Convolutional Neural Networks

Perumadura De Silva

Convolutional Neural Networks

Content

- Convolution operation
- Transpose convolution operation
- Feature-wise pooling operation
- Depth-wise pooling operation
- Design of a modern convolutional neural network
- Example Convolutional Layer design pattern
- Convolutional neural network example

Convolutional Neural Networks

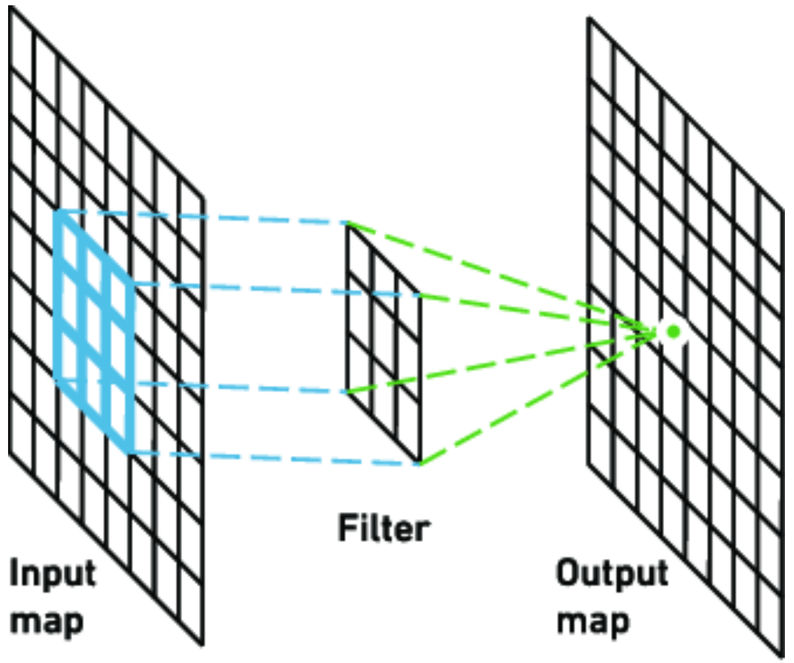
Convolution operation

- Issue of using a MLP layer:
  - For input 32x32x3 has 3072 connection weights
  - But if input 200x200x3 has 120000 connection weight
- Solution: Convolutional layer:
  - Made out of 2D or 1D filters
  - Convolutional filters has 2D structures (e.g 3x3, 5x5, 3x1, 1x1)
  - Used to extract features while preserving the structural consistency of the inputs

Convolutional Neural Networks

Convolution operation

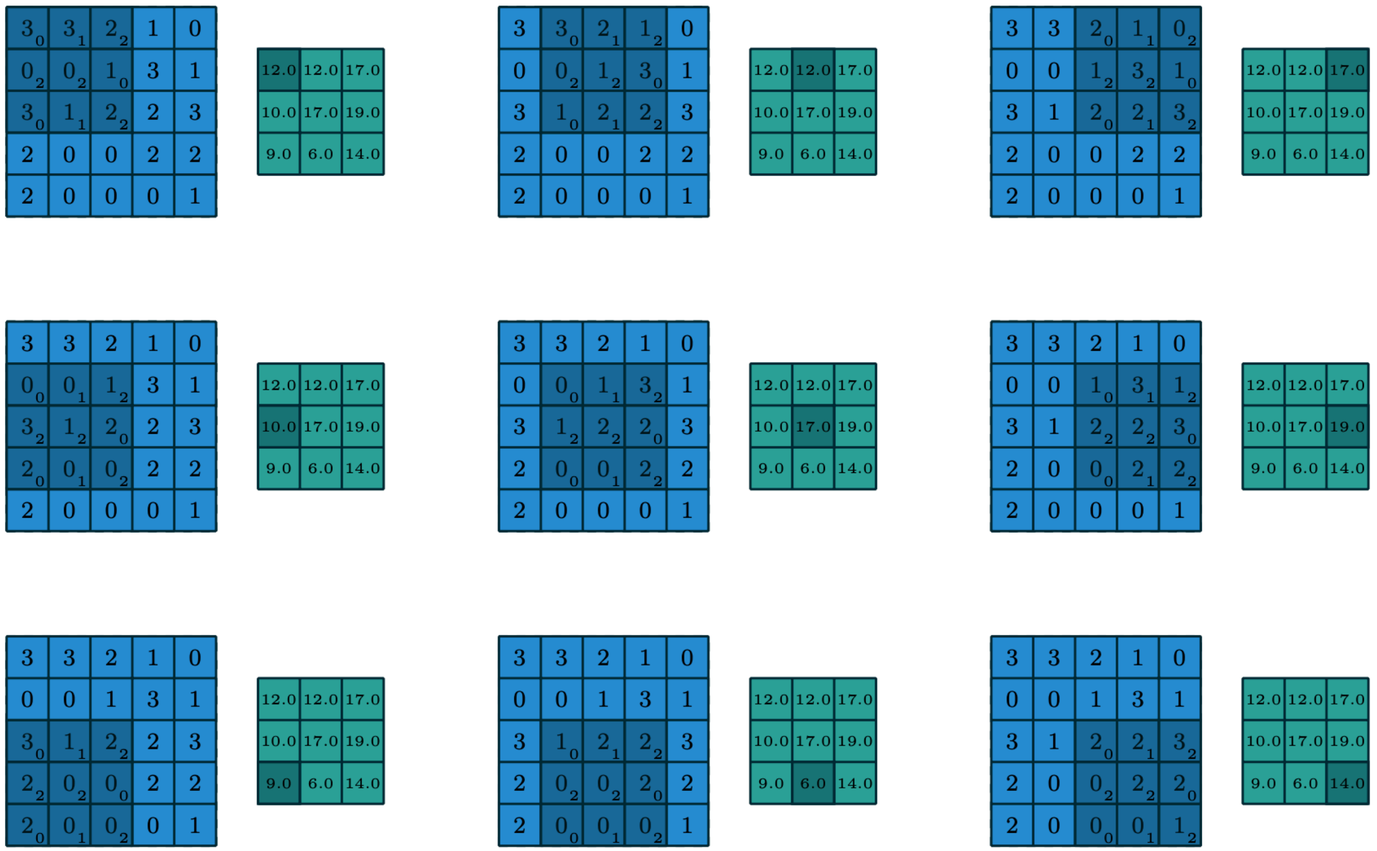
- Convolution filter:
  - Contains trainable parameters (trained using the gradient descent)
  - The filter is applied element wise on the inputs
  - The filter operation: element-wise multiplication of pixels and accumulation
  - The filter response is an activation map
  - A stack of filter can be used to extract different features from an image
  - The activation map:
    - At early layers extract edges, colour responses
    - At later layers filters combine low level features into high level objects (car, bird etc)



Ref.0: Input image/map processed by a filter to produce filter response/ activation map

Convolutional Neural Networks

Convolution operation



Ref.1: Convolution operation with a filter 3x3 on an input of 5x5 and stride of 1

Convolutional Neural Networks
Convolution operation

- Properties of convolution layer:
  - Unlike MLP filters only connect to a local region of the inputs
  - Filter parameters are shared across the spatial domain of the input:
    - Vertical edge at the top and bottom of the inputs are the same shape
  - Receptive field:
    - A local region a filter connects to
    - Usually odd values are preferred over the even values
  - E.g: Take Input 32x32x3 and 5x5 filter that produces one output activation map
    - Number of connections = 5x5x3 + 1 (bias) = 75 weights + 1 bias

Convolutional Neural Networks

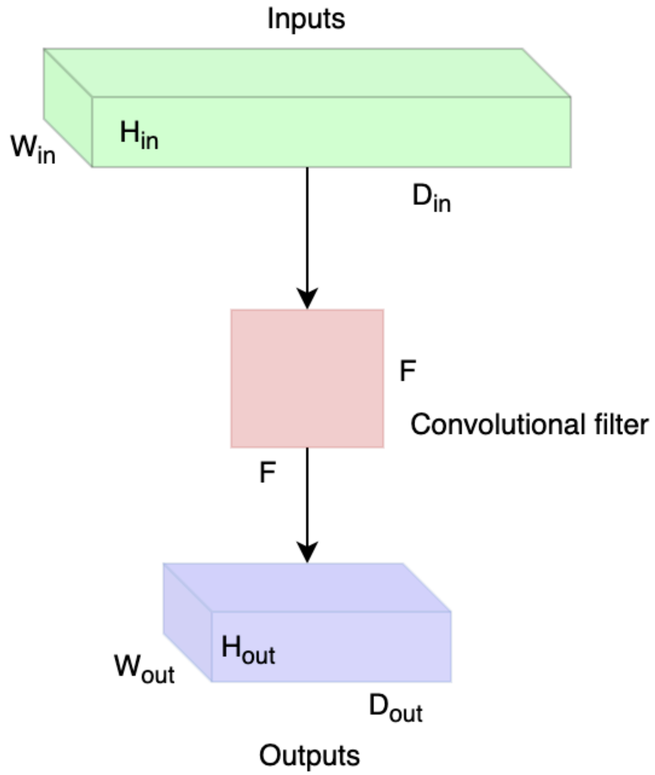
Convolution operation

- Size of the activation map depends on the depth, Filter size, stride and zero padding
- Filter size: Size of the filter (3x3, 5x5)
- Depth of output stack = Number of filters applied
- Stride: The number of pixels we slide along on the inputs
- Zero padding: Pad the inputs so we control the size of output activation map
- Activation output sizes:
  - $W_{out} = \frac{(W_{in} - F + 2P)}{s} + 1$ and $H_{out} = \frac{(H_{in} - F + 2P)}{s} + 1$
  - W_{in}/H_{in} input width/height, W_{out}/H_{out} output width/height
  - F filter size, s stride (number of pixels we slide), P padding

Convolutional Neural Networks

Convolution operation

- Takes an input volume of features either generated by a preceding layer or given as inputs
- Convolve and produce output feature volume
- Each convolutional filter is expressed as a stack of filters with dimensions:
  - $FxFxD_{input}$
- Convolutional filter produces $H_{output}W_{output}D_{output}$ feature volume as the output
- The depth-wise connections are fully connected



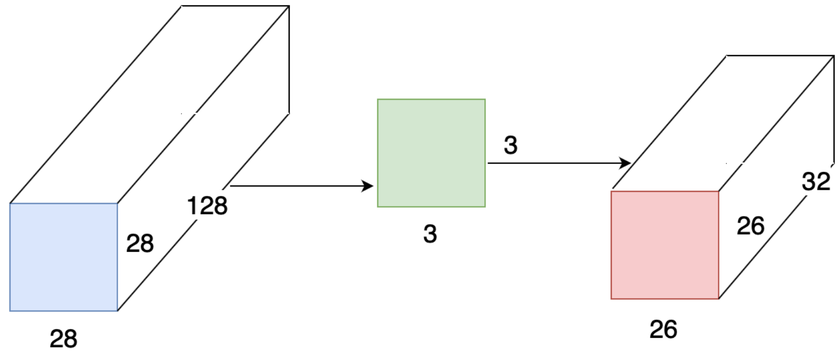
Convolutional Neural Networks
Convolution operation

- Number of parameters in a layer (total number of weights and biases):
  - $N_{parameters}^{total} = (F^2 D_{input} + 1) D_{output}$
- Number of operations in a convolutional layer
  - $N_{multiplications}^{weight} = H_{output} W_{output} F^2 D_{input} D_{output}$
  - $N_{accumulations}^{weight} = H_{output} W_{output} (F^2 - 1) D_{input} D_{output}$
  - $N_{summations}^{bias} = H_{output} W_{output} D_{output}$
  - $N_{ops}^{total} = N_{multiplications}^{weight} + N_{summations}^{weight} + N_{accumulations}^{bias}$

Convolutional Neural Networks

Convolution operation

- Example:
  - Find the number of parameters and total number of operations in the convolutional layer
  - Input size: 28x28
  - Input feature depth: 128
  - Output feature depth: 32
  - Filter size: 3x3
  - Padding: No padding
  - Stride: 1



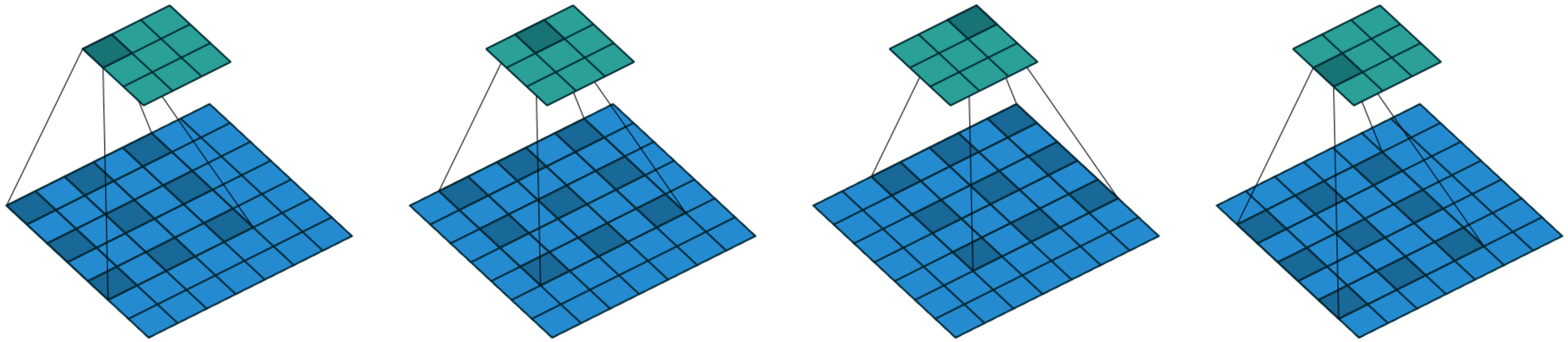
- **Number of parameters:**
  - $[(3x3x128) + 1]x32 = 36896$
- **Number of Multiplications:**
  - $(26x26)x[3x3x128x32] = 24920064$
- **Number of accumulations:**
  - $(26x26)x[(3x3 - 1)x128x32] = 22151168$
- **Number of bias sums:**
  - $(26x26x32) = 21632$

Convolutional Neural Networks
Convolution operation

- Dilated convolution
  - Inflate the filter by adding “0” in between filter parameters
  - Increase the receptive field without increasing the number of parameters
  - Number of spacing elements $= d - 1$; d is the dilation rate
  - $d = 1$ for normal convolution
  - Filter size after dilation $\hat{F} = F + (F - 1)(d - 1)$
  - $Output = \frac{Input - \hat{F} + 2P}{s} + 1$

Convolutional Neural Networks

Convolution operation



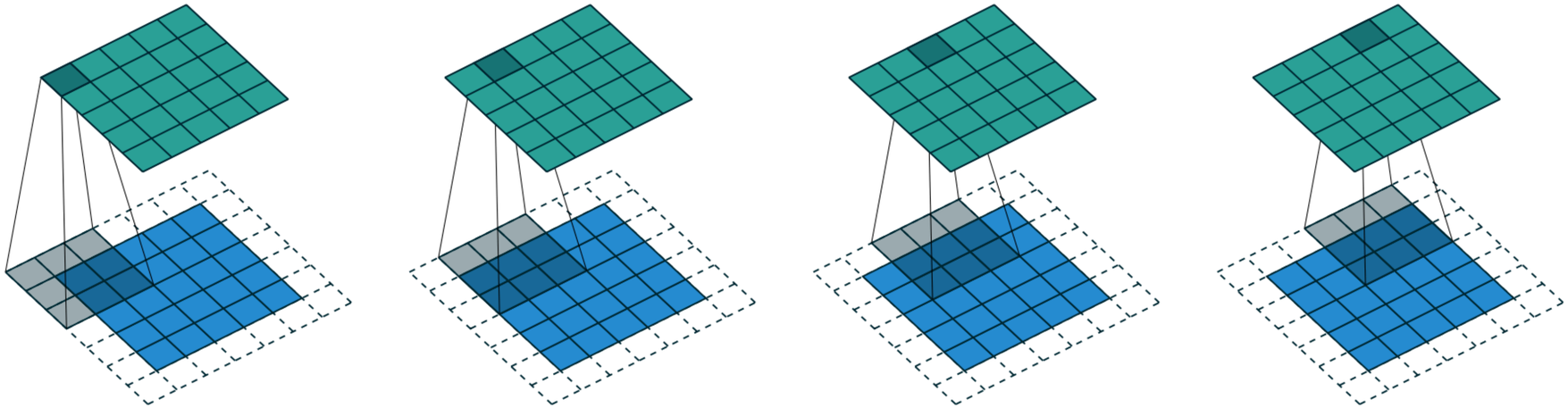
Ref.1: Dilated Convolution. Convolution on a 7x7 input with a 3x3 kernel and a stride of 1.
The dilation rate is 2

Convolutional Neural Networks
Convolution operation

- Controlling the output sizes
  - In some cases such as image enhancement, the output shapes are similar to the input shape
  - E.x: Take: input size 32x32, filter size 3x3 and stride 1
    - $Output = \frac{Input - F + 2P}{s} + 1$ -> $\frac{s(Output - 1) - Input + F}{2} = P$
    - So we get $P = \frac{F - 1}{2} = 1$ for stride 1 and $Output = Input$
    - Therefore we add the padding by increasing the hight and width by 1 pixel
    - The pixel value could set to 0 or any other constant value

Convolutional Neural Networks

Convolution operation



Ref.1: Convolution on 5x5 input with a 3x3 filter and a stride of 1. The image is padded to obtain a filter response that has the same size as the inputs

Convolutional Neural Networks

Transpose convolution operation

- Forward Convolutional filter encodes information from high dimension to low dimension
- Transpose Convolution filter decodes information from low dimension to high dimension
- Emulated using the convolutional filter with padding and spacing
- $O = s(I - 1) + k - 2P$

Convolutional Neural Networks

Transpose convolution operation

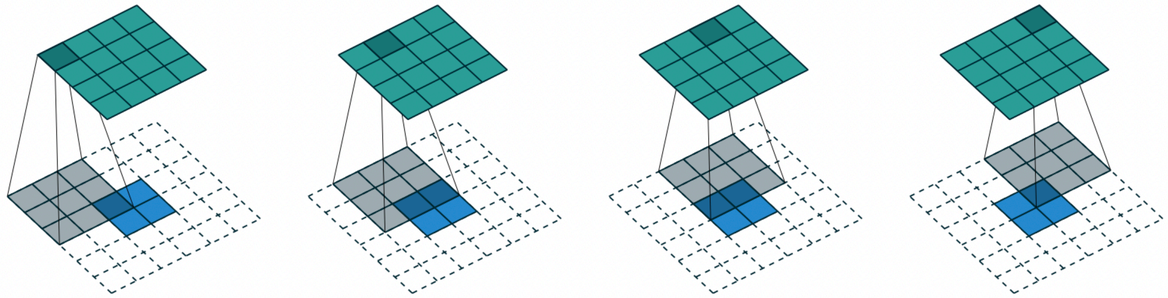


Figure 4.1: The transpose of convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input padded with a 2×2 border of zeros using unit strides (i.e., $i' = 2$, $k' = k$, $s' = 1$ and $p' = 2$).

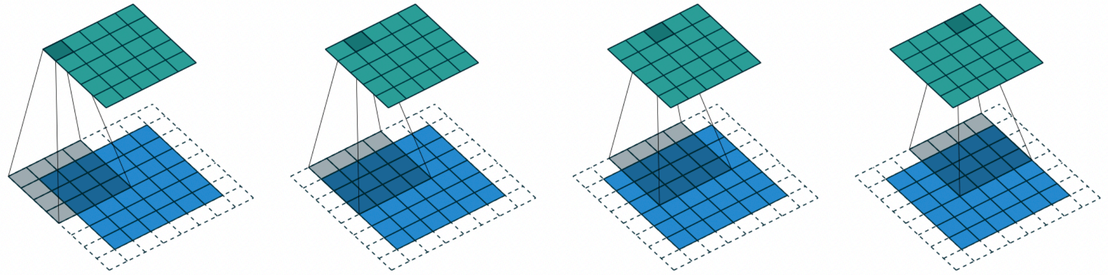


Figure 4.2: The transpose of convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$). It is equivalent to convolving a 4×4 kernel over a 6×6 input padded with a 1×1 border of zeros using unit strides (i.e., $i' = 6$, $k' = k$, $s' = 1$ and $p' = 1$).

Ref.1: Transpose Convolution Examples

Convolutional Neural Networks

Transpose convolution operation

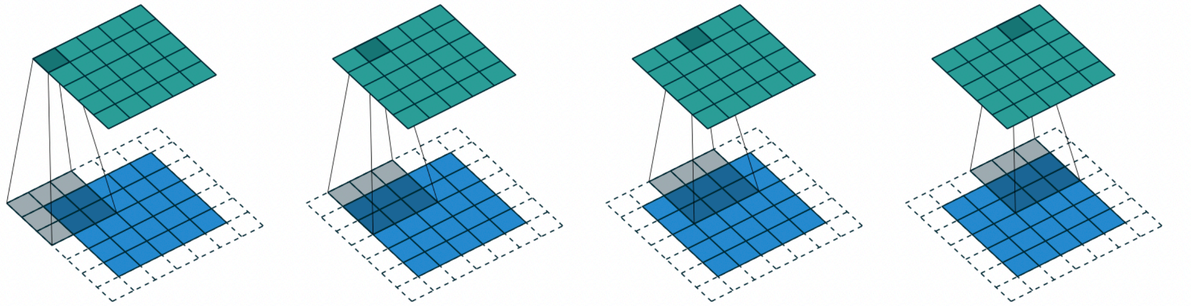


Figure 4.3: The transpose of convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$). It is equivalent to convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i' = 5$, $k' = k$, $s' = 1$ and $p' = 1$).

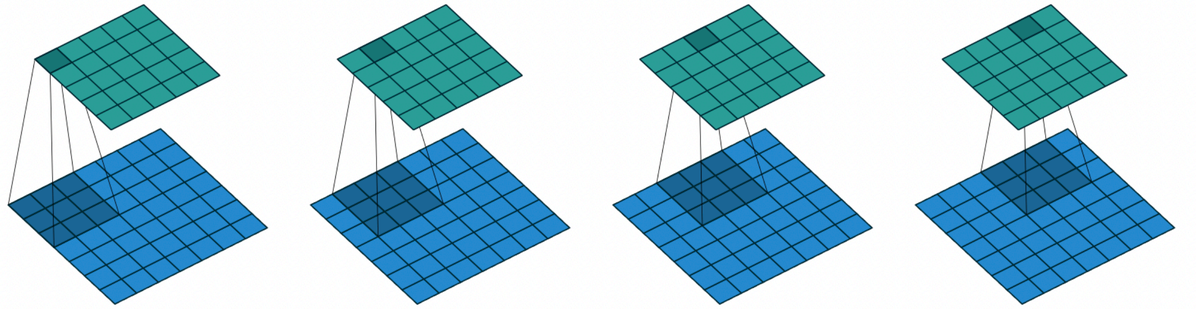


Figure 4.4: The transpose of convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$). It is equivalent to convolving a 3×3 kernel over a 7×7 input using unit strides (i.e., $i' = 7$, $k' = k$, $s' = 1$ and $p' = 0$).

Ref.1: Transpose Convolution Examples

Convolutional Neural Networks

Transpose convolution operation

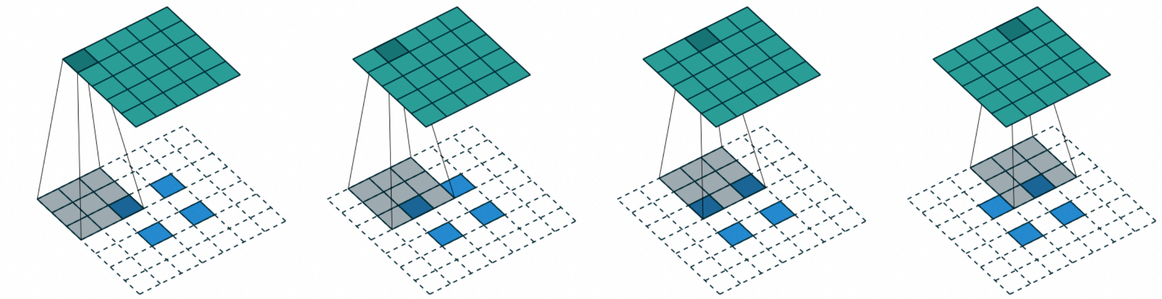


Figure 4.5: The transpose of convolving a 3×3 kernel over a 5×5 input using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input (with 1 zero inserted between inputs) padded with a 2×2 border of zeros using unit strides (i.e., $i' = 2$, $\tilde{i}' = 3$, $k' = k$, $s' = 1$ and $p' = 2$).

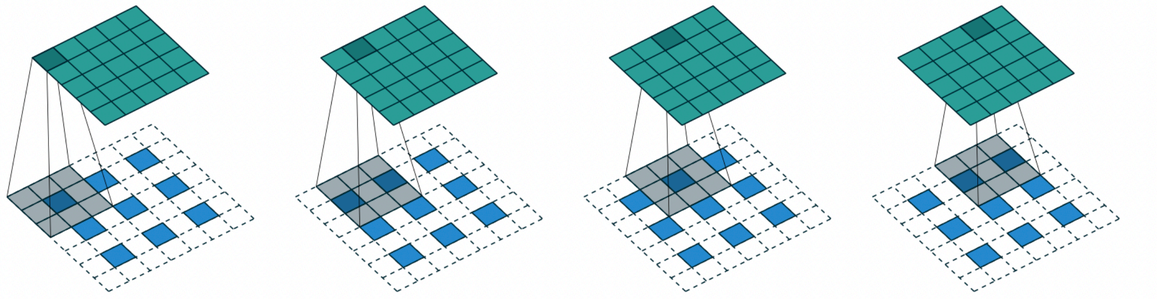


Figure 4.6: The transpose of convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 1$). It is equivalent to convolving a 3×3 kernel over a 3×3 input (with 1 zero inserted between inputs) padded with a 1×1 border of zeros using unit strides (i.e., $i' = 3$, $\tilde{i}' = 5$, $k' = k$, $s' = 1$ and $p' = 1$).

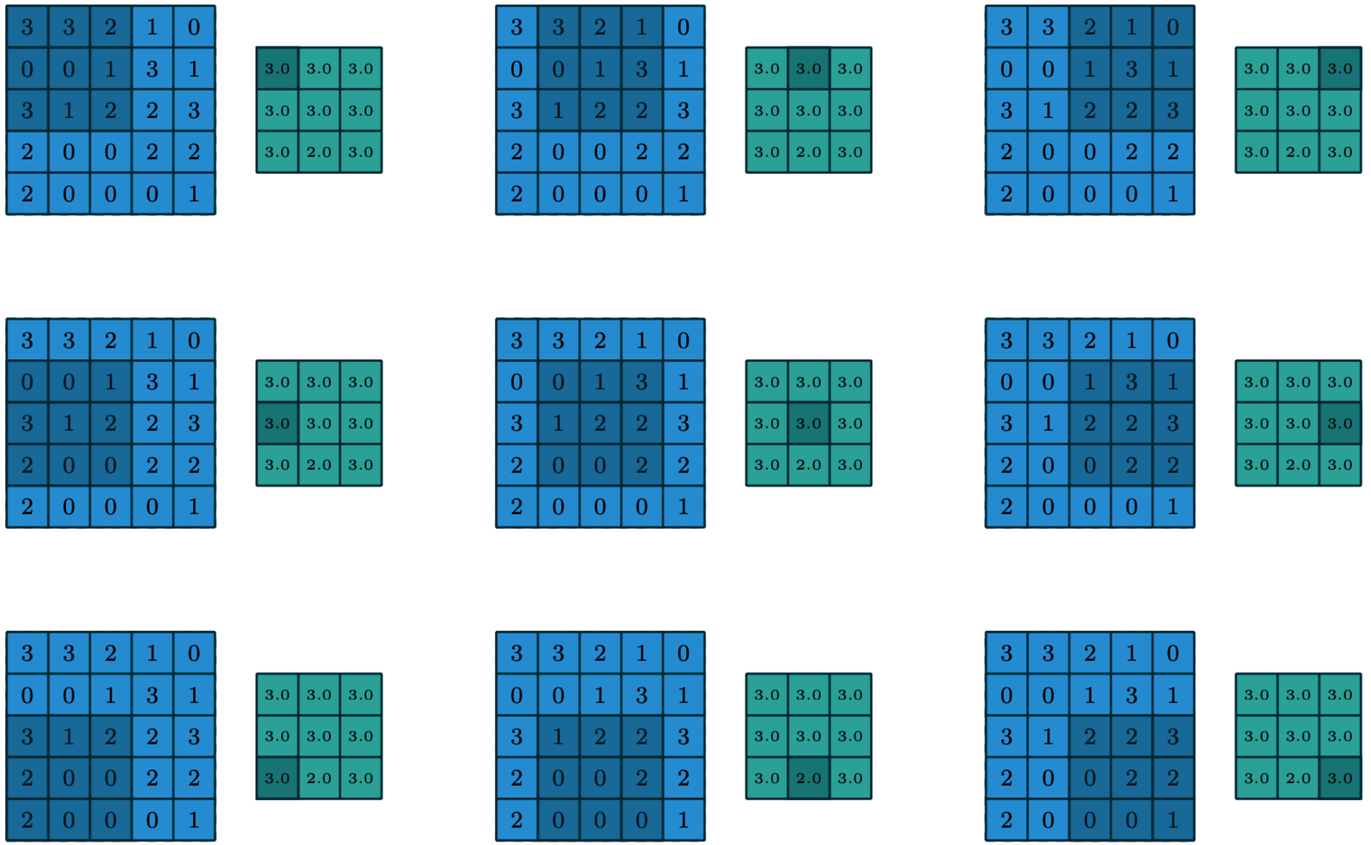
Ref.1: Transpose Convolution Examples

Convolutional Neural Networks
Feature-wise pooling operation

- Pooling layer summarises the response from a convolution filter
- A simple filter (commonly 2x2) that does one of the following operations on a receptive field
  - Max Pooling: Take the maximum value within a 2x2 receptive field
  - Average Pooling: Average a 2x2 receptive field
- Pooling does not introduce any new parameters
- Does not use padding in most cases
- Reduces the spatial dimension of the feature maps
- Invariant to small image transitions
- $Output = \frac{Input - F}{s} + 1$ When no padding in the pooling layers

Convolutional Neural Networks

Pooling operation

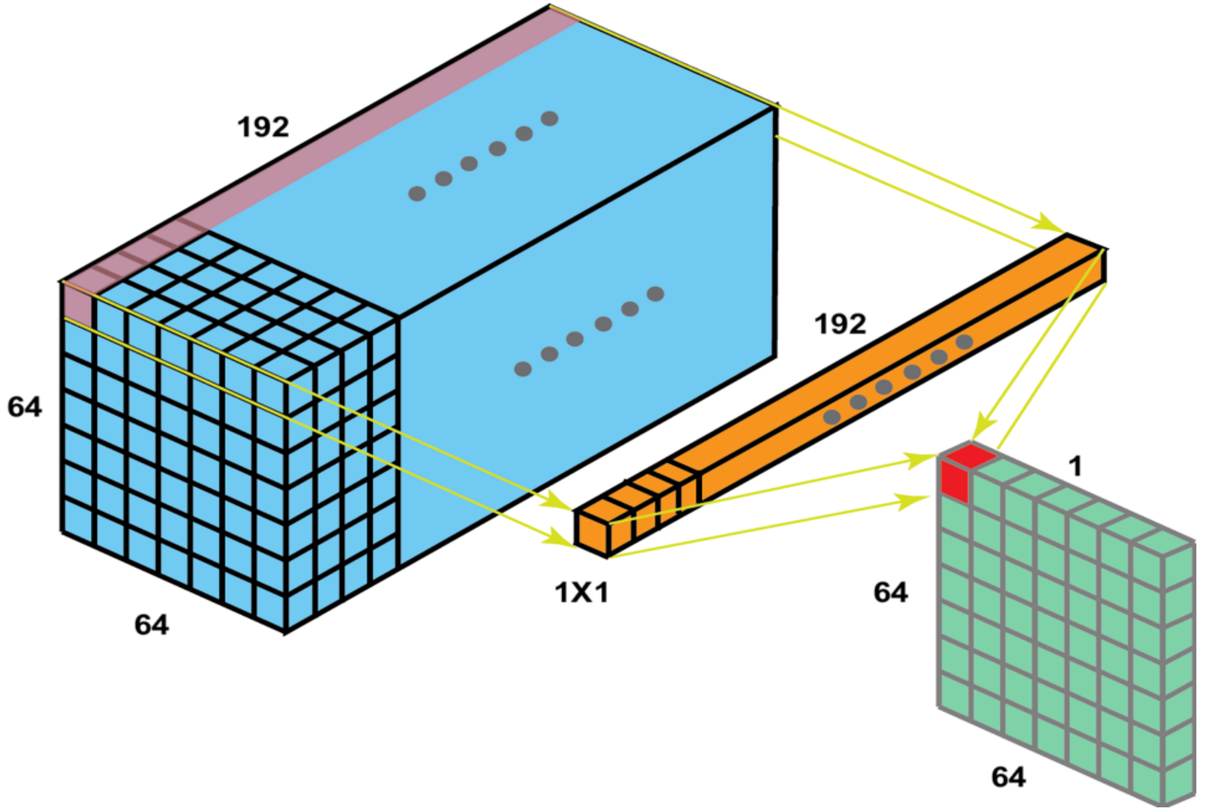


Ref.1: Max Pooling operation with a 3x3 pooling filter on a 5x5 input with stride of 1

Convolutional Neural Networks

Depth-wise pooling operation

- 1x1 Convolution filter
  - A popular design pattern of CNNs (Convolutional neural networks) is to increase the feature depth as the network goes deeper
  - Increasing feature depth adds parameters and operations which is expensive
  - Filter size: $1x1xN$
  - Multiply in the spatial domain
  - Accumulate in the depth domain



Convolutional Neural Networks

Depth-wise pooling operation

- 1x1 Convolution
  - 1x1 convolutional filter is designed for depth-wise pooling
    - It maintains the required output depth while reducing the number of parameters and operations
  - 1x1 convolutional layer is a linear projection layer
  - No padding is used
  - An activation function can be used to produce nonlinear outputs
  - Summarises the features in depth-wise

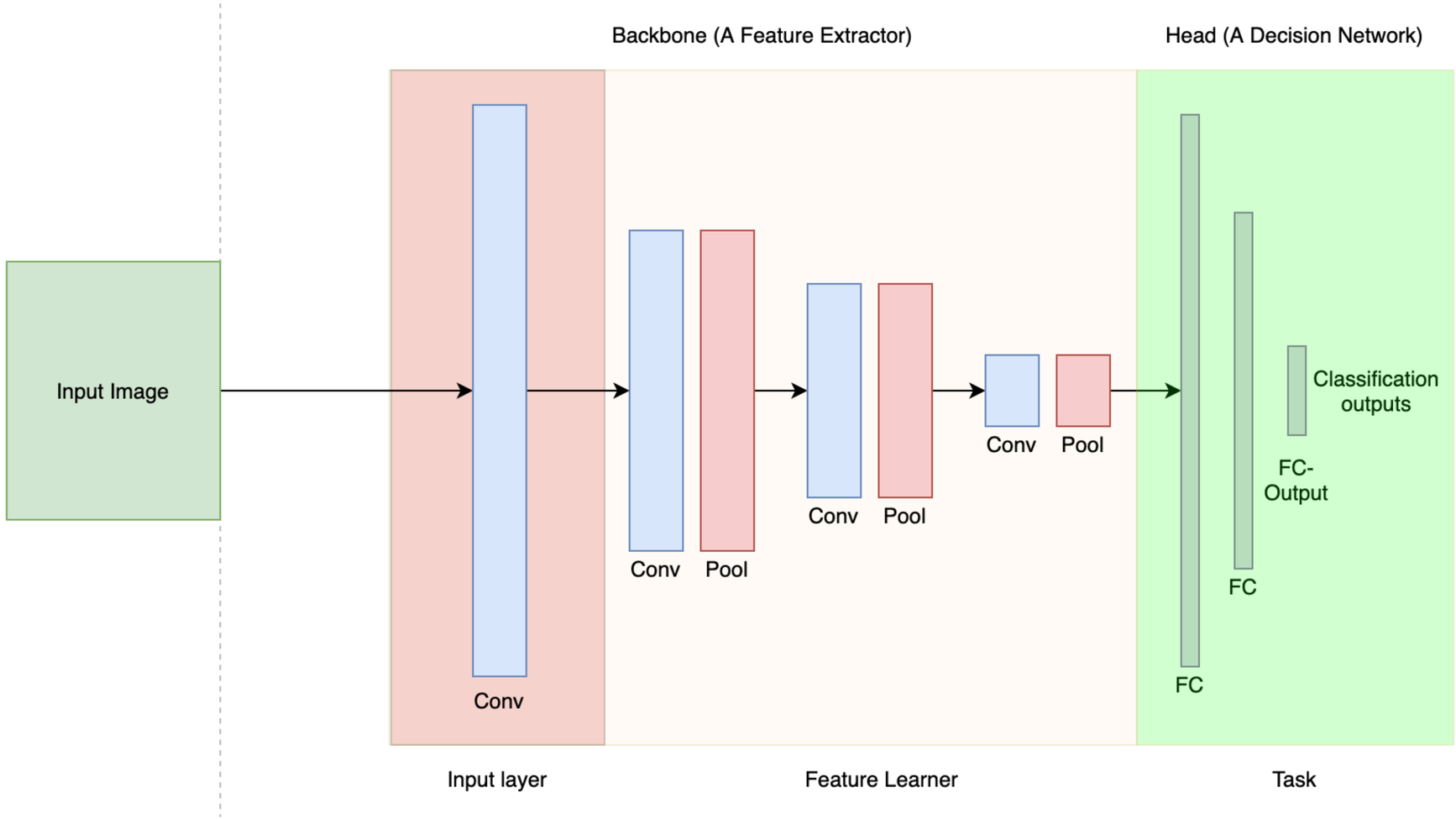
Convolutional Neural Networks

Design of a modern convolutional neural network

- Convolutional neural networks are made by combining a backbone with a task head
- A backbone is a hierarchical feature extractor made out of a feature learner and an input layer
- Input layer usually has a large convolutional filter to extract coarse features
  - Extract most relevant features for the feature learner to learn within a given model capacity
- The feature learner is made out of repeating blocks or group of blocks
  - Extract fine features from a coarser feature input to improve the task performance
- Each block contains one or more convolutional layers and other layers such as pooling and batch normalization
- The number of channels in the feature extractor increases as the depth increases
  - Usually the feature dimension reduces as the depth increases (A dimensionality reduction: retain only the useful features)
- Head utilises the features extracted using the feature learner to make predictions
- The design of the head depends on the task in hand

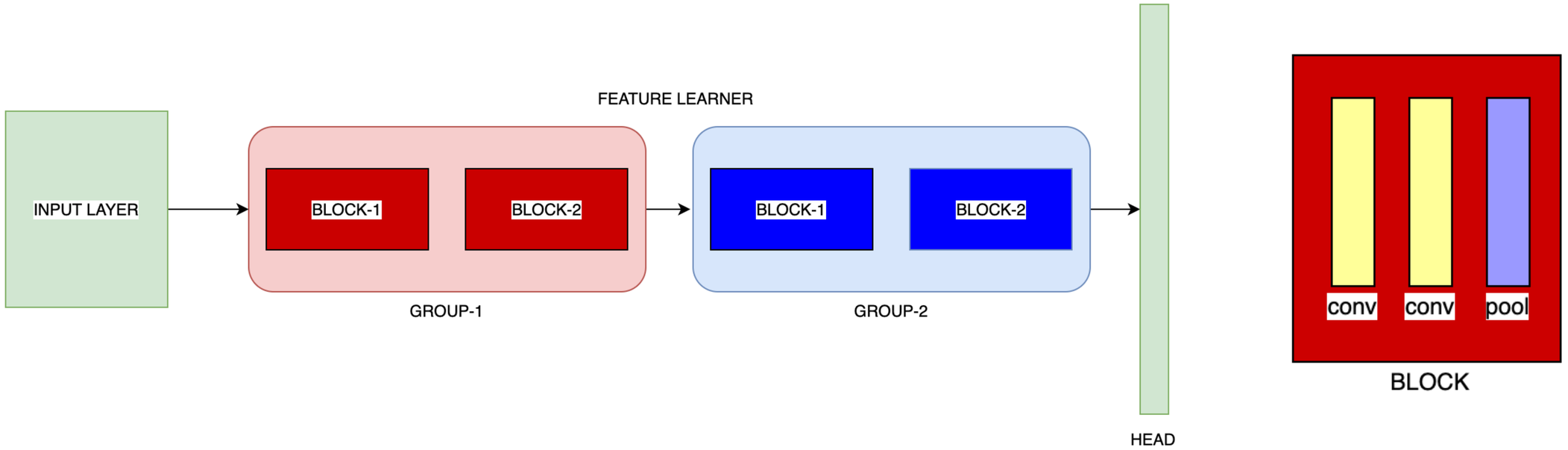
Convolutional Neural Networks

Design of a modern convolutional neural network



Convolutional Neural Networks

Design of a modern convolutional neural network

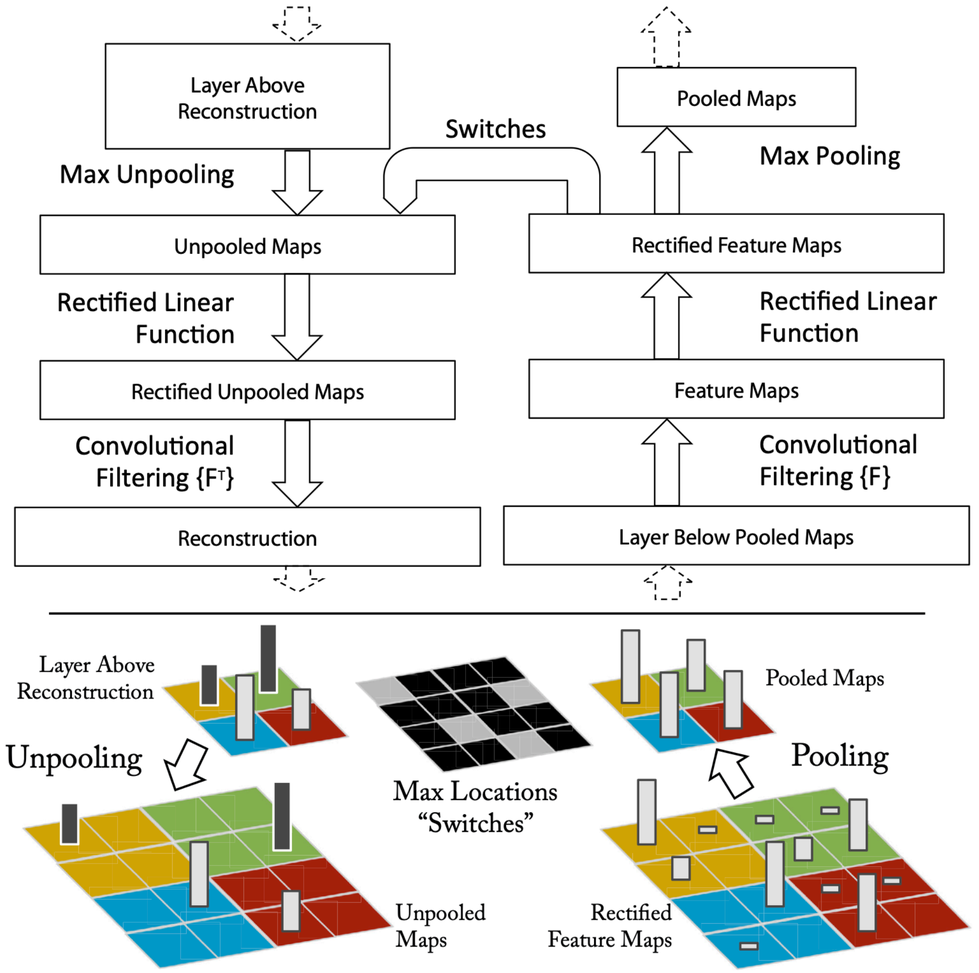


Convolutional Neural Networks Visualization

Ref.6 Filters with highest activation
at different layers

Convolutional Neural Networks
Visualization

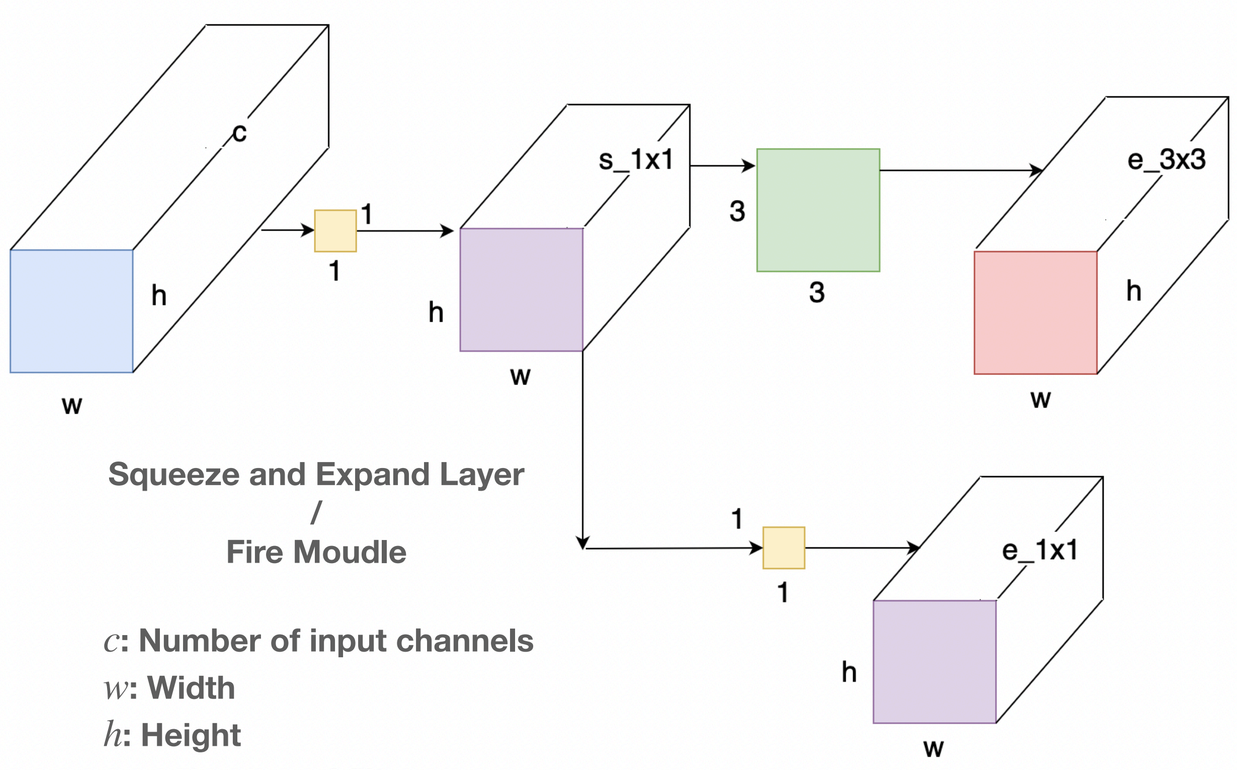
- Layer visualisation using Deconvolution and Unpooling
- For each layer, append a Unpooling and Deconvolution Layer
- Unpooling:
  - Record the indices of the maximums in the forward pass
  - Take the same indices from the layer above reconstruction, in the backward pass
- Deconvolution:
  - Upsample the Unpooled image using transpose convolution layer
- Deconvolution reconstructs the neurones with the largest activation for a given layer



Convolutional Neural Networks

Example Convolutional Layer design pattern

- Squeeze and Expand layer
  - Idea: Reduce the number of parameter
  - Bottleneck the channels with 1x1 convolution
  - Uncompress using a combination on 1x1 and 3x3 convolutions
  - Instead: input -> 3x3 -> output
  - Perform channel reduction at 3x3 input
  - Take: $s_{1x1} < (e_{1x1} + e_{3x3})$
  - Concatenate $1x1$ and $3x3$ in channel axis



Squeeze and Expand Layer / Fire Moudle

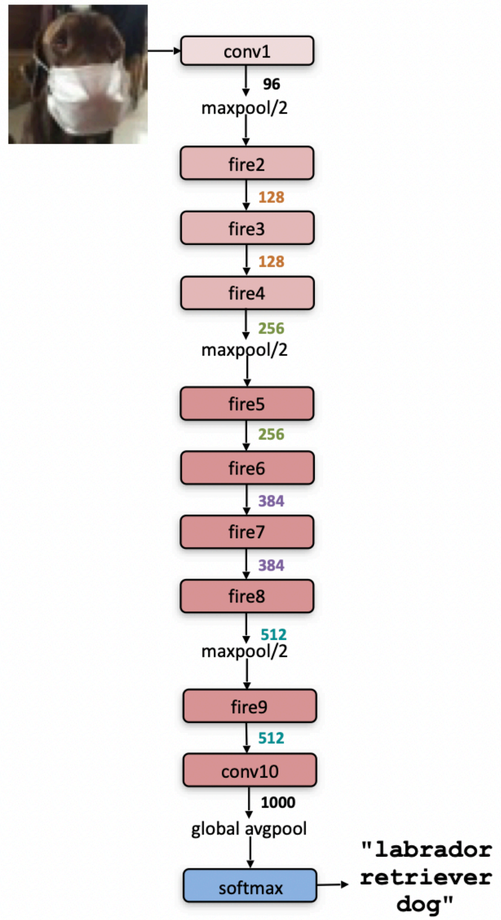
c: **Number of input channels**
w: **Width**
h: **Height**
s_{1x1}: **Number of filters in squeeze**
e_{1x1}: **Number of filters in expand** $1x1$
e_{3x3}: **Number of filters in expand** $3x3$

Convolutional Neural Networks

Example Convolutional Layer design pattern

Table 1: SqueezeNet architectural dimensions. (The formatting of this table was inspired by the Inception2 paper (Ioffe & Szegedy, 2015).)

layer name/type	output size	filter size / stride (if not a fire layer)	depth	s_{1x1} (#1x1 squeeze)	e_{1x1} (#1x1 expand)	e_{3x3} (#3x3 expand)	s_{1x1} sparsity	e_{1x1} sparsity	e_{3x3} sparsity	# bits	#parameter before pruning	#parameter after pruning
input image	224x224x3										-	-
conv1	111x111x96	7x7/2 (x96)	1				100% (7x7)			6bit	14,208	14,208
maxpool1	55x55x96	3x3/2	0									
fire2	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	11,920	5,746
fire3	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	12,432	6,258
fire4	55x55x256		2	32	128	128	100%	100%	**33%**	6bit	45,344	20,646
maxpool4	27x27x256	3x3/2	0									
fire5	27x27x256		2	32	128	128	100%	100%	**33%**	6bit	49,440	24,742
fire6	27x27x384		2	48	192	192	100%	**50%**	**33%**	6bit	104,880	44,700
fire7	27x27x384		2	48	192	192	**50%**	100%	**33%**	6bit	111,024	46,236
fire8	27x27x512		2	64	256	256	100%	**50%**	**33%**	6bit	188,992	77,581
maxpool8	13x12x512	3x3/2	0									
fire9	13x13x512		2	64	256	256	**50%**	100%	**30%**	6bit	197,184	77,581
conv10	13x13x1000	1x1/1 (x1000)	1				**20%** (3x3)			6bit	513,000	103,400
avgpool10	1x1x1000	13x13/1	0									
	activations		parameters				compression info				1,248,424 (total)	**421,098** (total)

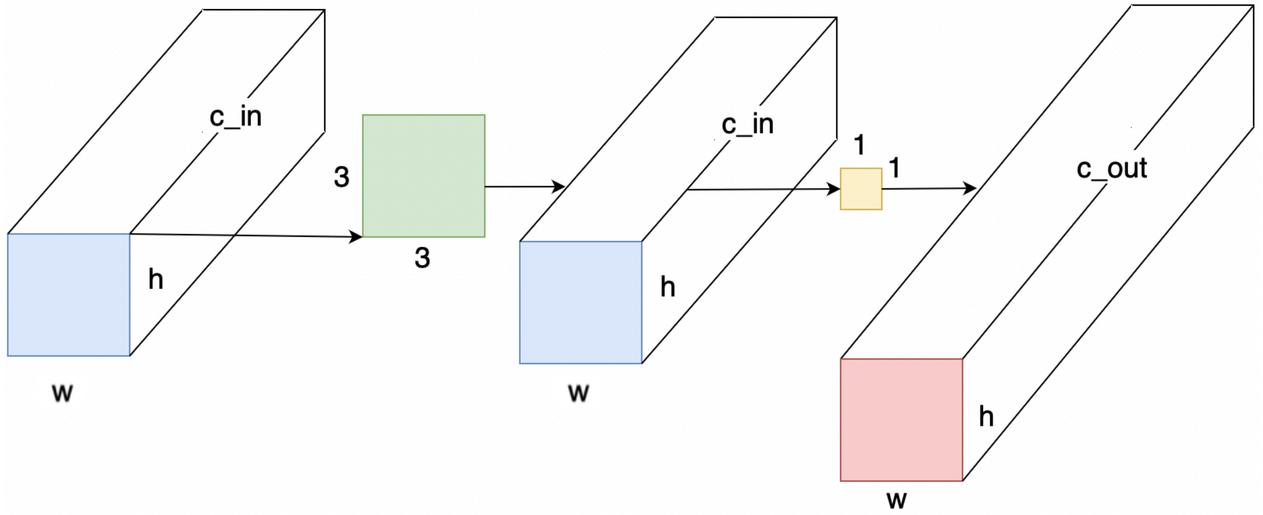


Ref.4: SqueezeNet v1.0 architecture and parameter table

Convolutional Neural Networks

Example Convolutional Layer design pattern

- Depthwise Separable Convolution
  - Idea: Reduce the number of computation (Make it mobile friendly)
  - Consist of:
    - Depth separable convolution
    - Pointwise (1x1) convolution layers
  - Depth separable convolution filters the input individually
    - Computationally efficient
  - Pointwise convolution combines the individually filtered channels
    - Expands the output to a desired number of channels

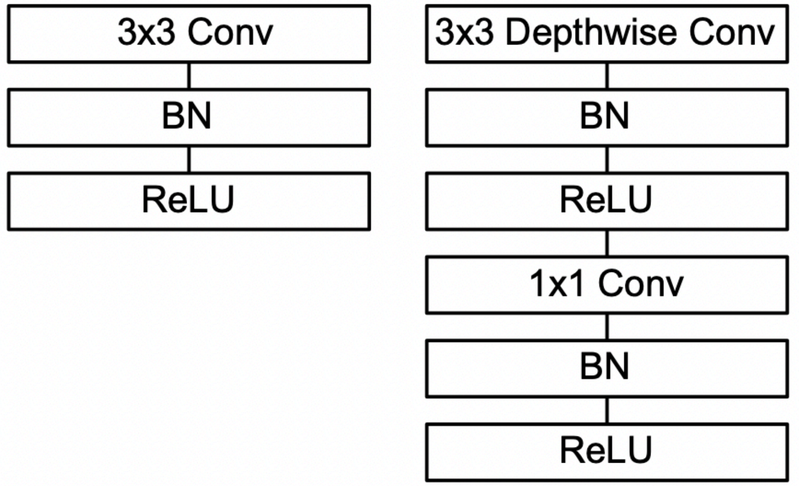


Convolutional Neural Networks

Example Convolutional Layer design pattern

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$



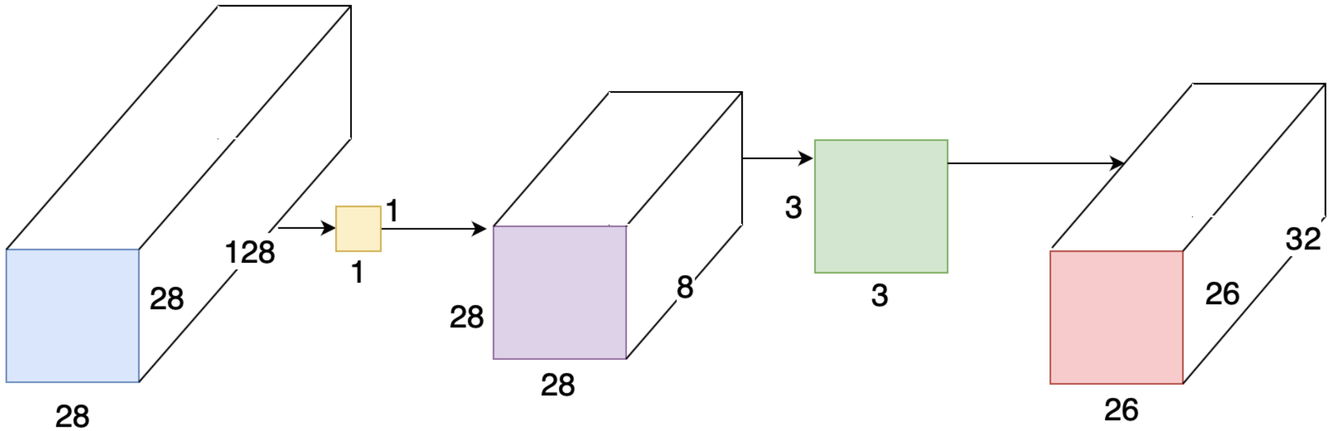
Ref.5: MobileNet v1 architecture and parameter table

Note: A stride $s > 1$ is used to reduce feature dimensions

Convolutional Neural Networks

Exercise: 1 Depth-wise pooling operation Exercise

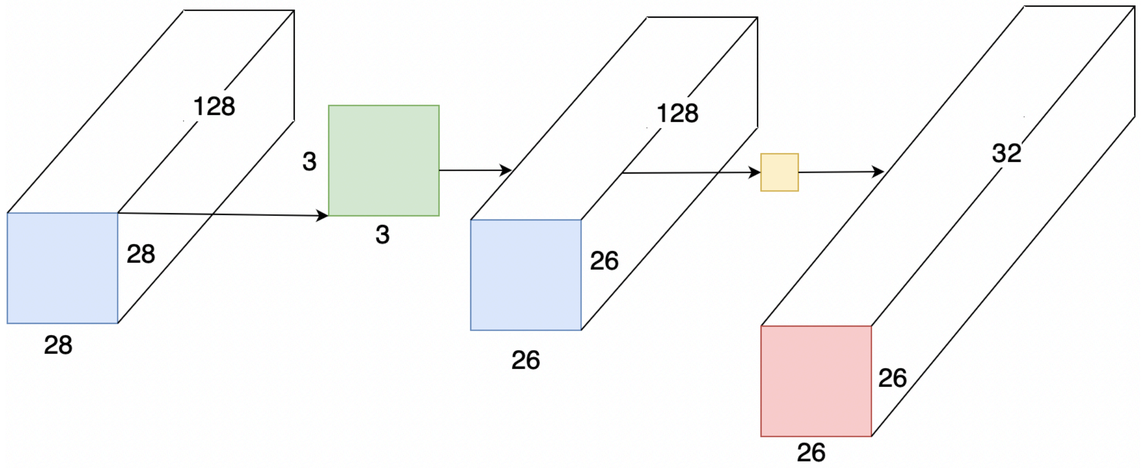
- Exercise-1:
  - Decrease the number of operations and parameters of the previous example using a 1x1 convolutional layer
  - Input size: 28x28
  - Input feature depth: 128
  - Output feature depth: 32
  - Filter size: 3x3
  - 1x1 convolutional layer filter depth 8
  - Padding: No padding
  - Stride: 1



Convolutional Neural Networks

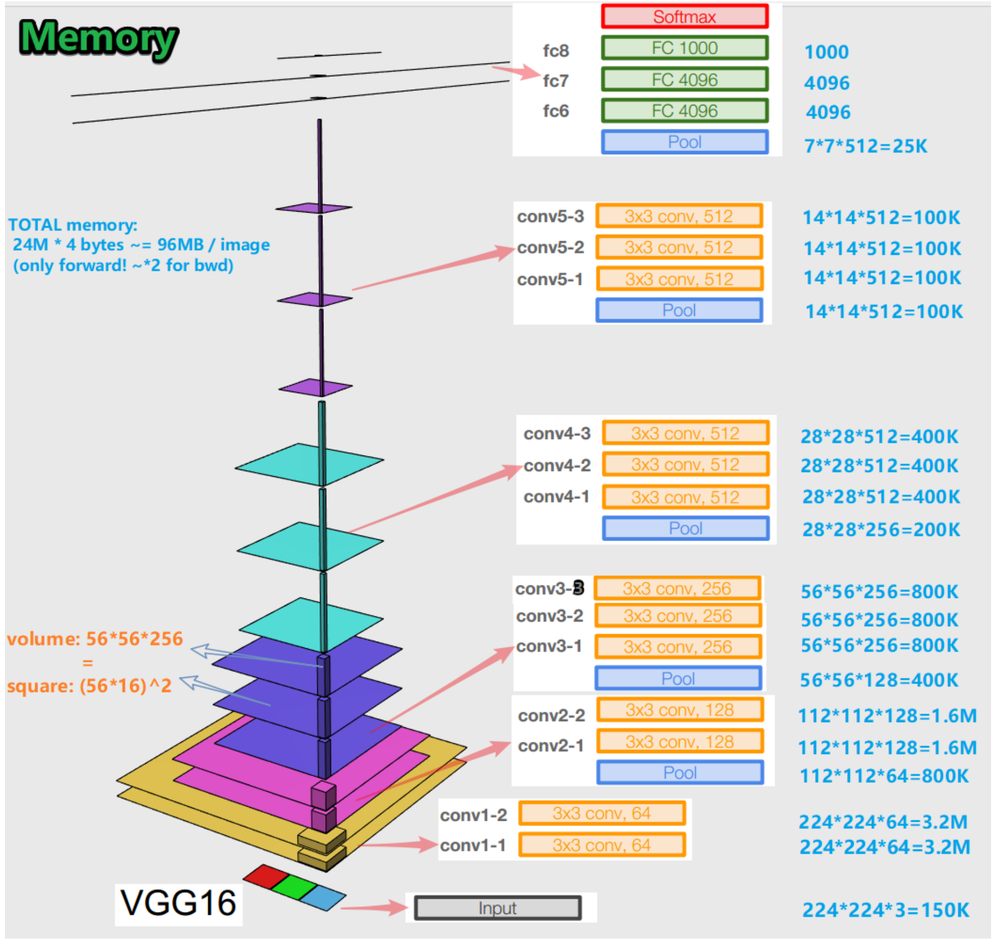
Exercise: 2 Depthwise separable convolution exercise

- Exercise-2:
- Derive the following Equation for the depth wise convolution
  - the number of parameters
  - Number of multiplications
  - Number of accumulations
- Use the equations to calculate the following in the depth wise separable convolution block:
  - Number of parameters
  - Number of Multiplications
- Present the efficiency gain (Comparison between normal convolution and efficient convolution)



Convolutional Neural Networks

Exercise-3: Build a VGG-16 network



Ref.7: Layers in the VGG-16 architecture

Reference

- Ref.0: https://www.researchgate.net/figure/Outline-of-the-convolutional-layer_fig1_323792694
- Ref.1: A guide to convolution arithmetic for deep learning Vincent Dumoulin and Francesco Visin
- Ref.2: https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215
- Ref.3: https://www.researchgate.net/figure/An-overview-of-the-VGG-16-model-architecture-this-model-uses-simple-convolutional-blocks_fig2_328966158
- Ref.4: SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE: Forrest N. Iandola1, Song Han2, Matthew W. Moskewicz1, Khalid Ashraf1, William J. Dally2, Kurt Keutzer1 (https://arxiv.org/pdf/1602.07360.pdf)
- Ref.6: https://www.groundai.com/project/visualizing-and-understanding-convolutional-networks/3
- Ref.7: https://forums.fast.ai/t/vgg16-memory-vs-parameter-in-3d-model/18117
- https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c
- Transpose convolution: https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md