

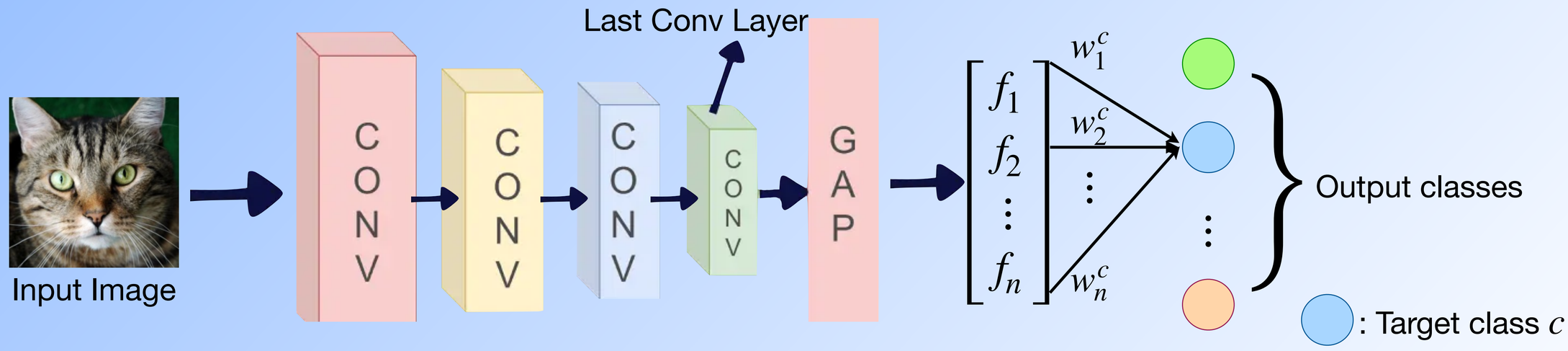
- The magnitude of the derivative w indicates which pixels of I need to be changed the least to affect the class scores the most.
- The derivative w is found using a single backpropagation using the trained weights of the CNN, and the weights thus computed are spatially rearranged to give the saliency map.
- In case of images having multiple color channels, the maximum of w across the channels are taken to compute the saliency map.

Saliency Map Visualization Using Backpropagation

Class Saliency Maps Obtained Using Backpropagation

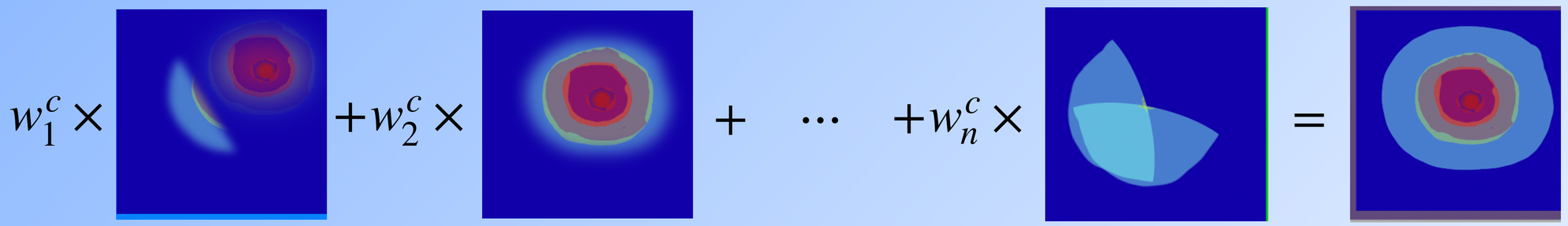
Class Activation Maps

- 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.

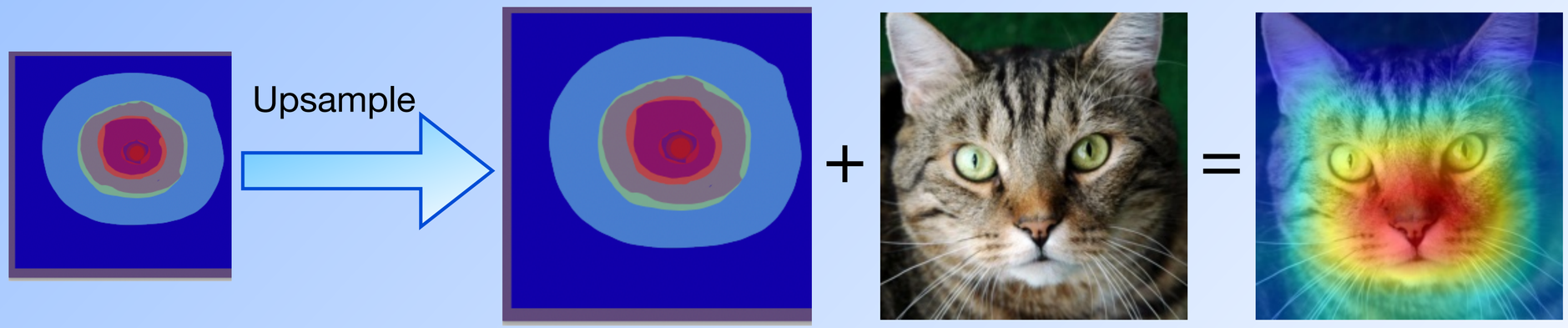


Class Activation Maps

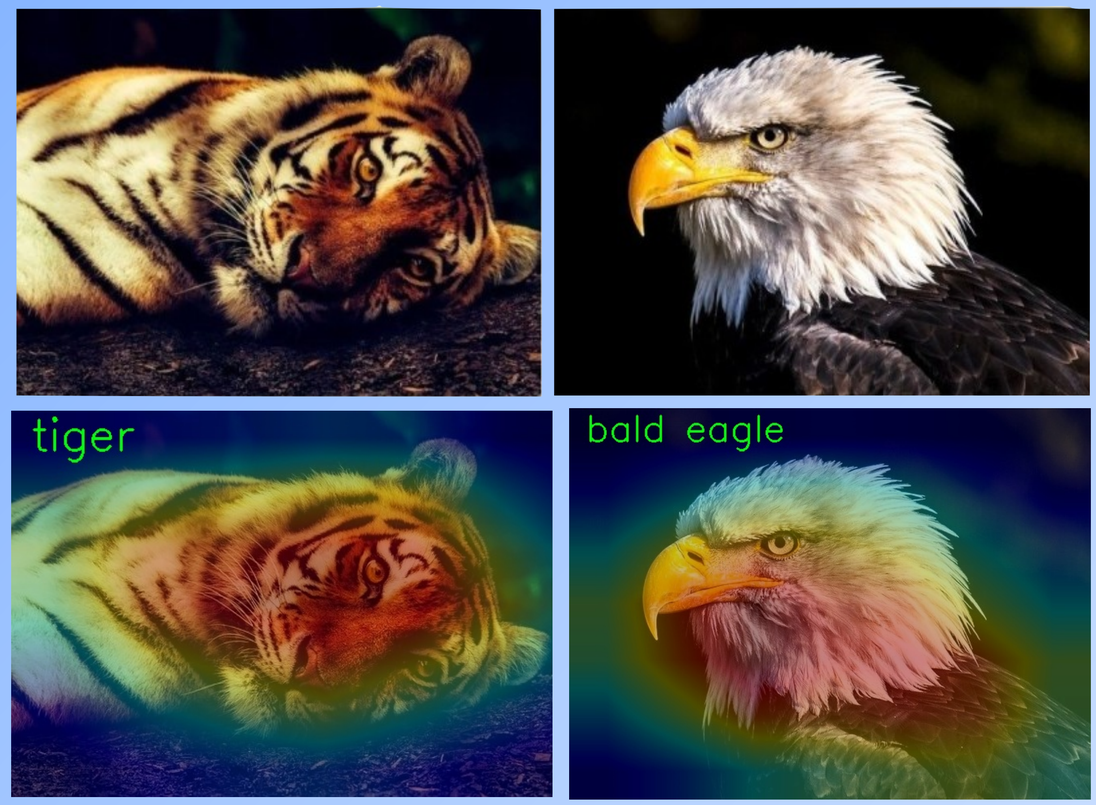
- To generate the CAM for a particular class, the weights learned by the linear classification layer are used to weight the feature maps learned by the last convolutional layer.



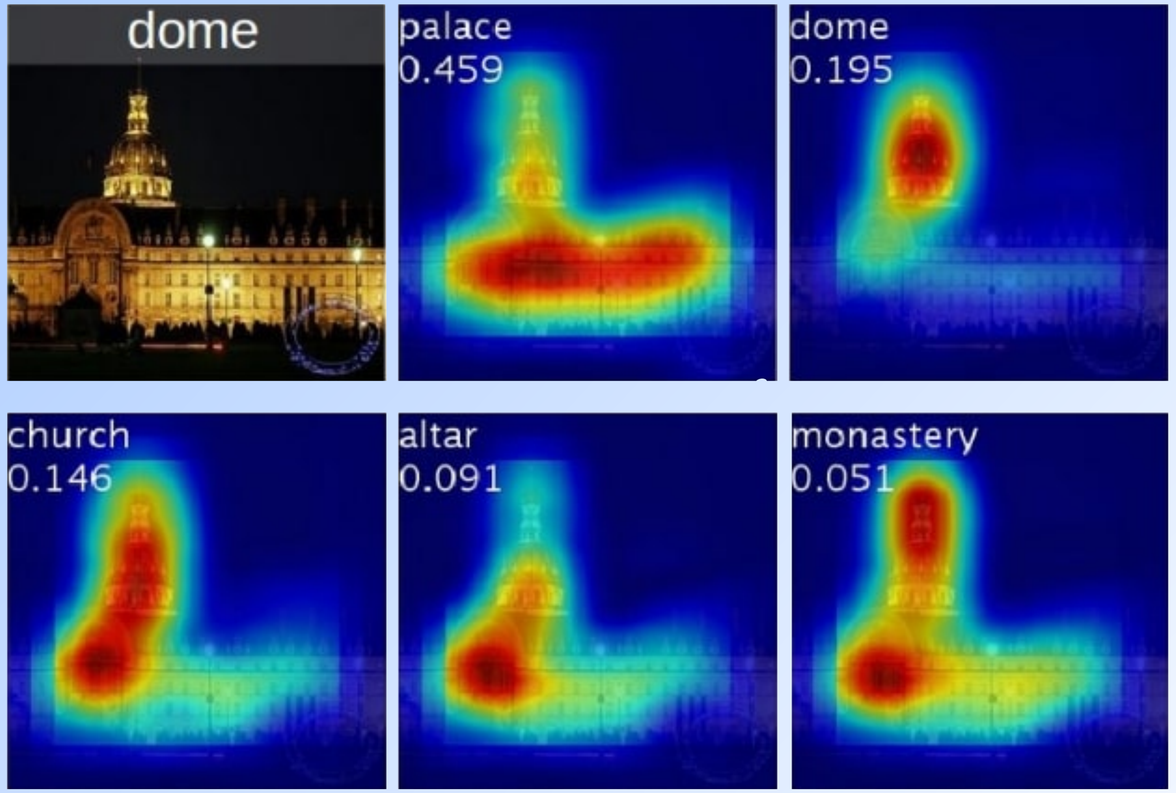
- The weighted feature map is finally upsampled to the size of the input image to get the CAM result.



Class Activation Maps



CAM visualizations



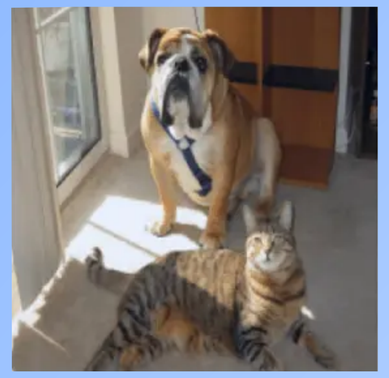
CAM with respect to different class labels

Gradient-Weighted CAMs

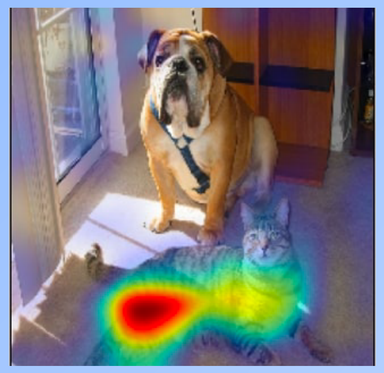
- CAMs require the CNN architecture to have a GAP layer before the final classification layer, which may not be suitable for all kinds of vision tasks.
- Gradient-weighted CAMs (Grad-CAMs) overcame the limitation of having a GAP layer by computing the gradient of the class logits with respect to the 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.

Gradient-Weighted CAMs

- Grad-CAMs can be computed with respect to any layer, not just the output layer.

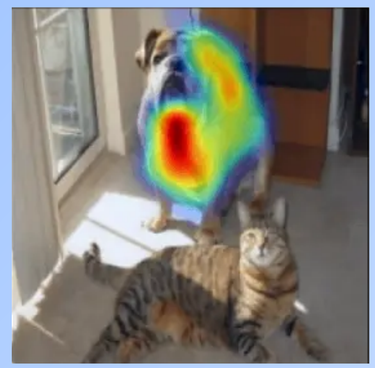


Input Image



Cat Grad-CAM

Dog Grad-CAM



Cat counterfactual explanation

Dog counterfactual explanation