Deep Learning with Neural Networks

Some important CNN variants

Pablo Martínez Olmos, pamartin@ing.uc3m.es

Style transfer

Image Style Transfer Using Convolutional Neural Networks

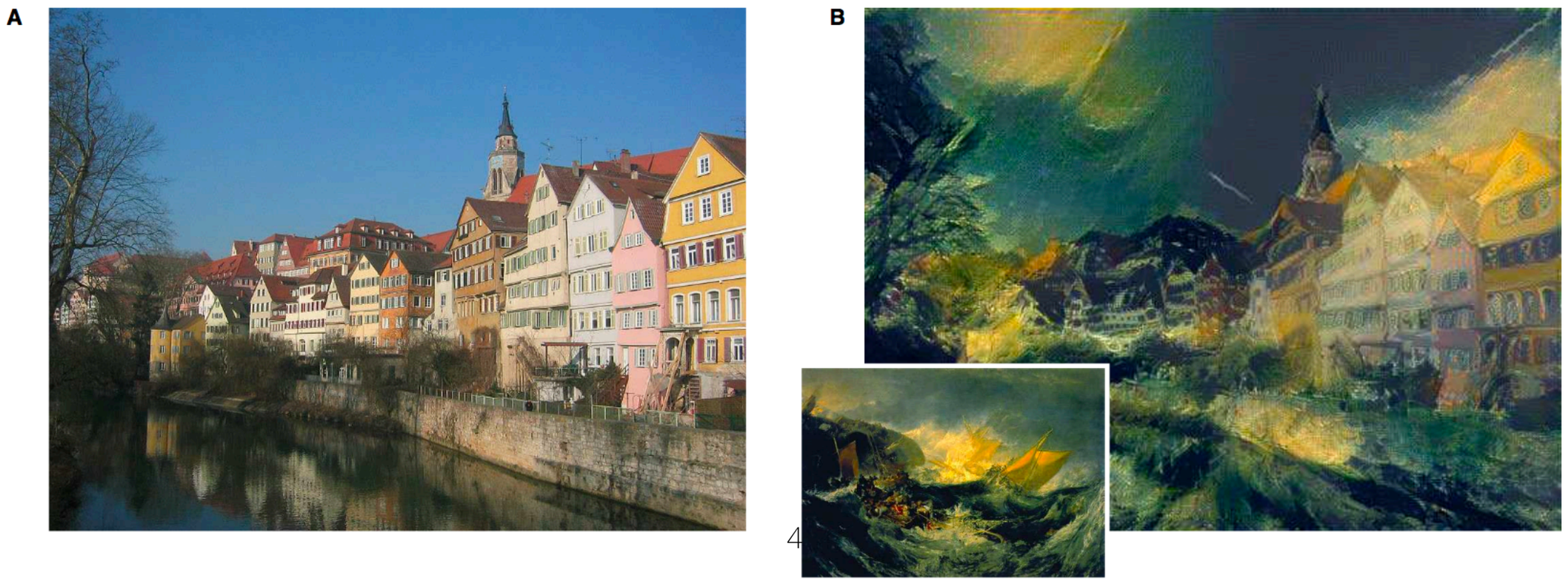
Leon A. Gatys
Centre for Integrative Neuroscience, University of Tübingen, Germany
Bernstein Center for Computational Neuroscience, Tübingen, Germany
Graduate School of Neural Information Processing, University of Tübingen, Germany
leon.gatys@bethgelab.org

Alexander S. Ecker
Centre for Integrative Neuroscience, University of Tübingen, Germany
Bernstein Center for Computational Neuroscience, Tübingen, Germany
Max Planck Institute for Biological Cybernetics, Tübingen, Germany
Baylor College of Medicine, Houston, TX, USA

Matthias Bethge
Centre for Integrative Neuroscience, University of Tübingen, Germany
Bernstein Center for Computational Neuroscience, Tübingen, Germany
Max Planck Institute for Biological Cybernetics, Tübingen, Germany

Style Transfer

- "We use image representations derived from CNNs optimised for object recognition, which make **high level image information explicit**."
- "Our algorithm can **separate and recombine the image content and style** of images"
- "The algorithm allows us to produce new images of high perceptual quality that **combine the content** of an arbitrary photograph **with the appearance** of numerous well-known artworks"
- "Our results provide new insights into the deep image representations learned by CNNs and demonstrate their **potential for high level image synthesis and manipulation**"

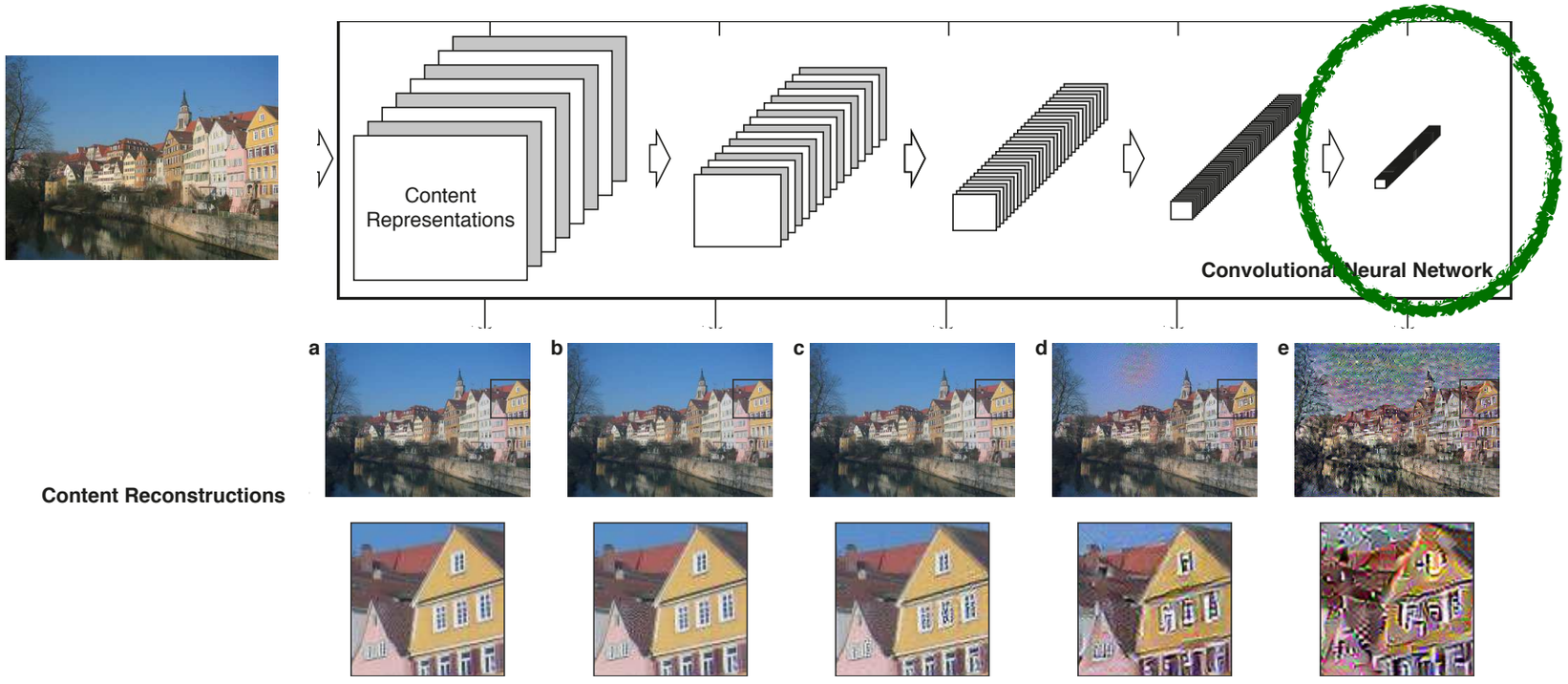


Extracting content from an image

- The input image is transformed into **representations that are increasingly sensitive** to the actual content of the image, but become relatively invariant to its precise appearance.

- **Higher layers** in the network capture the high-level content in terms of objects, but do not constrain the exact pixel values of the reconstruction very much.

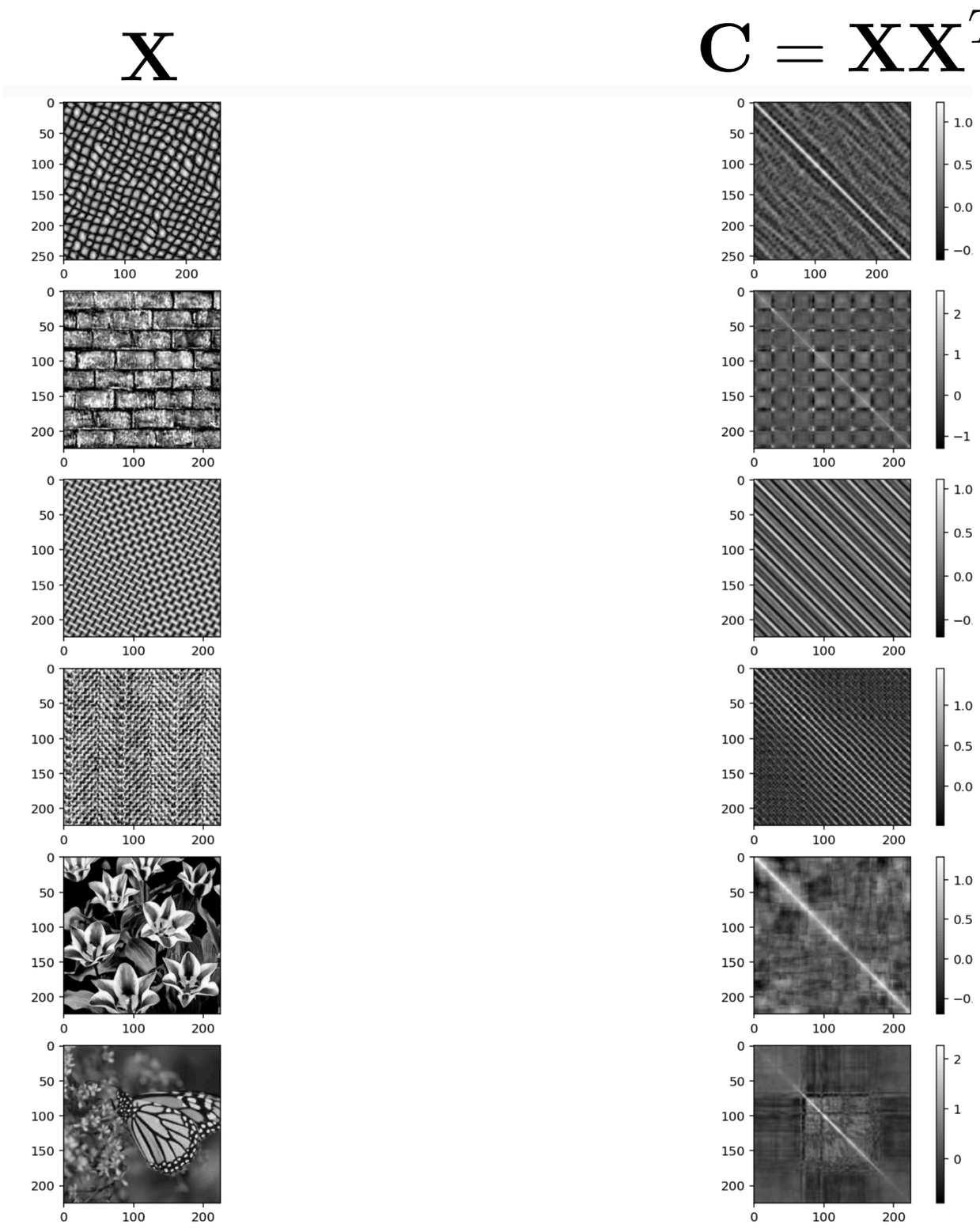
- Reconstructions from the **lower layers simply reproduce the exact pixel values of the original image**.



Textures as image correlations

- The **correlation matrix is sensitive to textures** (spatial patterns that happen all over the image).

- **Right:** normalised images and the correlation matrix.



Extracting style from an image

- Style feature space can be built on top of the filter responses in any layer of the network. It consists of the **correlations between the different filter responses**.

- Feature correlations are given by the Gram matrix G, where G^l_{ij} is the inner product between the vectorised featured maps *i* and *j* in layer *l*:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}.$$

Extracting style from an image

- Style feature space can be built on top of the filter responses in any layer of the network. It consists of the **correlations between the different filter responses**.

- Feature correlations are given by the Gram matrix G, where G^l_{ij} is the inner product between the vectorised featured maps *i* and *j* in layer *l*:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}.$$

Sum across all pixels in feature maps

Extracting style from an image

- Style feature space can be built on top of the filter responses in any layer of the network. It consists of the **correlations between the different filter responses**.

- Feature correlations are given by the Gram matrix G, where G^l_{ij} is the inner product between the vectorised featured maps *i* and *j* in layer *l*:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}.$$

Sum across all pixels in feature maps

- We obtain a stationary, multi-scale representation of the input image, which **captures its texture information but not the global arrangement**.

Extracting style from an image

- Style feature space can be built on top of the filter responses in any layer of the network. It consists of the **correlations between the different filter responses**.

- Feature correlations are given by the Gram matrix G, where G^l_{ij} is the inner product between the vectorised featured maps *i* and *j* in layer *l*:

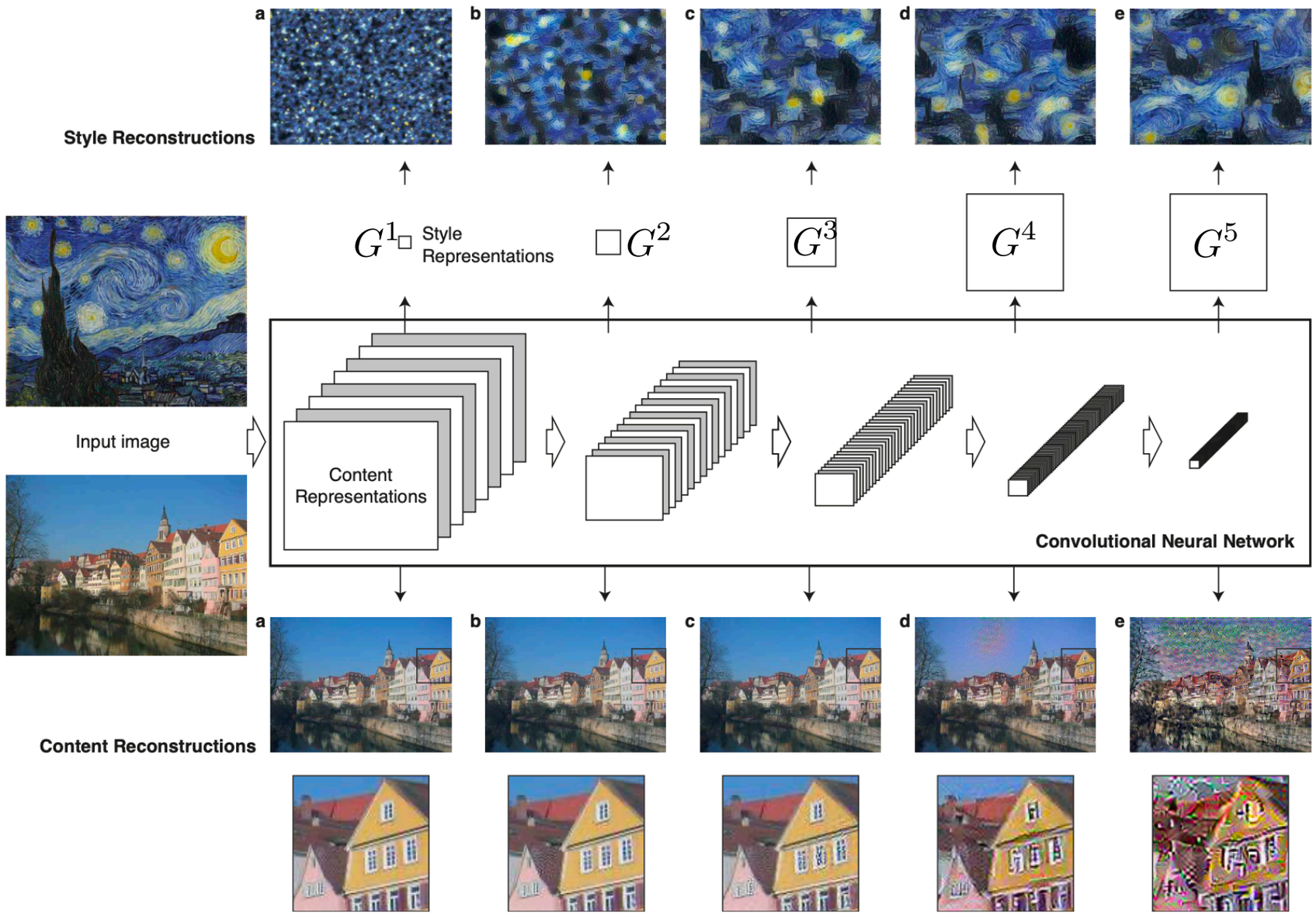
$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}.$$

Sum across all pixels in feature maps

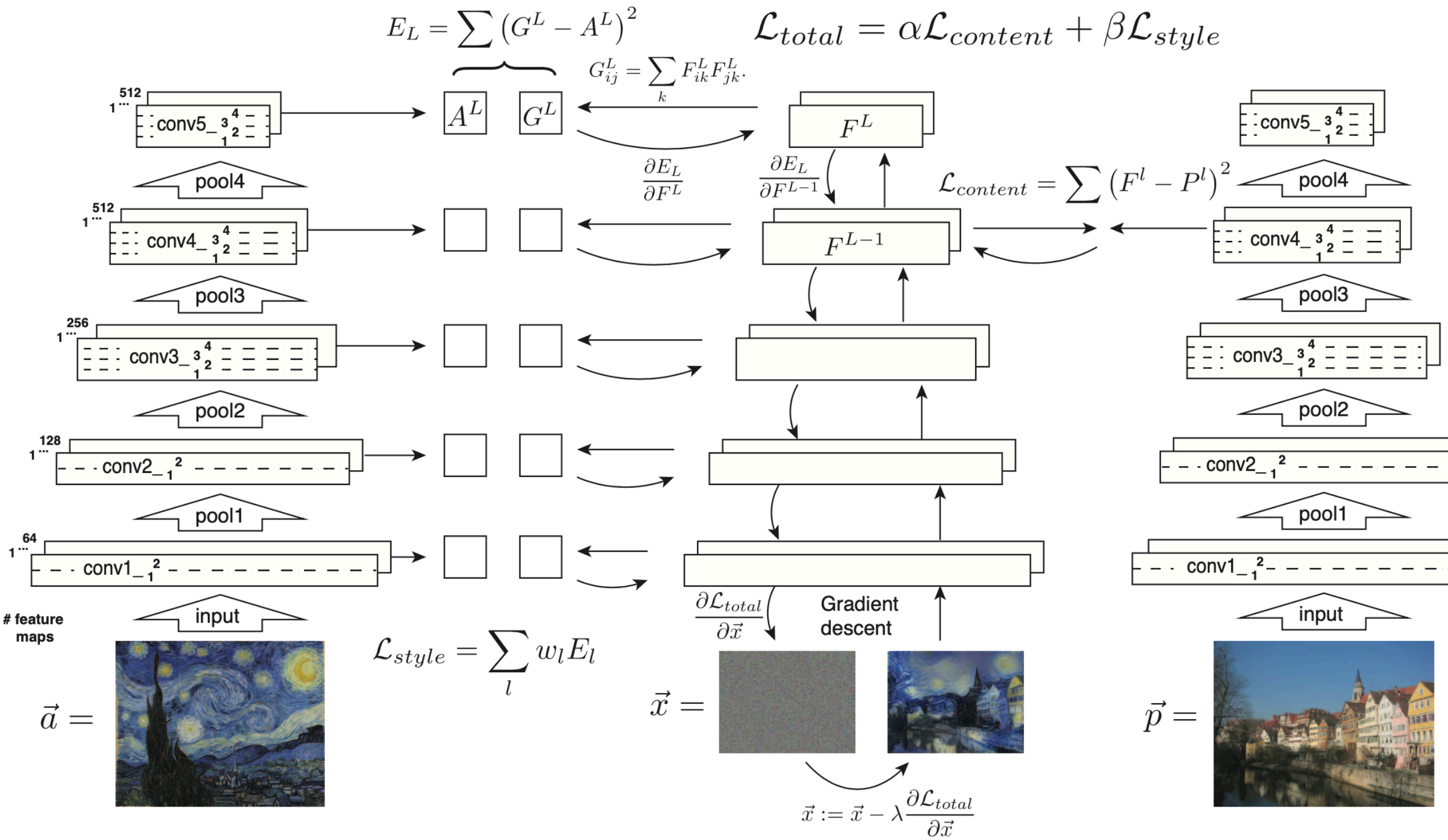
- We obtain a stationary, multi-scale representation of the input image, which **captures its texture information but not the global arrangement**.

- As with content, we can visualise the information captured by these style feature spaces built on different layers of the network by constructing an image that matches the style representation of a given input image

Extracting style and content from an image



Style transfer algorithm



Style transfer

[Github repository (one of many)]

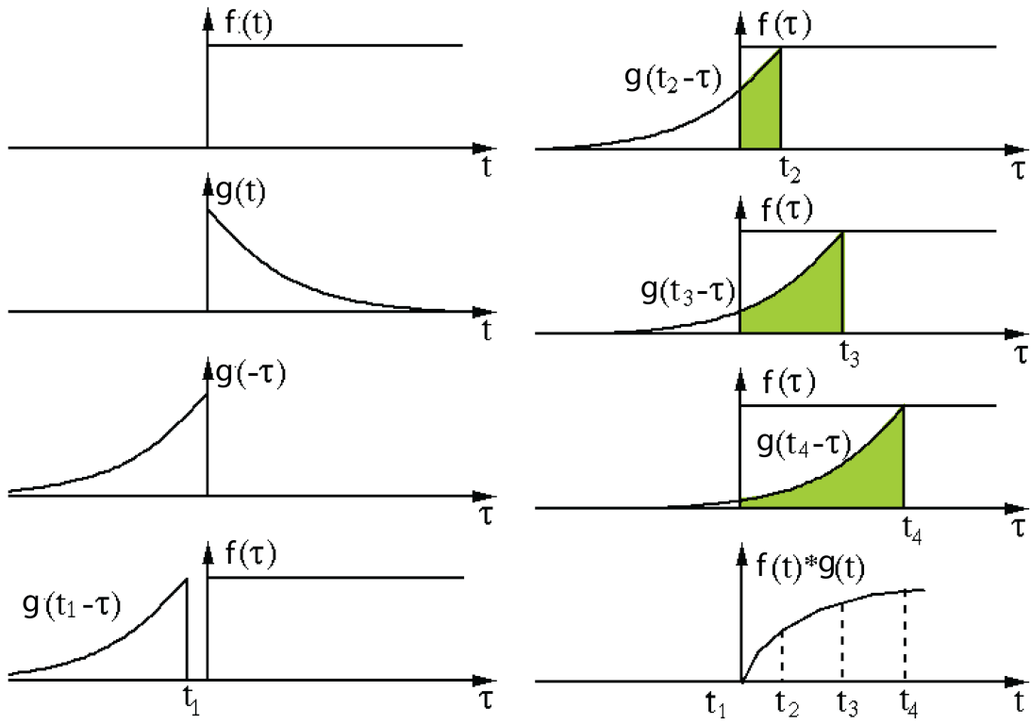
1D CNNs

1D CNNs

- CNNs perform well on computer vision problems, due to their ability to operate convolutionally **extract features from local input patches**.
- The same properties that make CNNs excel at computer vision also make them **highly relevant to sequence processing**
- 1D CNNs, typically used with **dilated kernels**, have been used with great success for audio generation and machine translation
- They can offer a **fast alternative to RNNs** for simple tasks such as text classification and timeseries forecasting

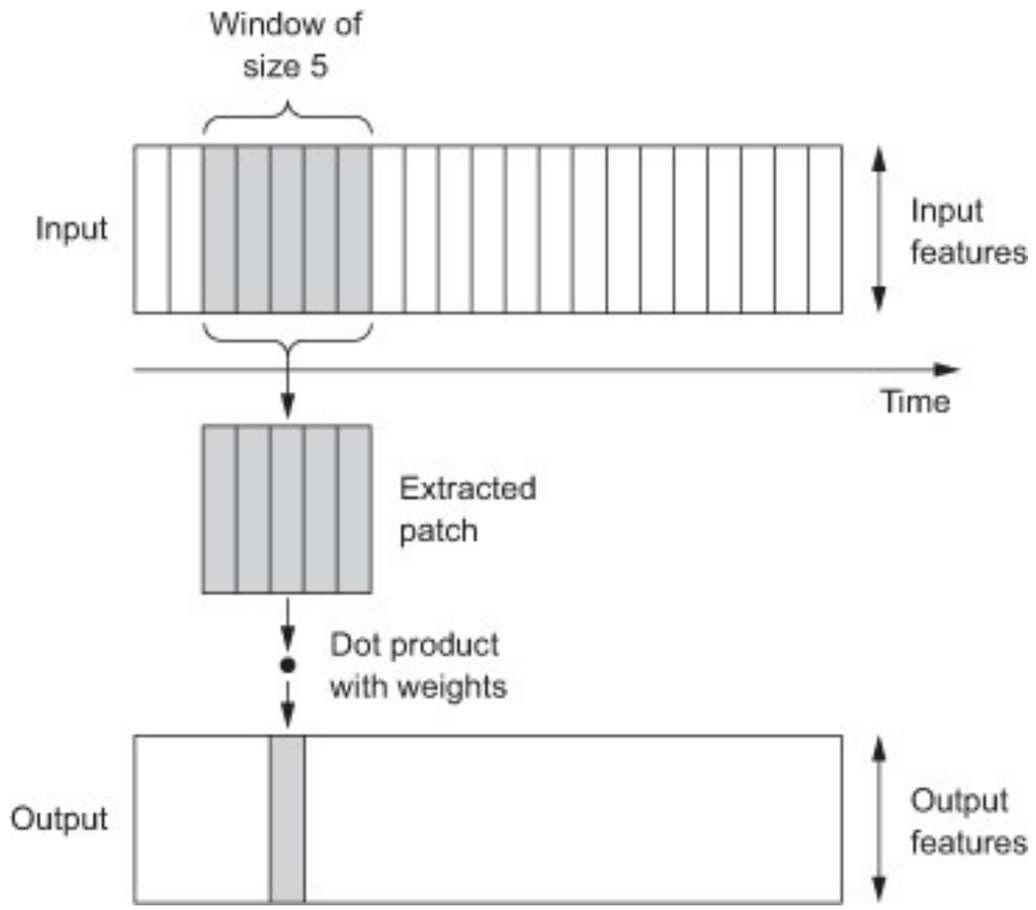
$$y(t) = f(t) * g(t) = \int f(\tau)g(t-\tau)d\tau$$

$$y[n] = f[n] * g[n] = \sum_{k=-\infty}^{\infty} f[k]g[n-k]$$



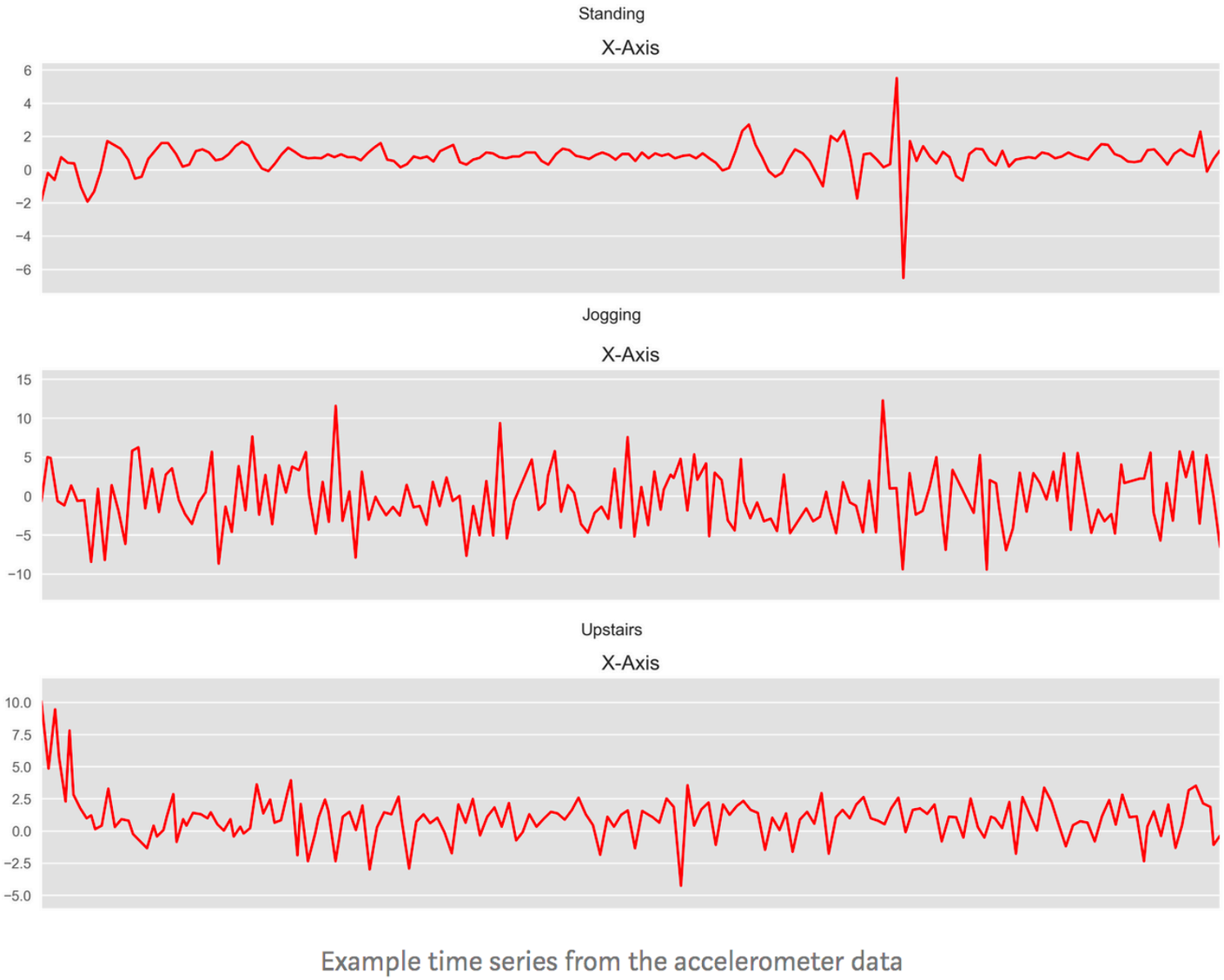
1D CNNs

- Combine finite-length filters with 1D patches (subsequences) from sequences
- Each output timestep is obtained from a temporal patch in the input sequence
- Convolution layers can recognize local patterns in a sequence
- 1D pooling layers, batch normalization, ….



1D CNNs. An Example

- Example borrowed from **this blog** (also posted in Aula Global)



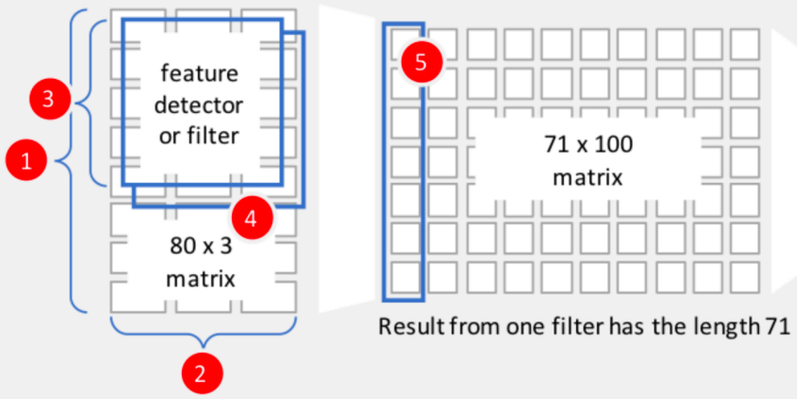
Example time series from the accelerometer data

Predict the type of activity a user is performing (“Walking”, “Jogging” or “Standing”, …)

There are 6 activities in total.

depicted above. The following picture provides a high level overview of the

Conv Layer

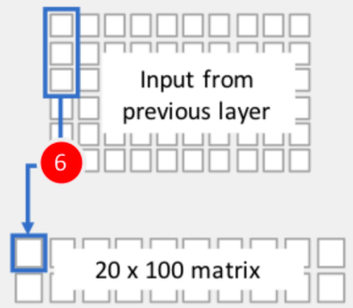


1D convolutional layer with 100 feature detectors (a kernel size of 10 applied to each feature detector). With a height of 80 and a kernel size of 10, the window will slide through the data for 71 steps (80-10+1). Therefore resulting in a 71 x 100 output matrix. This initial layer will learn basic features.

Conv Layer

Another 1D convolutional layer with 100 filters (a kernel size of 10). Having a second convolutional layer prior pooling allows the model to learn more complex features.

Pooling



The pooling layer slides a window of height 3 across our data and replaces it with the max value, therefore discarding 66% of our values from the previous layer. Results in a 20 x 100 matrix.

Conv Layer & Pooling

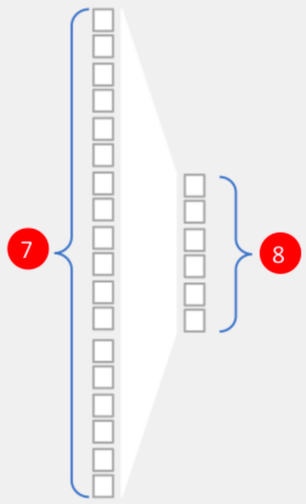
Additional set of 1D convolutional layers. Same concept applies as for the first two convolutional layers.

Another pooling layer (average instead of max pooling window in this case).

Dropout

To further reduce the likelihood of overfitting, a drop layer is added at the end. A fraction of the input units is randomly set to 0 (also called dropout rate). In our example, the rate is set to 0.5, therefore 50% of the neurons will drop out. The different neurons are less likely to influence each other since any of them might drop out at random. The network becomes less sensitive to smaller variations in the data.

Fully Connected



1 Height

The length of one data set which is fed into the network. In our case, the length is 80. Is also referred to as the input height.

2 Width

The width of the data set which is fed into the network. In our case, the width is 3 (x, y and z accelerometer reading). Is also referred to as the depth.

3 Kernel Size

This is the size / height of the sliding window that convolves across the data. Also referred to as kernel size or filter length. 10 entries in one window in our case.

4 Filters

Defines how many sliding windows will run through the data. In our case 100, therefore 100 different features can be detected in this layer. Also called feature detectors.

5 Output

The kernel size and the height of the original data determines the amount of neurons in the output.

6 Max Pooling

This step will prevent overfitting of the learned features by taking the max value of multiple features. Again, a sliding window approach is used.

7 Dense

The last layer is a fully connected layer (Dense in Keras). It uses a Softmax activation function to produce a probability distribution over the 6 output classes.

8 Final Output

The final output layer consists of 6 neurons (one for each label “Walking“, „Jogging“, etc.) including its probability. All probabilities add up to 1.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_45 (Reshape)         (None, 80, 3)             0
_________________________________________________________________
conv1d_145 (Conv1D)          (None, 71, 100)           3100
_________________________________________________________________
conv1d_146 (Conv1D)          (None, 62, 100)           100100
_________________________________________________________________
max_pooling1d_39 (MaxPooling (None, 20, 100)           0
_________________________________________________________________
conv1d_147 (Conv1D)          (None, 11, 160)           160160
_________________________________________________________________
conv1d_148 (Conv1D)          (None, 2, 160)            256160
_________________________________________________________________
global_average_pooling1d_29  (None, 160)               0
_________________________________________________________________
dropout_29 (Dropout)         (None, 160)               0
_________________________________________________________________
dense_29 (Dense)             (None, 6)                 966
=================================================================
Total params: 520,486
Trainable params: 520,486
Non-trainable params: 0
_________________________________________________________________
None
```

```
Accuracy on test data: 0.92

Loss on test data: 0.39
```

Graph CNNs

Published as a conference paper at ICLR 2017

SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

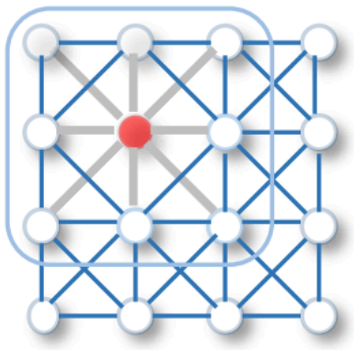
Thomas N. Kipf
University of Amsterdam
T.N.Kipf@uva.nl

Max Welling
University of Amsterdam
Canadian Institute for Advanced Research (CIFAR)
M.Welling@uva.nl

uc3m

Graph CNNs (GCN)

- A graph convolutional network (GCN) is a neural network that operates on graphs.
- Given a graph *G* = (V, E), a GCN takes as input
    1. An input feature matrix **X** of dimension N x F , where N is the number of nodes and F is the number of features per node
    2. An N x N matrix representation of the graph structure (adjacency matrix **A**).
- The first hidden layer in the GCN process each node in *G* according to **X** and **A**, **H**=f(**X,A**)
- **H** is a node feature matrix NxF' where each row is a new feature representation of the node



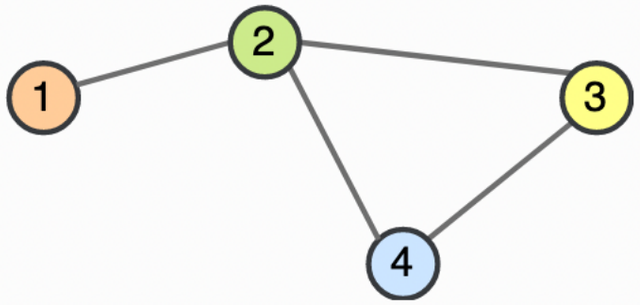
(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

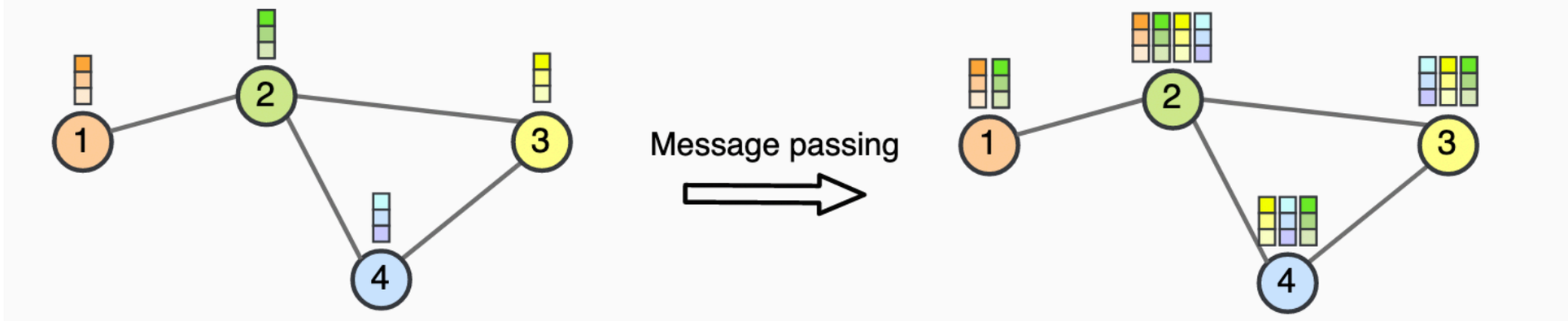
Fig. 1: 2D Convolution vs. Graph Convolution.

- At each layer, these features are aggregated to form the next layer's features using a propagation rule *f*.

Graph CNNs (GCN)



$$\mathbf{x}_v^{(\ell+1)} = \mathbf{W}^{(\ell+1)} \sum_{w \in \mathcal{N}(v) \cup \{v\}} \frac{1}{c_{w,v}} \cdot \mathbf{x}_w^{(\ell)}$$



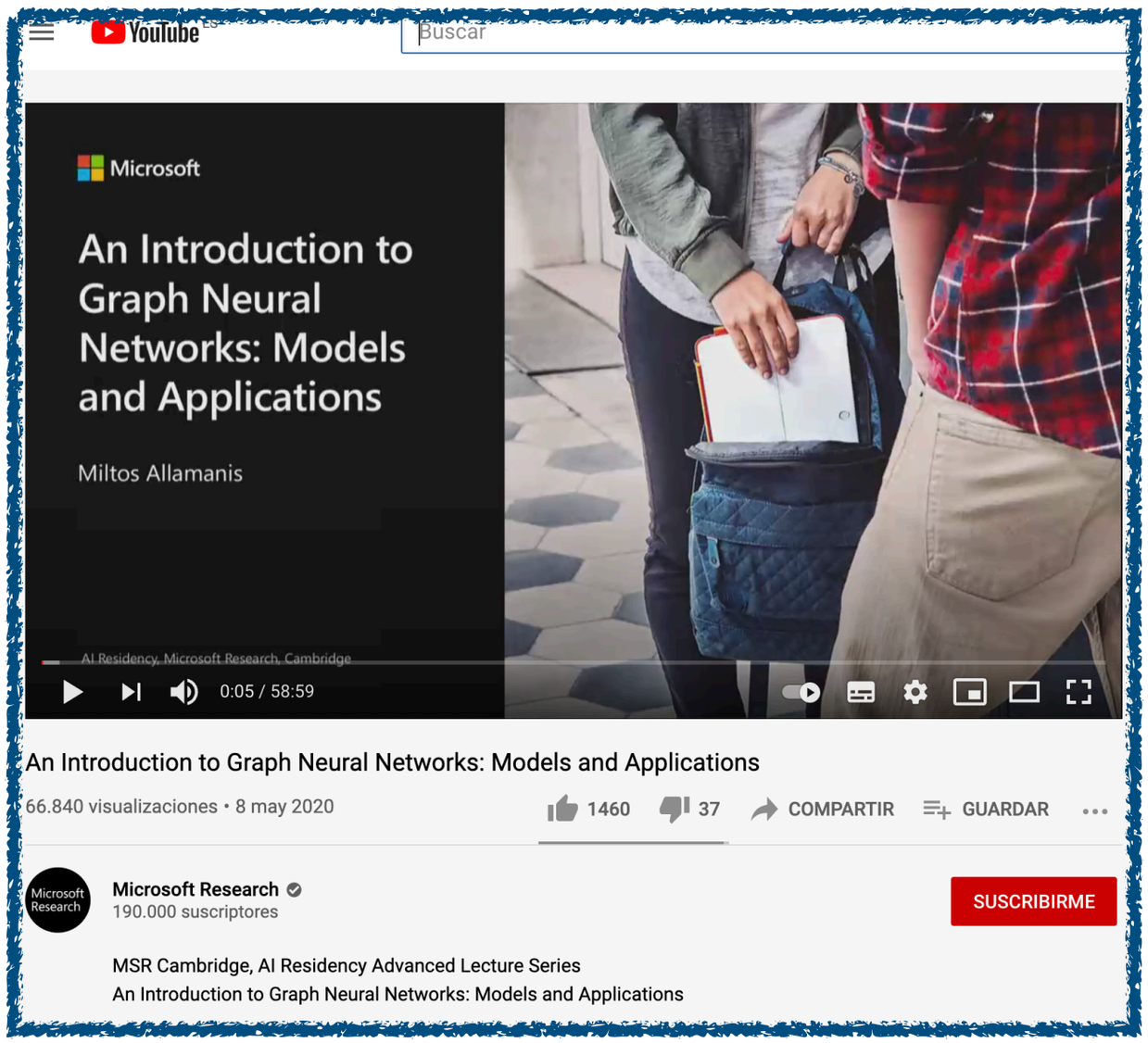
PyTorch geometric

Graph Neural Networks

Graph CNNs (GCN)

A Comprehensive Survey on Graph Neural Networks

Zonghan Wu, Shirui Pan, *Member, IEEE*, Fengwen Chen, Guodong Long, Chengqi Zhang, *Senior Member, IEEE*, and Philip S. Yu, *Life Fellow, IEEE*



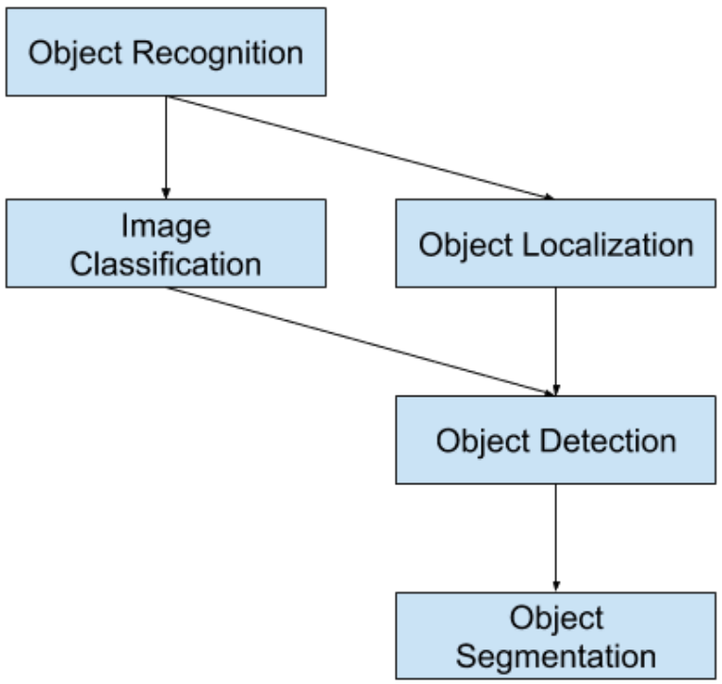
Learning Convolutional Neural Networks for Graphs

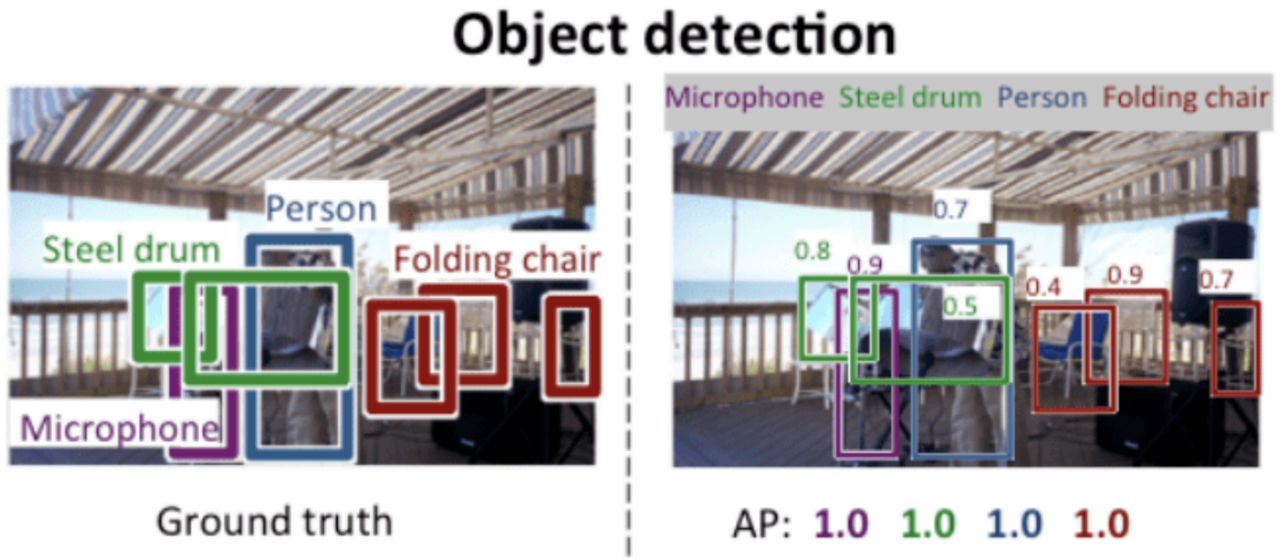
Mathias Niepert MATHIAS.NIEPERT@NECLAB.EU
Mohamed Ahmed MOHAMED.AHMED@NECLAB.EU
Konstantin Kutzkov KONSTANTIN.KUTZKOV@NECLAB.EU
NEC Labs Europe, Heidelberg, Germany

Object detection with Region CNNs (RCNNs)

uc3m

Object Recognition





- **Region-based CNNs** (R-CNN) and extensions such as Fast R-CNN and YOLO (you only look once) are the most dominant approach for **object localisation and recognition**.

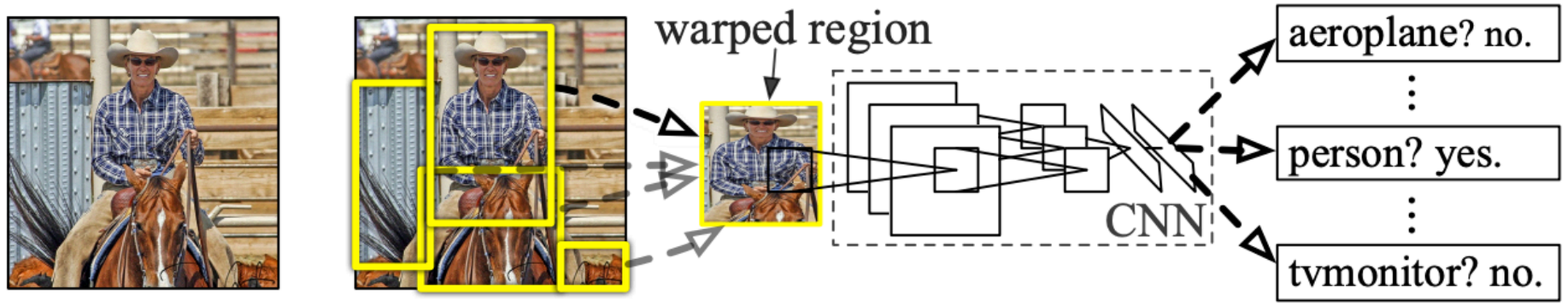
R-CNN

Rich feature hierarchies for accurate object detection and semantic segmentation
Tech report (v5)

Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik
UC Berkeley
{rbg,jdonahue,trevor,malik}@eecs.berkeley.edu

- Locate regions of interest
- Classify using a pre-trained CNN for object clarification

R-CNN: *Regions with CNN features*



1. Input image | **2**. Extract region proposals (~2k) | **3**. Compute CNN features | **4**. Classify regions

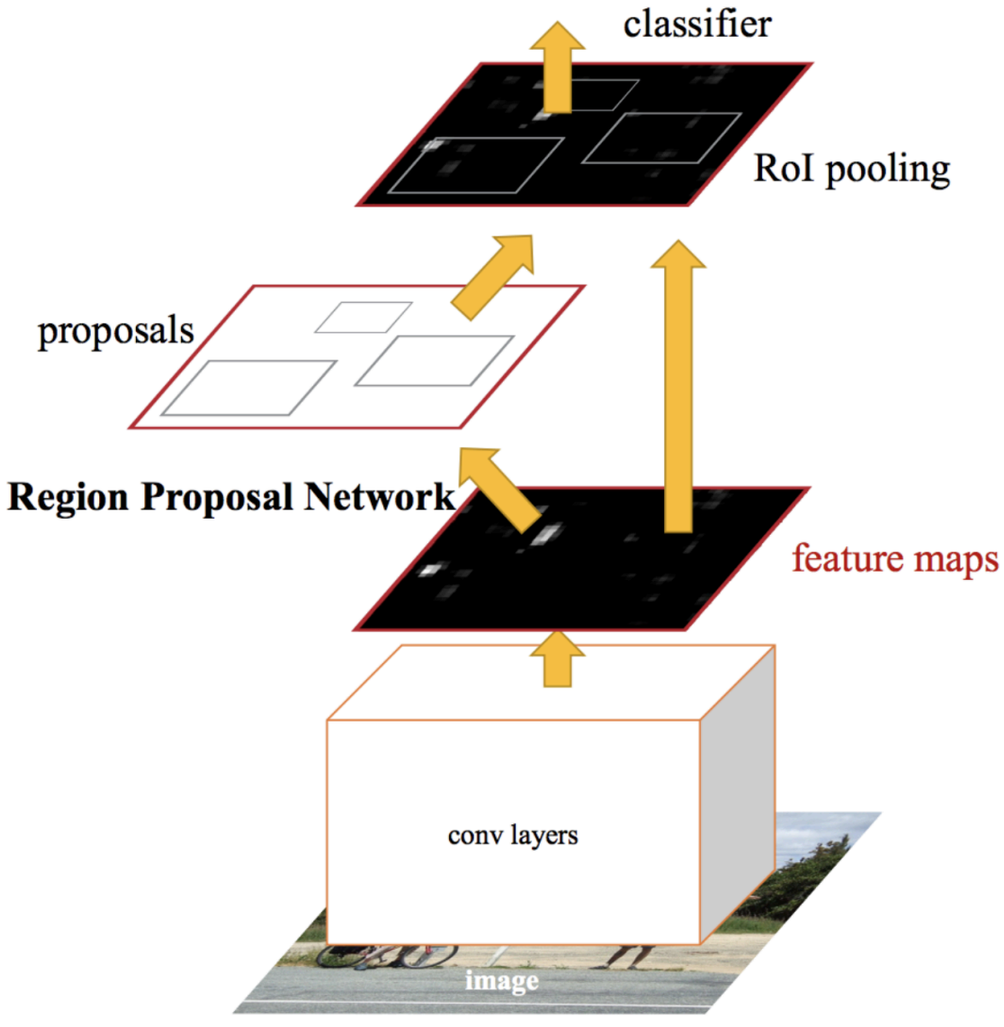
R-CNN: obtain region proposals

Selective Search for Object Recognition

J. R. R. Uijlings · K. E. A. van de Sande · T. Gevers · A. W. M. Smeulders

- One of the simplest
- First, **segment** different areas in the image
- Tons of algorithms for **multi-scale image segmentation** (thresholding, K-means, histogram based, edge detection)
- Second, define “squared” regions based on segmented areas.

Fast/Faster R-CNN: obtain region proposals



Faster R-CNN

- Same idea, but you define regions proposals over the feature maps
- You only have to filter the input image once
- In Faster R-CNN they replace selective search by **regions learned by the network itself**

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

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

Image segmentation using CNNs

Image Segmentation Using Deep Learning: A Survey

Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos

- Most powerful methods are based on **encoder-decoder networks**
- We will explore this in the block on **unsupervised deep learning**

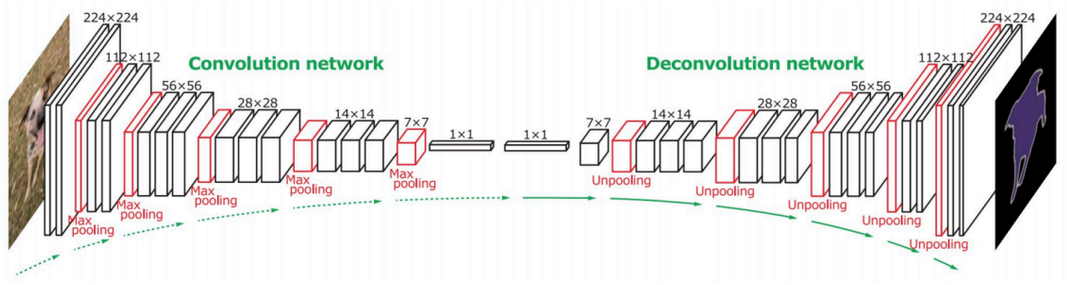


Fig. 11. Deconvolutional semantic segmentation. Following a convolution network based on the VGG 16-layer net, is a multi-layer deconvolution network to generate the accurate segmentation map. From [42].

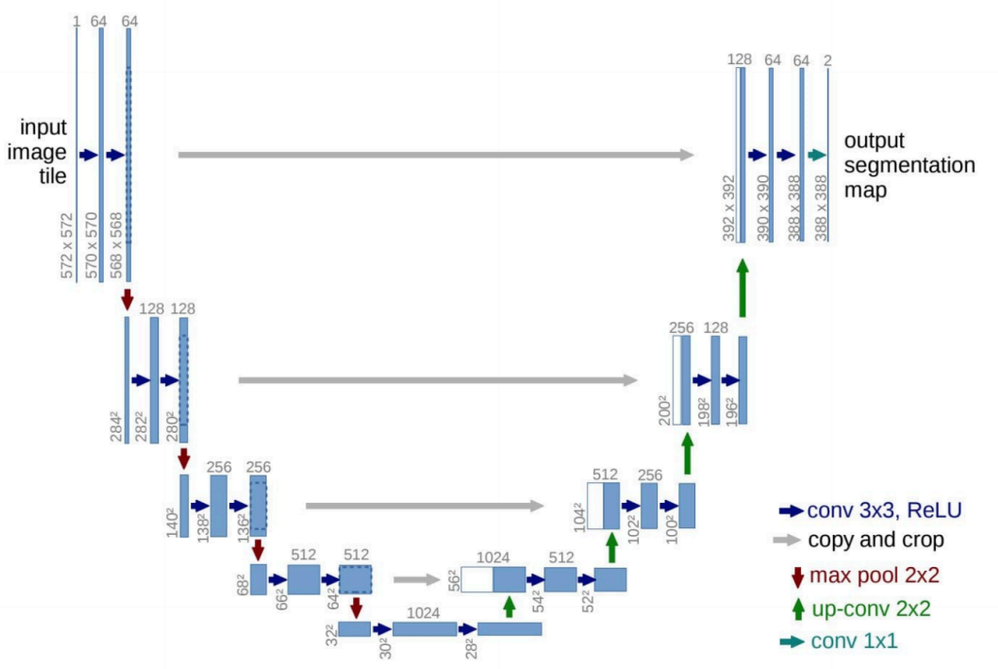


Fig. 14. The U-net model. The blue boxes denote feature map blocks with their indicated shapes. From [49].