LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Deep Learning

Chapter 6: Convolutional neural networks II

Bernd Bischl

Department of Statistics – LMU Munich

Winter term 2020

Important types of convolutions

1D CONVOLUTIONS

Data situation: Sequential, 1-dimensional tensor data.

- Data consists of tensors with shape [channels, xdim]
- Single channel:
  - Univariate time series, e.g. development of a single stock price over time
  - Functional / curve data
- Multi-channel:
  - Multivariate time series, e.g.
    - Movement data measured with multiple sensors for human activity recognition [Wang et al., 2017]
    - Temperature and humidity in weather forecasting
  - Text encoded as character-level one-hot-vectors [Zhang et al., 2015]

$\rightarrow$ Convolve the data with a 1D-kernel

1D CONVOLUTIONS – SENSOR DATA

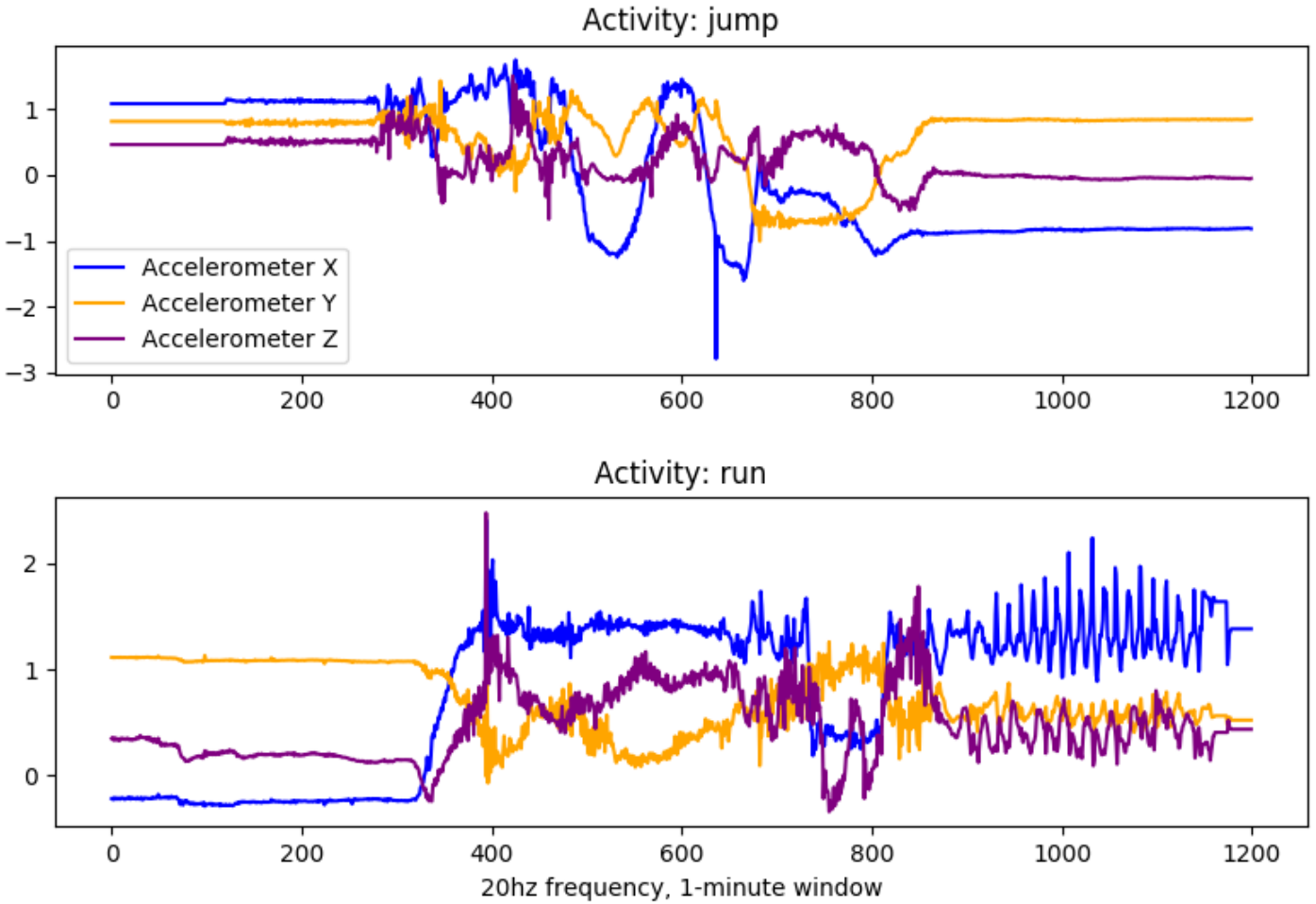


Figure: Illustration of 1D movement data with three channels measured with an accelerometer sensor belonging to a human activity recognition task.

1D CONVOLUTIONS – SENSOR DATA

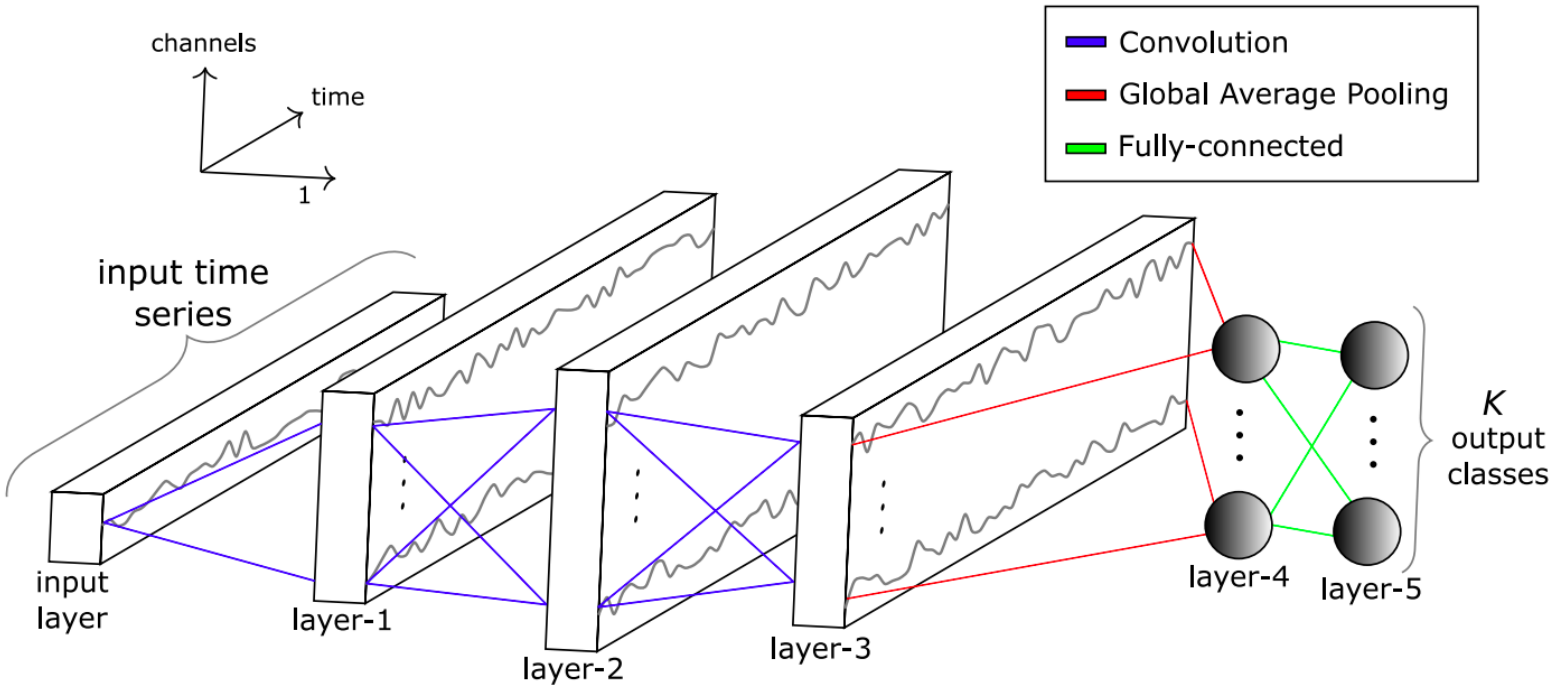


Figure: Time series classification with 1D CNNs and global average pooling (explained later). An input time series is convolved with 3 CNN layers, pooled and fed into a fully connected layer before the final softmax layer. This is one of the classic time series classification architectures [Wang et. al , 2017].

1D CONVOLUTIONS – TEXT MINING

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

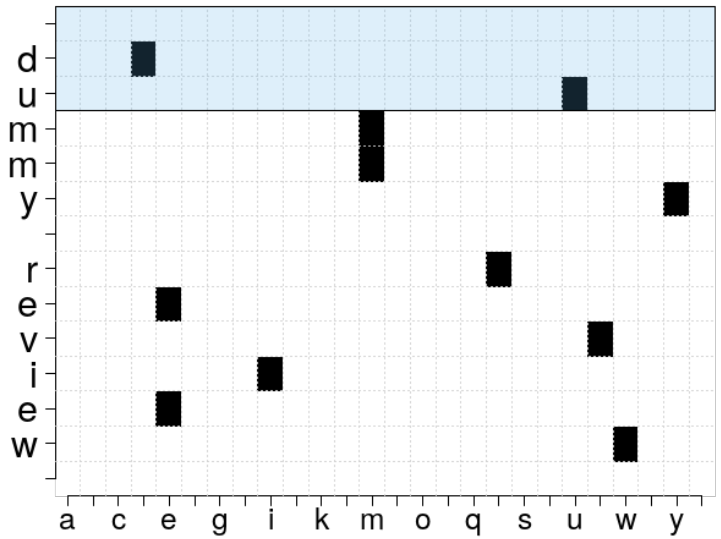
Miriam L.
Munich, Germany
57 friends
437 reviews
450 photos

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.

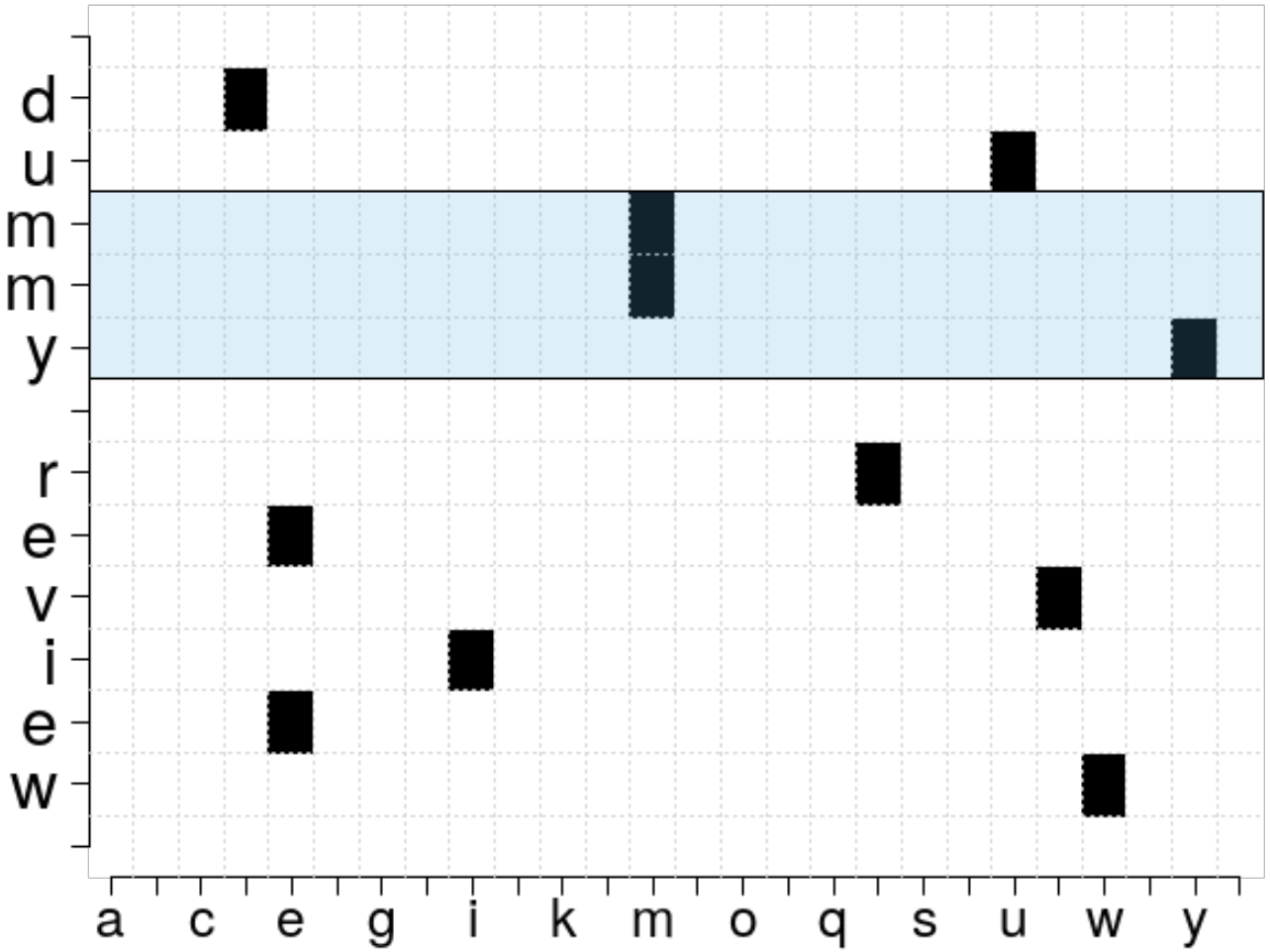
Figure: Sentiment classification: can we teach the net that this a positive review?

1D CONVOLUTIONS – TEXT MINING



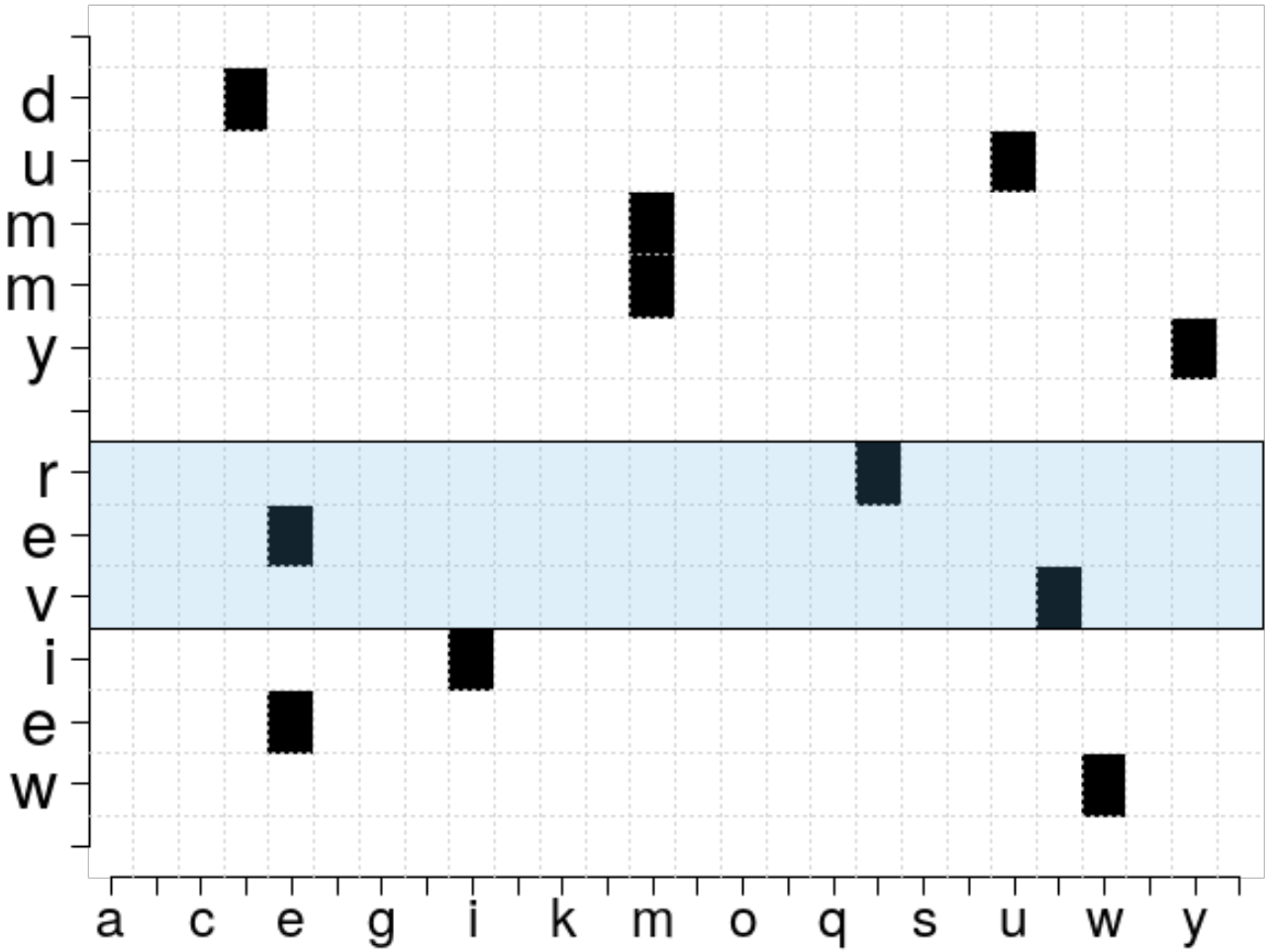
- We use a given alphabet to encode the text reviews (here: *“dummy review”*).
- Each character is transformed into a one-hot vector. The vector for character *d* contains only 0’s at all positions except for the 4th position.
- The maximum length of each review is set to 1014: shorter texts are padded with spaces (zero-vectors), longer texts are simply cut.

1D CONVOLUTIONS – TEXT MINING



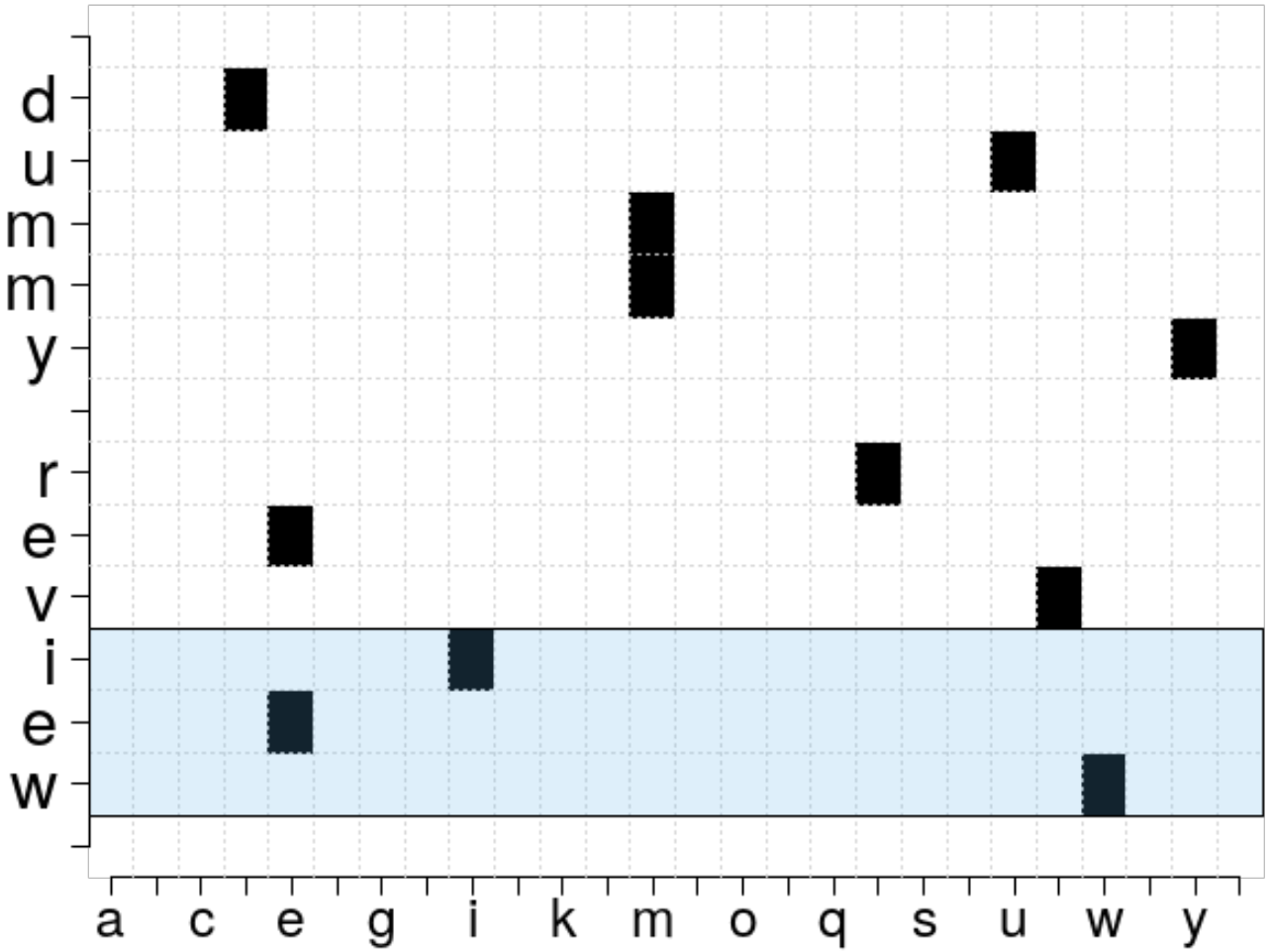
- The data is represented as 1D signal with channel size = size of the alphabet.

1D CONVOLUTIONS – TEXT MINING



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

1D CONVOLUTIONS – TEXT MINING



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

1D CONVOLUTIONS – TEXT MINING

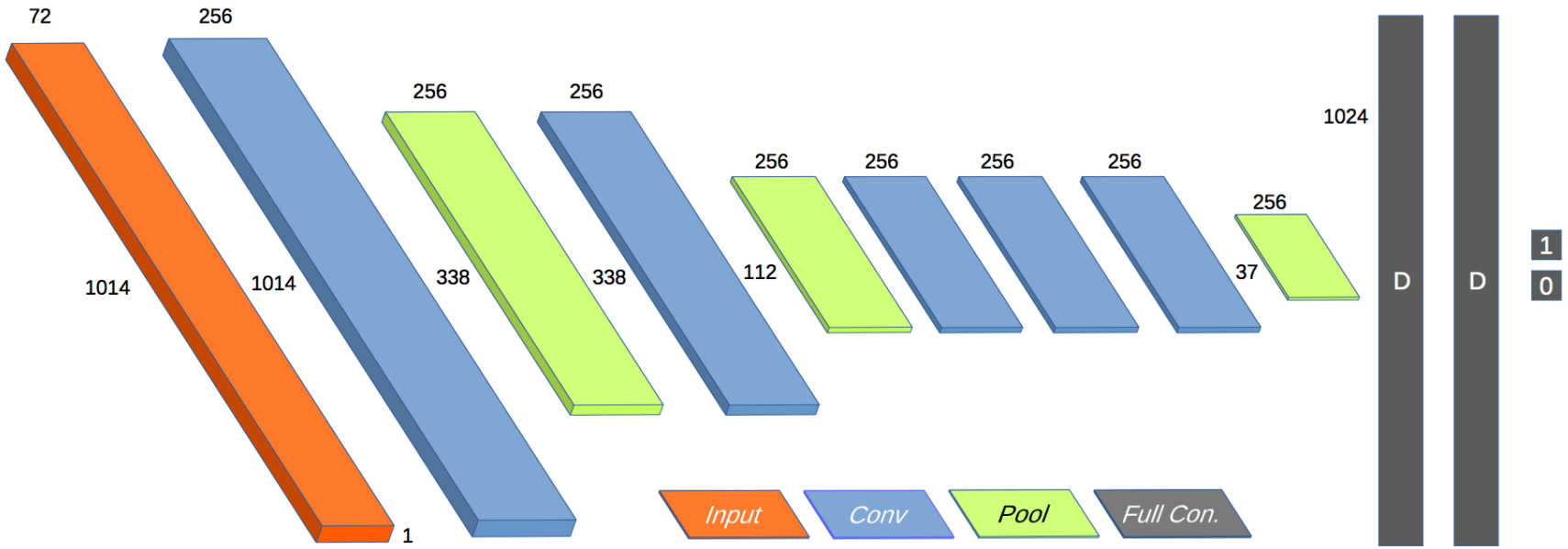


Figure: An alphabet containing 72 (special-) characters is used for encoding and the text length is fixed to 1014. This yields an accuracy of 96% with a net trained on 500K reviews [Zhang et al., 2015].

2D CONVOLUTIONS

Data situation: 2-dimensional tensor data such as image data.

→ Refer to the previous lecture for a comprehensive introduction to 2-dimensional convolutions.

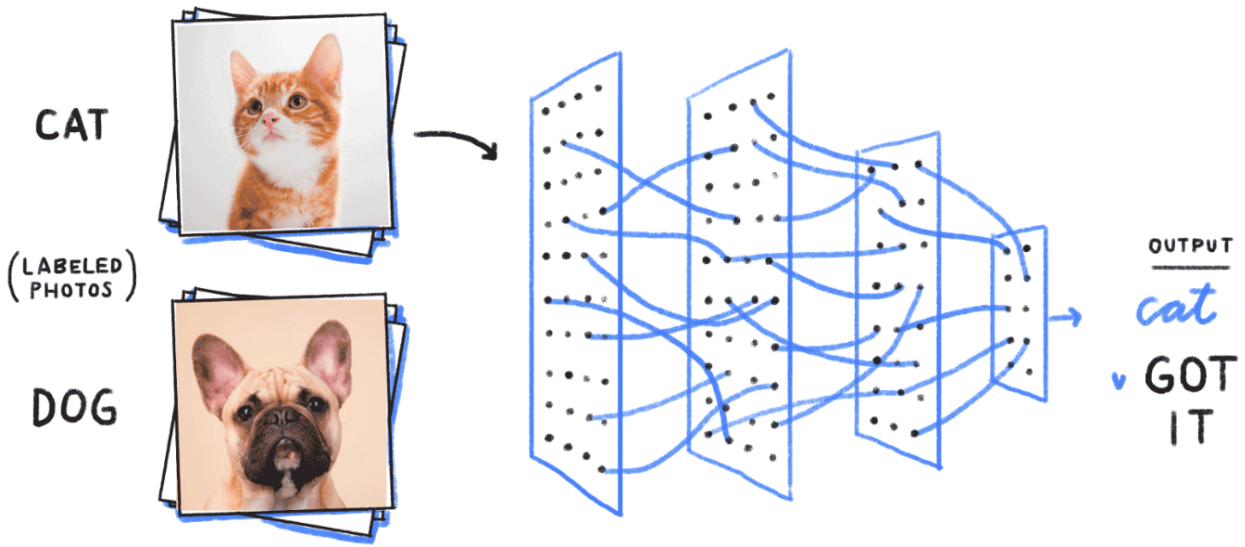


Figure: Simple binary classification problem on 2D image data: cat vs. dog detection as illustrated here.

3D CONVOLUTIONS

Data situation: 3-dimensional tensor data.

- Data consists of tensors with shape [channels, xdim, ydim, zdim].
- Dimensions can be both temporal (e.g. video frames) or spatial (e.g. MRI)
- Examples:
  - Human activity recognition in video data [Tran et al., 2015]
  - Disease classification or tumor segmentation on MRI scans [Milletari et al., 2016]

Solution: Move a 3D-kernel in x, y and z direction to capture all important information.

3D CONVOLUTIONS – DATA

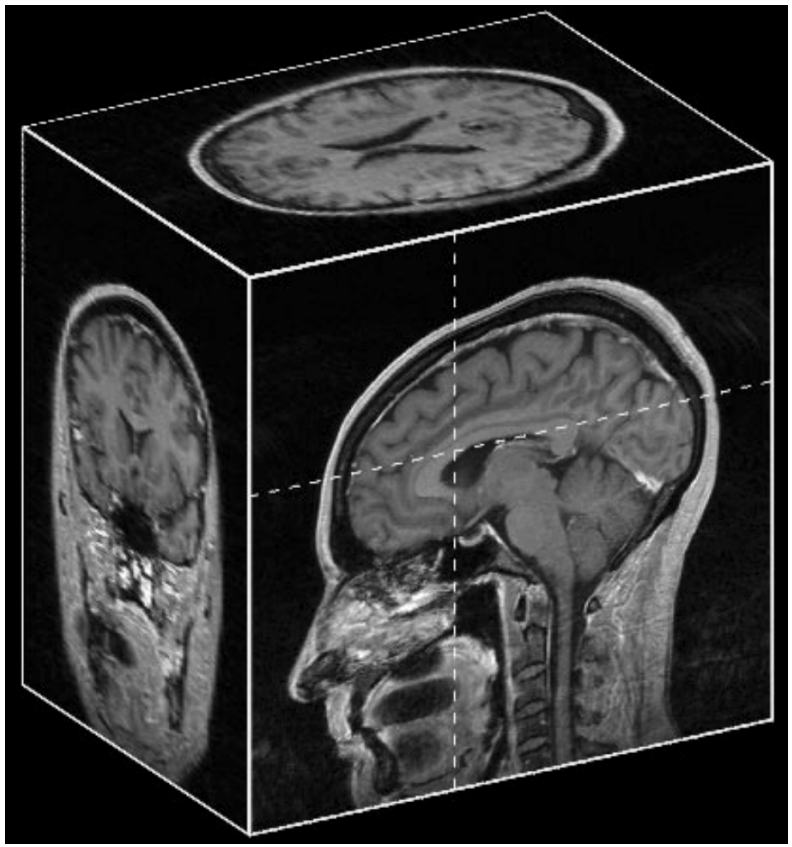


Figure: Illustration of single-channel volumetric data: MRI scan [Gennart et al., 1996]. Each slice of the stack has one channel, as the frames are black-white.

3D CONVOLUTIONS – DATA

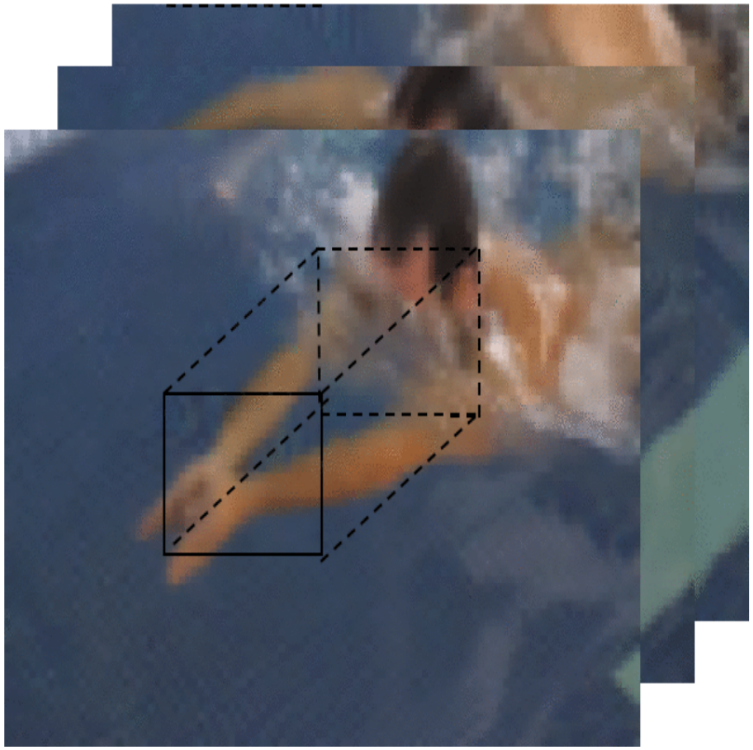
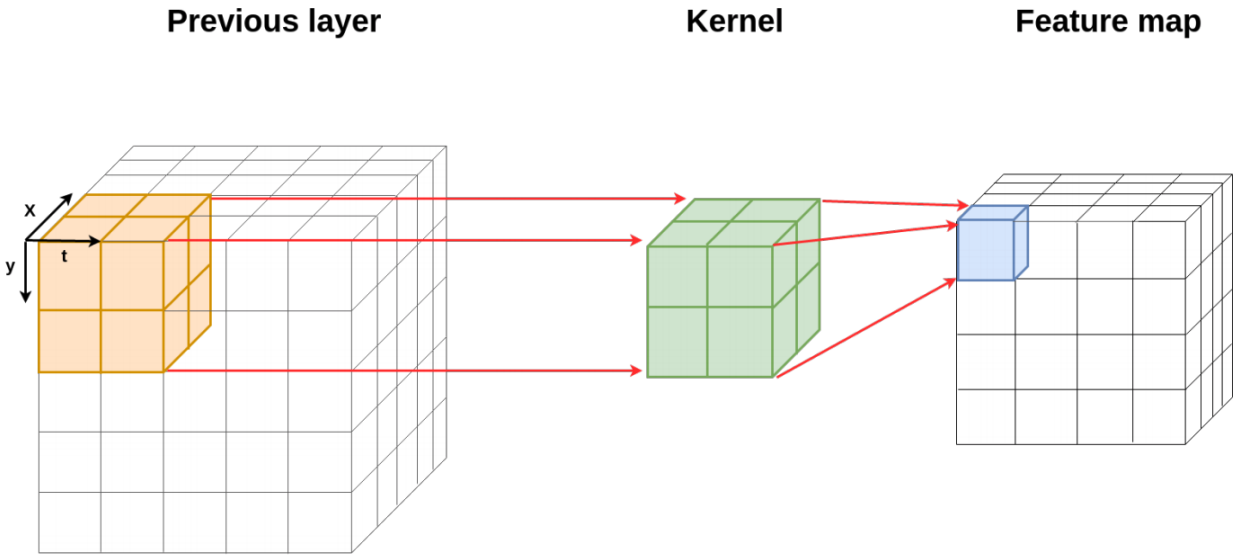


Figure: Illustration of multi-channel volumetric data: video snippet of an action detection task. The video consists of several slices, stacked in temporal order. Frames have three channels, as they are RGB.

3D CONVOLUTIONS



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

3D CONVOLUTIONS

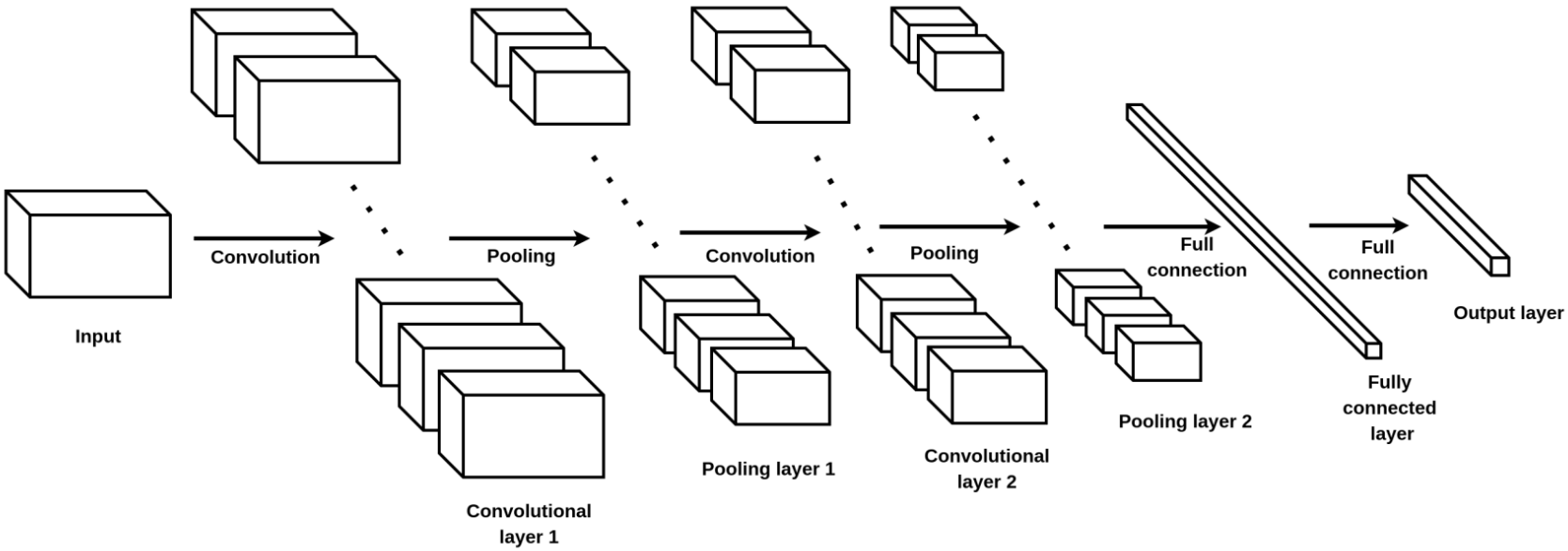


Figure: Basic 3D-CNN architecture.

- 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.

DILATED CONVOLUTIONS

- The **receptive field** of a single neuron comprises all inputs that have an impact on this neuron.
- Neurons in the first layers capture less information of the input, while neurons in the last layers have huge receptive fields and can capture a lot more global information from the input.
- The size of the receptive fields depends on the filter size.

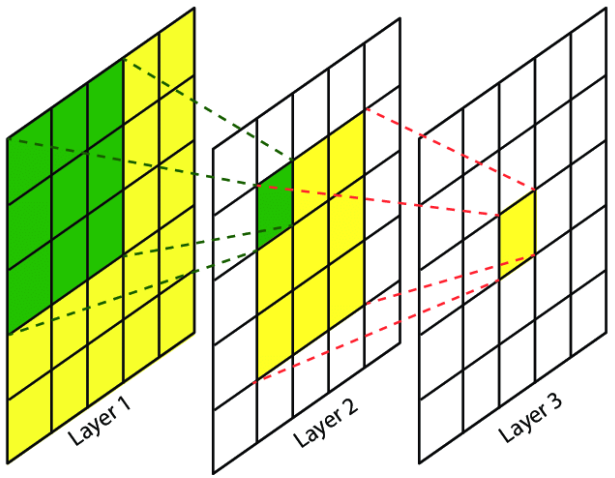


Figure: Receptive field of each convolution layer with a 3×3 kernel. The green area marks the receptive field of one pixel in Layer 2, the yellow area marks the receptive field of one pixel in layer 3 [Lin et al., 2017].

DILATED CONVOLUTIONS

- By increasing the filter size, the receptive fields of the neurons increase as well and more contextual information can be captured.
- However, increasing the filter size increases the number of parameters, which leads to increased runtime.
- Idea: artificially increase the receptive field of the net without using more filter weights by adding a new dilation parameter to the kernel $\mathcal{K}$ that skips pixels during convolution.

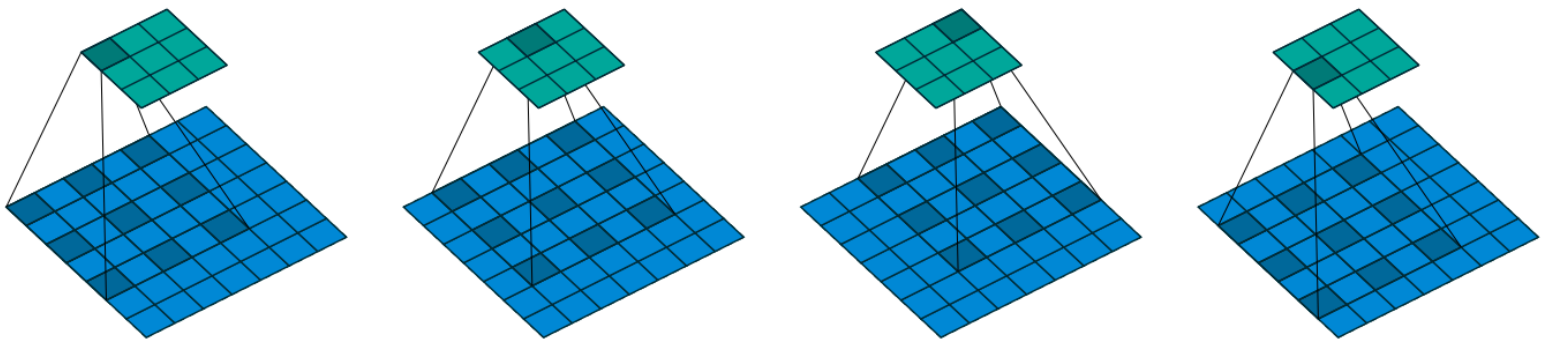


Figure: Dilated convolution on 2D data [Dumoulin et al., 2016]. A dilated kernel is a regular convolutional kernel interleaved with zeros.

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 [van den Oord et al., 2016].
  - Time series classification and forecasting [Bai et. al, 2018].
  - Image segmentation [Yu et. al, 2015].

DILATED CONVOLUTIONS

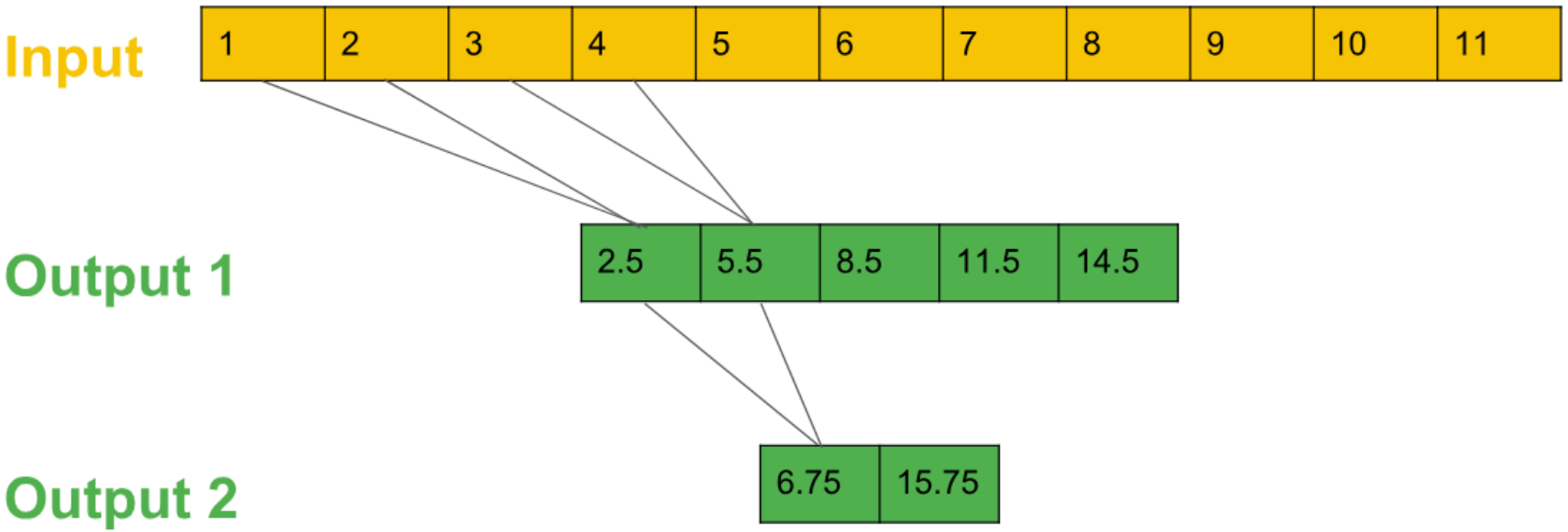


Figure: Regular convolution: we use a convolutional kernel of size 2 with the fixed weights $\{0.5, 1.0\}$ and convolve the input vector with a stride of 2. We do not dilate the kernel in this example (the dilation factor is one) and one neuron in layer 2 has a receptive field of size 4 after two stacked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

DILATED CONVOLUTIONS

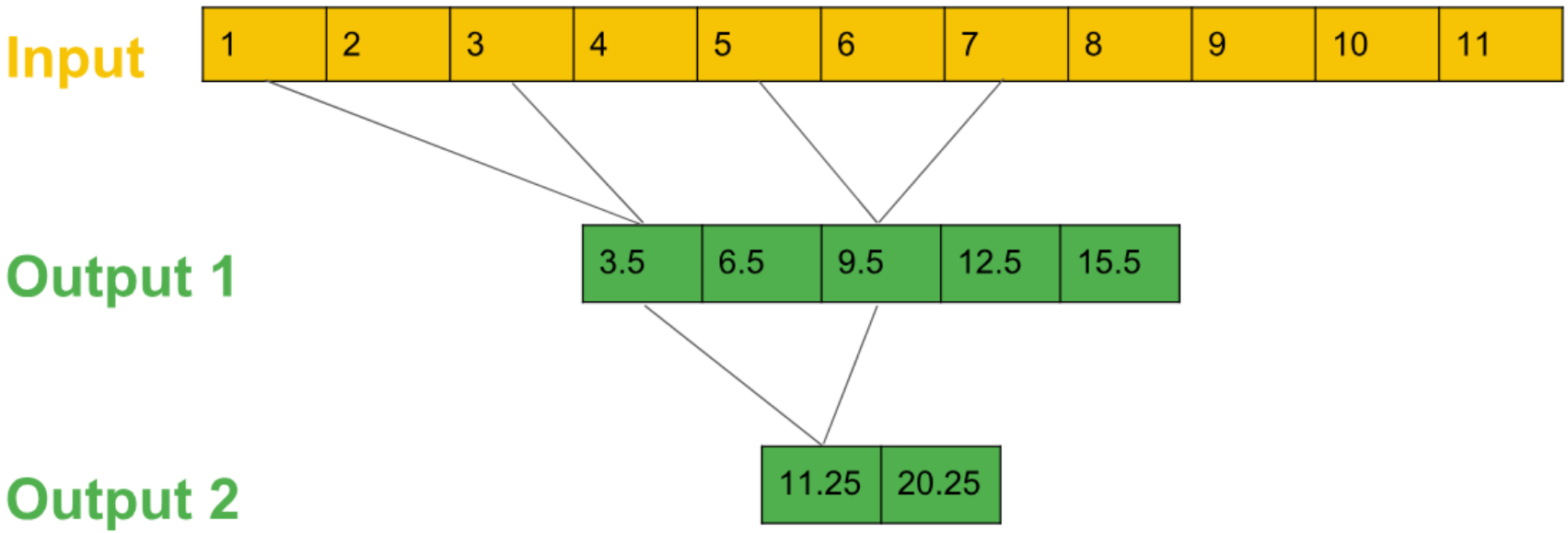


Figure: Dilated convolution: we use a convolutional kernel of size 2 with the fixed weights $\{0.5, 1.0\}$ and convolve the input vector with a stride of 2. We dilate the kernel with a factor of 2 in this example and one neuron in layer 2 has a receptive field of size 8 after two stacked layers.

DILATED CONVOLUTIONS

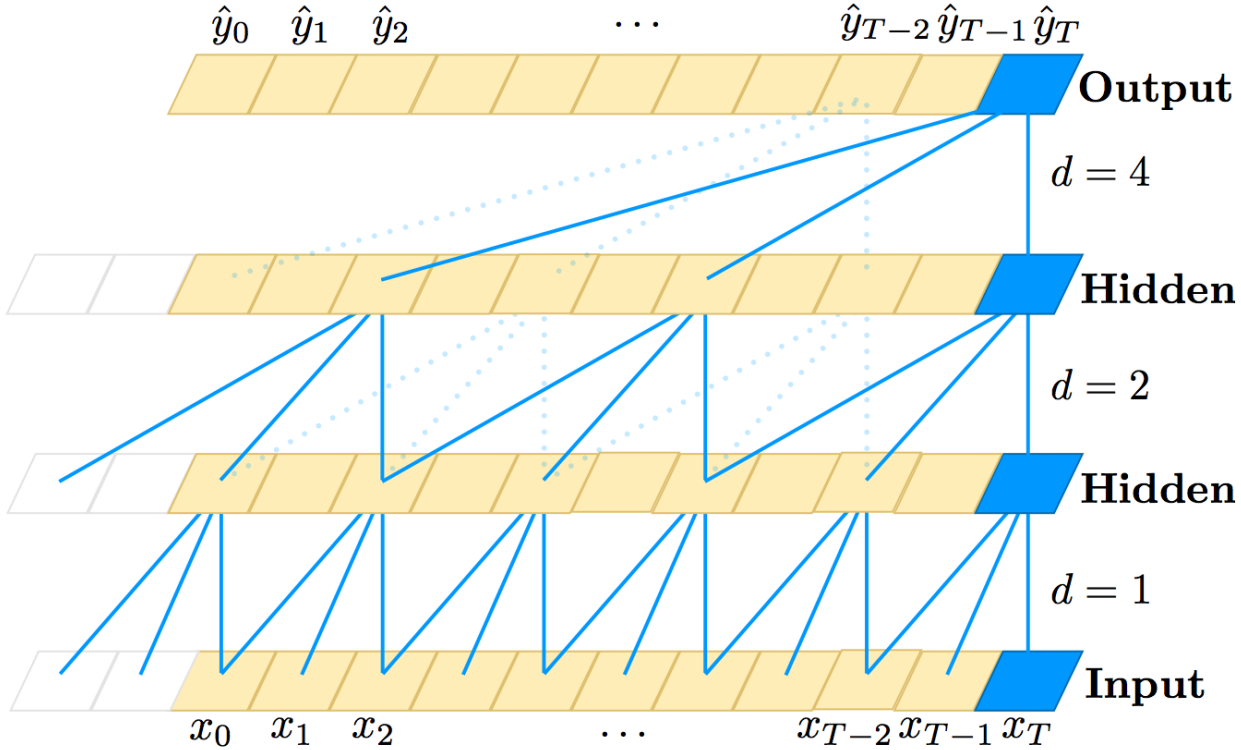


Figure: Application of dilated convolutions on time series for classification or seq2seq prediction [Bai et. al, 2018]. The dilations are used to drastically increase the context information for each output neuron y_i with relatively few layers.

TRANSPOSED CONVOLUTIONS

- Problem setting:
  - In some encoder-decoder architectures such as (variational) Autoencoders, Segmentation Nets, GANs, dimensions of the feature maps are getting reduced in a first step,
  - before being re-increased to the original dimensionality in a second step.
- Instead of decreasing dimensionality as with regular convolutions, **transposed convolutions** are used to re-increase dimensionality back to the initial dimensionality.
- Note: Do not confuse this with deconvolutions (which are mathematically defined as the inverse of a convolution).

TRANSPOSED CONVOLUTIONS

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

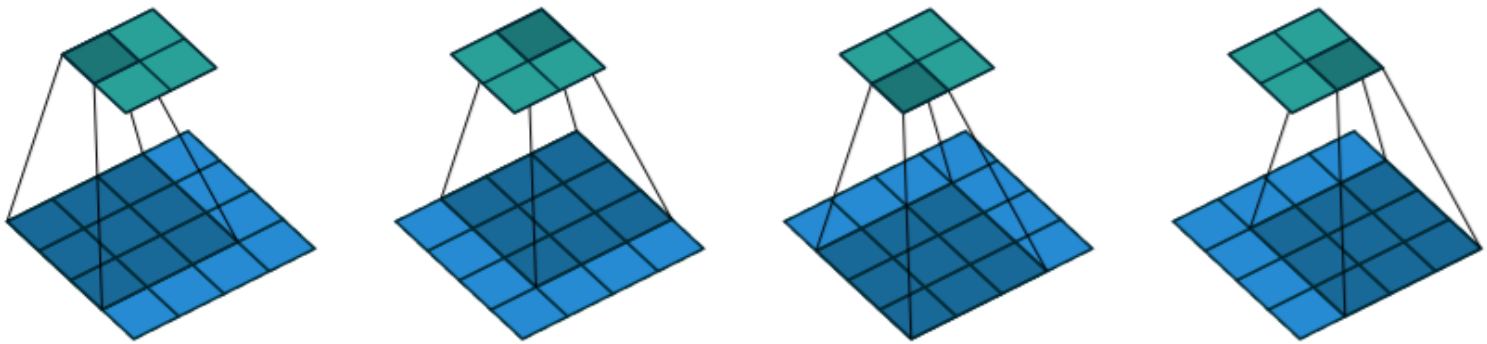


Figure: A **regular** convolution with kernel-size $k = 3$, padding $p = 0$ and stride $s = 1$.

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

TRANSPOSED CONVOLUTIONS

- Example 1:
  - Now, let us 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.

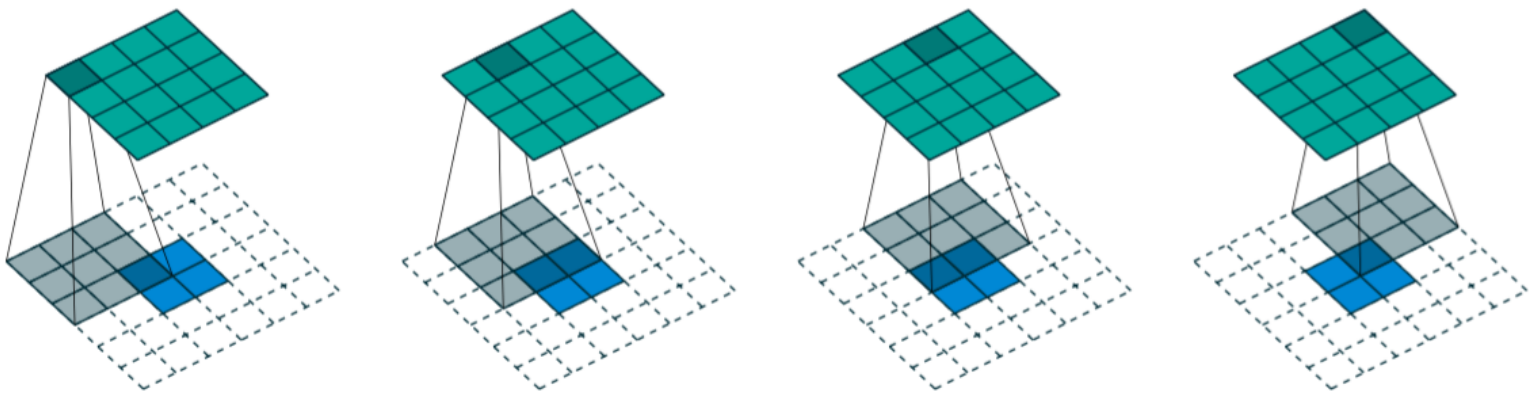


Figure: Transposed convolution can be a seen as a regular convolution. Convolution (above) with $k' = 3, s' = 1, p' = 2$ re-increases dimensionality from 2×2 to 4×4 as shown in [Dumoulin et al., 2016]

TRANSPOSED CONVOLUTIONS

Example 2: Transposed Convolution as matrix multiplication:

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

K z

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.

TRANSPOSED CONVOLUTIONS

Example 2: Transposed Convolution as matrix multiplication:

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

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

Note:

- 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 optimized by backpropagation).
- 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.

TRANSPOSED CONVOLUTIONS

Example 2: Transposed Convolution as matrix multiplication:

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

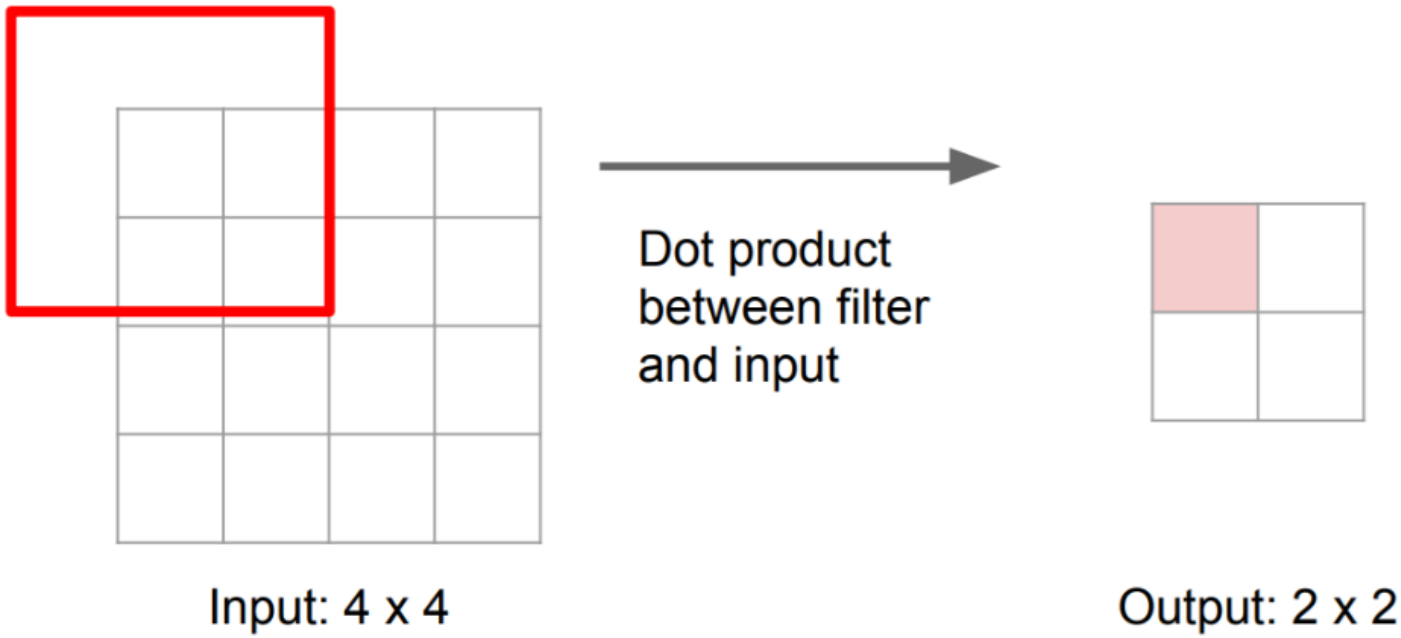
Figure: A transposed convolution can be used to upsample the feature vector of length 4 back to a feature vector of length 6.

Note:

- 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 $\tilde{z}_3$.
- In general, transposing the original matrix does not 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).

TRANSPOSED CONVOLUTIONS

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

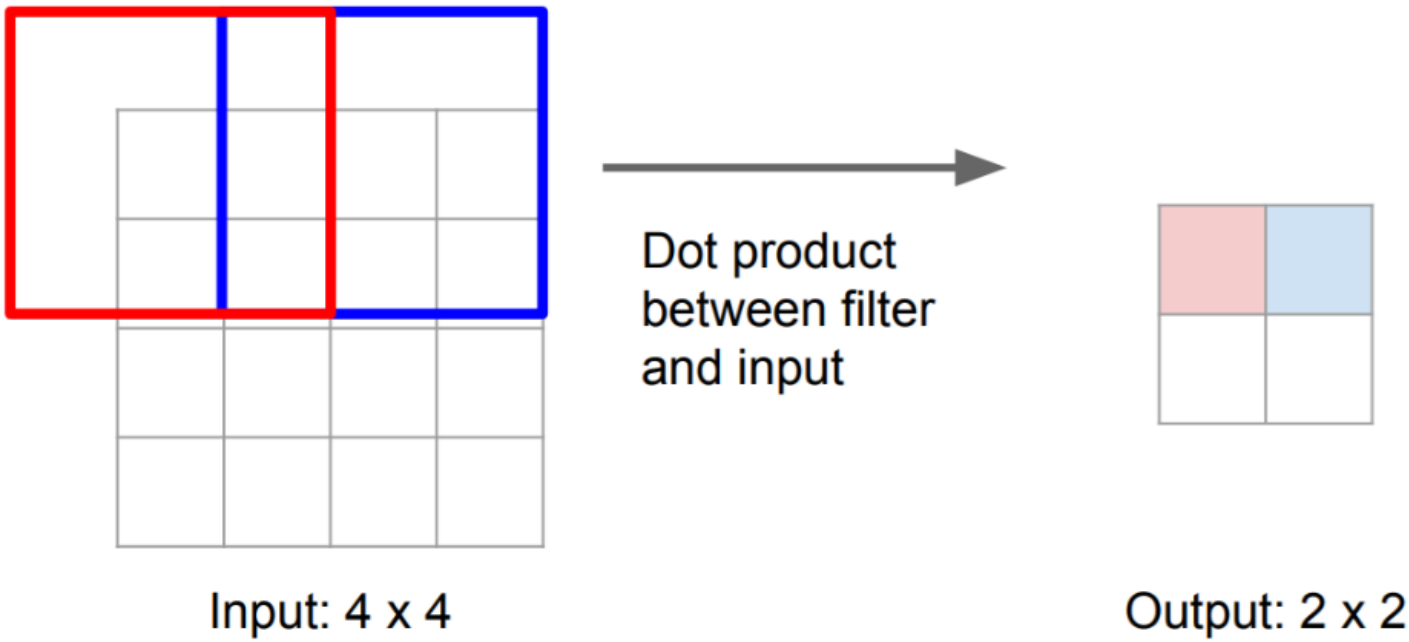


credit: Stanford University

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

TRANSPOSED CONVOLUTIONS

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

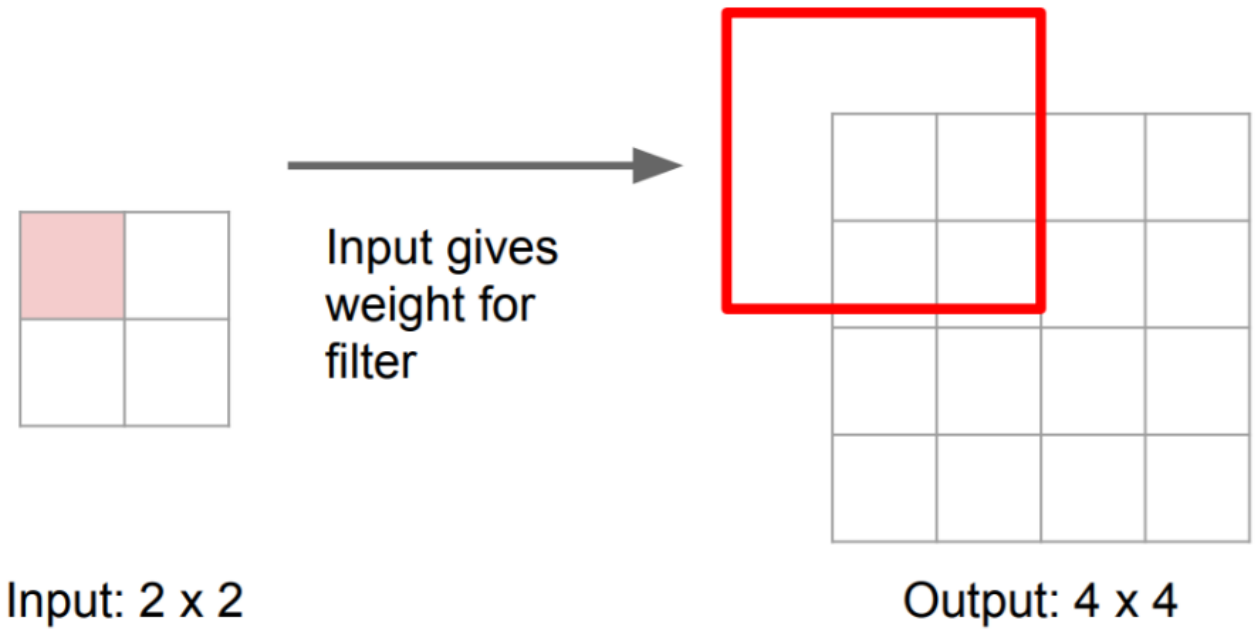


credit: Stanford University

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

TRANSPOSED CONVOLUTIONS

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



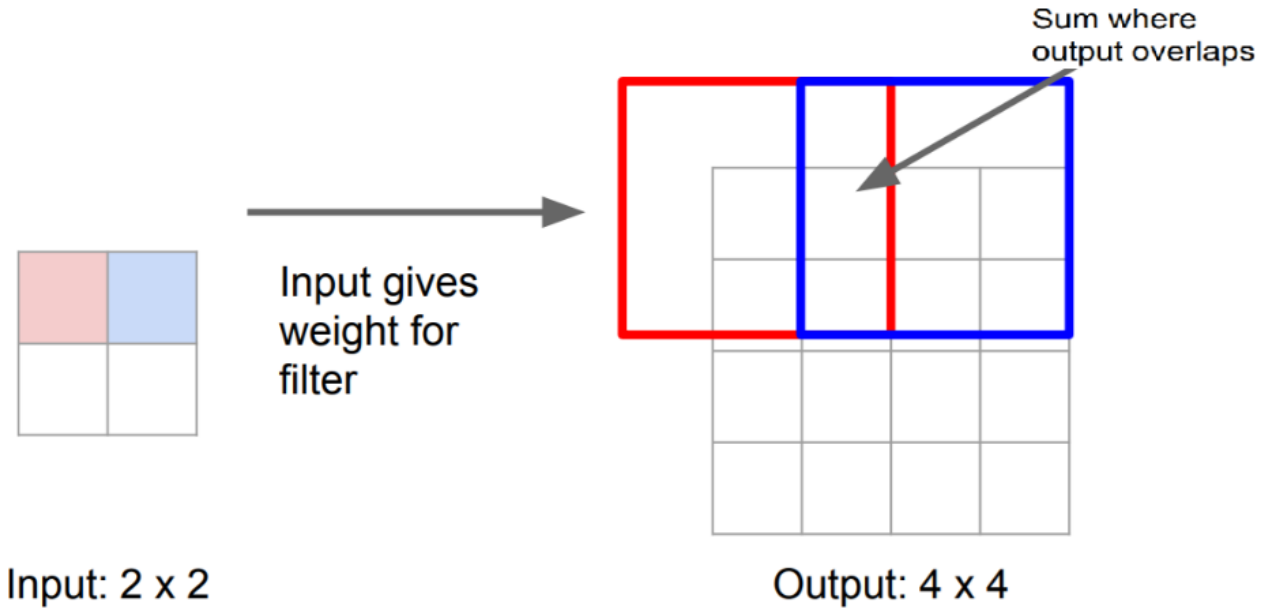
credit: Stanford University

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

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

TRANSPOSED CONVOLUTIONS

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



credit: Stanford University

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

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

TRANSPOSED CONVOLUTIONS – DRAWBACK

Figure: Artifacts produced by transposed convolutions [Odena et. al , 2017].

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

TRANSPOSED CONVOLUTIONS – 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.

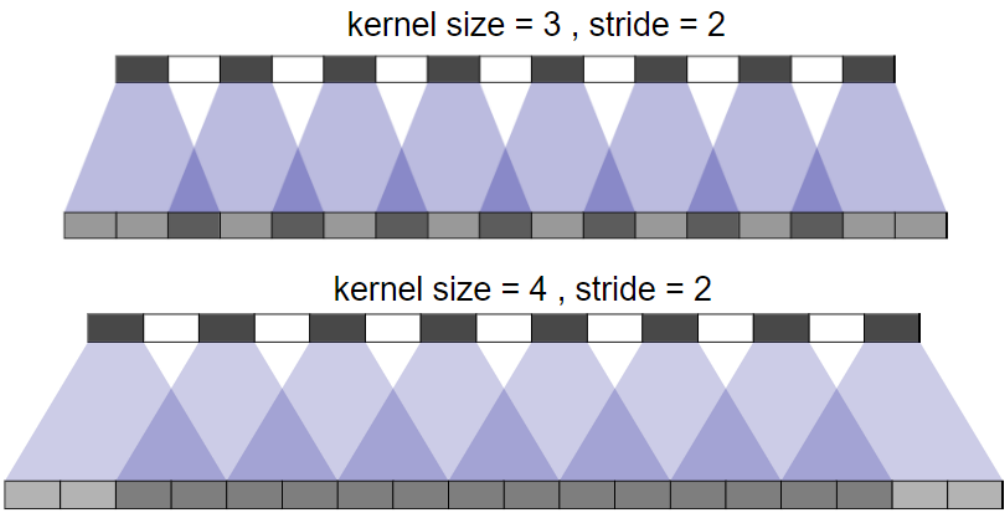


Figure: 1D example. In both images, top row = input and bottom row = output. *Top*: Here, kernel weights overlap unevenly which results in a checkerboard pattern. *Bottom*: There is no checkerboard pattern as the kernel size is divisible by the stride.

Advanced components of CNNs

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 [Szegedy et. al , 2014].

INCEPTION MODULES

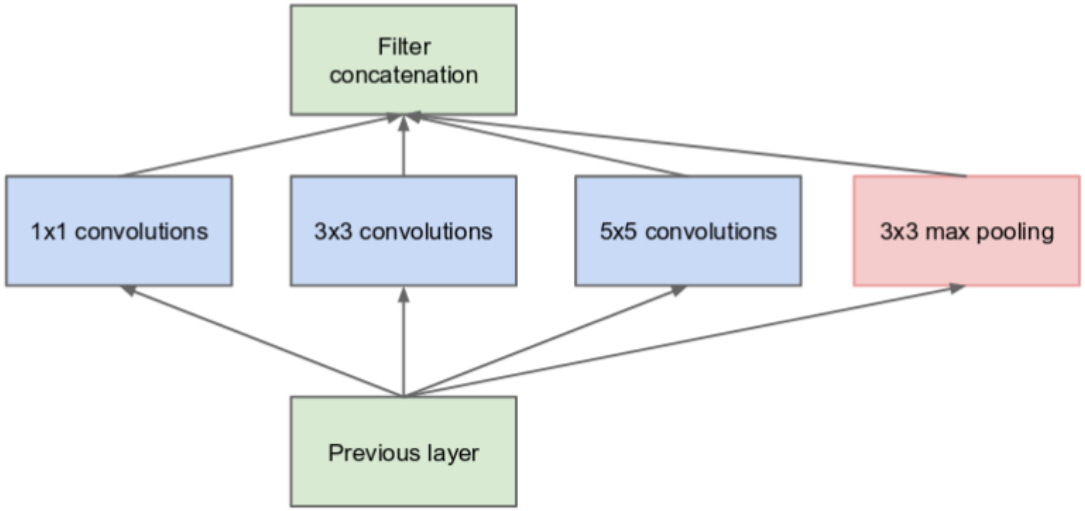


Figure: Naive inception module. The model can “choose” from kernels of different sizes.

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.

INCEPTION MODULES

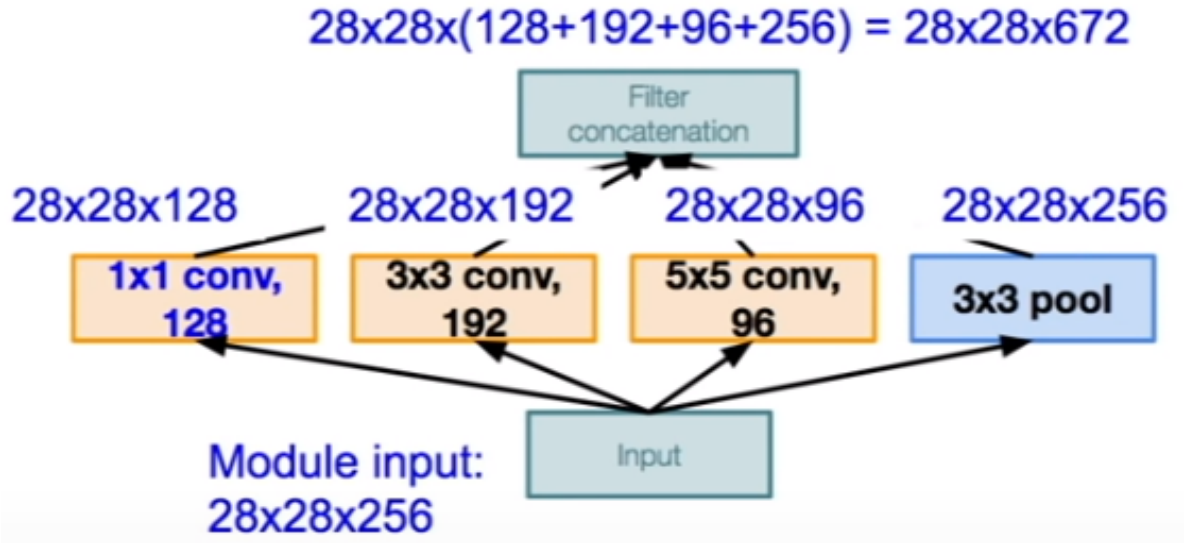


Figure: Naive Inception module - Example

- 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.

INCEPTION MODULES

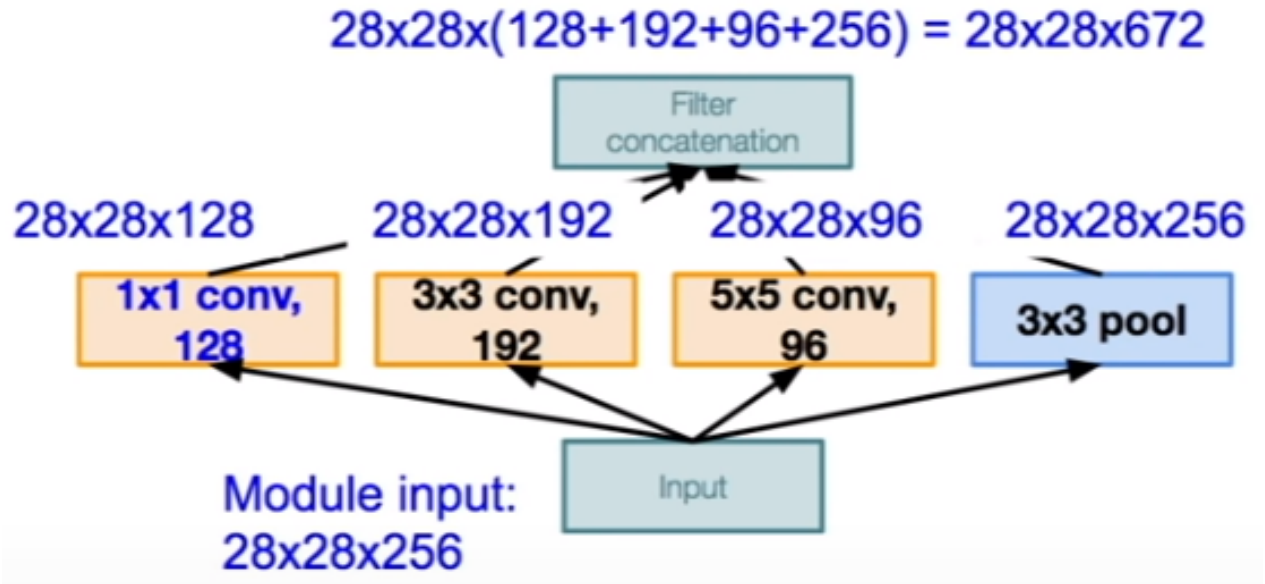


Figure: Naive Inception module - Example

- 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.

INCEPTION MODULES

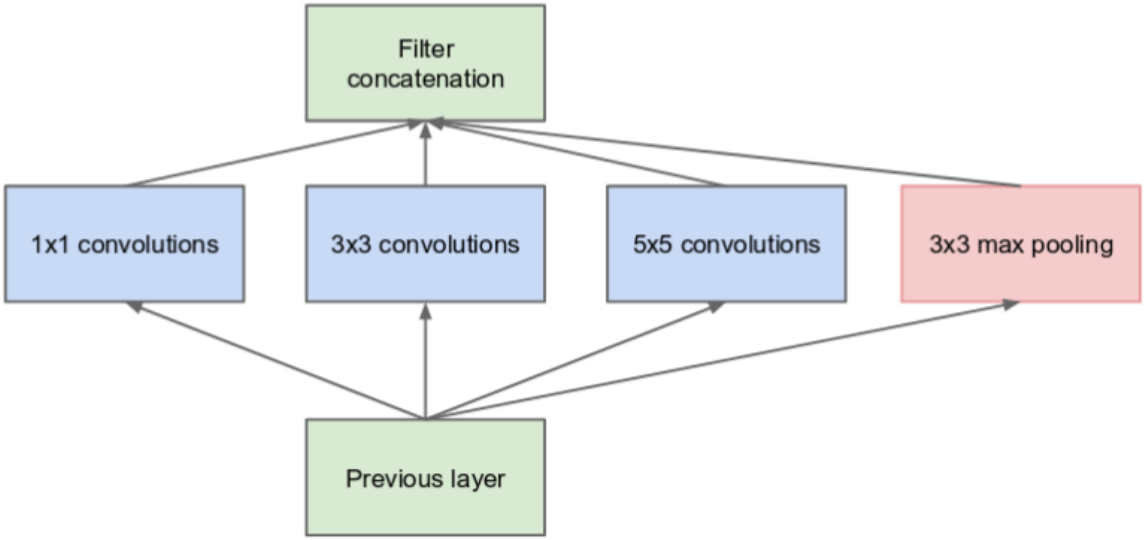


Figure: Naive inception module.

- 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.

INCEPTION MODULES

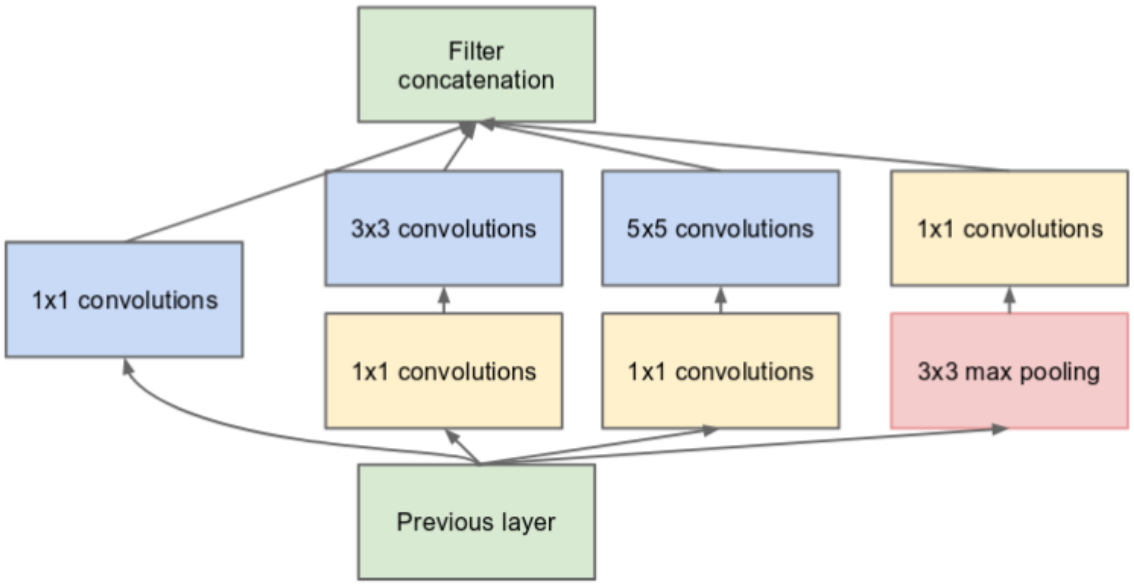


Figure: Dimensionality reduced inception module.

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

INCEPTION MODULES

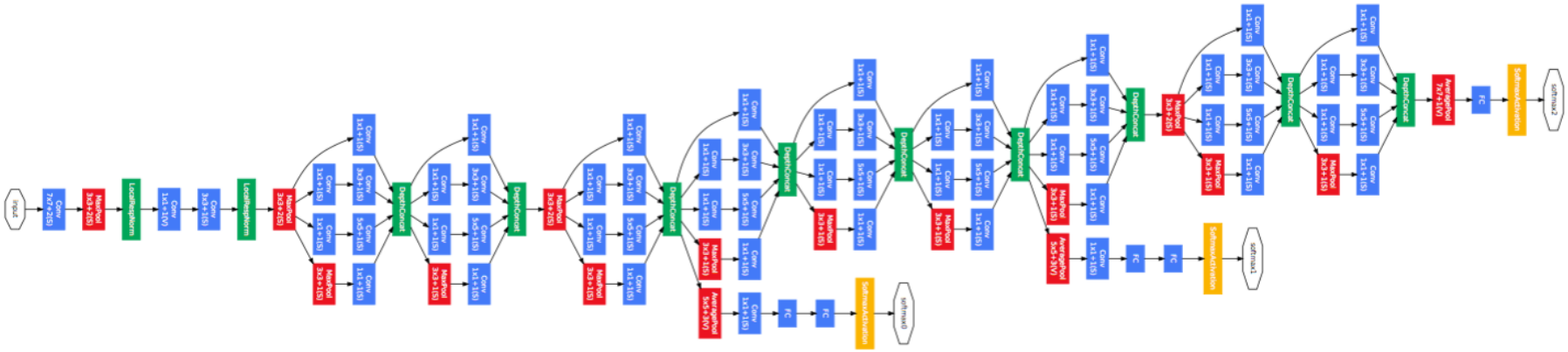


Figure: Inception modules are the integral part of the famous GoogLeNet (2014), also known as Inception net, one of the first very deep net architectures.

INCEPTION MODULES

- Let us 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: $1^2 \cdot 28^2 \cdot 192 \cdot 16 + 5^2 \cdot 28^2 \cdot 16 \cdot 32 = 12.443.648$

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 $\mathbf{x} = \mathcal{F}(\mathbf{x})$. 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 et. al , 2015] and motivated by the observation that stacking evermore layers increases the test- as well as the train-error ($\neq$ overfitting).

SKIP CONNECTIONS

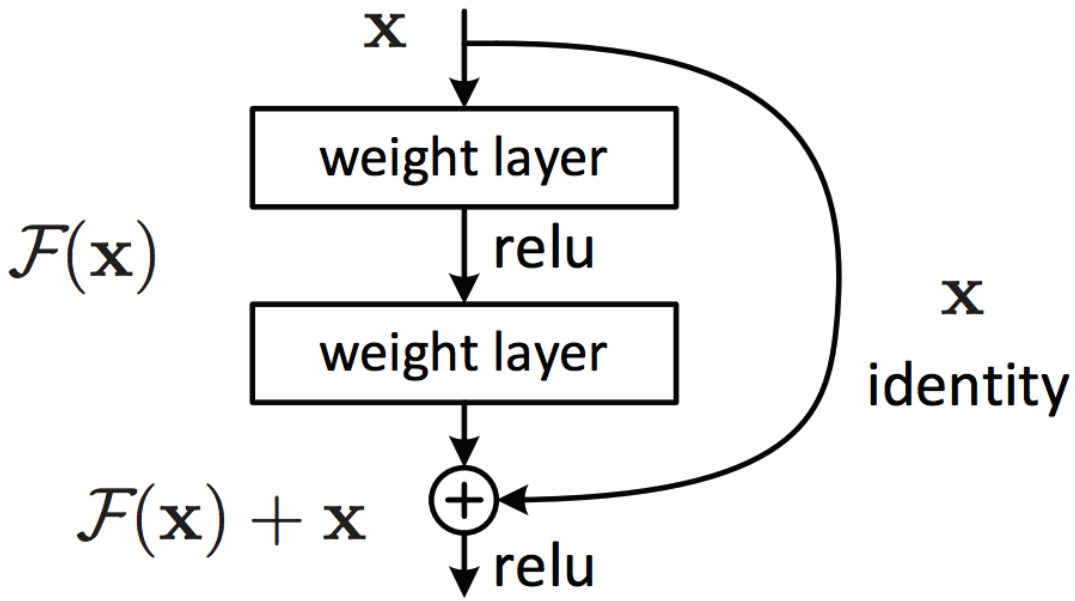


Figure: Skip connection/ residual learning module. The information flows through two layers and the identity function. Both streams of information are then element-wise summed and jointly activated.

SKIP CONNECTIONS

- Let $\mathcal{H}(\mathbf{x})$ 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).
- $\mathcal{H}(\mathbf{x})$ is the output from layer l.
- Instead of fitting $\mathcal{H}(x)$, the net is ought to learn the residual mapping $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$ whilst x is added via the identity mapping.
- Thus, $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$, as formulated on the previous slide.
- The model should only learn the **residual mapping** $\mathcal{F}(\mathbf{x})$
- Thus, the procedure is also referred to as **Residual Learning**.

SKIP CONNECTIONS

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

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

- 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 $\mathcal{F}(\mathbf{x})$).
- This idea applies to fully connected layers as well as to convolutional layers.

SKIP CONNECTIONS

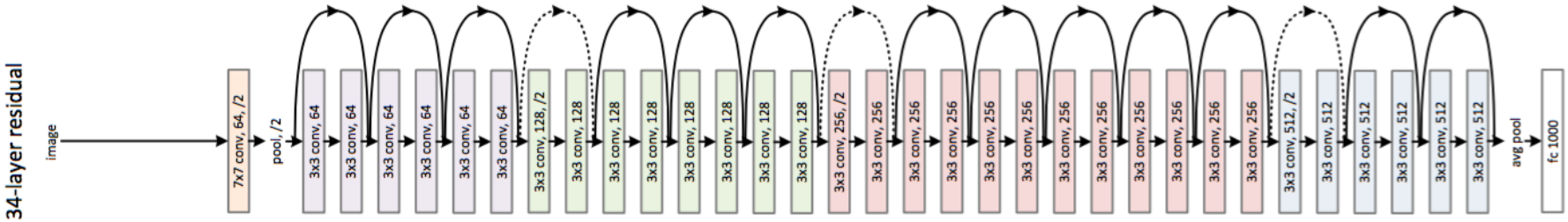


Figure: Skip connetions are the integral part of ResNet architectures [Christian Szegedy et al., 2014].

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 25.6×10^6 weights for the final dense layer.

GLOBAL AVERAGE POOLING

- 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.

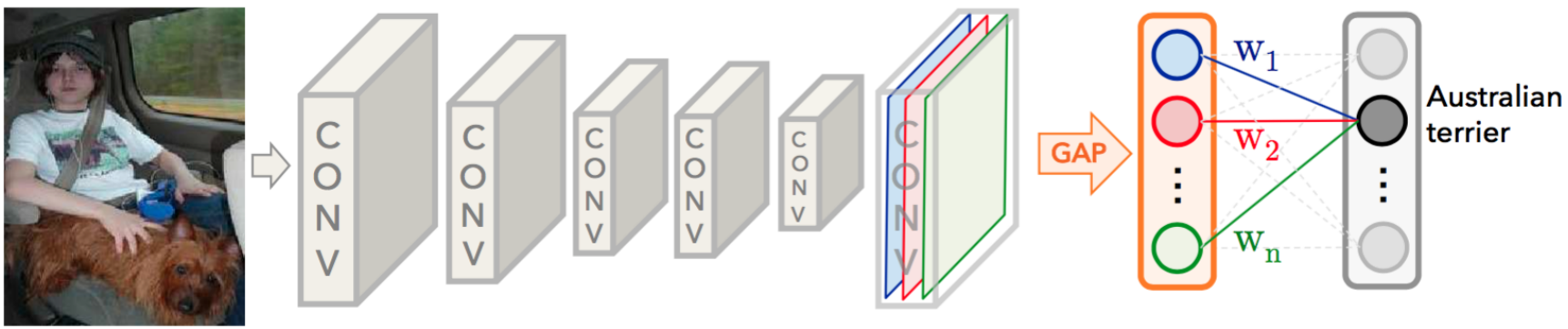


Figure: Illustration of GAP [Zhou et. al , 2016]. Each feature map representing one feature category averaged into one final vector. No pooling operations are applied throughout the net. The dimensionality of the input reduces solely due to the convolution operations.

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).

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 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

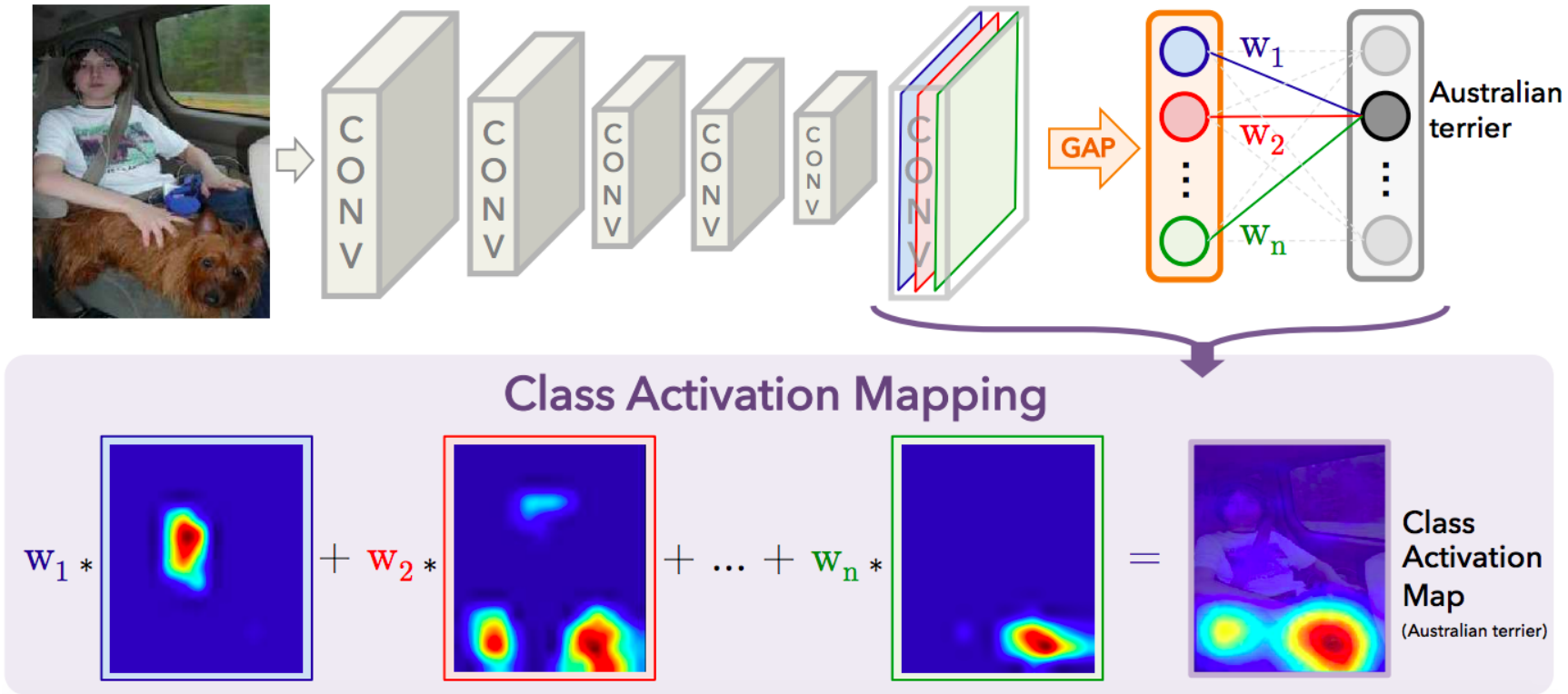


Figure: Illustration of the class activation mapping. The activated regions from the feature maps are summed up weighted by their connection strength with the final output classes and upsampled back to the dimension of the input image. No max-pooling is applied throughout the architecture, the downsampling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

REFERENCES

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

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

REFERENCES

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

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

REFERENCES

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

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

http://arxiv.org/abs/1509.01626

Augustus Odena, Vincent Dumoulin and Chris Olah (2016)

Deconvolution and Checkerboard Artifacts

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

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

Lin, Haoning and Shi, Zhenwei and Zou, Zhengxia (2017)

Maritime Semantic Labeling of Optical Remote Sensing Images with Multi-Scale Fully Convolutional Network

REFERENCES

Olaf Ronneberger, Philipp Fischer, Thomas Brox (2015)

U-Net: Convolutional Networks for Biomedical Image Segmentation

http://arxiv.org/abs/1505.04597

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, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich (2014)

Going deeper with convolutions

https://arxiv.org/abs/1409.4842

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

Deep Residual Learning for Image Recognition

https://arxiv.org/abs/1512.03385

REFERENCES

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