LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Deep Learning

Chapter 6: Convolutional neural networks II

Bernd Bischl

Department of Statistics – LMU Munich

Winter term 2020

CNNS - APPLICATION

- As CNNs are one of the workhorses of deep learning, lots of research focuses on this model class.
- In this chapter, we will :
  - introduce different CNN topologies.
  - discuss different use cases and suitable prominent architectures.
  - give practical tips and tricks on the application of CNNs on real world data.

TYPOLOGY OF CONVOLUTIONS

1. 1D convolutions
2. 2D convolutions
3. 3D convolutions
4. Dilated convolutions
5. Separable convolutions
6. Transposed convolutions
7. Inception modules
8. Skip connections
9. Global average pooling

TYPES: 1D

- Problem setting: data that contains sequential, 1-dimensional information.
- Data is convolved with a 1D-kernel.
- Data consists of tensors with shape [channels, xdim].
- Single channel:
  - Analogon to univariate time series.
  - Development of a single stock price over time.
  - Text encoded in character-level one-hot-vectors [Zhang Xiang et al., 2015].
- Multichannel:
  - Analogon to multivariate time series.
  - Movement data measured with multiple sensors for human activity recognition [Wang Zhiguang et al., 2017].
  - Temperature and humidity in weather forecasting.

TYPES: 1D - SENSOR DATA

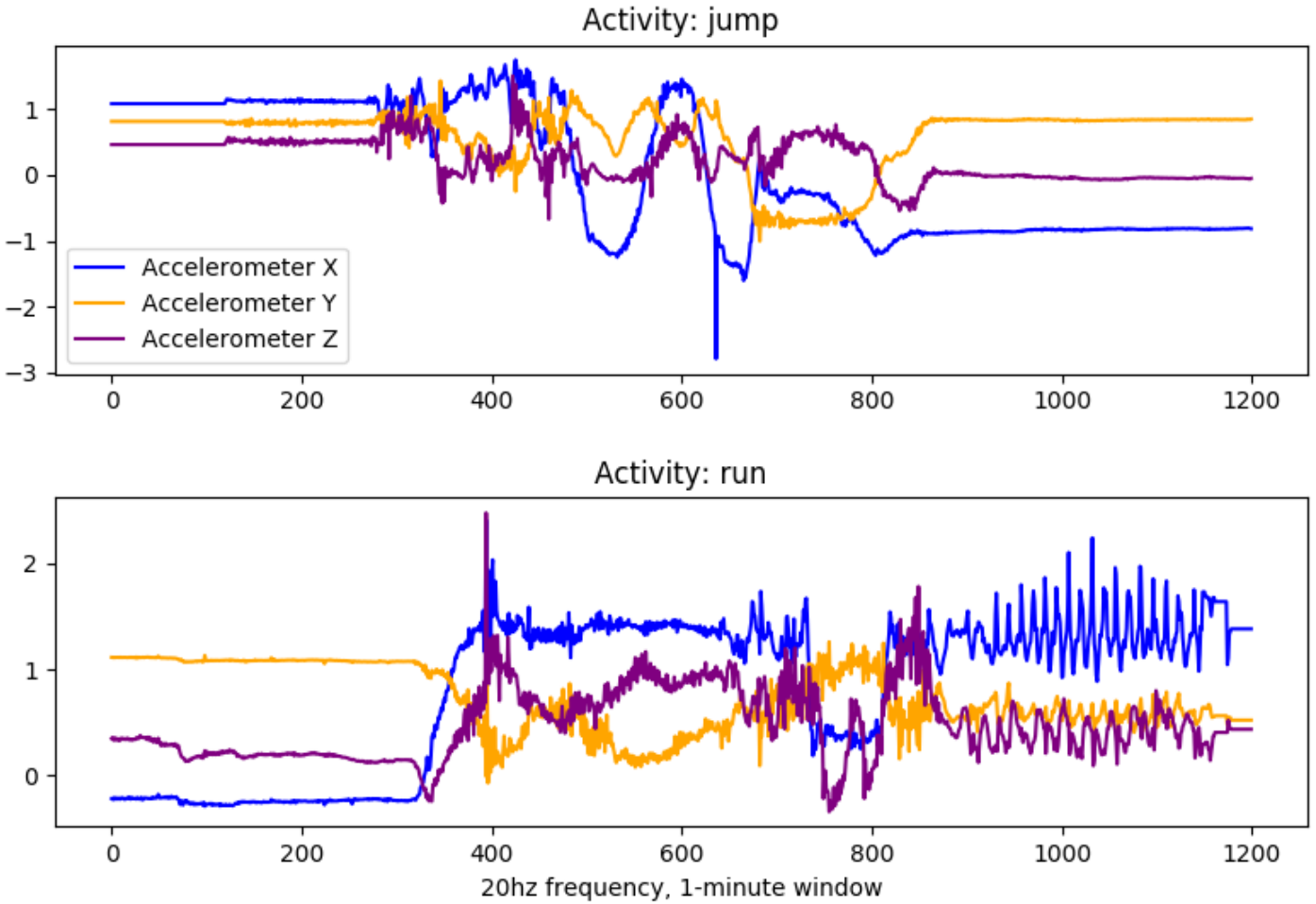


Figure: Illustration of 1D movement data with three channels measured with an accelerometer sensor in the course of a human activity recognition task.

TYPES: 1D - SENSOR DATA

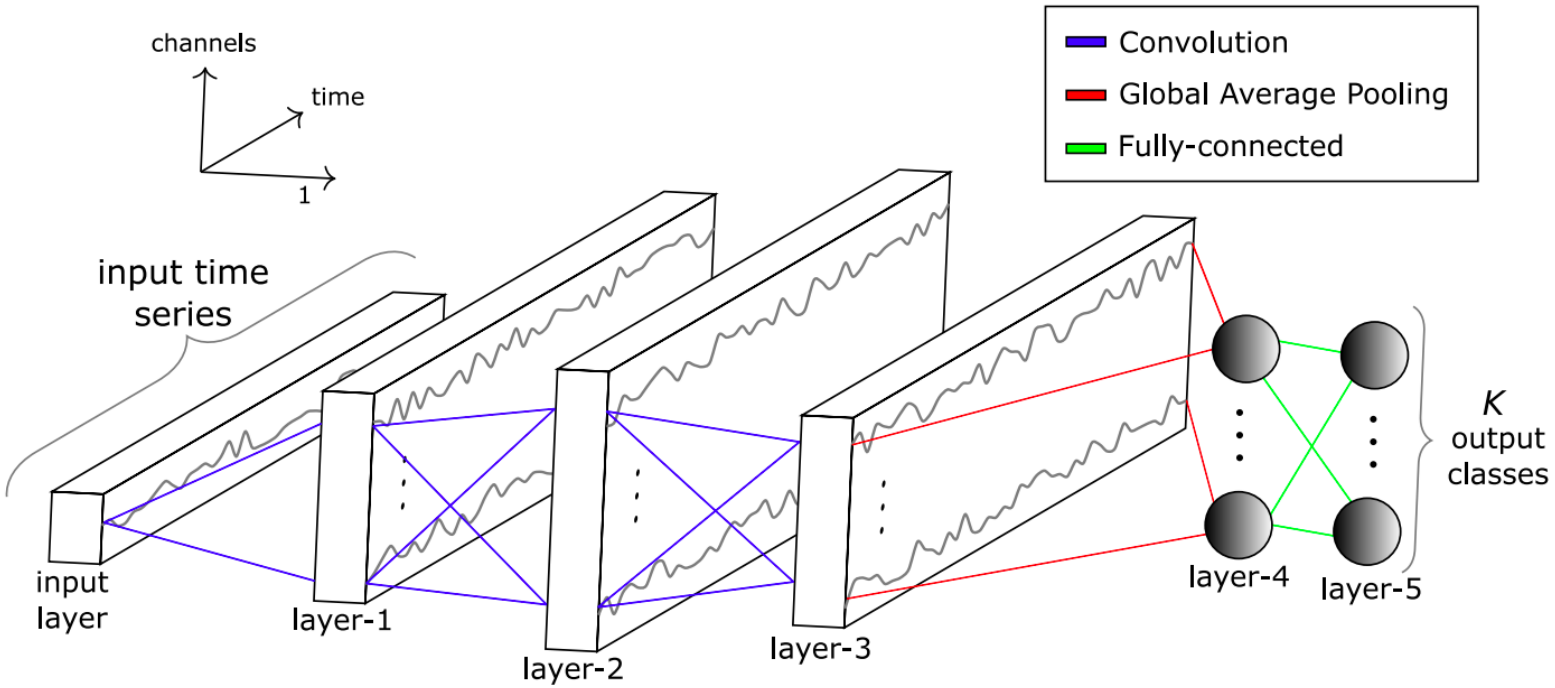


Figure: Time series classification with 1D CNNs and global average pooling (explained later). An input time series is convolved with 3 CNN layers, pooled and fed into an MLP layer for classification. This is one of the classic time series classification architectures as proposed in
[Wang Zhiguang et. al , 2017].

TYPES: 1D - TEXT MINING

- 1D convolutions also have an interesting application in text mining [Zhang Xiang et al., 2015].
- For example, they can be used to classify the sentiment of text snippets such as yelp reviews.

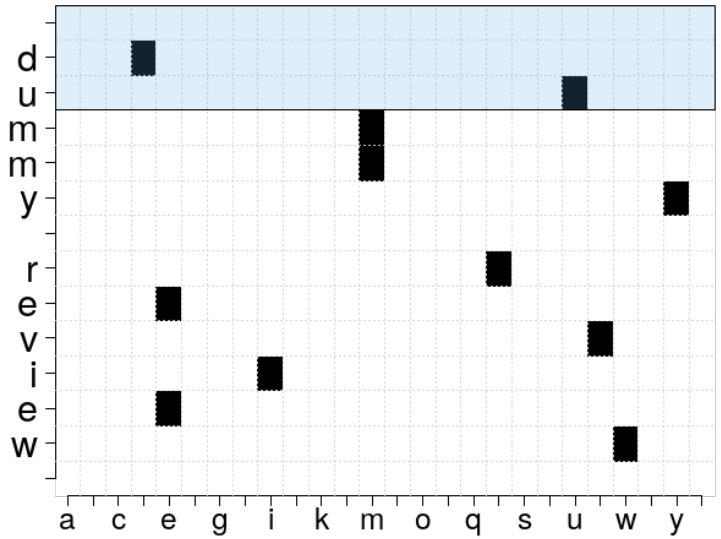
Miriam L.
Munich, Germany
57 friends
437 reviews
450 photos

7/18/2010

The LMU is main building one of the most beautiful buildings in München...nicht only in relation to the architecture just great, but above all also if the history here has taken place, is a conscious.

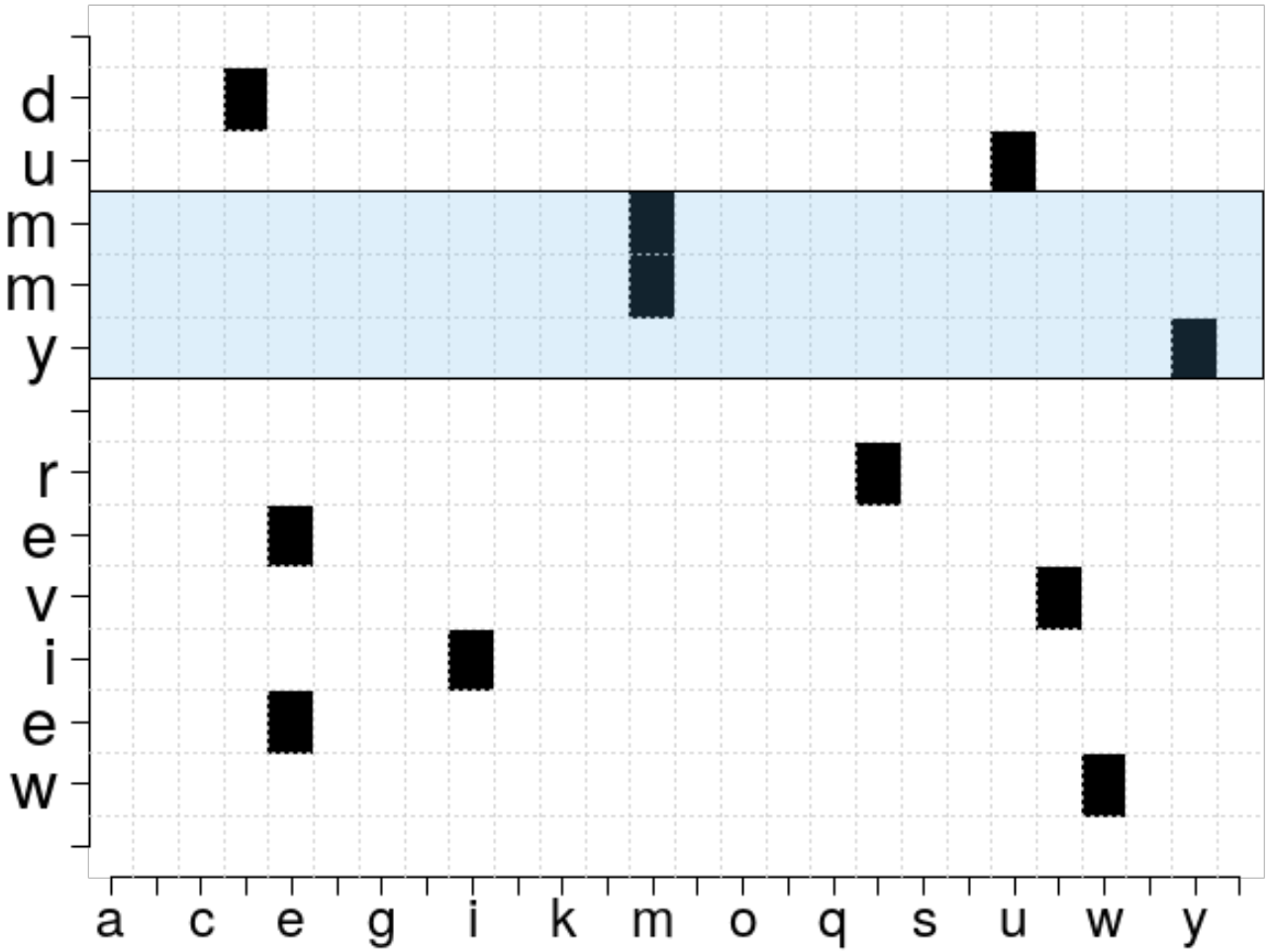
Figure: Sentiment classification: can we teach the net that this a positive review?

TYPES: 1D - TEXT MINING



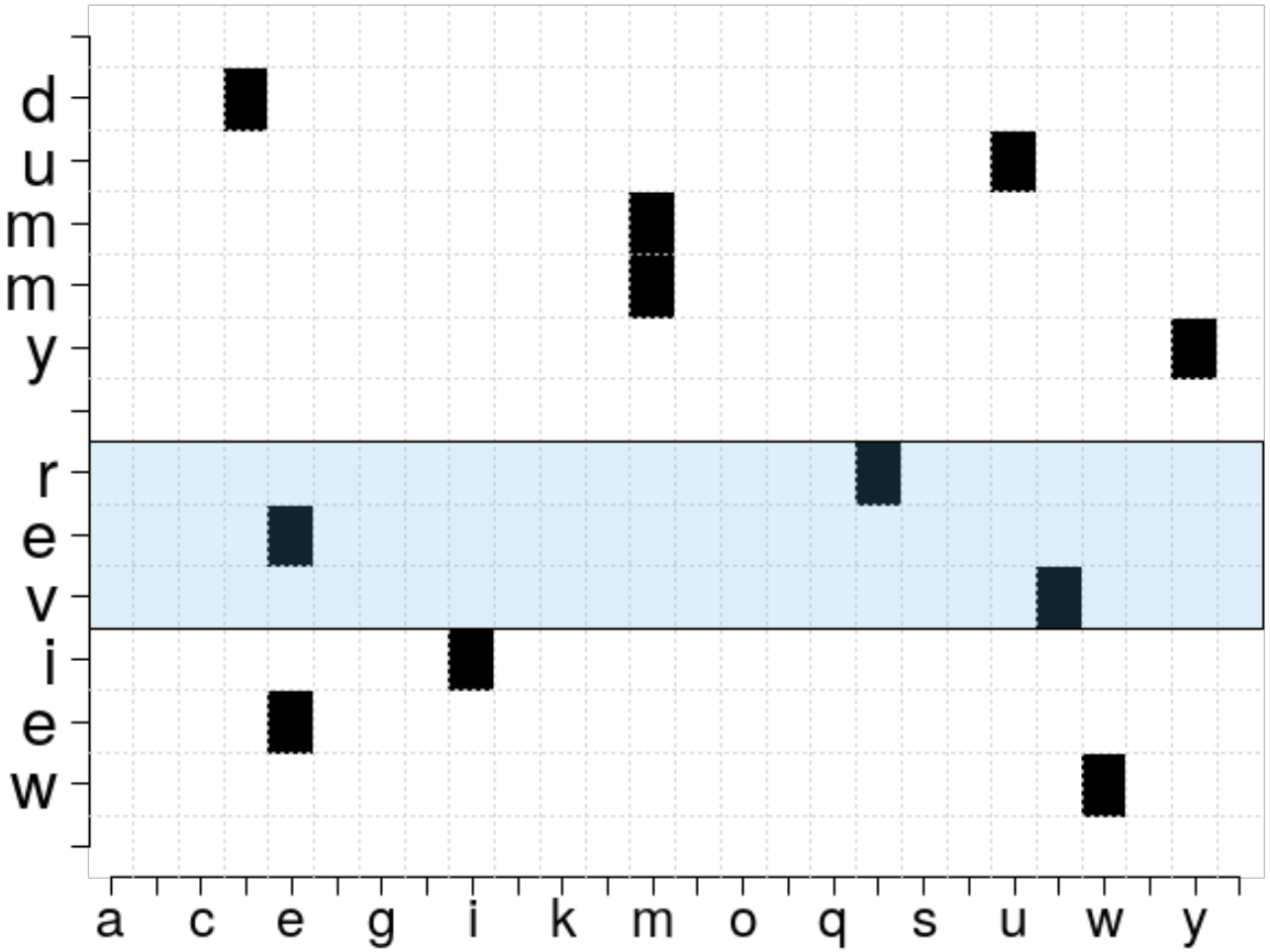
- We encode the text reviews (here: *dummy review*) using a given alphabet.
- Each character is transformed into a one-hot vector. Character *d* contains only zero values for all positions in the vector except position four.
- The max length of each review is set to 1014: shorter texts are padded with spaces (zero-vectors) and longer texts are simply cut.

TYPES: 1D - TEXT MINING



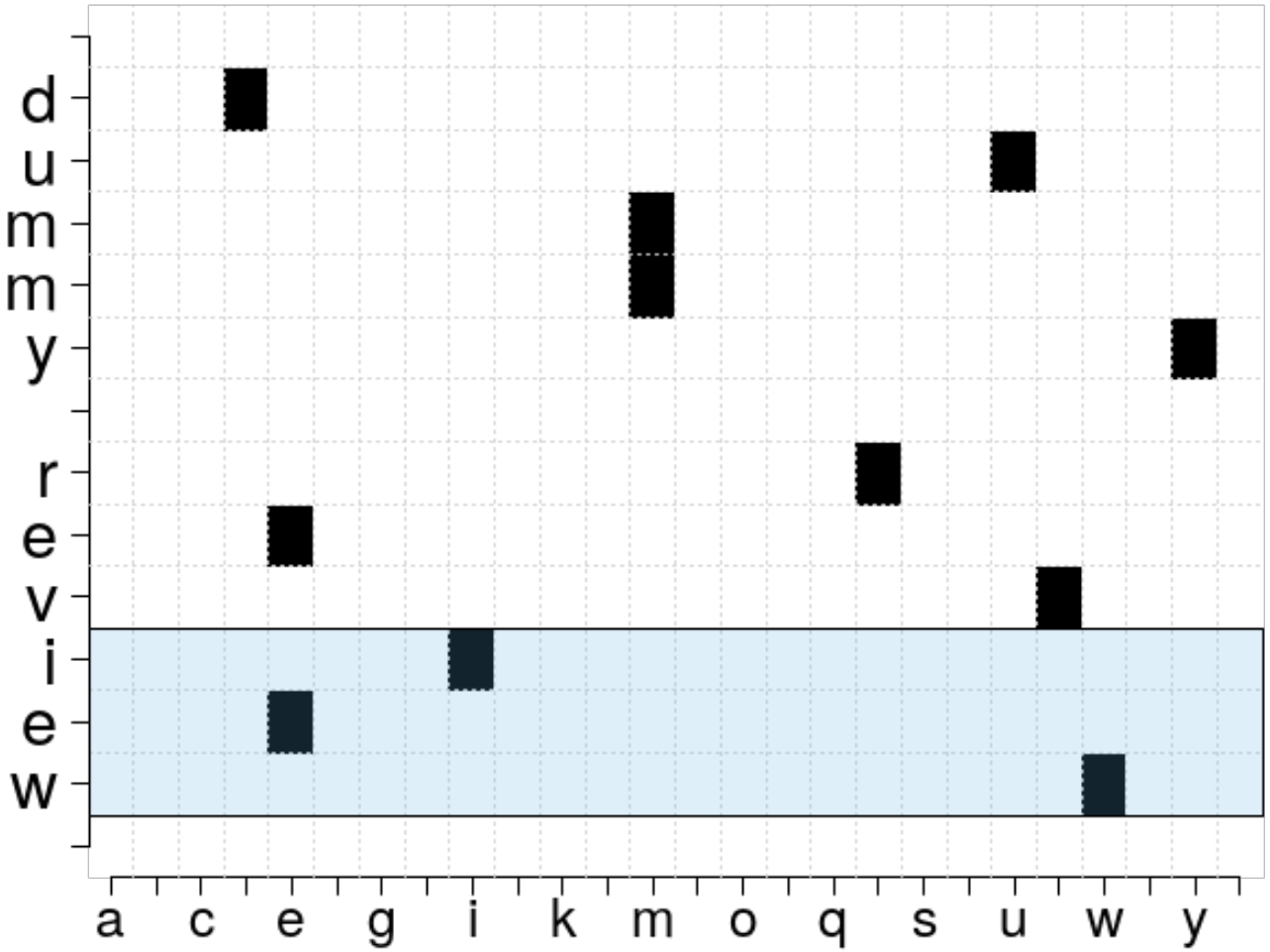
- The data is represented as 1D signal with channel size = size of the alphabet.

TYPES: 1D - TEXT MINING



- The temporal dimension is shown as the y dimension for illustrative purposes.

TYPES: 1D - TEXT MINING



- The 1D-Kernel (blue) convolves the input in the temporal y-dimension yielding a 1D feature vector.

TYPES: 1D - TEXT MINING

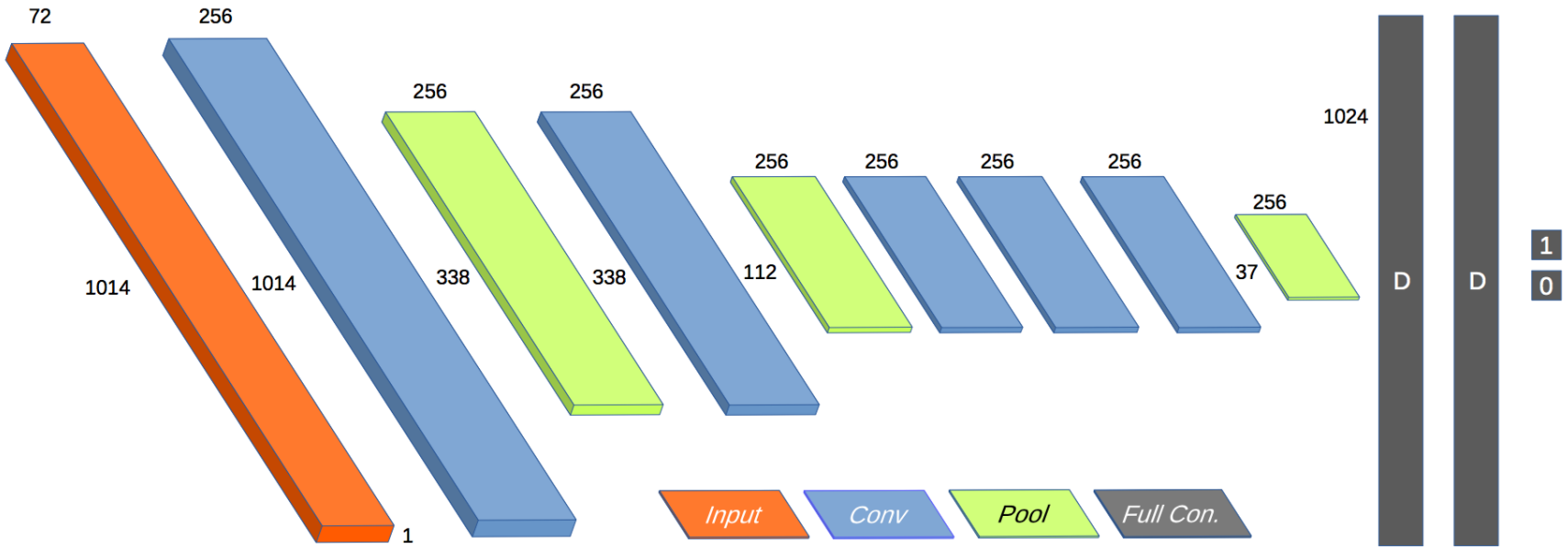


Figure: The architecture proposed in [Zhang Xiang et al., 2015]: an alphabet containing 72 (special-) characters is used for encoding and the text length is fixed to 1014. This yields an accuracy of 96% with a net trained on 500K reviews.

TYPES: 2D

- Problem setting: work with 2 dimensional data such as images.
- Refer to the previous lecture for a comprehensive introduction to 2 dimensional convolutions.

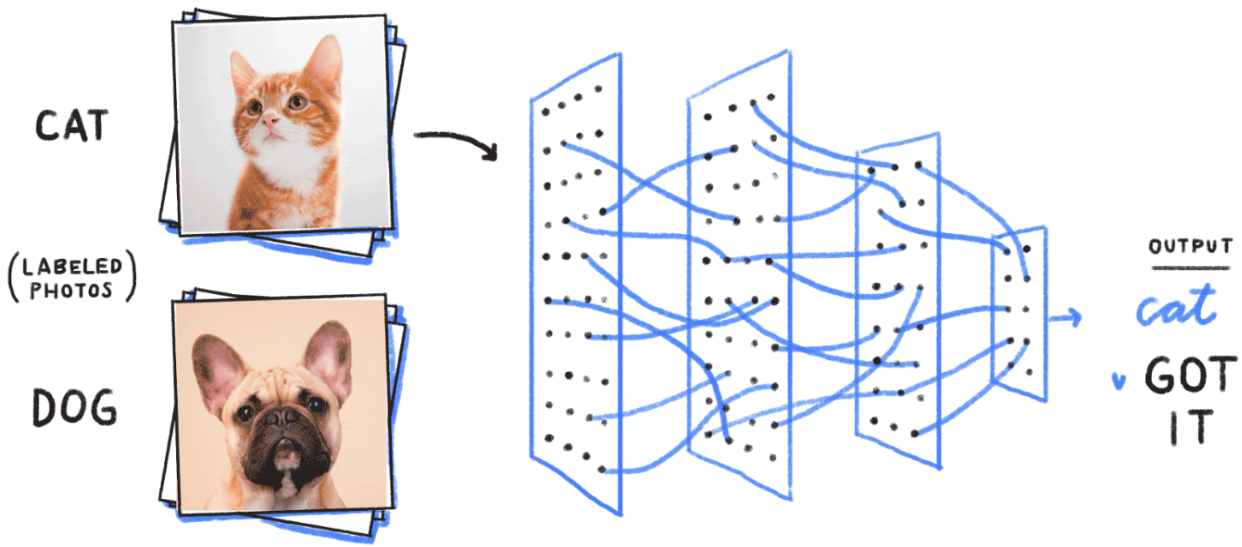


Figure: Simple binary classification problem on 2D image data: cat vs. dog detection as illustrated here.

TYPES: 3D

- Problem setting: settings with data containing 3 dimensions.
- Solution: move the three-dimensional kernel in x, y and z direction to capture all important information.
- Data consists of tensors with shape [channels, xdim, ydim, zdim].
- Dimensions can be both, temporal (video frames) or spatial (MRI).
- Examples:
  - Human activity recognition in video data in one of the first papers in this domain [Du Tran et al., 2015].
  - Disease classification or tumor segmentation on MRI scans [Milletari Fausto et al., 2016].

TYPES: 3D - DATA

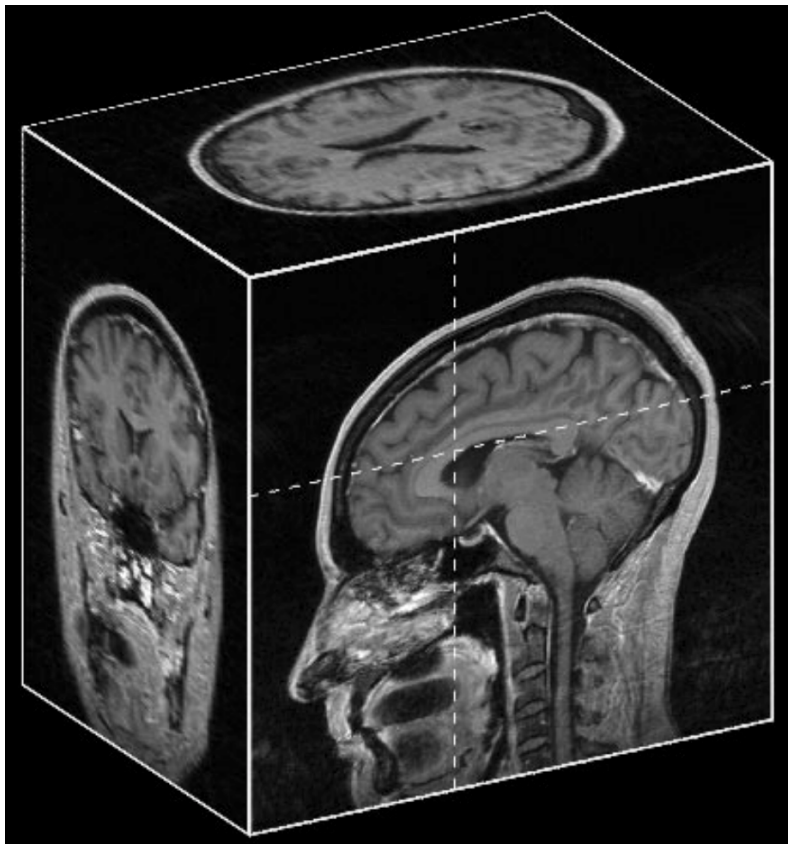


Figure: Illustration of single-channel volumetric data: MRI scan by [Benoit A. Gennart et al., 1996]. Each slice of the stack has one channel, as the frames are black-white.

TYPES: 3D - DATA

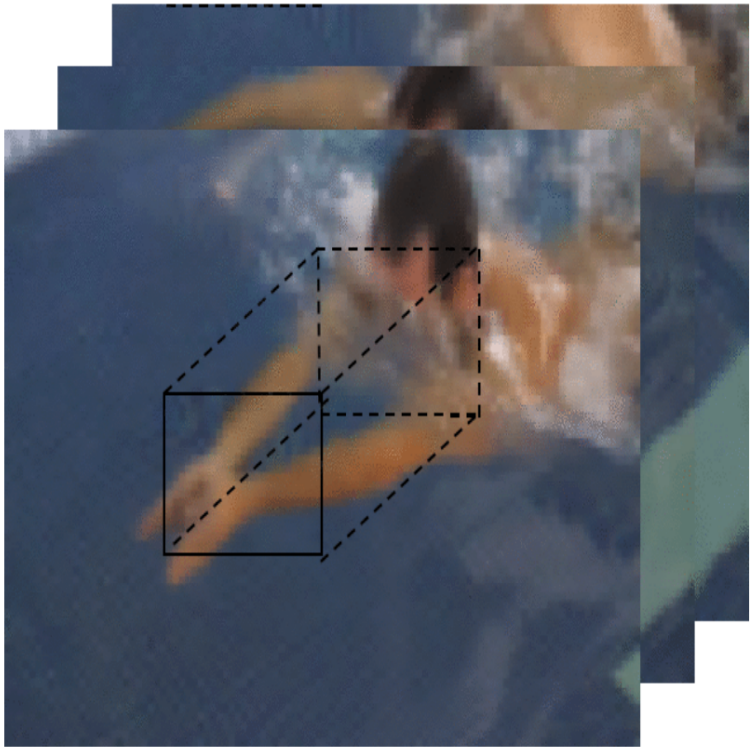


Figure: Illustration of multi-channel volumetric data: video snippet of an action detection task. The video consists of several slices, stacked in temporal order. Frames have three channels, as they are RGB.

TYPES: 3D

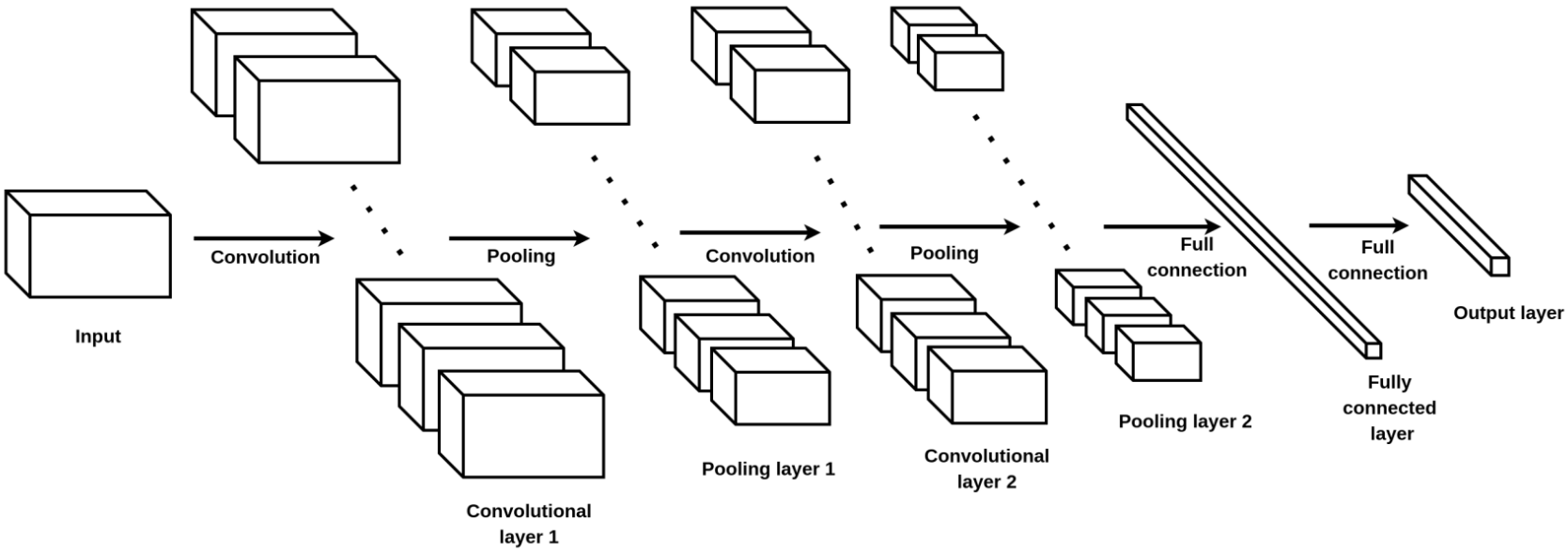
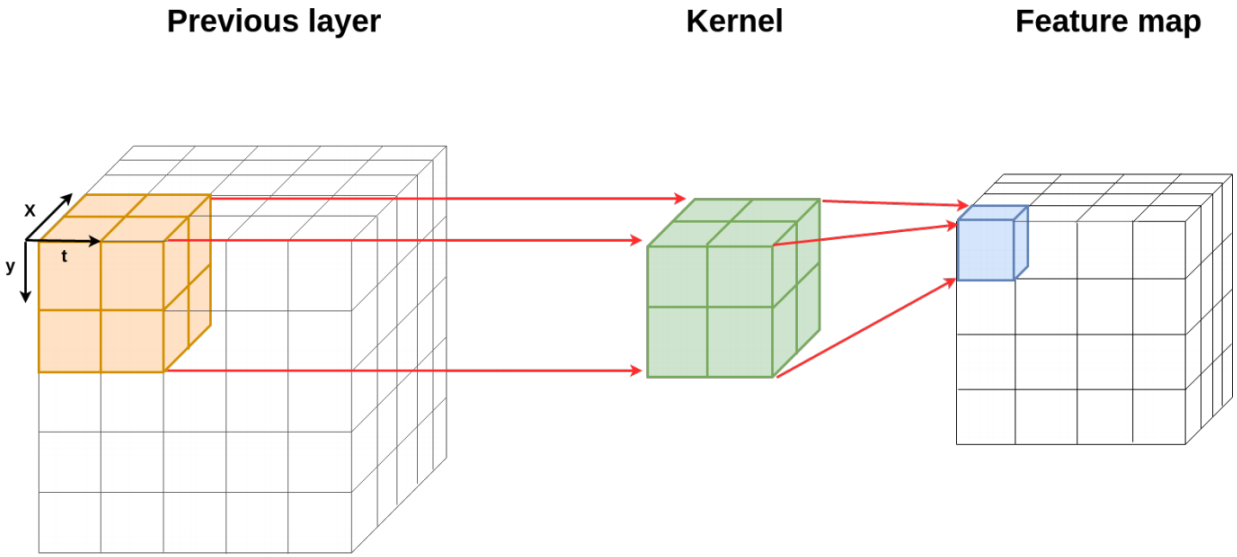


Figure: Basic 3D-CNN architecture.

- Basic architecture of the CNN stays the same.
- 3D convolutions output 3D feature maps which are element-wise activated and then (eventually) pooled in 3 dimensions.

TYPES: 3D



- 3D convolution can be expressed as:

$$H(i,j,k) = (\mathcal{I} \star \mathcal{G})(i,j,k) = \sum_x \sum_y \sum_z \mathcal{I}(x,y,z)\mathcal{G}(i-x, j-y, k-z)$$

- Note : 3D convolutions yield a 3D output.

TYPES: DILATED CONVOLUTIONS

- Idea : artificially increase the receptive field of the net without using more filter weights.
- Benefit :
  - Detection of fine-details by processing inputs in higher resolutions.
  - Broader view of the input to capture more contextual information with less depth.
  - Improved run-time-performance due to less parameters.
- Idea : Add new dilation parameter to the kernel $\mathcal{K}$ that skips pixels during convolution.
- Dilated kernel is basically a regular convolutional kernel interleaved with zeros.

TYPES: DILATED CONVOLUTIONS

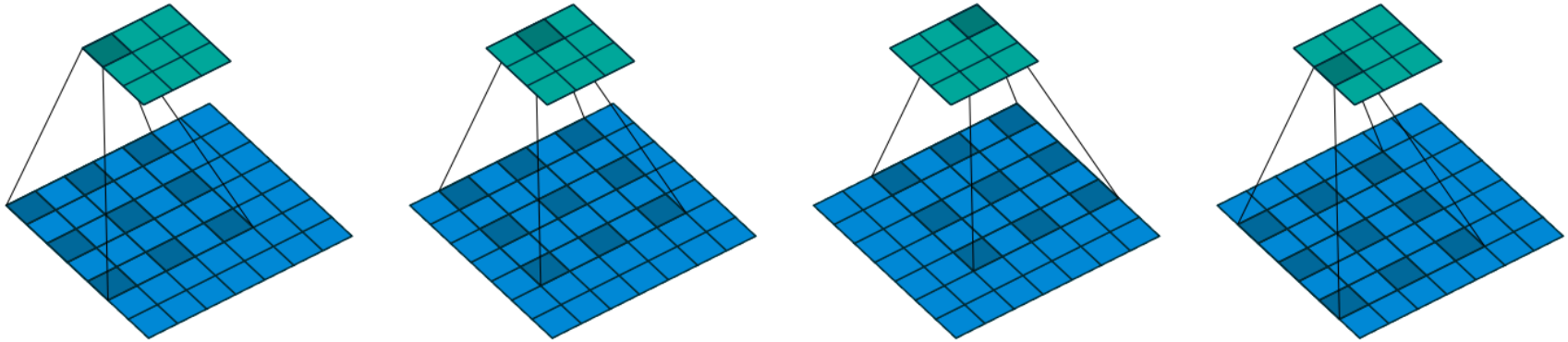


Figure: Dilated convolution on 2-dimensional data as shown in [Vincent Dumoulin et al., 2016]. It is observable, that the dilation can be interpreted as an increased kernel filled with zero values.

TYPES: DILATED CONVOLUTIONS

- Useful in applications where the global context is of great importance for the model decision.
- This component finds application in:
  - Generation of audio-signals and songs within the famous Wavenet developed by DeepMind [Aaron van den Oord et al., 2016].
  - Time series classification and forecasting [Bai Shaojie et. al, 2018].
  - Image segmentation [Fisher Yu et. al, 2015].

TYPES: DILATED CONVOLUTIONS

- Dilation increases the receptive field of the model - what is a receptive field?
- Receptive field: the visual field of a single neuron in a specific layer.
- Huge receptive field of neurons in the last hidden layer: they can capture a lot more global information from the input.

TYPES: DILATED CONVOLUTIONS

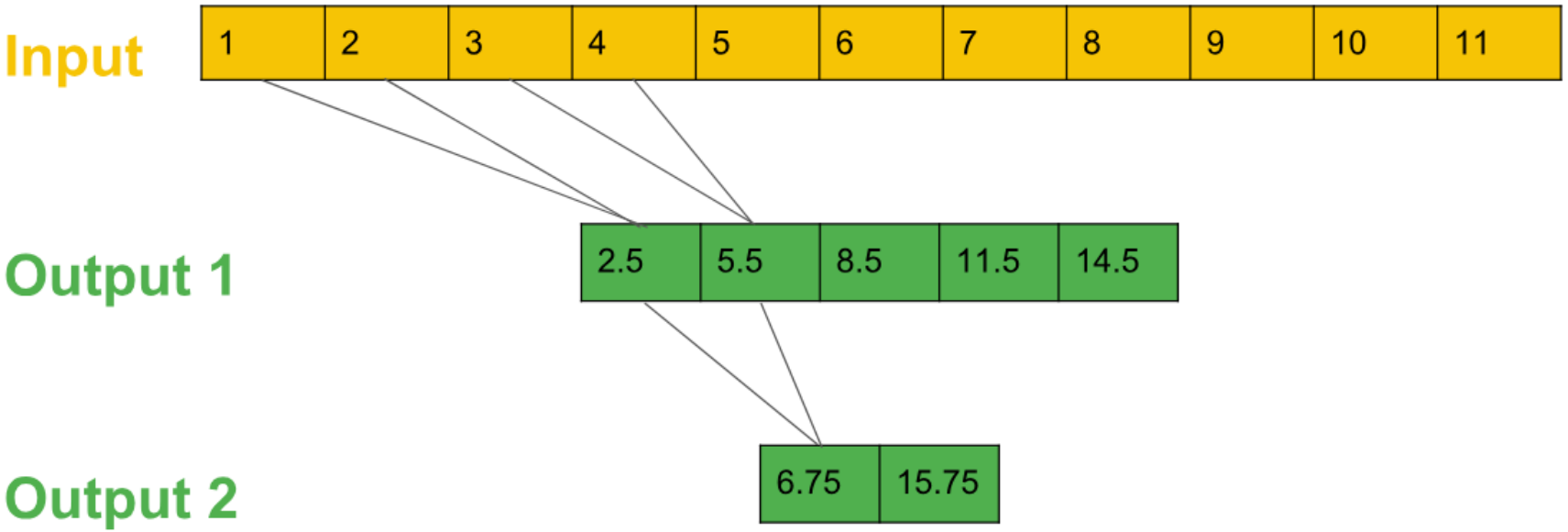


Figure: Regular convolution: we use a convolutional kernel of size 2 with the fixed weights $\{0.5, 1.0\}$ and convolve the input vector with a stride of 2. We do not dilate the kernel in this example (the dilation factor is one) and one neuron in layer 2 has a receptive field of size 4 after two stacked layers.

TYPES: DILATED CONVOLUTIONS

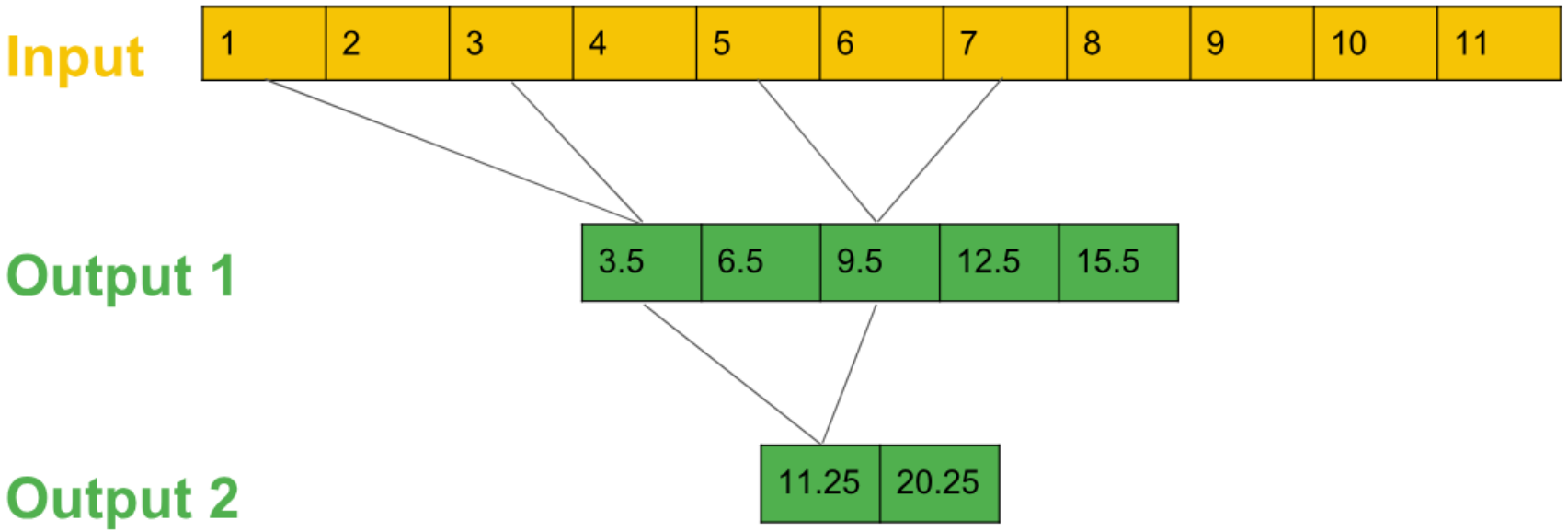


Figure: Dilated convolution: we use a convolutional kernel of size 2 with the fixed weights $\{0.5, 1.0\}$ and convolve the input vector with a stride of 2. We dilate the kernel with a factor of 2 in this example and one neuron in layer 2 has a receptive field of size 8 after two stacked layers.

TYPES: DILATED CONVOLUTIONS

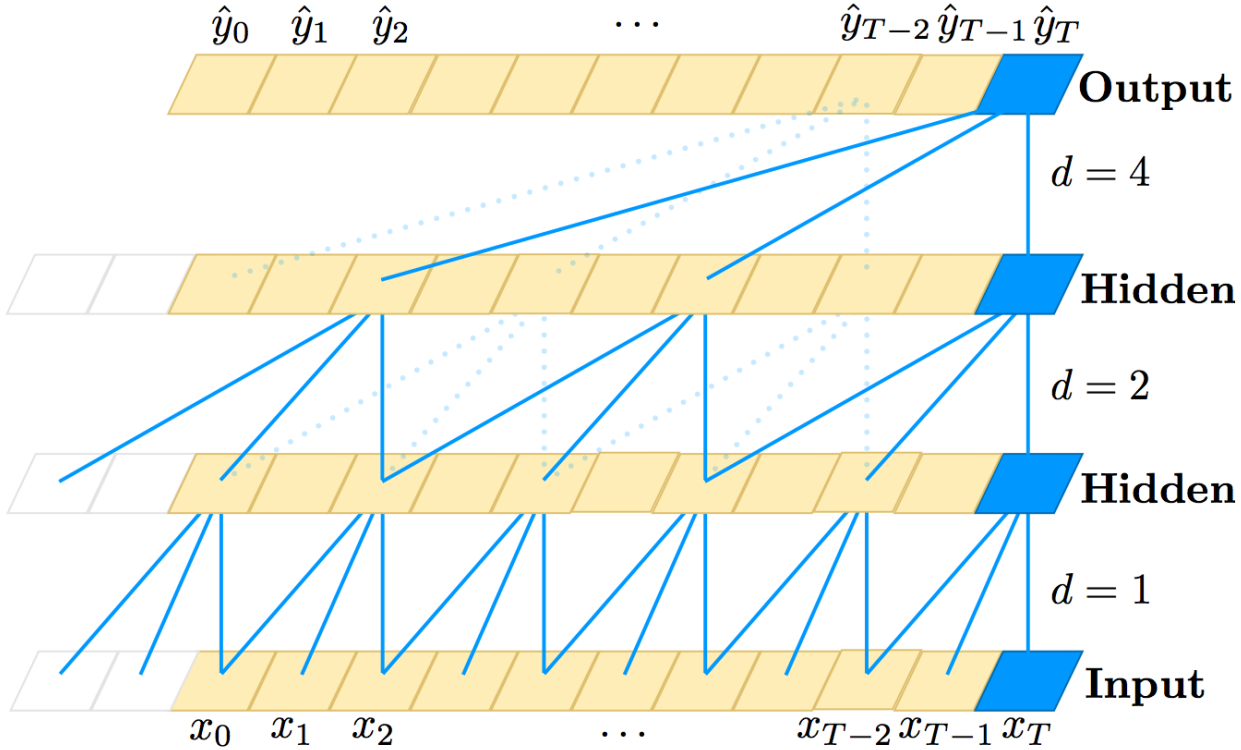


Figure: Application of dilated convolutions on time series for classification or seq2seq prediction [Bai Shaojie et. al, 2018]. The dilations are used to drastically increase the context information for each output neuron y_i with relatively few layers.

TYPES: TRANSPOSED CONVOLUTIONS

- Idea: re-increase dimensionality of a feature map instead of decreasing it as with regular convolutions.
- Problem setting: required module in Encoder-Decoder architectures such as (variational) Autoencoders, Segmentation Nets, GANs (covered in later lectures).
- In such architectures, dimensions of the feature maps are getting reduced throughout the architecture. Transposed convolutions are used to get back to the initial dimensionality.
- Note : Do not confuse this with deconvolutions (which are mathematically defined as the inverse of a convolution).

TYPES: TRANSPOSED CONVOLUTIONS

- Example 1:
  - Input: blue feature map with dim 4×4.
  - Output: turquoise feature map with dim 2×2.

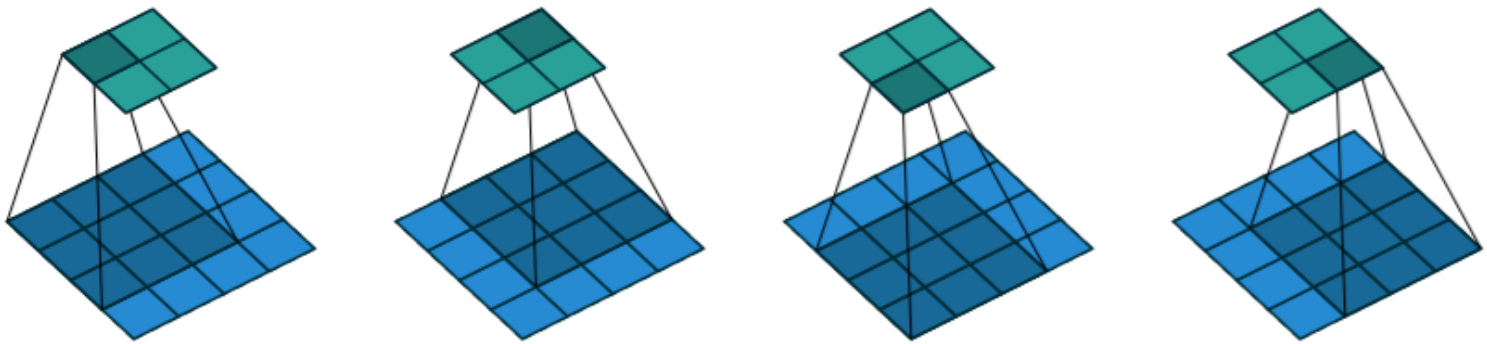


Figure: A **regular** convolution with kernel-size $k = 3$, padding $p = 0$ and stride $s = 1$.

Here, the feature map shrinks from 4×4 to 2×2.

TYPES: TRANSPOSED CONVOLUTIONS

- Example 1:
  - Now, let's upsample the 2×2 feature map back to a 4×4 feature map.
  - Input: 2×2 (blue). Output : 4×4 (turquoise).
- One way to upsample is to use a regular convolution with various padding strategies.

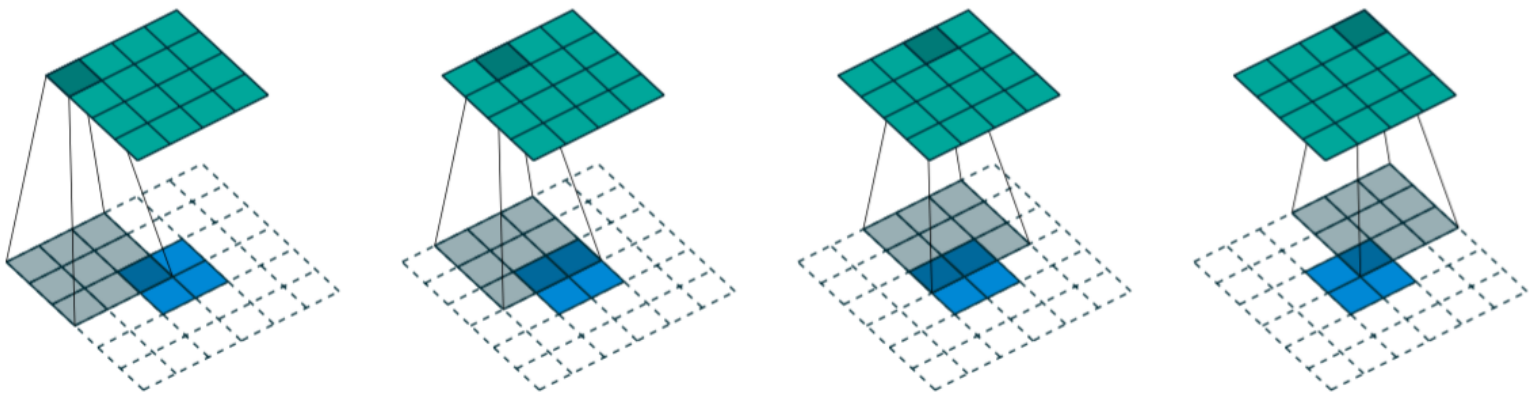


Figure: Transposed convolution can be a seen as a regular convolution.

Convolution (above) with $k' = 3, s' = 1, p' = 2$ re-increases dimensionality from 2×2 to 4×4 as shown in [Vincent Dumoulin et al., 2016].

TYPES: TRANSPOSED CONVOLUTIONS

Example 2 : Transposed Convolution as matrix multiplication :

$$\underbrace{\begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \end{bmatrix}}_{z} = \begin{bmatrix} w_1 z_1 + w_2 z_2 + w_3 z_3 \\ w_1 z_2 + w_2 z_3 + w_3 z_4 \\ w_1 z_3 + w_2 z_4 + w_3 z_5 \\ w_1 z_4 + w_2 z_5 + w_3 z_6 \end{bmatrix} = \begin{bmatrix} z_7 \\ z_8 \\ z_9 \\ z_{10} \end{bmatrix}$$

Figure: A regular 1D convolution. kernel size = 3, stride = 1 , padding = 0. The vector *z* is in the input feature map. The matrix *K* represents the convolution operation.

A regular convolution decreases the dimensionality of the feature map from 6 to 4.

TYPES: TRANSPOSED CONVOLUTIONS

Example 2 : Transposed Convolution as matrix multiplication :

$$\underbrace{\begin{bmatrix} w_1 & 0 & 0 & 0 \\ w_2 & w_1 & 0 & 0 \\ w_3 & w_2 & w_1 & 0 \\ 0 & w_3 & w_2 & w_1 \\ 0 & 0 & w_3 & w_2 \\ 0 & 0 & 0 & w_3 \end{bmatrix}}_{K^T} \begin{bmatrix} z_7 \\ z_8 \\ z_9 \\ z_{10} \end{bmatrix} = \begin{bmatrix} w_1 z_7 \\ w_2 z_7 + w_1 z_8 \\ w_3 z_7 + w_2 z_8 + w_1 z_9 \\ w_3 z_8 + w_2 z_9 + w_1 z_{10} \\ w_3 z_9 + w_2 z_{10} \\ w_3 z_{10} \end{bmatrix} = \begin{bmatrix} \tilde{z}_1 \\ \tilde{z}_2 \\ \tilde{z}_3 \\ \tilde{z}_4 \\ \tilde{z}_5 \\ \tilde{z}_6 \end{bmatrix}$$

Figure: A transposed convolution can be used to upsample the feature vector of length 4 back to a feature vector of length 6.

Few important things to note :

1) : Even though the transpose of the original matrix is shown in this example, the actual values of the weights are different from the original matrix (and tuned by backpropagation).

2) : The goal of the transposed convolution here is simply to get back the original dimensionality. It is *not* necessarily to get back the original feature map itself.

TYPES: TRANSPOSED CONVOLUTIONS

Example 2 : Transposed Convolution as matrix multiplication :

$$\underbrace{\begin{bmatrix} w_1 & 0 & 0 & 0 \\ w_2 & w_1 & 0 & 0 \\ w_3 & w_2 & w_1 & 0 \\ 0 & w_3 & w_2 & w_1 \\ 0 & 0 & w_3 & w_2 \\ 0 & 0 & 0 & w_3 \end{bmatrix}}_{K^T} \begin{bmatrix} z_7 \\ z_8 \\ z_9 \\ z_{10} \end{bmatrix} = \begin{bmatrix} w_1 z_7 \\ w_2 z_7 + w_1 z_8 \\ w_3 z_7 + w_2 z_8 + w_1 z_9 \\ w_3 z_8 + w_2 z_9 + w_1 z_{10} \\ w_3 z_9 + w_2 z_{10} \\ w_3 z_{10} \end{bmatrix} = \begin{bmatrix} \tilde{z}_1 \\ \tilde{z}_2 \\ \tilde{z}_3 \\ \tilde{z}_4 \\ \tilde{z}_5 \\ \tilde{z}_6 \end{bmatrix}$$

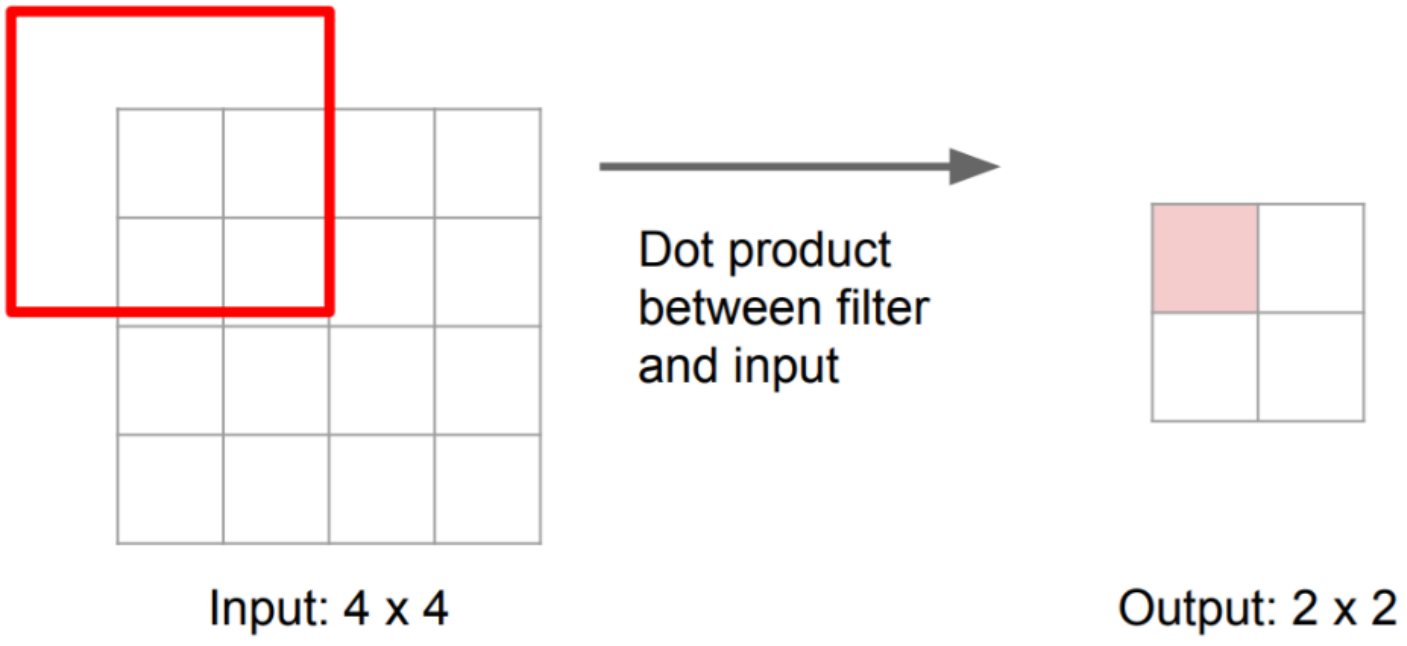
Figure: A transposed convolution can be used to upsample the feature vector of length 4 back to a feature vector of length 6.

3) : The elements in the downsampled vector only affect those elements in the upsampled vector that they were originally "derived" from. For example, z_7 was computed using z_1 , z_2 and z_3 and it is only used to compute $\tilde{z}_1$, $\tilde{z}_2$ and $\tilde{z}_3$.

4) : In general, transposing the original matrix doesn't result in a convolution. But a transposed convolution can always be implemented as a regular convolution by using various padding strategies (this would not be very efficient, however).

TYPES: TRANSPOSED CONVOLUTIONS

Example 3 : Let's now view transposed convolutions from a different perspective.

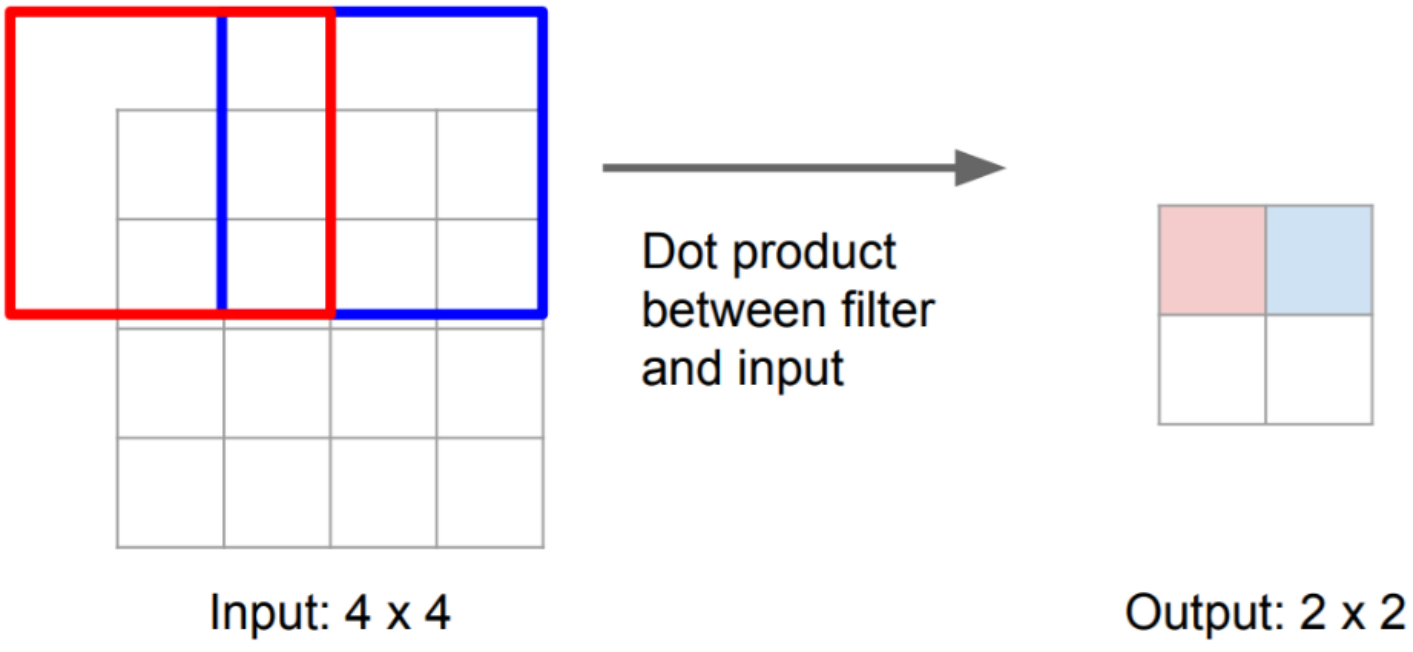


credit:Stanford

Figure: Regular 3 $\times$ 3 convolution, stride 2 , pad 1.

TYPES: TRANSPOSED CONVOLUTIONS

Example 3 : Let's now view transposed convolutions from a different perspective.

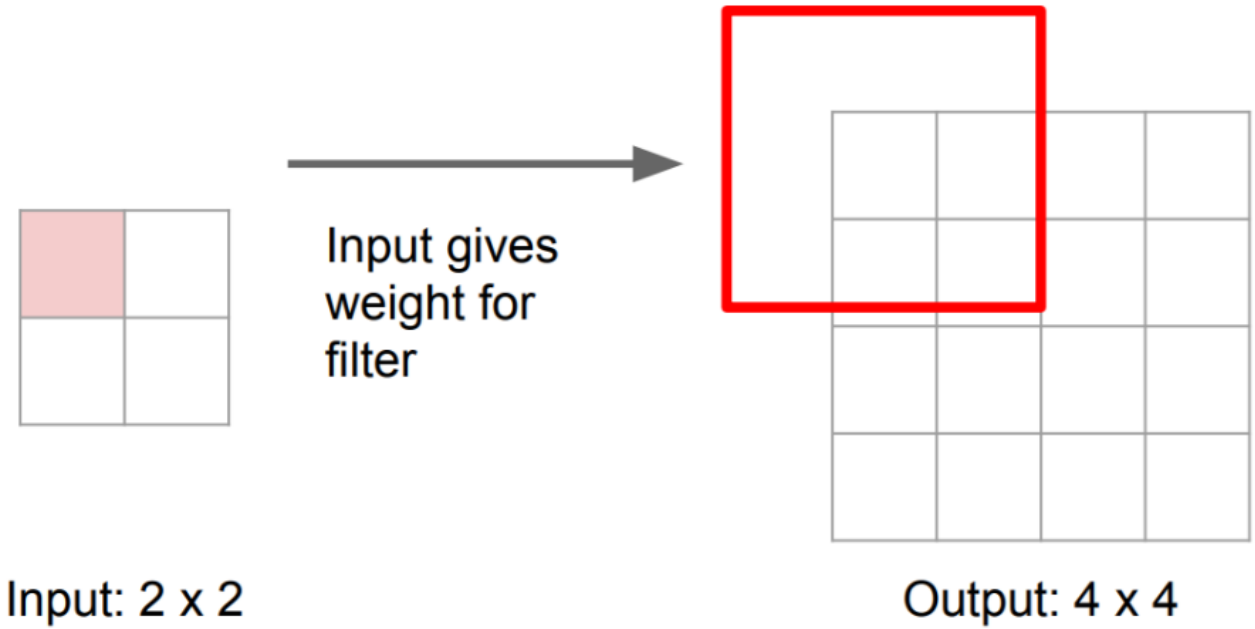


credit:Stanford

Figure: Regular 3 $\times$ 3 convolution, stride 2 , pad 1.

TYPES: TRANSPOSED CONVOLUTIONS

Example 3 : Let's now view transposed convolutions from a different perspective.



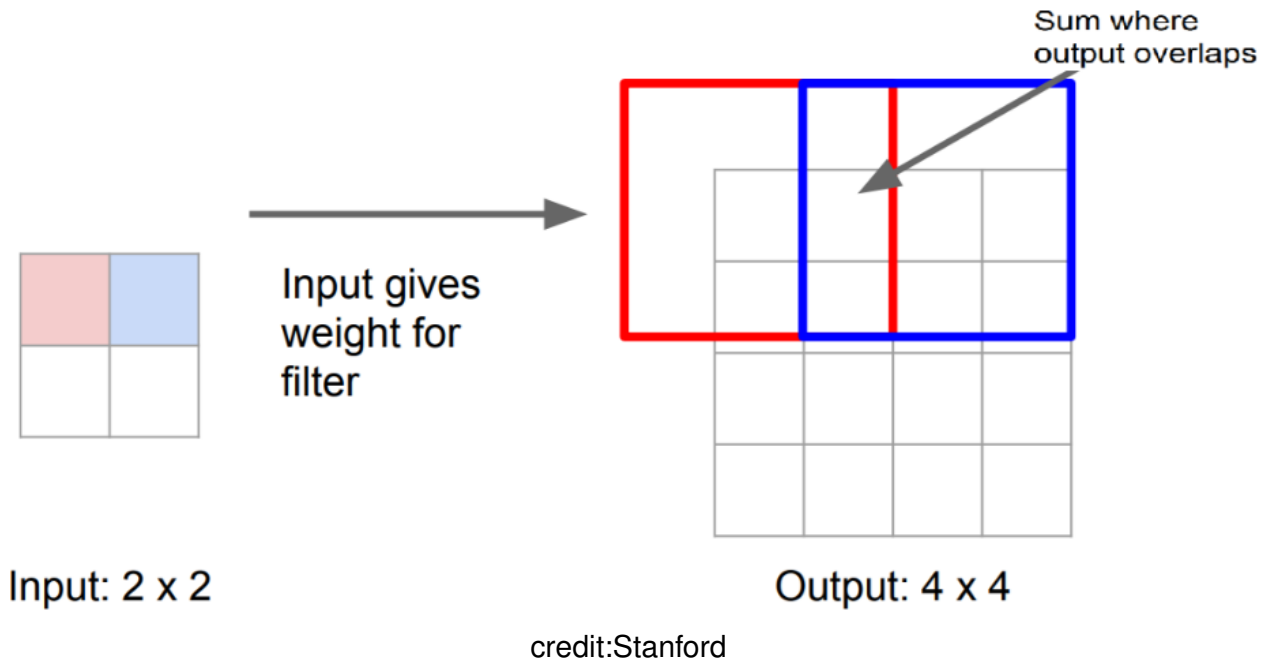
credit:Stanford

Figure: *Transposed* 3×3 convolution, stride 2 , pad 1. Note : stride now refers to the "stride" in the *output*.

Here, the filter is *scaled* by the input.

TYPES: TRANSPOSED CONVOLUTIONS

Example 3 : Let's now view transposed convolutions from a different perspective.



credit:Stanford

Figure: *Transposed* 3×3 convolution, stride 2 , pad 1. Note : stride now refers to the "stride" in the *output*.

Here, the filter is *scaled* by the input.

TYPES: TRANSPOSED CONVS - DRAWBACK

Figure: Artifacts produced py transposed convolutions as shown in [Distill.pub et. al , 2017].

- Transposed convolutions lead to checkerboard-style artifacts in resulting images.

TYPES: TRANSPOSED CONVS - DRAWBACK

- Explanation: transposed convolution yields an overlap in some feature map values.
- This leads to higher magnitude for some feature map elements than for others, resulting in the checkerboard pattern.
- One solution is to ensure that the kernel size is divisible by the stride.

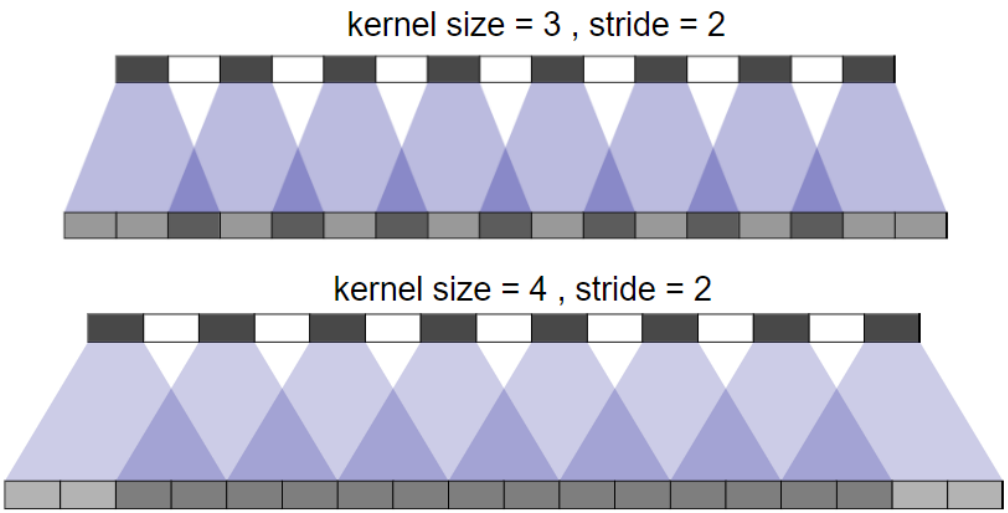


Figure: 1D example. In both images, top row = input and bottom row = output. *Top* : Here, kernel weights overlap unevenly which results in a checkerboard pattern. *Bottom* : There is no checkerboard pattern as the kernel size is divisible by the stride.

TYPES: INCEPTION MODULES

- Problem setting: how do we choose the kernel size in each layer?
- This is often an arbitrary decision.
- Solution: offer the model kernels of different sizes in each layer through which it can propagate information and let it decide, which one to use to which extent.
- Side-effect: massive parameter reduction allowing for deeper architectures.
- First proposed in [Christian Szegedy et. al , 2014].

TYPES: INCEPTION MODULES

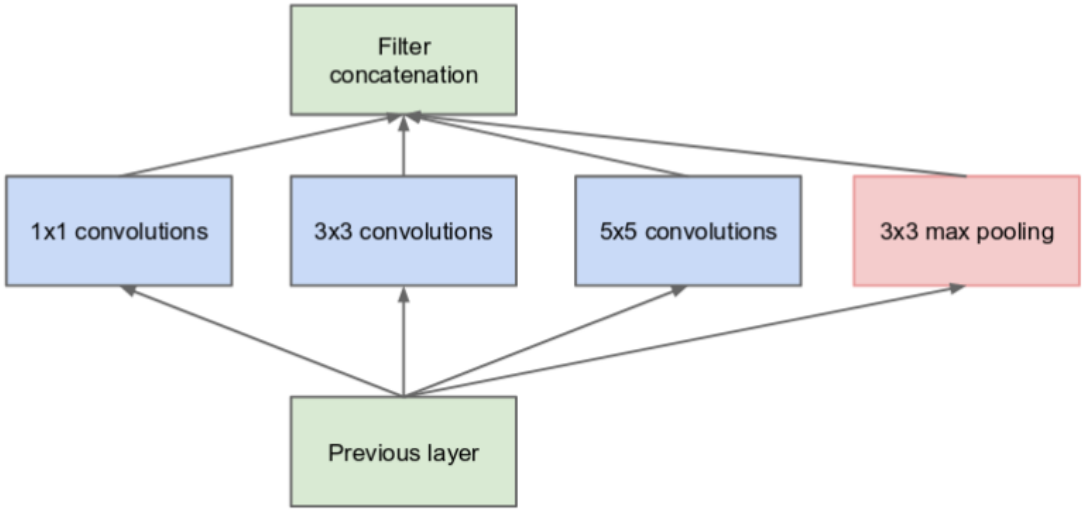


Figure: Naive inception module. The model can “choose” from kernels of different sizes.

Idea : Do several convolutions in parallel and concatenate the resulting feature maps in the depth dimension. This requires equal dimensions of the feature maps created by the parallel convolutions.Thus, same padding is used throughout the parallel convolutions.

TYPES: INCEPTION MODULES

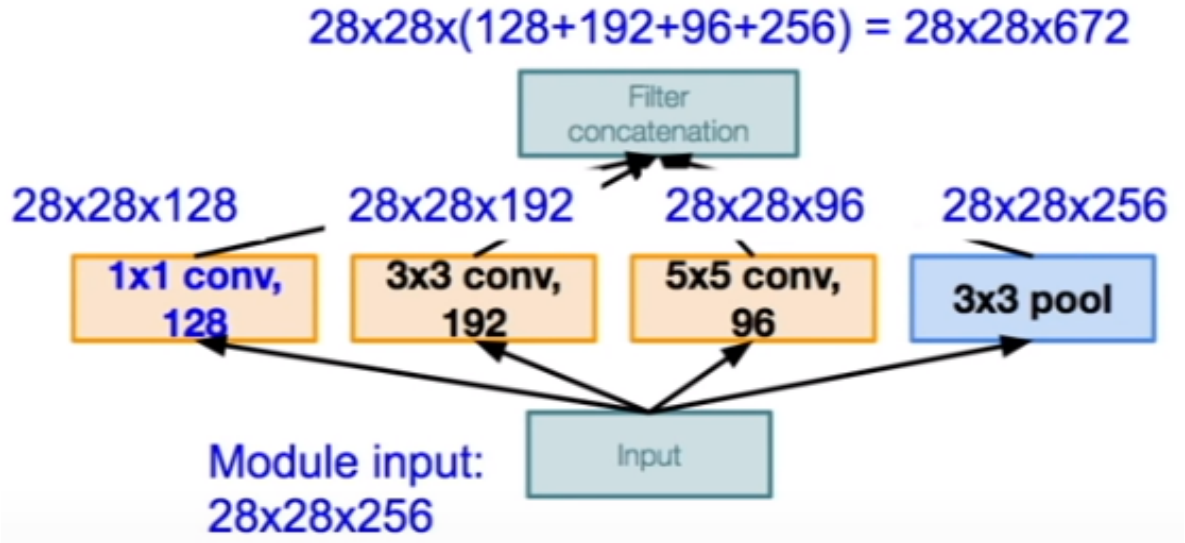


Figure: Naive Inception module - Example

- To allow for the bypass of information throughout one inception module, an 1x1 convolutional layer is also included.
- Max-pooling is used as it is ought to increase the robustness of the feature maps. The kernels are padded accordingly to yield feature maps of equal dimensions.

TYPES: INCEPTION MODULES

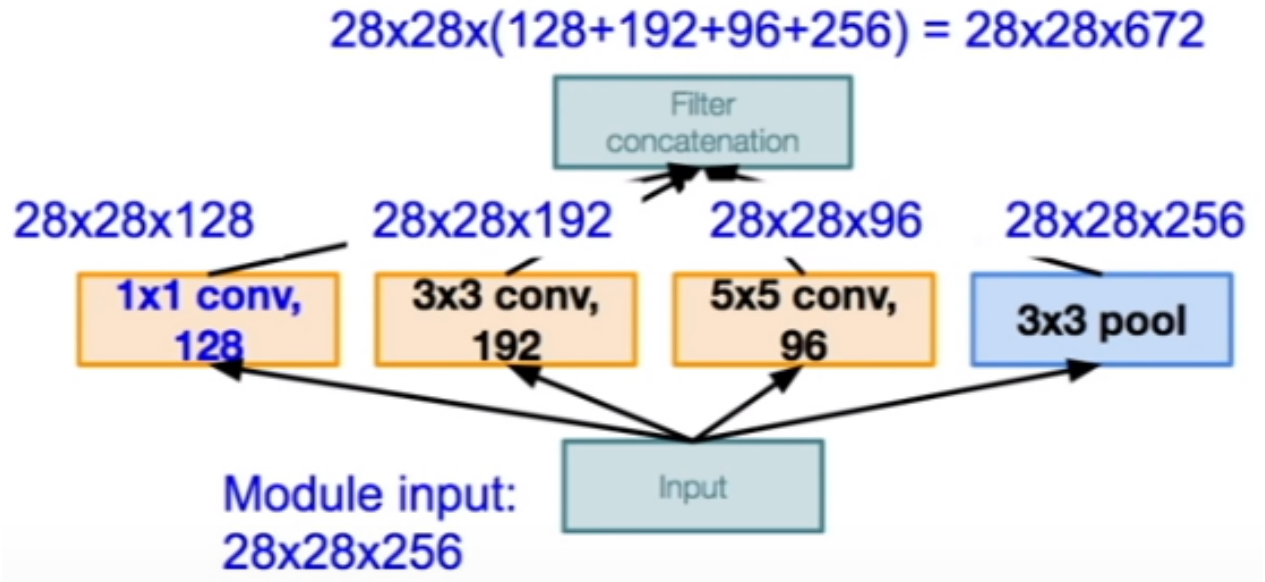


Figure: Naive Inception module - Example

- Resulting feature map blocks are restricted to have the same dimensionality but can be of varying depth.
- The different feature maps are finally concatenated in the depth-dimension and fed to the next layer.

TYPES: INCEPTION MODULES

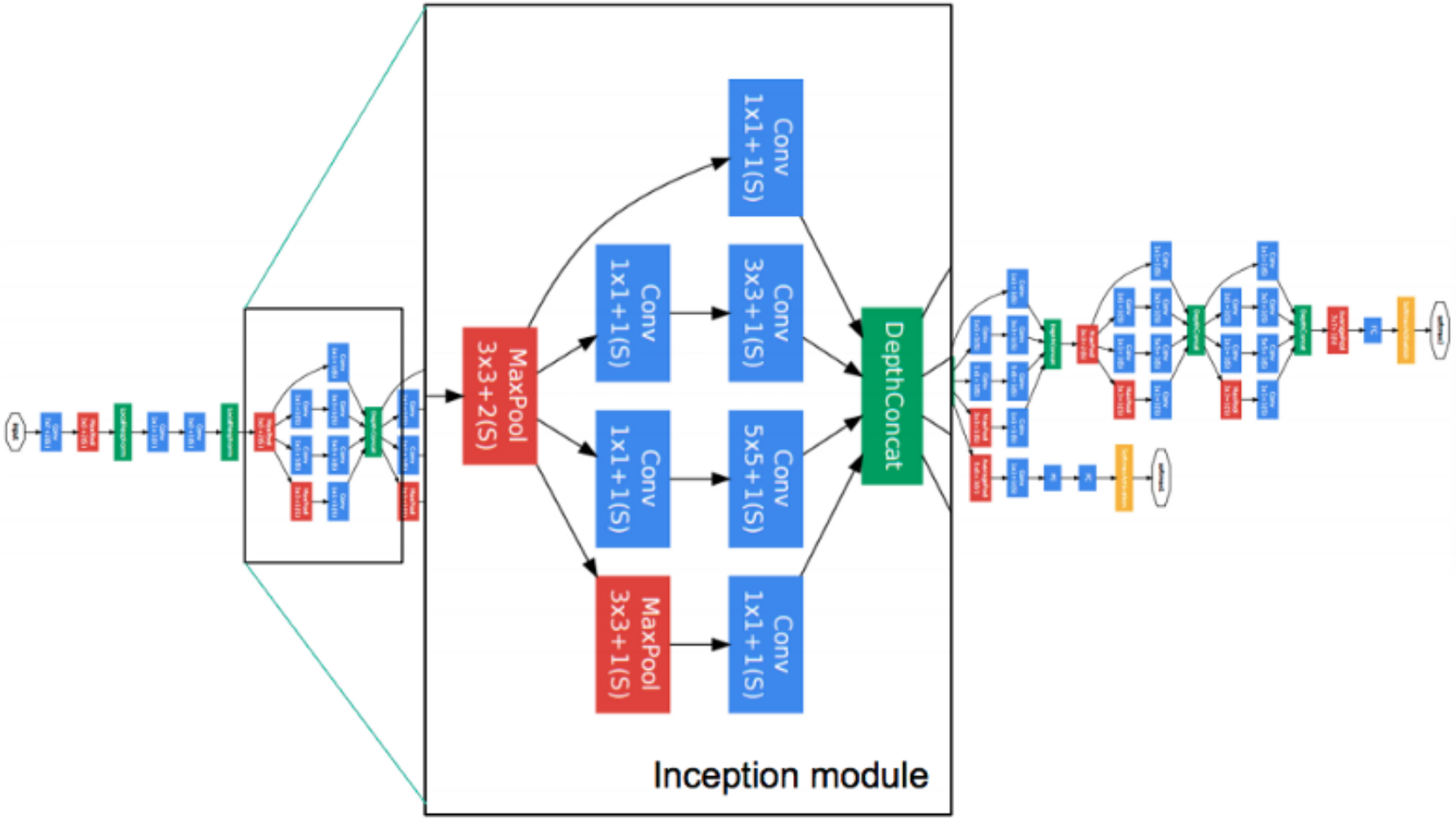


Figure: Inception modules are the integral part of the infamous GoogLeNet (2014), one of the first very deep net architectures.

TYPES: INCEPTION MODULES

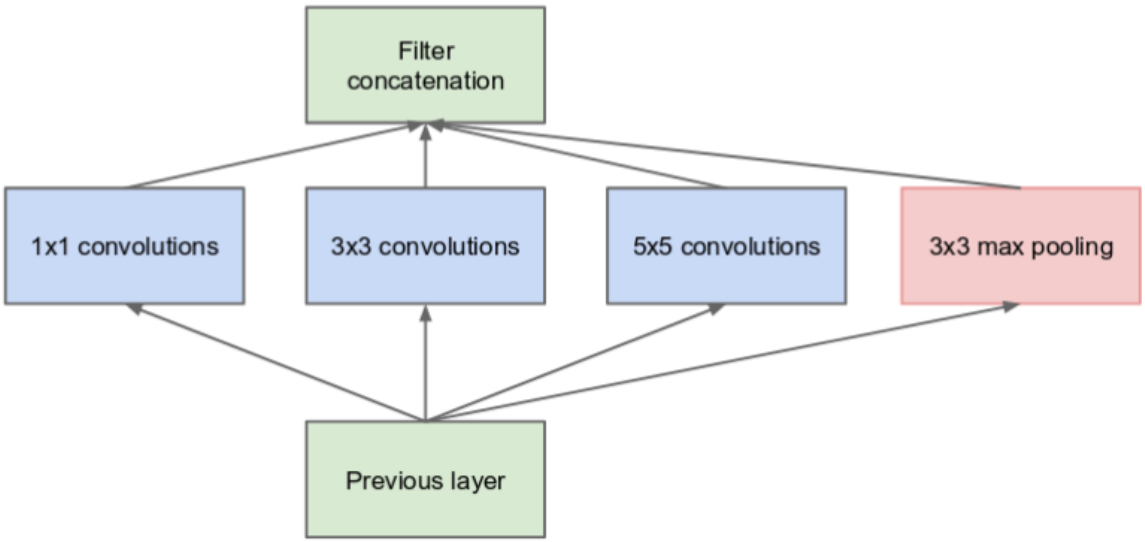


Figure: Naive inception module.

- Problem: 3x3 and 5x5 convolutions are expensive operations, especially when executed on very deep input blocks such as many feature maps from the previous layer.

TYPES: INCEPTION MODULES

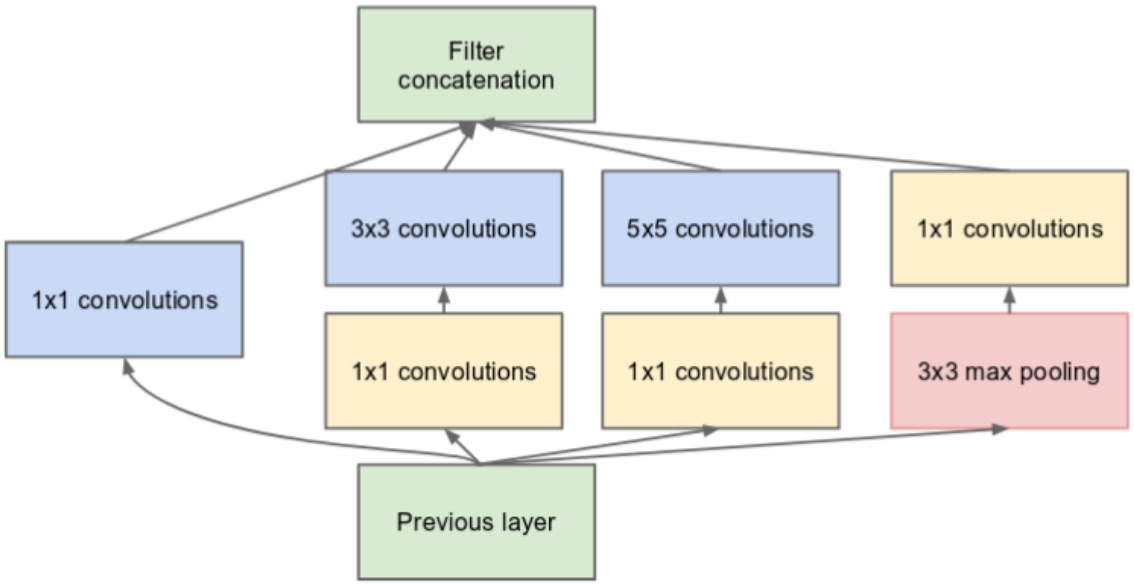


Figure: Dimensionality reduced inception module.

- Solution: apply 1x1 convolutions beforehand to reduce the depth of the previous feature map.

TYPES: INCEPTION MODULES

- Let's understand this with a little numerical example.
- Output dimensions of the previous layer: [28, 28, 192].
- Output dimensions of the 5x5 convolution from the inception module: [28, 28, 32].
- The 5x5 convolution has stride 1 and same padding.
- To improve speed, we first convolve the [28, 28, 192] input with 16 1x1 kernels which results in a [28, 28, 16] block. We then apply the 32 5x5 kernel convolution on this "thinner" block.
- Required operations:
  - Naive: $5^2 \cdot 28^2 \cdot 192 \cdot 32 = 120.422.400$
  - Improved version with 1x1 convolution and depth 16: $1^2 \cdot 28^2 \cdot 192 \cdot 16 + 5^2 \cdot 28^2 \cdot 16 \cdot 32 = 12.443.648$

SEPARABLE CONVOLUTIONS

- Problem setting: make convolution computationally more efficient.
- Remember the sobel kernel from the previous lecture:

$$K_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

- This 3x3 dimensional kernel can be replaced by the outer product of two 3x1 and 1x3 dimensional kernels:

$$\begin{bmatrix} +1 \\ +2 \\ +1 \end{bmatrix} * \begin{bmatrix} +1 & 0 & -1 \end{bmatrix}$$

- Convolving with both filters subsequently has a similar effect, reduces the amount of parameters to be stored and thus improves speed.

SEPARABLE CONVOLUTIONS

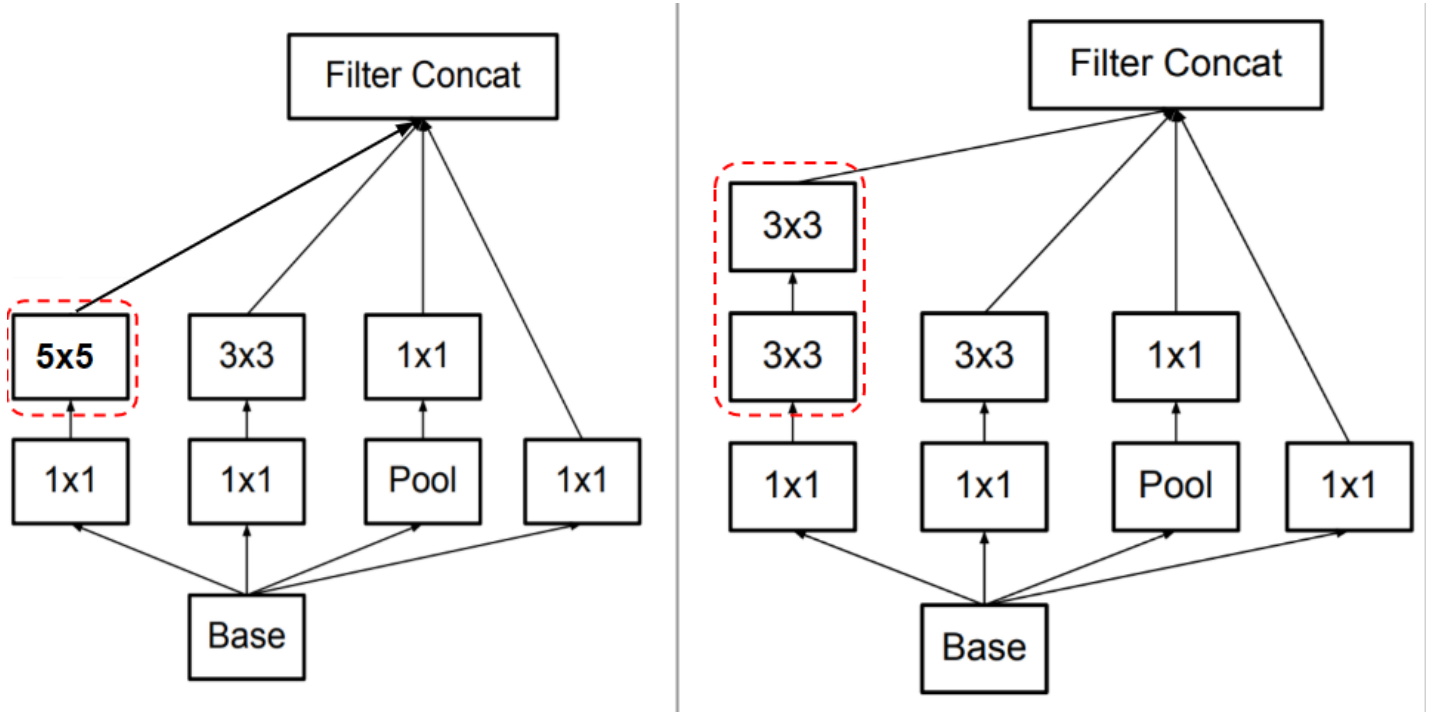


Figure: *Left* : Regular Inception module . *Right* : Inception module where each 5x5 convolution is replaced by two 3x3 convolutions.

Separable convolutions (also: factorized) find application in the inception net V4 [Szegedy Christian et. al , 2015]. The authors find that it increases computational speed but suggest to use this trick in medium to late stage layers only.

SEPARABLE CONVOLUTIONS

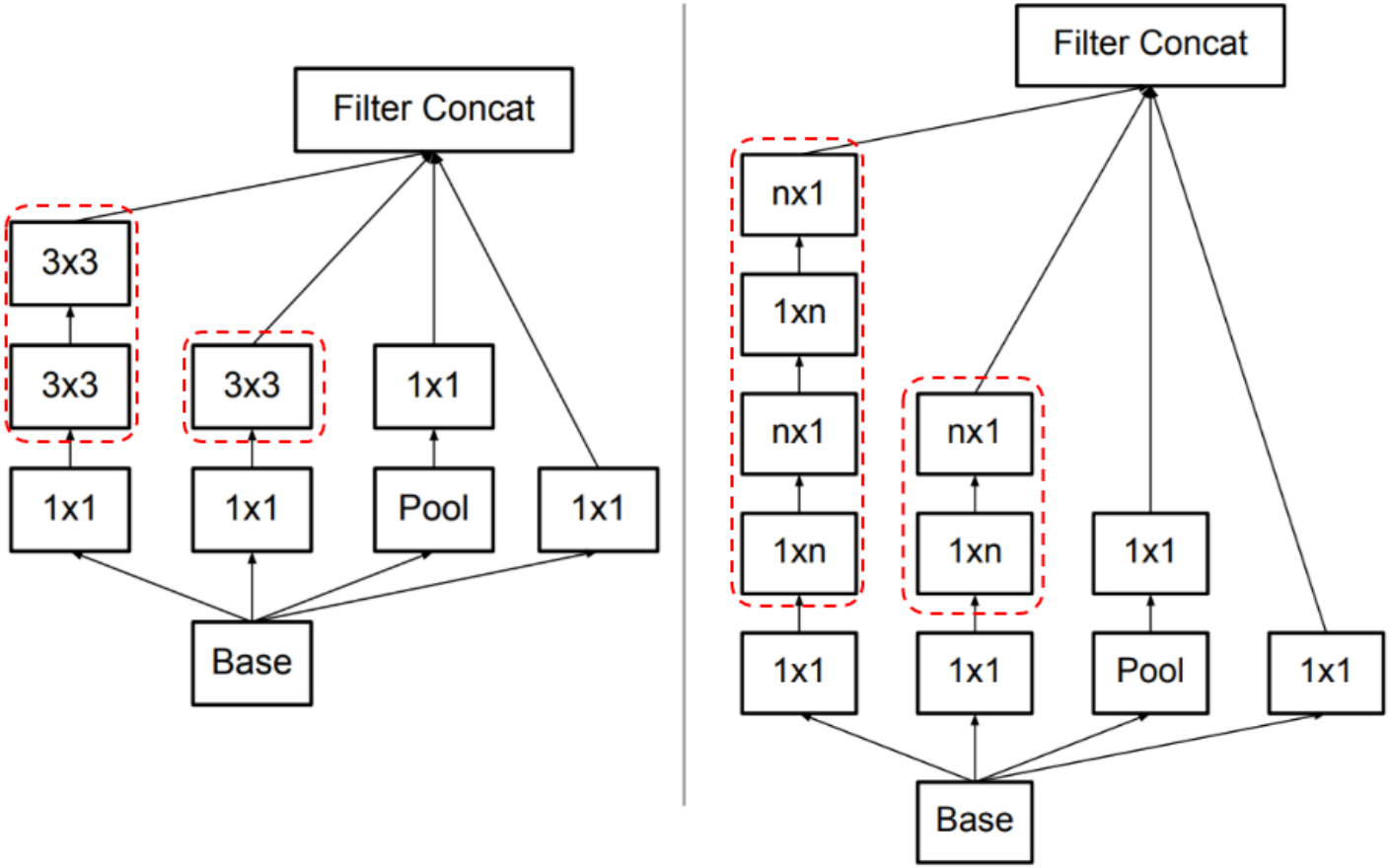


Figure: *Left* : Inception module from the previous slide. *Right* : Inception module after the factorization of the *nxn* convolutions (n = 3, here).

TYPES: SKIP CONNECTIONS

- Problem setting: theoretically, we could build infinitely deep architectures as the net should learn to pick the beneficial layers and skip those that do not improve the performance automatically.
- But: this skipping would imply learning an identity mapping $x = \mathcal{F}(x)$. It is very hard for a neural net to learn such a 1:1 mapping through the many non-linear activations in the architecture.
- Solution: offer the model explicitly the opportunity to skip certain layers if they are not useful.
- Introduced in [He Kaiming et. al , 2015] and motivated by the observation that stacking evermore layers increases the test- as well as the train-error ($\neq$ overfitting).

TYPES: SKIP CONNECTIONS

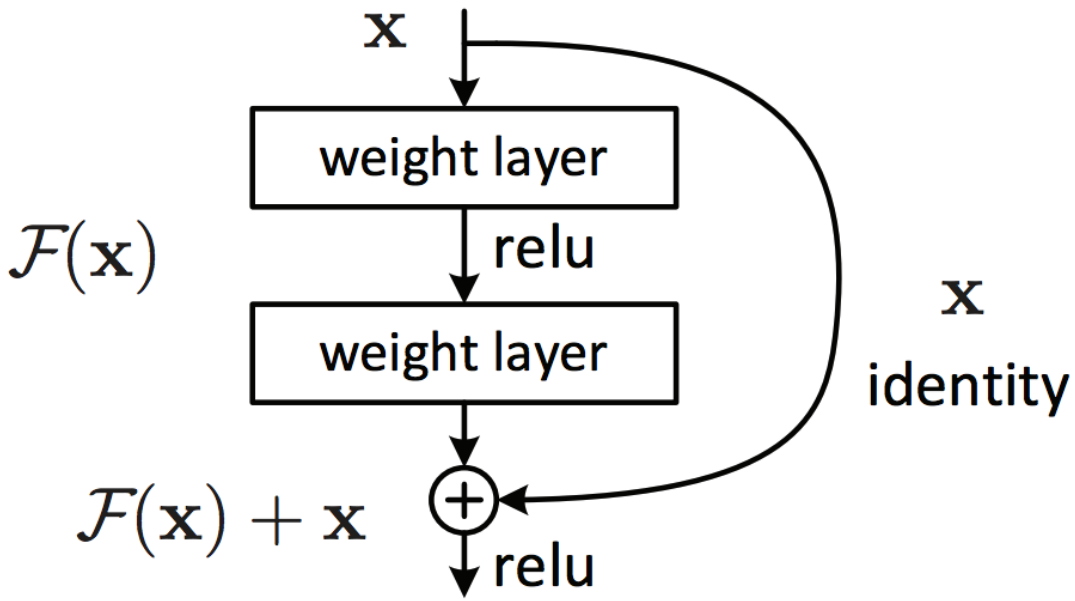


Figure: Skip connection/ residual learning module. The information flows through two layers and the identity function. Both streams of information are then element-wise summed and jointly activated.

TYPES: SKIP CONNECTIONS

- Let $\mathcal{H}(x)$ be the optimal underlying mapping that should be learned by (parts of) the net.
- x is the input in layer l (can be raw data input or the output of a previous layer).
- $\mathcal{H}(x)$ is the output from layer l.
- Instead of fitting $\mathcal{H}(x)$, the net is ought to learn the residual mapping $\mathcal{F}(x) := \mathcal{H}(x) - x$ whilst x is added via the identity mapping.
- Thus, $\mathcal{H}(x) = \mathcal{F}(x) + x$, as formulated on the previous slide.
- The model should only learn the *residual mapping* $\mathcal{F}(x)$
- Thus, the procedure is also referred to as *Residual Learning*..

TYPES: SKIP CONNECTIONS

- The element-wise addition of the learned residuals $\mathcal{F}(x)$ and the identity-mapped data x requires both to have the same dimensions.
- To allow for downsampling within $\mathcal{F}(x)$ (via pooling or valid-padded convolutions), the authors introduce a linear projection layer W_s .
- W_s ensures that x is brought to the same dimensionality as $\mathcal{F}(x)$ such that:

$$y = \mathcal{F}(x) + W_s x$$

- where y is the output of the skip module and W_s represents the weight matrix of the linear projection (# rows of W_s = dimensionality of $\mathcal{F}(x)$).
- This idea applies to fully connected layers as well as to convolutional layers.

TYPES: GLOBAL AVERAGE POOLING

- Problem setting: tackle overfitting in the final fully connected layer.
  - Classic pooling removes spatial information and is mainly used for dimension and parameter reduction.
  - The elements of the final feature maps are connected to the output layer via a dense layer. This could require a huge number of weights increasing the danger of overfitting.
  - Example: 256 feature maps of dim 100x100 connected to 10 output neurons lead to 25.6e6 weights for the final dense layer.
- Solution:
  - Average each final feature map to the element of one global average pooling (GAP) vector.
  - Do not use pooling throughout the net.
  - Example: 256 feature maps are now reduced to GAP-vector of length 256 yielding a final dense layer with 2560 weights.

TYPES: GLOBAL AVERAGE POOLING

- GAP preserves whole information from the single feature maps whilst decreasing the dimension.
- Mitigates the possibly *destructive* effect of pooling.
- Each element of the GAP output represents the activation of a certain feature on the input data.
- Acts as an additional regularizer on the final fully connected layer.
- Allows for interpretation of the model via Class Activation Maps (more on this later).

TYPES: GLOBAL AVERAGE POOLING

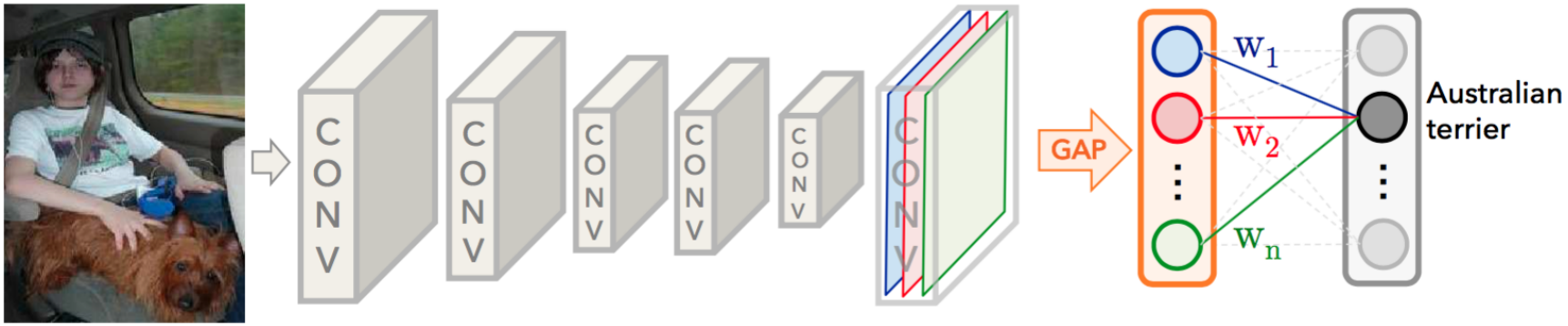


Figure: Illustration of GAP as in [B. Zhou et. al , 2016]. Each feature map representing one feature category averaged into one final vector. No pooling operations are applied throughout the net. The dimensionality of the input reduces solely due to the convolution operations.

CLASS ACTIVATION MAPPING

- We want to understand the decision-making of a net, e.g. **why does it classify image X as a cat?**
- Simplest method based on GAP was introduced in [Zhou Bolei et. al, 2016].
- Idea:
  - the final GAP vector stores the activation of each feature map category that was learnt throughout the net.
  - the dense layer that connects the output classes with the GAP vector stores information about how much each feature contributes to each class.
  - exploit this information to show which parts of the input image would be activated for each class.

CLASS ACTIVATION MAPPING

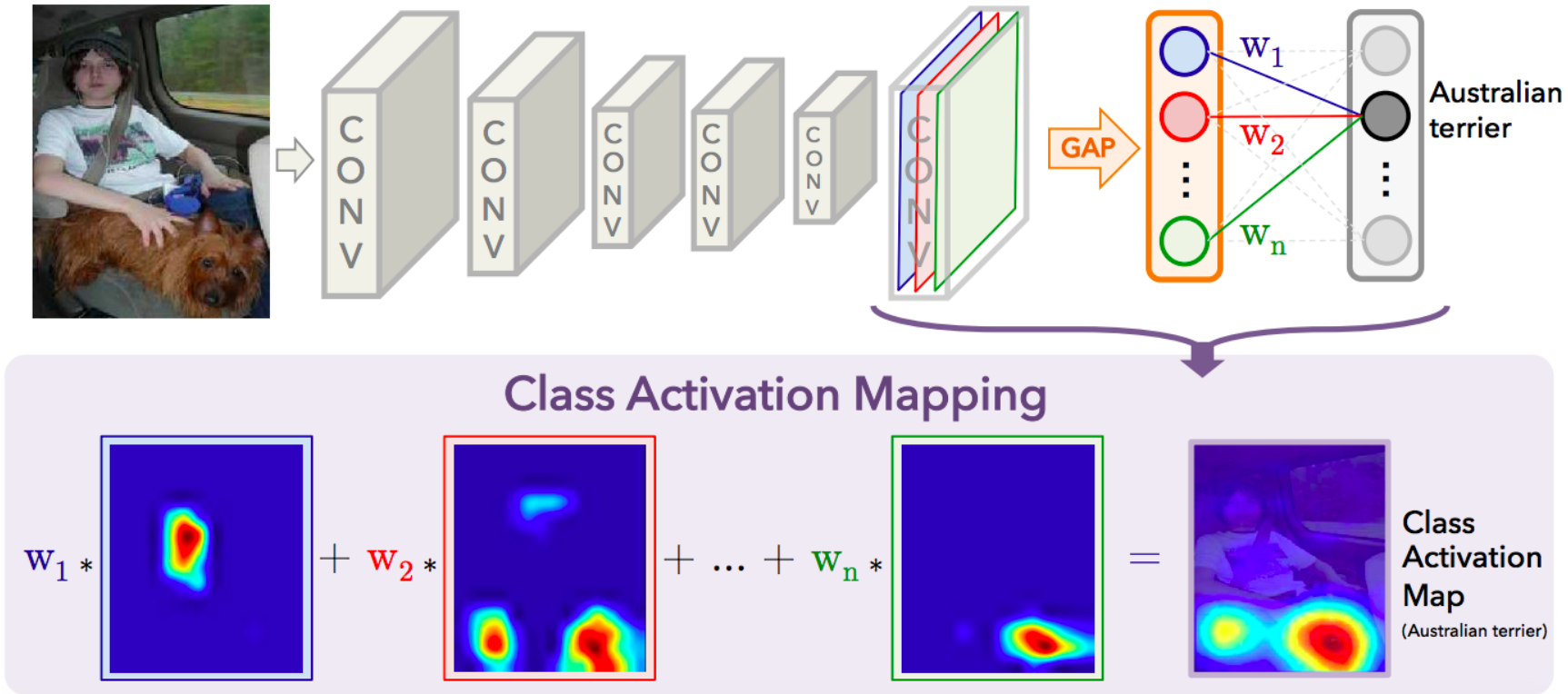


Figure: Illustration of the class activation mapping. The activated regions from the feature maps are summed up weighted by their connection strength with the final output classes and upsampled back to the dimension of the input image. No max-pooling is applied throughout the architecture, the downsampling is due to the CNN layers.

CLASS ACTIVATION MAPPING

1. Train a net with GAP pooling end-to-end.
2. Run a forward-pass with the image i you would like to understand.
3. Take the final l feature maps $f_1, \ldots, f_l$ for this input.
4. Get the *feature weights* $w_{j1}, \ldots, w_{jl}$ that connect the GAP layer with the final class output j that you would like to interpret (e.g. terrier).
5. Create the CAM for class j on input image i:

$$\text{CAM}_{j,i} = \sum_{k=1}^{l} w_{jk} * f_k$$

6. Normalize the values such that $\text{CAM}_{j,i} \in [0, 1]$.
7. In case of valid convolutions, the resulting CAM will be smaller than the input image. Linear upsampling is then used to map it back to the input dimension.
8. Overlay the input image with the CAM and interpret the activation.

APPLICATION - ROAD SEGMENTATION

- Problem setting: train a neural net to pixelwise segment roads in satellite imagery.
- Answer the questions **where is the road map?**

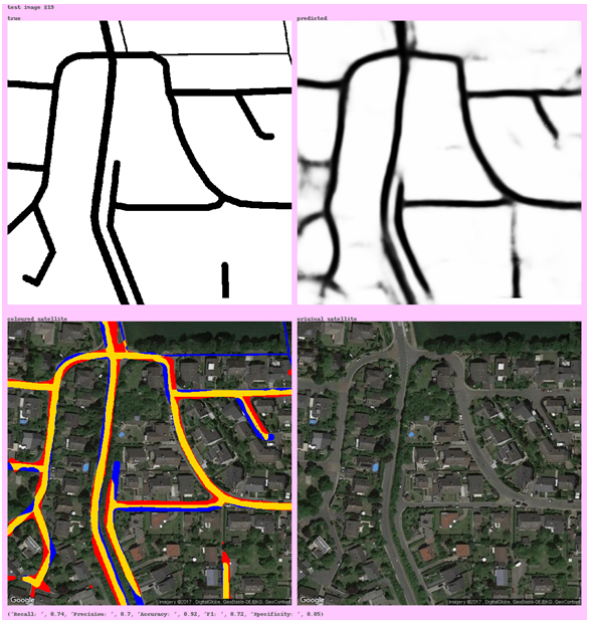
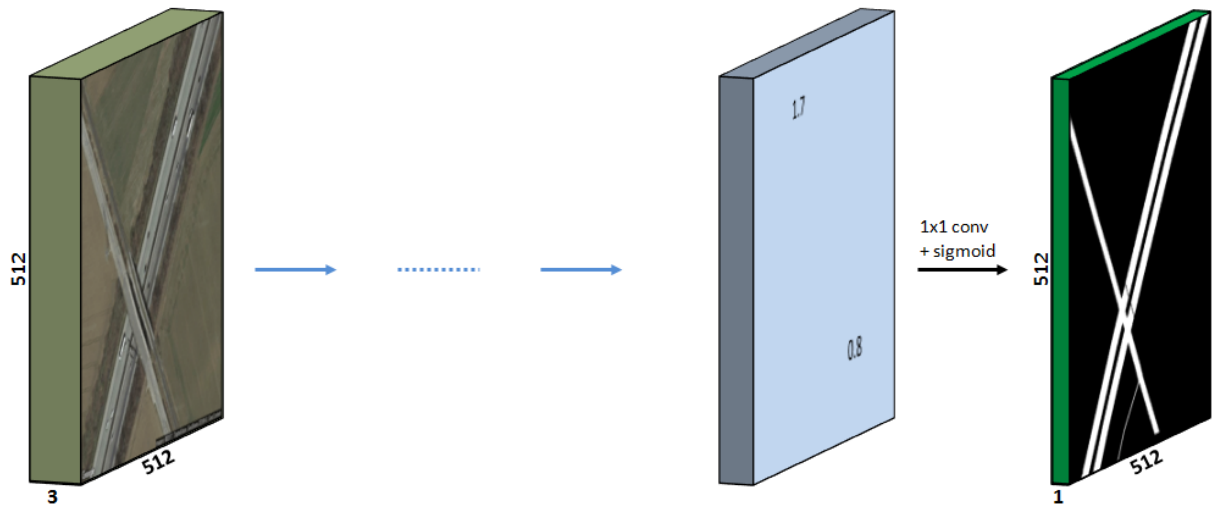


Figure: Model prediction on a test satellite image. Yellow are correctly identified pixels, blue false negatives and red false positives.

APPLICATION - ROAD SEGMENTATION

- The net takes an RGB image [512, 512, 3] and outputs a binary (road / no road) probability mask [512, 512, 1] for each pixel.
- The model is trained via a binary cross entropy loss which was combined over each pixel.



$$Output_{i,j} = \frac{1}{1+exp(-1.7 \cdot \mathbf{w})} \in (0,1)$$

Figure: Scheme for the input/ output of the net architecture.

APPLICATION - ROAD SEGMENTATION

- The architecture is inspired by the U-Net [Olaf Ronneberger et al., 2015], a fully convolutional net that makes use of upsampling via transposed convolutions as well as skip connections.
- Input images are getting convolved and down-sampled in the first half of the architecture.
- Then, they are getting upsampled and convolved again in the second half to get back to the input dimension .
- Skip connections throughout the net ensure that high-level features from the start can be combined with low-level features throughout the architecture.
- Only convolutional and no dense layers are used.

APPLICATION - ROAD SEGMENTATION

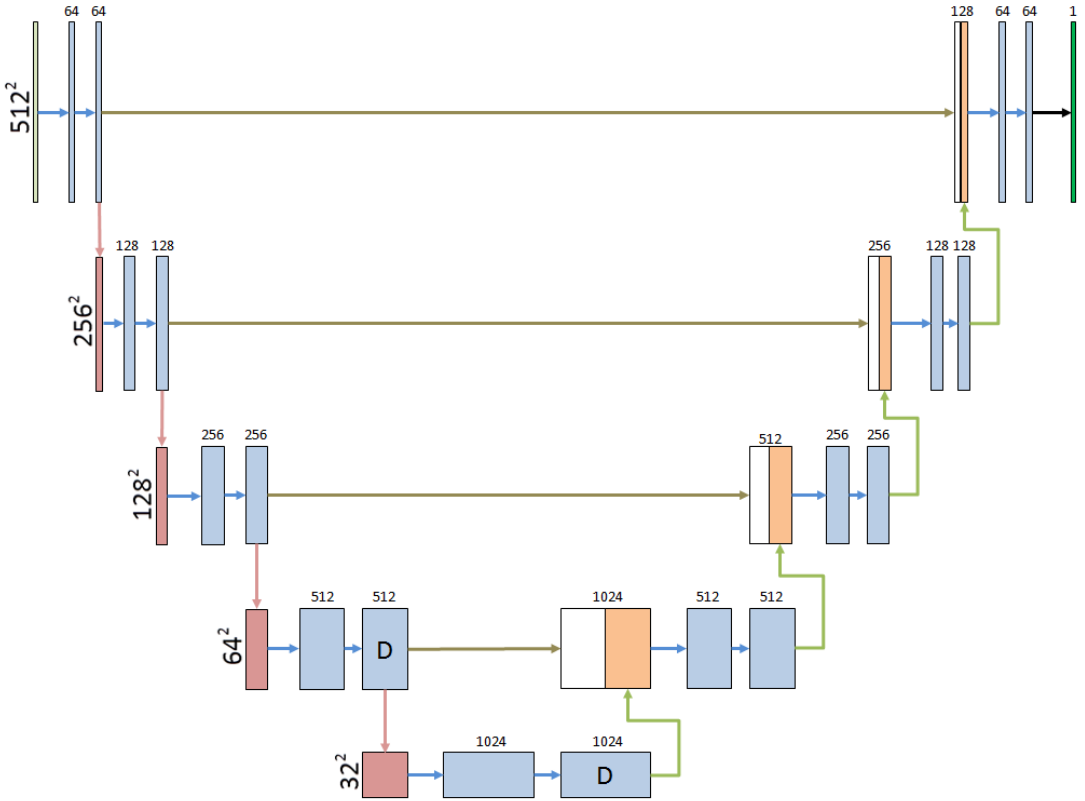


Figure: Illustration of the architecture. Blue arrows are convolutions, red arrows max-pooling operations, green arrows upsampling steps and the brown arrows merge layers with skip connections. The height and width of the feature blocks are shown on the vertical and the depth on the horizontal. D are dropout layers.

FAMOUS ARCHITECTURES - LENET

- Pioneering work on CNNs by [LeCunn Yann et. al, 1998].
- Applied on the famous MNIST data set for automated handwritten digit recognition.
- Consists of convolutional, max-pooling and dense layers.
- Complexity and depth of the net was mainly restricted by limited computational power back in the days.

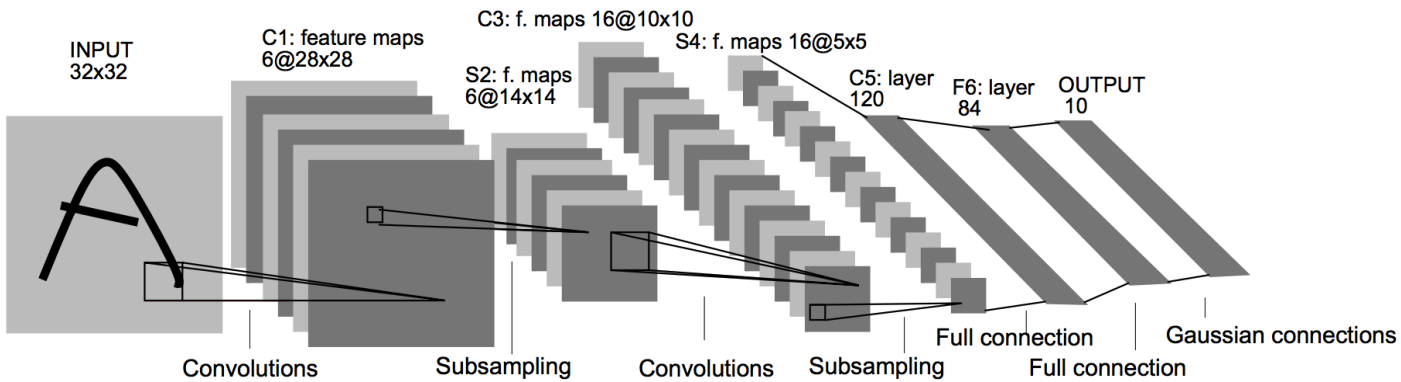


Figure: LeNet architecture: two CNN layers followed by max-pooling, a hidden dense and a softmax layer.

FAMOUS ARCHITECTURES - ALEXNET

- Introduced by [Alex Krizhevsky et al., 2012], won the ImageNet challenge in 2012 and is basically a deeper version of the LeNet.
- Trained in parallel on two GPUs, using two streams of convolutions which are partly interconnected.
- Contains ReLU activations and makes use of data set augmentation strategies.

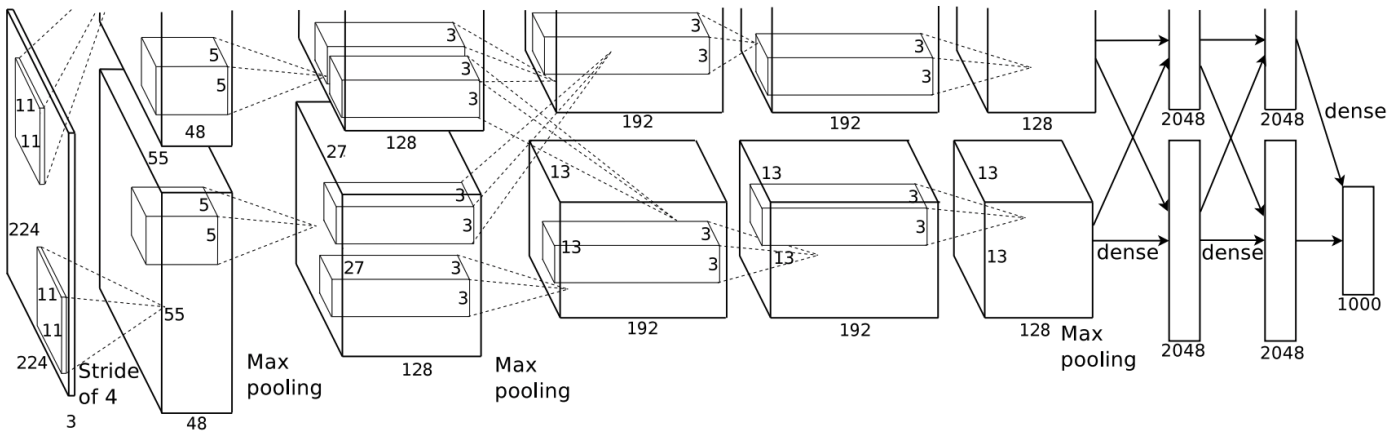


Figure: AlexNet architecture by [Krizhevsky Alex et. al, 2012].

FAMOUS ARCHITECTURES - VGG

- Architecture introduced by [Simoyan Karen et. al, 2014] as “Very Deep Convolutional Network”.
- A deeper variant of the AlexNet.
- Mainly uses many convolutional layers with a small kernel size 3x3.
- Performed very well in the ImageNet Challenge 2014.
- Exists in a small version (VGG16) with a total of 16 layers and a larger version (VGG19) with 19 layers.

FAMOUS ARCHITECTURES - VGG

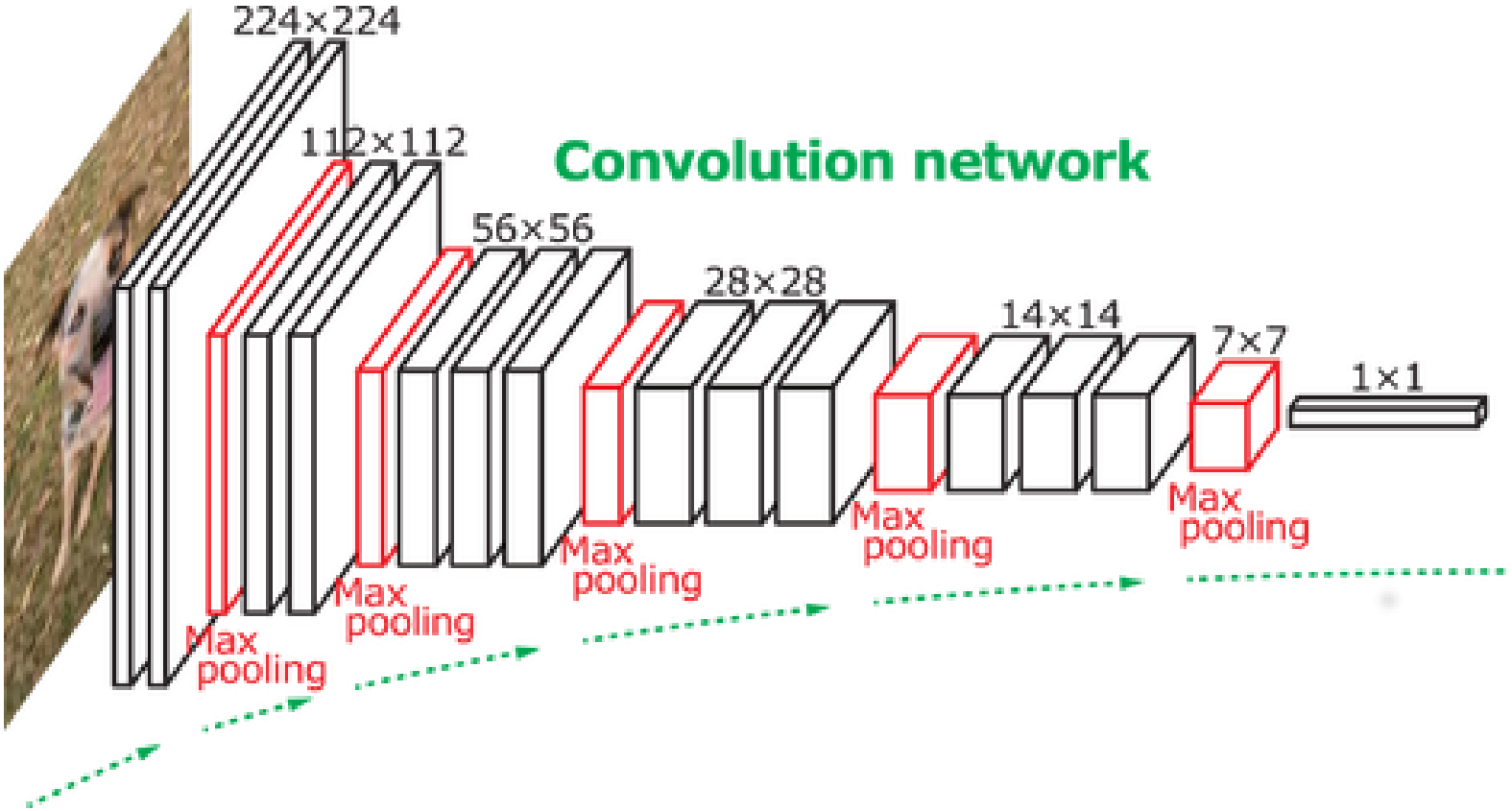


Figure: VGG Net 16 with 10 CNN layers.

FAMOUS ARCHITECTURES - VGG

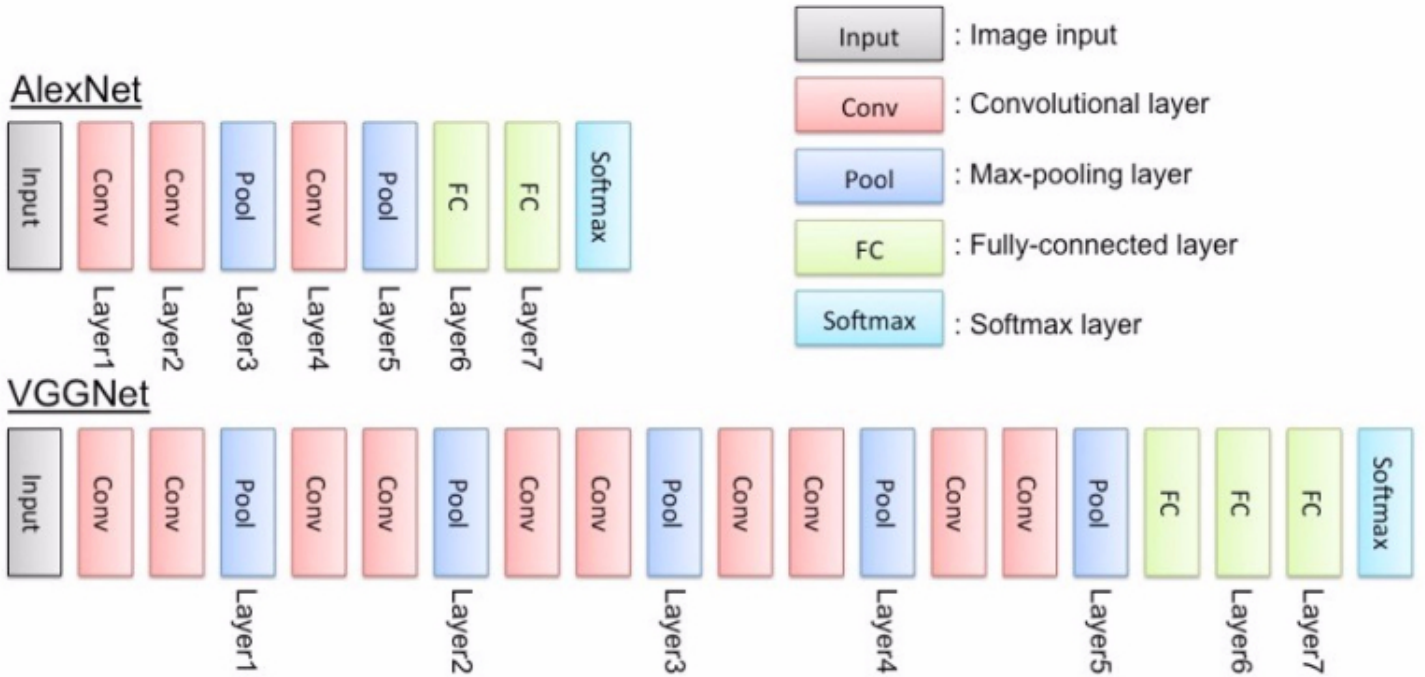


Figure: VGG Net 16 with 10 CNN layers in comparison with AlexNet.

FAMOUS ARCHITECTURES - GOOGLENET

- The net architecture that first made use of inception modules [Christian Szegedy et. al , 2014].
- Also referred to as Inception Net.
- Also includes Batch Normalization to improve training.
- Uses auxiliary losses: branches throughout the net that consist of softmax layers that make predictions using early stage layers.
- Those losses are jointly optimized and the output from the final head is used for deployment and testing of the architecture.
- Inception modules allow the net to “choose” between different kernels.

FAMOUS ARCHITECTURES - GOOGLENET

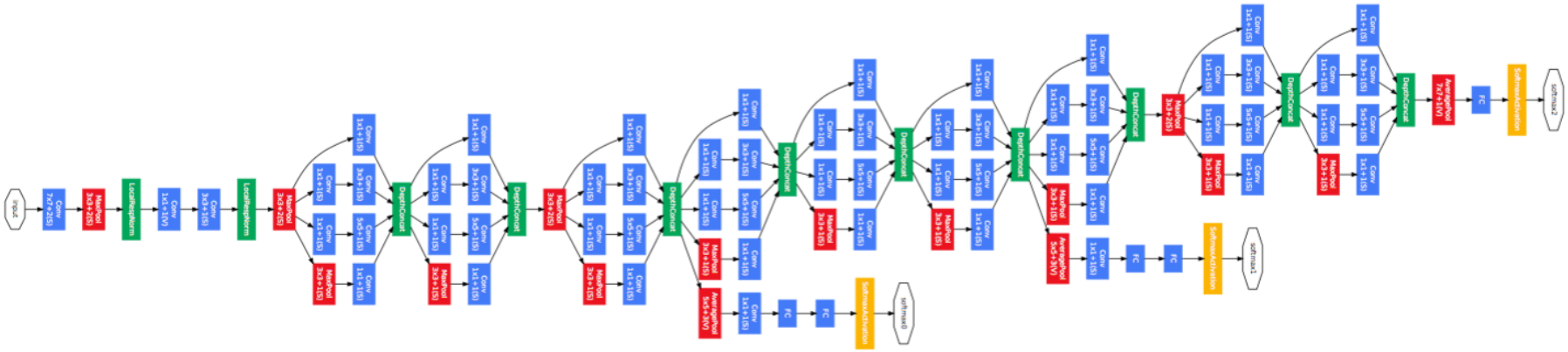


Figure: GoogleNet architecture. Yellow rectangles mark the auxiliary losses which are combined during training.

FAMOUS ARCHITECTURES - GOOGLENET

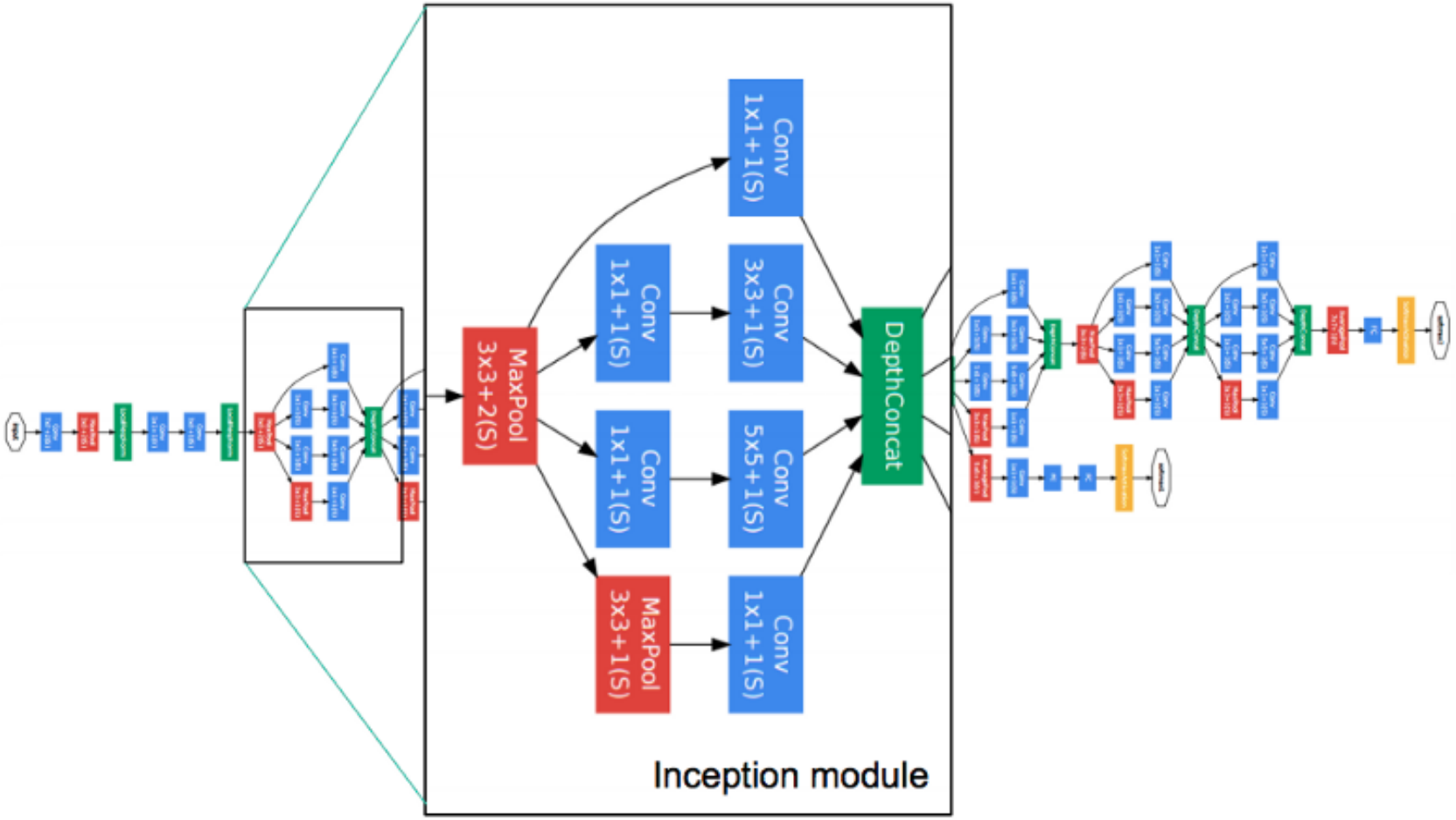


Figure: Zoom view in one of the inception modules upon which the GoogLeNet architecture is build.

FAMOUS ARCHITECTURES - RESNET

- Net architecture that makes use of skip connections / residual learning.
- This allowed [Christian Szegedy et al., 2014] to create a very deep net architectures of up to 152 layers.
- Batch normalization and global average pooling is used.

FAMOUS ARCHITECTURES - RESNET

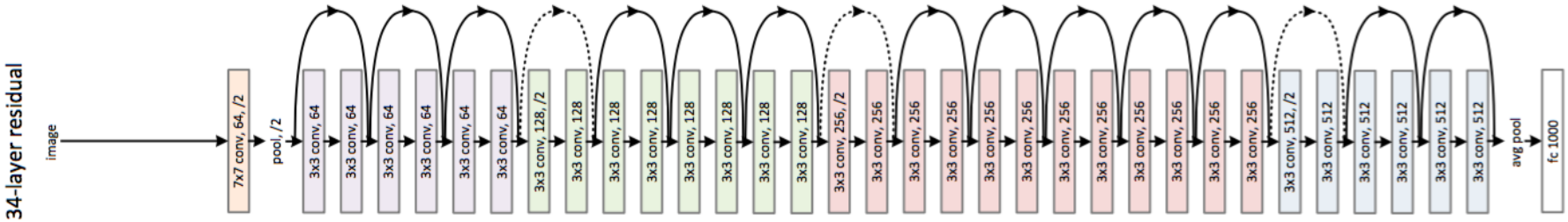


Figure: A deep ResNet architecture with a total of 34 layers.

FAMOUS ARCHITECTURES - RESNET

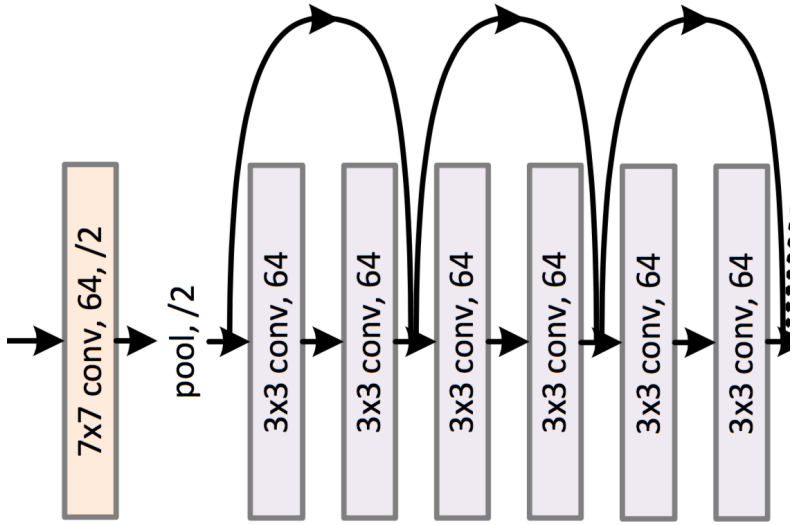


Figure: Zoom view in the ResNet architecture. The arrows mark residual connections which allow the net to “skip” layers which do not contribute to an improvement in the prediction performance.

FAMOUS ARCHITECTURES - SUMMARY

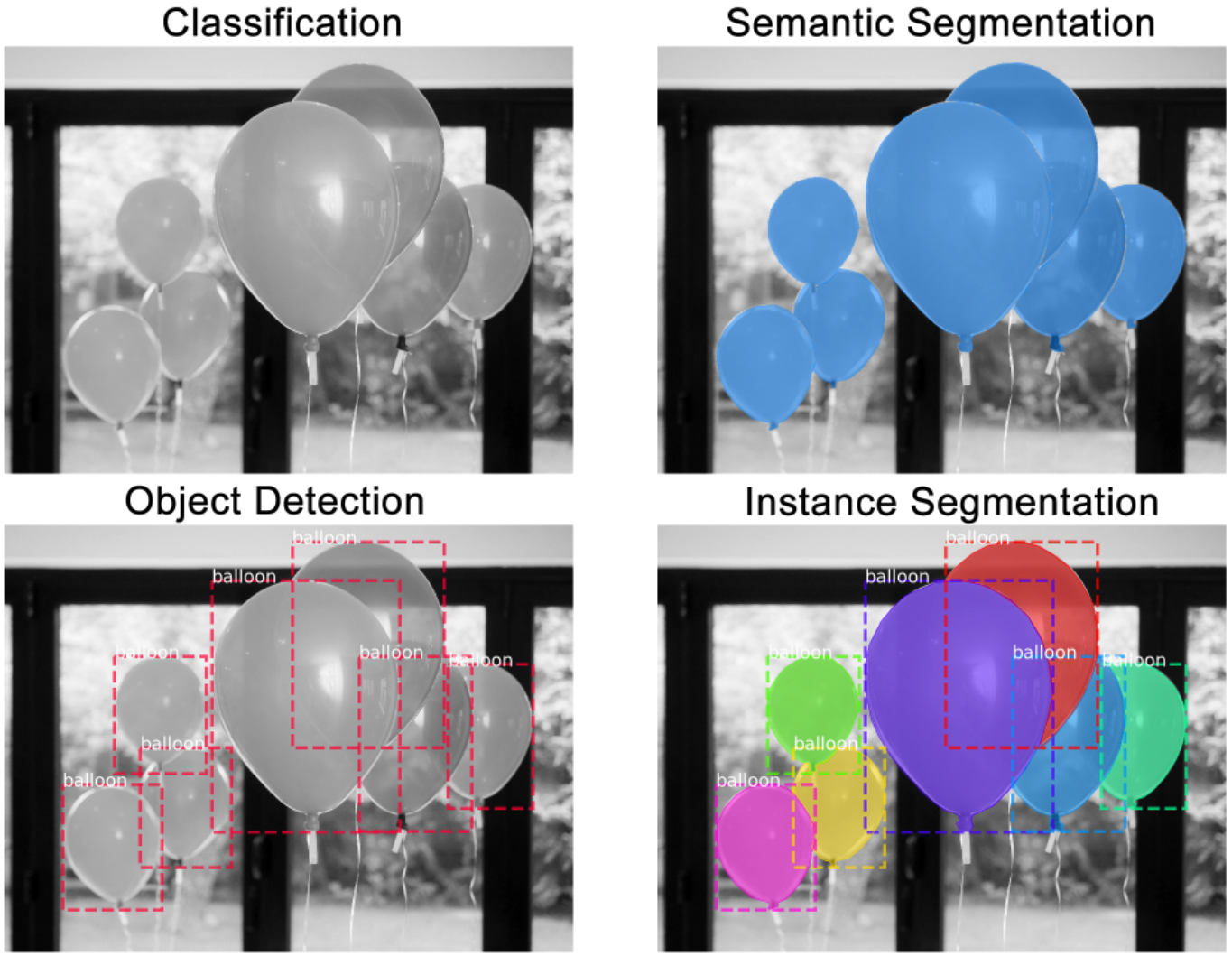
- Main points from the previous architectures:
  - ResNet makes heavy use of skip connections.
  - GoogLeNet uses inception modules.
  - LeNet was one of the first successful applications of CNNs.
  - AlexNet is a deeper version of LeNet.
  - VGG is a deeper version of AlexNet.
  - Batch normalization is often used in modern architectures.
  - Global average pooling is also a prominent module in modern architectures.
- There exists a great variety of different architectures which perform very well on different tasks.
- Almost all of the above described architectures exist in extended versions such as the Inception V1 - V4, deeper and shallower ResNets, ResNets for time series ... it is up to you to discover them all.

Case Study - Mask R-CNN

COMMON MACHINE VISION TASKS

- Image Classification : Assign a *single* label to the whole image.
- Object localization / detection : Draw **bounding boxes** around (and classify) one or more objects in the image.
- Semantic Segmentation : The goal here is to draw **outlines** between classes *without* differentiating between different instances of a given class.
- Instance segmentation : This is a hybrid of the previous two tasks. The goal is to classify *each* object in the image and draw a *separate* outline around it.
- Note : This is not an exhaustive list. There are *many* others.

COMMON MACHINE VISION TASKS



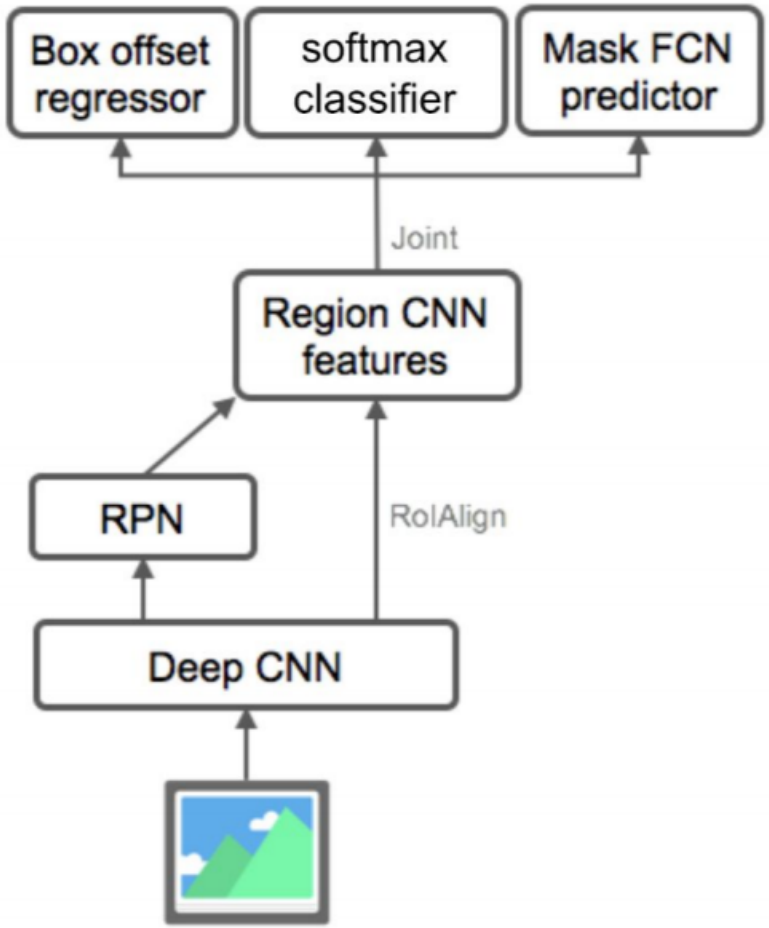
credit : Waleed Abulla

Instance segmentation is the most challenging task!

MASK R-CNN - OVERVIEW

- Mask R-CNN (He et al. 2017) belongs to an important class of CNNs known as Region-based CNNs (or R-CNNs).
- It extends the Faster R-CNN (Ren et al. 2015) architecture (which performs classification and detection) by adding a branch for segmentation (more on this later).
- Therefore, Mask R-CNN performs classification, detection *and* instance segmentation in parallel!
- It was developed at Facebook in 2017 and trained using the COCO (Common Objects in Context) dataset.
- Note : Each training example has ground truth class labels, bounding boxes and segmentation masks.
- This case-study is only meant to be a 'quick tour' of Mask R-CNN. If any of the details are unclear, please read the paper.

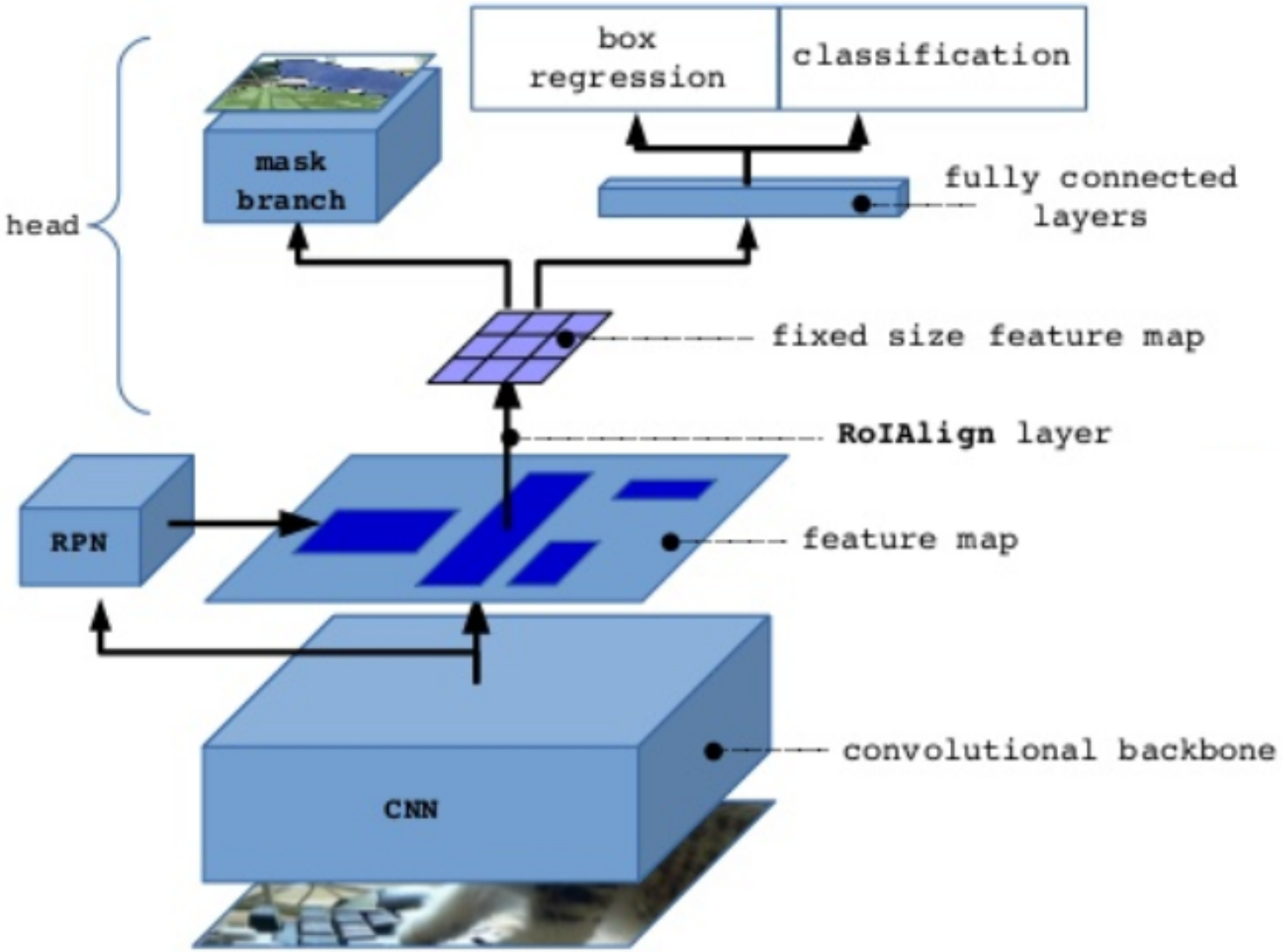
MASK R-CNN ARCHITECTURE



credit: UCSD

Figure: Mask R-CNN architecture.

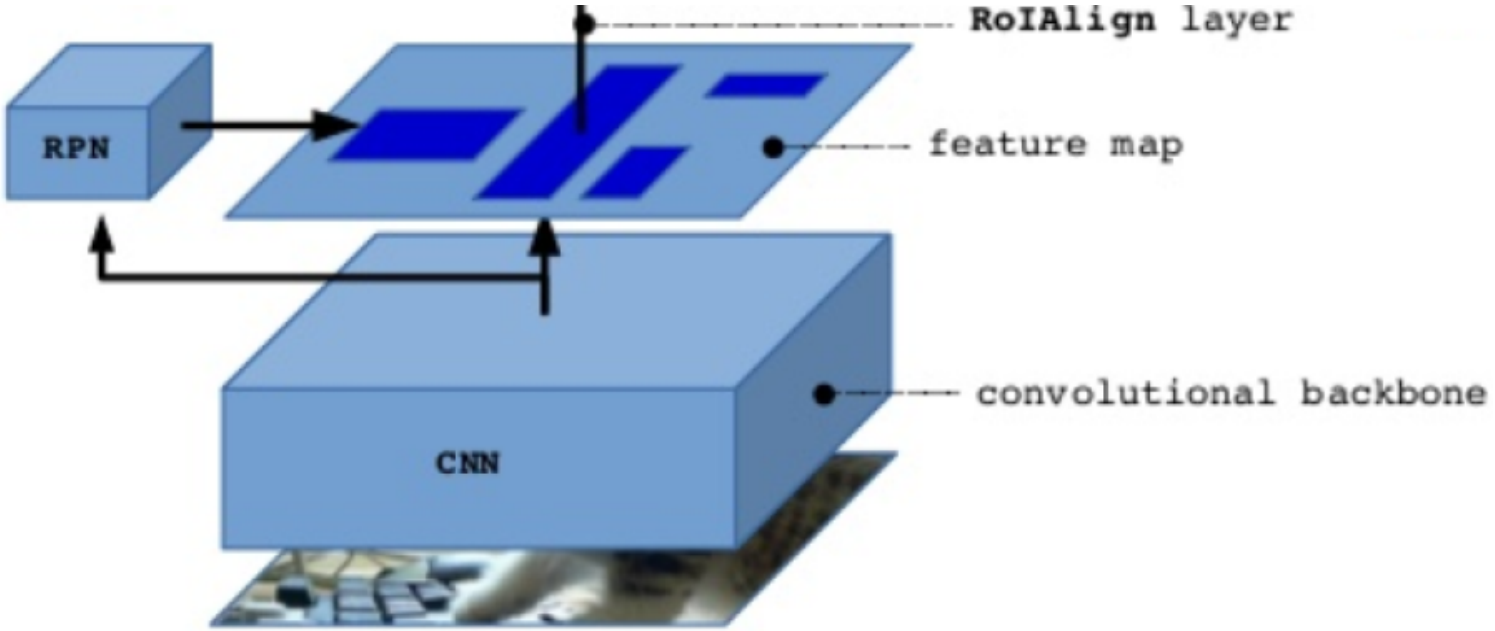
MASK R-CNN ARCHITECTURE



credit: Georgia Gkioxari

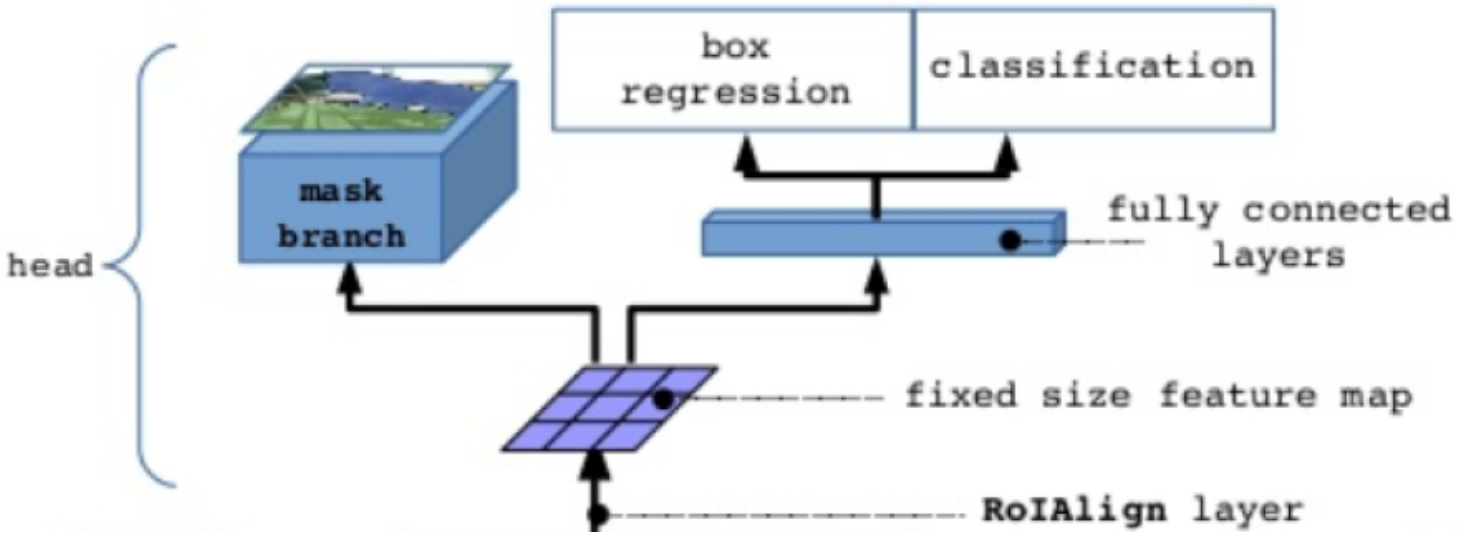
Figure: Mask R-CNN architecture.

MASK R-CNN ARCHITECTURE - STAGE ONE



- *Stage One* : Consists of a "backbone" network and a Region Proposal Network (RPN).
- The backbone network extracts the feature maps from an image and the RPN identifies promising regions, or Regions of Interest (ROIs) that might contain interesting objects.
- The ROIAlign layer then resizes the ROIs so that they all have the same spatial dimensions.

MASK R-CNN ARCHITECTURE - STAGE TWO



- The resized ROIs are then fed to the *Stage Two* network (pictured above).
- This has three branches : one each for classification, detection/localization and segmentation.

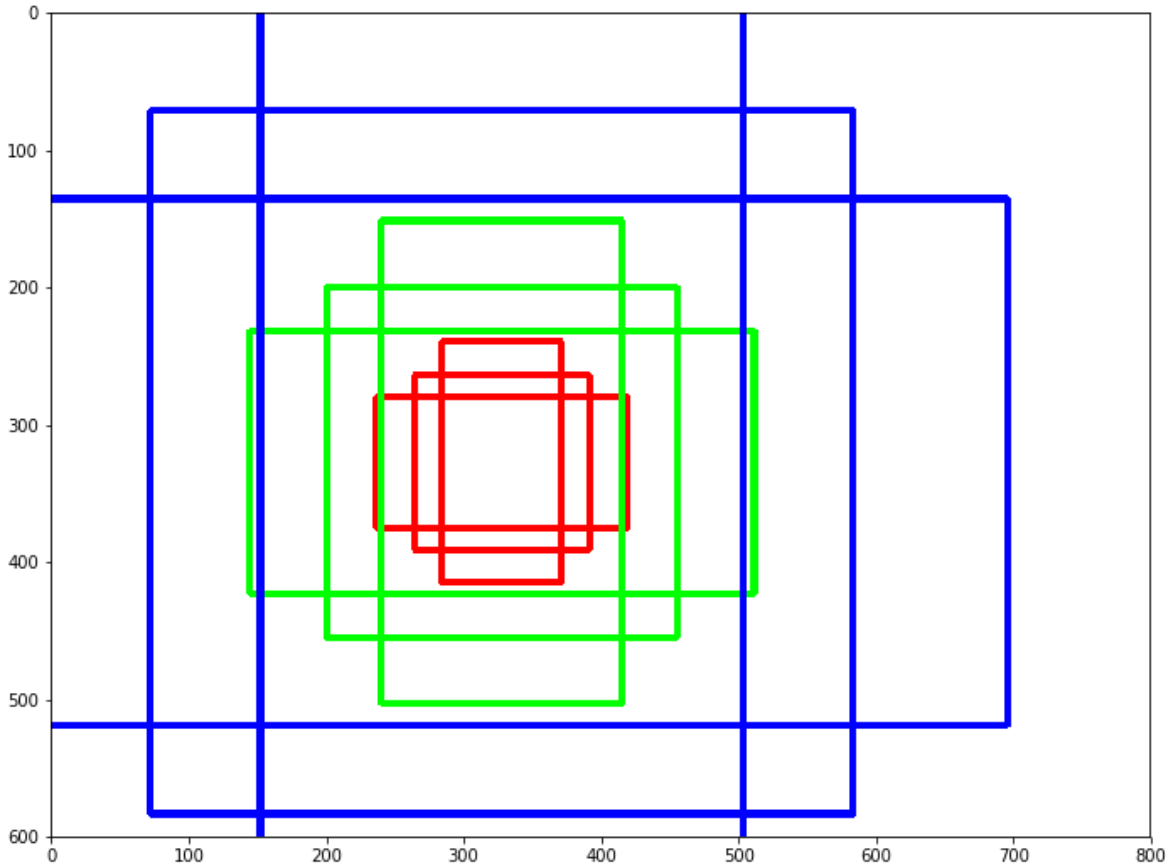
STAGE ONE - BACKBONE

- The backbone architecture can be based on any generic CNN.
- In the paper, the authors implement Mask R-CNN using ResNet.
- ResNets are divided into 5 "stages" where each stage consists of several convolutional layers.
- The authors use the first 4 stages of ResNet-50 or ResNet-101 as the backbone network.
- The output of the 4th stage will now serve as the input to the Region Proposal Network.

STAGE ONE - REGION PROPOSAL NETWORK

- Lightweight network that works directly on the final feature map generated by the backbone network.
- Consists of a 3×3 convolutional layer and 1×1 convolutional layers.
- The RPN looks at small $n \times n$ regions of the input feature map (n = 3, in our case).
- At **each** such location, the RPN simultaneously proposes multiple regions of interest (ROIs) that might contain objects.
- In order to accomplish this, each such $n \times n$ region is associated with k **anchor boxes**.
- The number of anchor boxes and their locations, sizes and aspect ratios are all hyperparameters.

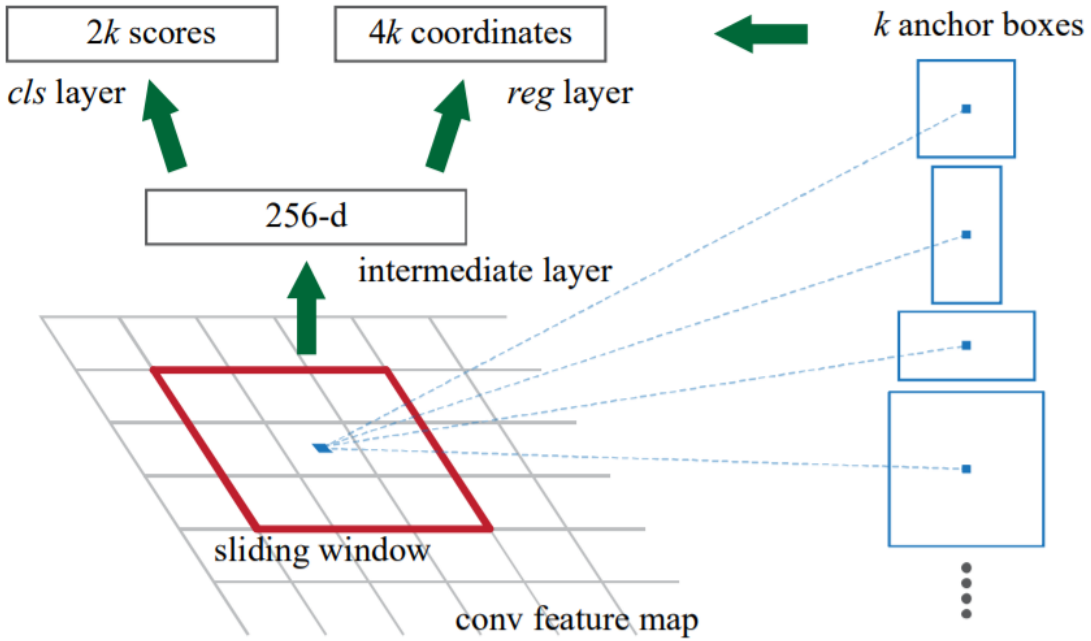
ANCHOR BOXES



credit:Hao Gao

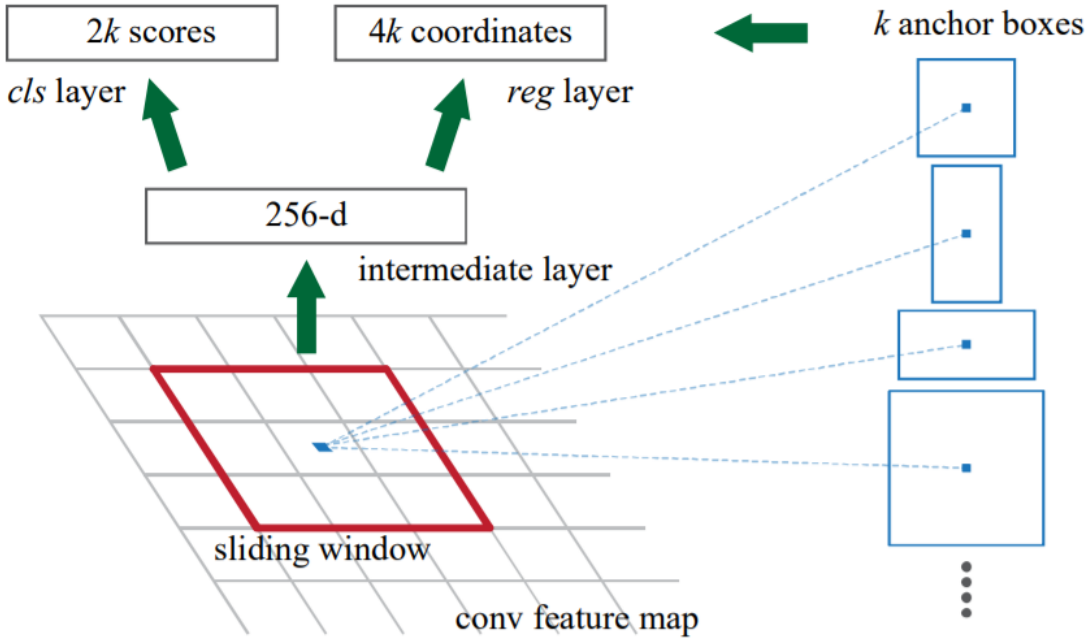
Figure: An example set of anchor boxes in a 600x800 pixel image. Three colors represent three scales or sizes: 128x128, 256x256, 512x512. For a given color, the three boxes correspond to height:width ratios of 1:1, 1:2 and 2:1.

STAGE ONE - REGION PROPOSAL NETWORK



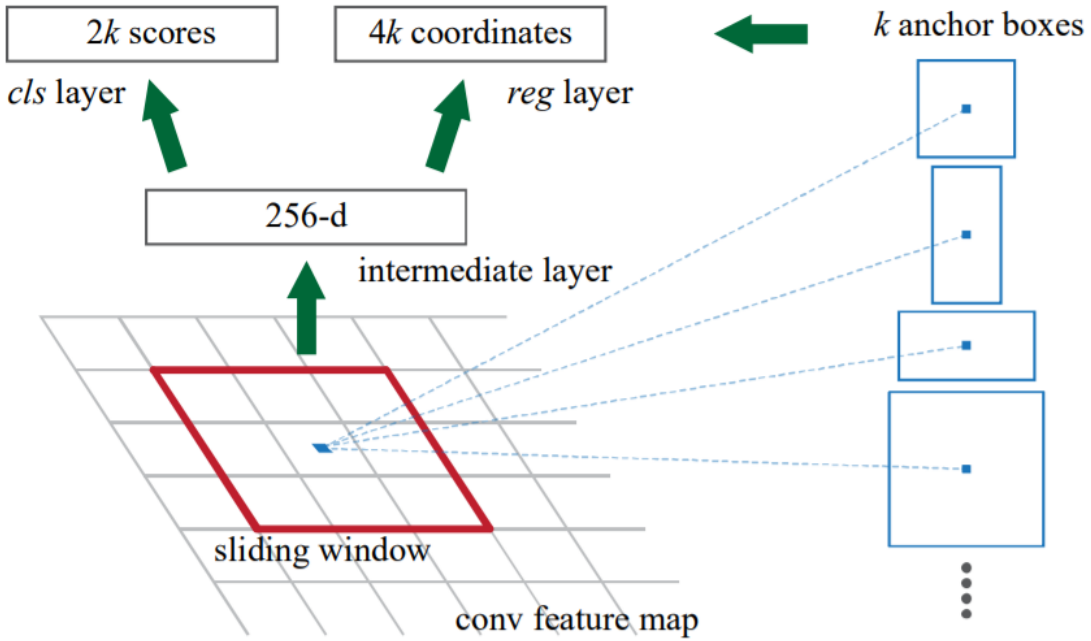
- Every location/'sliding window' in the final feature map is associated with $k = 15$ anchor boxes of varying sizes and aspect ratios.
- For a convolutional feature map of size $W \times H$, there will be $W \times H \times k$ anchor boxes.
- Cross-boundary anchors are ignored during training.

STAGE ONE - REGION PROPOSAL NETWORK



- The classifier ('cls' layer) is a dense layer which outputs $2k$ scores (to which a softmax is applied) indicating the probability of an object being present or *not* present in each of the k anchor boxes.
- Of course, it's also possible to simply output k scores (to which a logistic sigmoid is applied) instead indicating simply whether an object is present.

STAGE ONE - REGION PROPOSAL NETWORK



- The regression ('reg') layer is a dense layer which outputs $4k$ outputs encoding the 4 coordinates of the k region proposals.
- These four values represent the *offsets* to the location of the upper-left corner and the height and width of a (preconfigured, rectangular) anchor box.

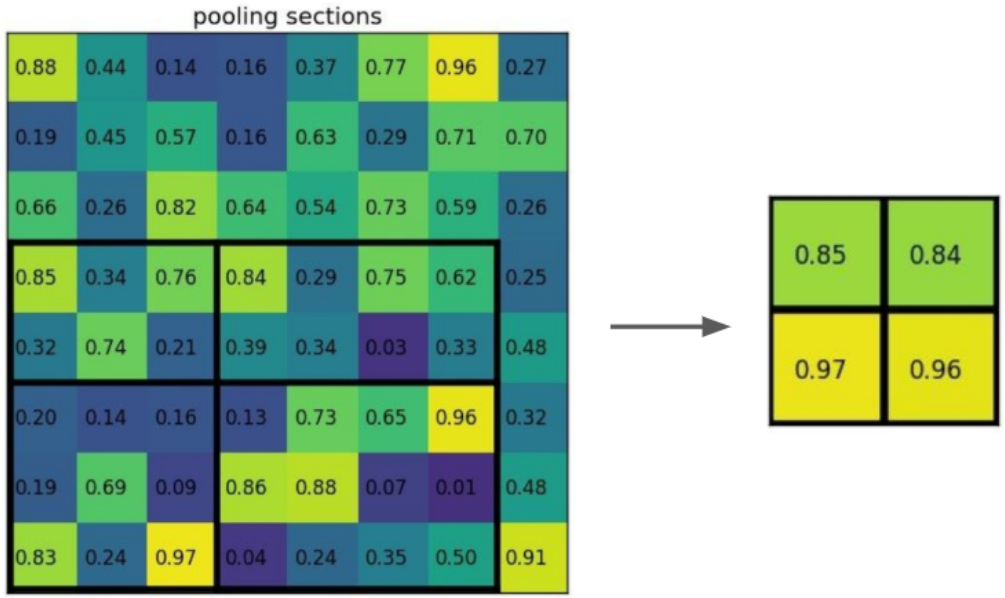
ROI POOLING

- The RPN has its own loss function and can be trained either separately or jointly with the rest of the network.
- Stage One was computing the region proposals. Once we have have them, we enter Stage Two.
- Only the most promising region proposals are chosen for Stage Two using a technique called 'non-max suppression'.
- Please read the paper for details.

ROI POOLING

- The region proposals are in pixel space, not feature space. Therefore, a region of the output feature map (of the backbone network) which roughly corresponds to the region proposal must first be computed.
- These computed regions in the feature map will be of different sizes and aspect ratios. However, the Stage Two network expects a fixed size input.
- Therefore, these regions (in the feature map) must then be *reshaped* so that they all have the same size.

ROI POOLING



credit:deepsense.ai

Figure: A rectangular piece of the feature map which roughly corresponds to a region proposal is further subdivided into four unequal sections.

- To reshape, one option is to divide each rectangular region into unequal sections and perform max-pooling in each section. This is called ROIPool.
- However, the authors implement a more sophisticated method called ROIAlign to perform the reshaping.

STAGE TWO

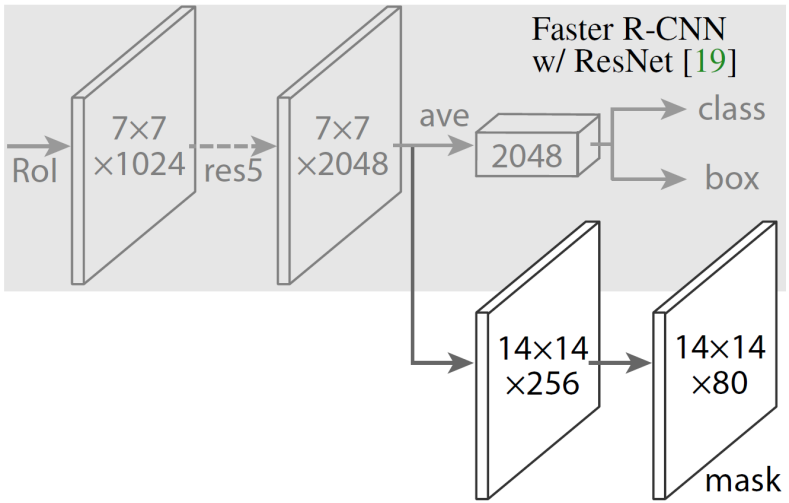
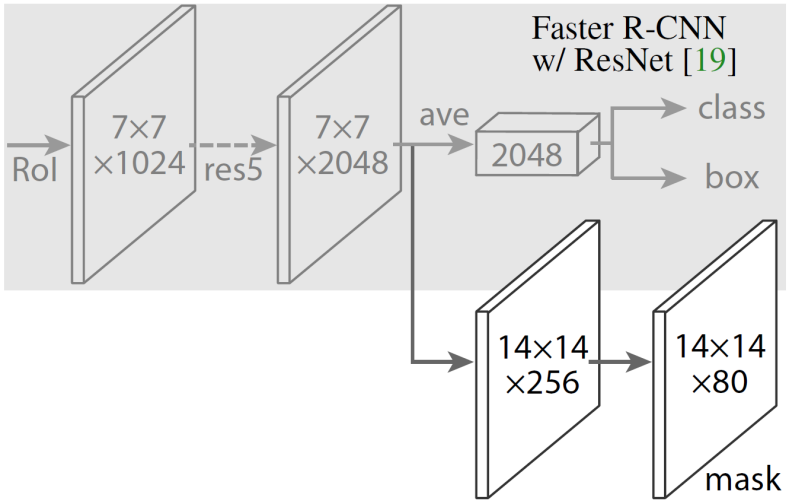


Figure: The three "heads" for classification ('class'), bounding box regression ('box') and instance segmentation ('mask'). The first two heads are also present in the Faster-RCNN architecture from which Mask R-CNN is derived.

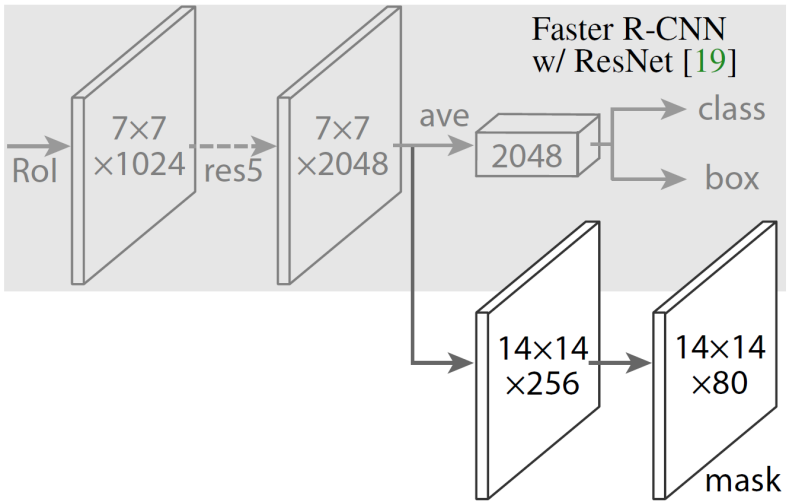
- After reshaping, a $7 \times 7 \times 1024$ ROI is fed to the fifth stage of ResNet ('res5') which outputs a $7 \times 7 \times 2048$ feature map.
- The resulting feature map is then fed to the classification, regression and mask branches.

STAGE TWO



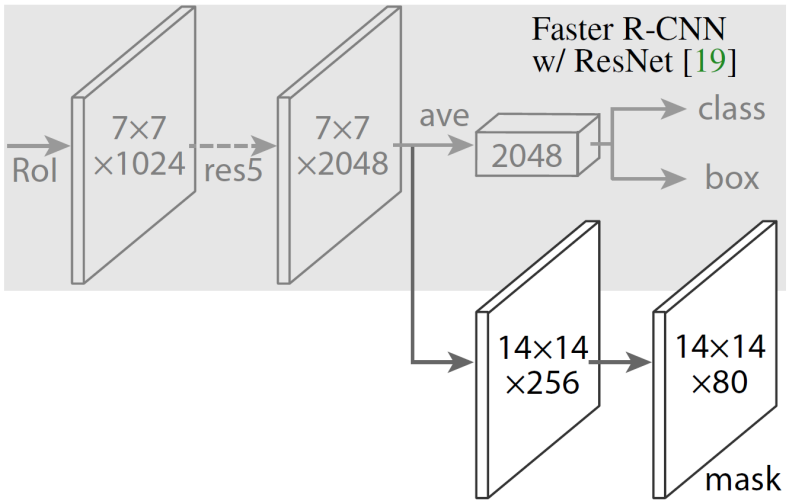
- The $7 \times 7 \times 2048$ output of 'res5' is fed to a dense layer of size 2048.
- The output of the dense layer is then fed to two 'sibling' branches which contain additional dense layers.
- The classification head outputs a vector of length $K + 1$, where K is the number of classes.
- This vector contains the probabilities for each class and, additionally, *no* class / "background" class.

STAGE TWO



- The regression head outputs 4 real valued numbers for each of the K classes.
- Each set of 4 values encodes refined bounding box predictions for one of the classes.

STAGE TWO



- The segmentation head contains a transposed convolution layer which upsamples the 7×7 feature maps to 14×14.
- Finally, the output layer, which is a regular 1×1 convolution layer with a sigmoid activation, outputs a a Km^2 dimensional output ($K = 80$ and $m = 14$ here).
- This encodes K masks of resolution $m \times m$, one for each class.
- The $m \times m$ mask is then resized to the ROI size and binarized (with a threshold of 0.5).

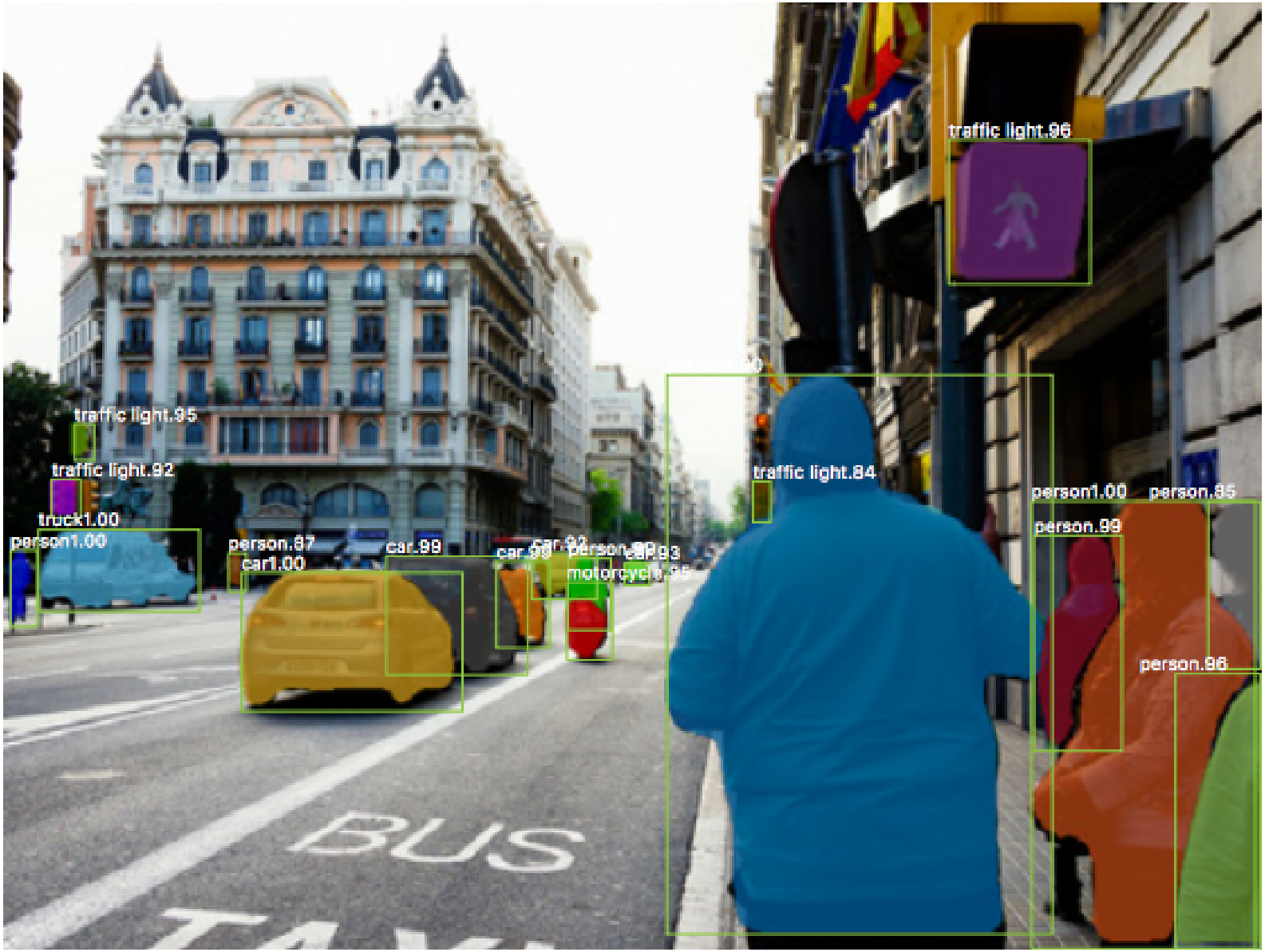
MASK R-CNN - LOSS FUNCTION

- The multi-task loss function of Mask R-CNN combines the losses of the classification, localization and segmentation masks.
- For each ROI, the loss is $L = L_{cls} + L_{box} + L_{mask}$.
- L_{cls} is the negative log likelihood for the true class.
- L_{box} is a modified $L1$ loss between the predicted and true bounding box coordinates (for the true class) which is summed over the 4 values.
- Finally, L_{mask} is the average binary cross-entropy loss.
- For an RoI associated with ground-truth class k, L_{mask} is only defined on the k-th mask (other mask outputs do not contribute to the loss).
- Note : The component losses may be weighted differently.

MASK R-CNN - TRAINING AND INFERENCE

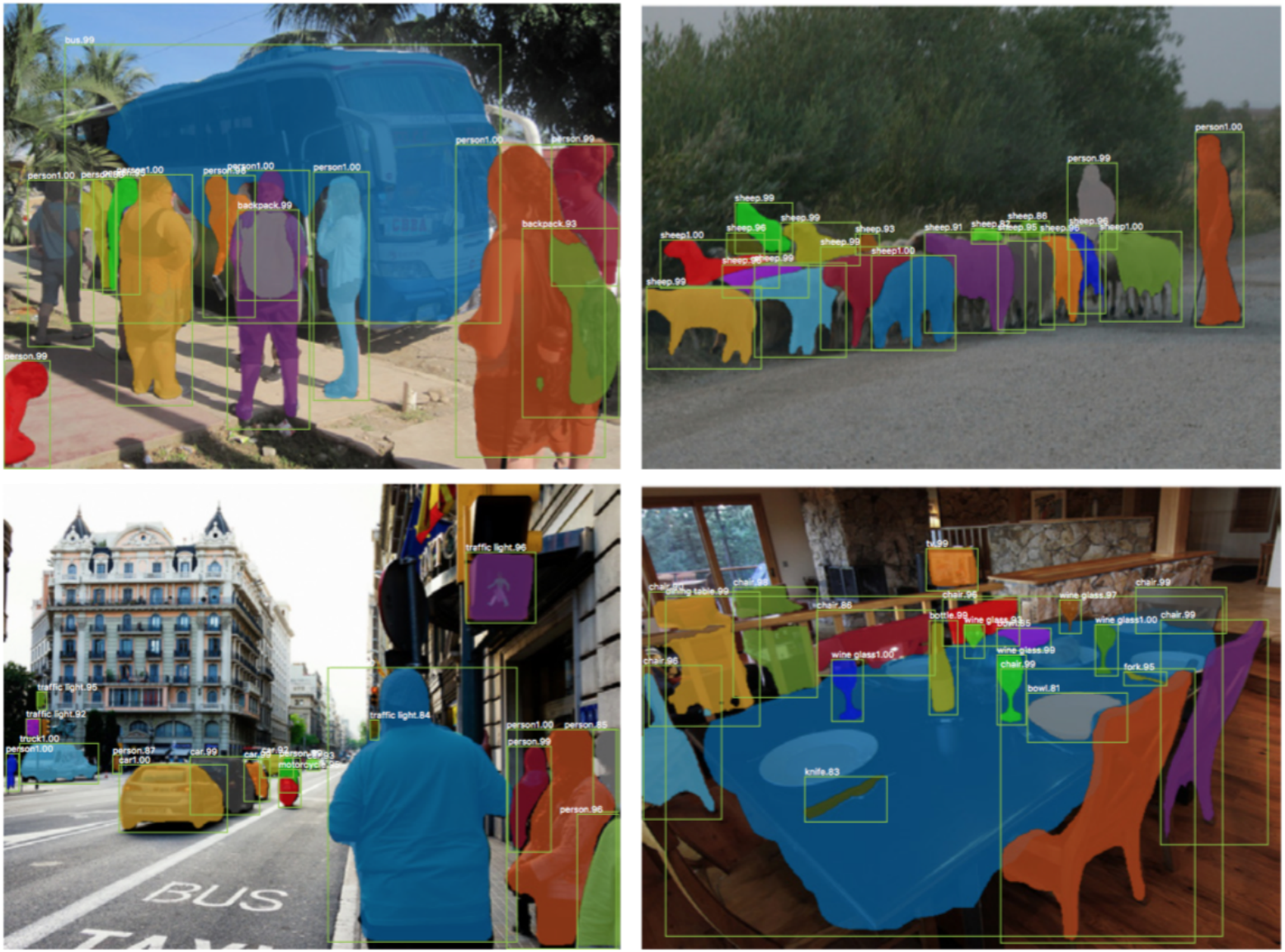
- Dataset : COCO ($\sim$ 100k training images).
- Trained on 8 GPUs with a minibatch size of 16 for 160k iterations.
- Learning rate : 0.02 for the first 120k, 0.002 thereafter.
- Weight Decay : 0.0001, Momentum : 0.9.
- Training time : 32 - 44 hours .
- Inference time : 200 ms - 400 ms on a Tesla M40.

MASK R-CNN - EXAMPLES



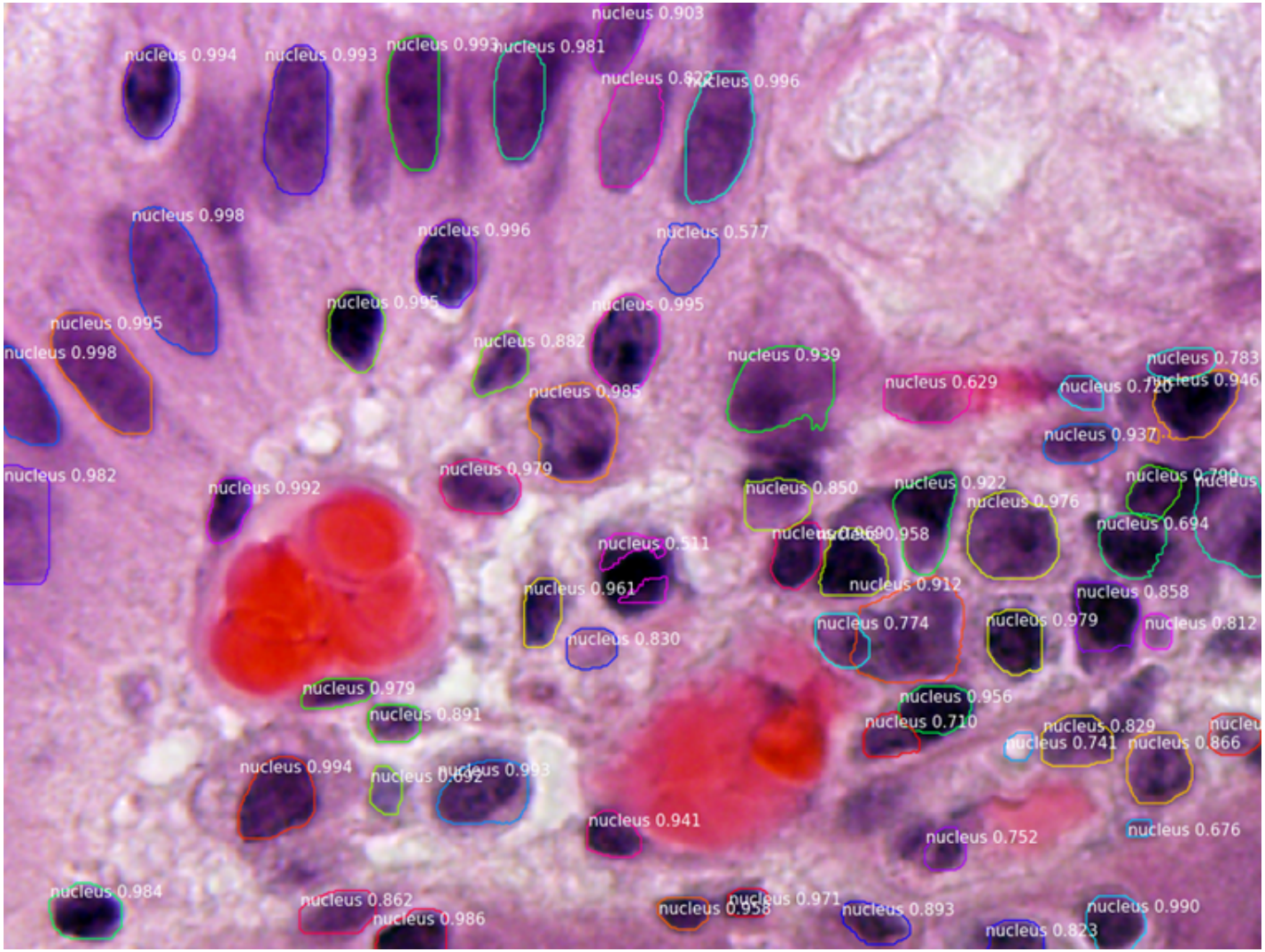
credit : He et al. 2017

MASK R-CNN - EXAMPLES



credit : He et al. 2017

MASK R-CNN - EXAMPLES



credit : Matterport

Figure: Segmenting nuclei in microscopy images.

REFERENCES

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)

Deep Learning

http://www.deeplearningbook.org/

Otavio Good (2015)

How Google Translate squeezes deep learning onto a phone

https://research.googleblog.com/2015/07/how-google-translate-squeezes-deep.html

Zhang, Richard and Isola, Phillip and Efros, Alexei A (2016)

Colorful Image Colorization

https://arxiv.org/pdf/1603.08511.pdf

Mariusz Bojarski, Davide Del Testa,Daniel Dworakowski,Bernhard Firner,Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba (2016)

End to End Learning for Self-Driving Cars

https://arxiv.org/abs/1604.07316

REFERENCES

Namrata Anand and Prateek Verma (2016)

Convolutional and recurrent nets for detecting emotion from audio data

http://cs231n.stanford.edu/reports/2015/pdfs/Cs_231n_paper.pdf

Alex Krizhevsky (2009)

Learning Multiple Layers of Features from Tiny Images

https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

Matthew D. Zeiler and Rob Fergus (2013)

Visualizing and Understanding Convolutional Networks

http://arxiv.org/abs/1311.2901

Mnih Volodymyr (2013)

Machine Learning for Aerial Image Labeling

https://www.cs.toronto.edu/~vmnih/docs/Mnih_Volodymyr_PhD_Thesis.pdf

REFERENCES

Hyeonwoo Noh, Seunghoon Hong and Bohyung Han (2015)
Learning Deconvolution Network for Semantic Segmentation
http://arxiv.org/abs/1505.04366

Karen Simonyan and Andrew Zisserman (2014)
Very Deep Convolutional Networks for Large-Scale Image Recognition
https://arxiv.org/abs/1409.1556

Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton (2012)
ImageNet Classification with Deep Convolutional Neural Networks
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Olaf Ronneberger, Philipp Fischer, Thomas Brox (2015)
U-Net: Convolutional Networks for Biomedical Image Segmentation
http://arxiv.org/abs/1505.04597

REFERENCES

Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury and Chris Pal (2016)

The Importance of Skip Connections in Biomedical Image Segmentation

http://arxiv.org/abs/1608.04117

Dumoulin, Vincent and Visin, Francesco (2016)

A guide to convolution arithmetic for deep learning

https://arxiv.org/abs/1603.07285v1

Van den Oord, Aaron, Sander Dielman, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, and Koray Kavukocuoglu (2016)

WaveNet: A Generative Model for Raw Audio

https://arxiv.org/abs/1609.03499

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhouck and Andrew Rabinovich (2014)

Going Deeper with Convolutions

http://arxiv.org/abs/1409.4842

REFERENCES

Benoit A., Gennart, Bernard Krummenacher, Roger D. Hersch, Bernard Saugy, J.C. Hadorn and D. Mueller (1996)

The Giga View Multiprocessor Multidisk Image Server

https://www.researchgate.net/publication/220060811_The_Giga_View_Multiprocessor_Multidisk_Image_Server

Tran, Du, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani and Paluri Manohar (2015)

Learning Spatiotemporal Features with 3D Convolutional Networks

https://arxiv.org/pdf/1412.0767.pdf

Milletari, Fausto, Nassir Navab and Seyed-Ahmad Ahmadi (2016)

V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation

https://arxiv.org/pdf/1606.04797.pdf

Zhang, Xiang, Junbo Zhao and Yann LeCun (2015)

Character-level Convolutional Networks for Text Classification

http://arxiv.org/abs/1509.01626

REFERENCES

Wang, Zhiguang, Weizhong Yan and Tim Oates (2017)

Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline

http://arxiv.org/abs/1509.01626

Fisher, Yu and Vladlen Koltun (2015)

Multi-Scale Context Aggregation by Dilated Convolutions

https://arxiv.org/abs/1511.07122

Bai, Shaojie, Zico J. Kolter eand Vladlen Koltun (2018)

An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

http://arxiv.org/abs/1509.01626

Distill, pub (2018)

Deconvolution and Checkerboard Artifacts

https://distill.pub/2016/deconv-checkerboard/ https://distill.pub/2016/deconv-checkerboard/

REFERENCES

B. Zhou, Khosla, A., Labedriza, A., Oliva, A. and A. Torralba (2016)

Deconvolution and Checkerboard Artifacts

http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf

Christian Szegedy (2014)

Going deeper with convolutions

https://arxiv.org/abs/1409.4842

Jie Hu, Shen, Li and Gang Sun (2017)

Squeeze-and-Excitation Networks

https://arxiv.org/abs/1709.01507

Christian Szegedy, Vanhoucke, Vincent, Sergey, Ioffe, Shlens, Jonathan and Wojna Zbigniew (2015)

Rethinking the Inception Architecture for Computer Vision

https://arxiv.org/abs/1512.00567

REFERENCES

Kaiming He, Zhang, Xiangyu, Ren, Shaoqing, and Jian Sun (2015)

Deep Residual Learning for Image Recognition

https://arxiv.org/abs/1512.03385

Zhiguang Wang, Yan, Weizhong and Tim Oates (2017)

Time series classification from scratch with deep neural networks: A strong baseline

https://arxiv.org/1611.06455

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva and Antonio Torralba (2016)

Learning Deep Features for Discriminative Localization

http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf

LeCunn Yann, Bottou, Leon, Bengio, Yoshua and Patrick Haffner (1998)

Global training of document processing systems using graph transformer networks

http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

REFERENCES

Krizhevsky Alex, Sutskever, Ijla, and Geoffray Hinton (2012)

ImageNet Classification with Deep Convolutional Neural Networks

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Simoyan Karen and Andrew Zisserman (2014)

Very Deep Convolutional Networks for Large-Scale Image Recognition

https://arxiv.org/abs/1409.1556

Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun (2015)

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

https://arxiv.org/abs/1506.01497

Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick (2017)

Mask R-CNN

https://arxiv.org/abs/1703.06870