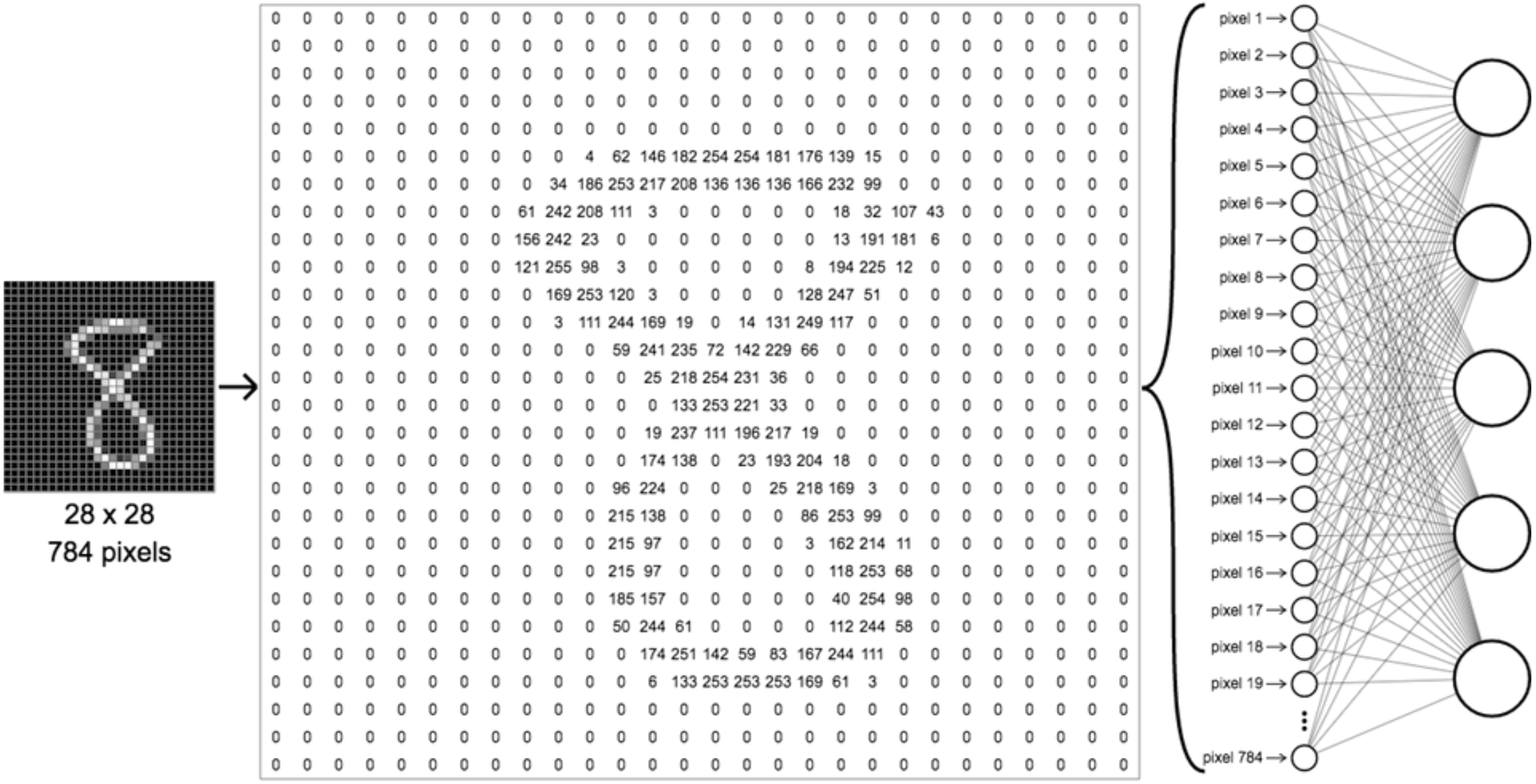
Deep Learning

Convolutional Neural Networks

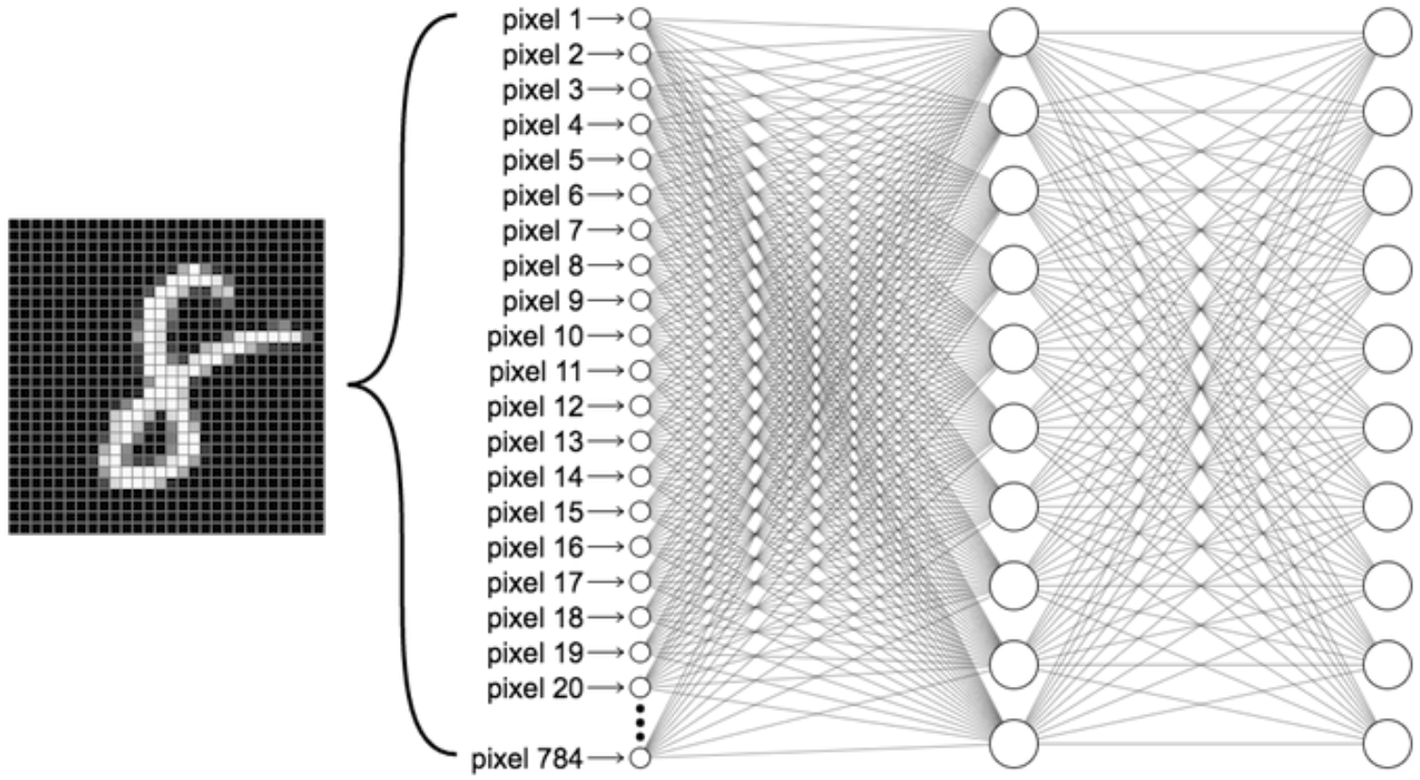
Images and FC neural networks



Flattening operation

https://mc.ai/what-is-deep-learning/

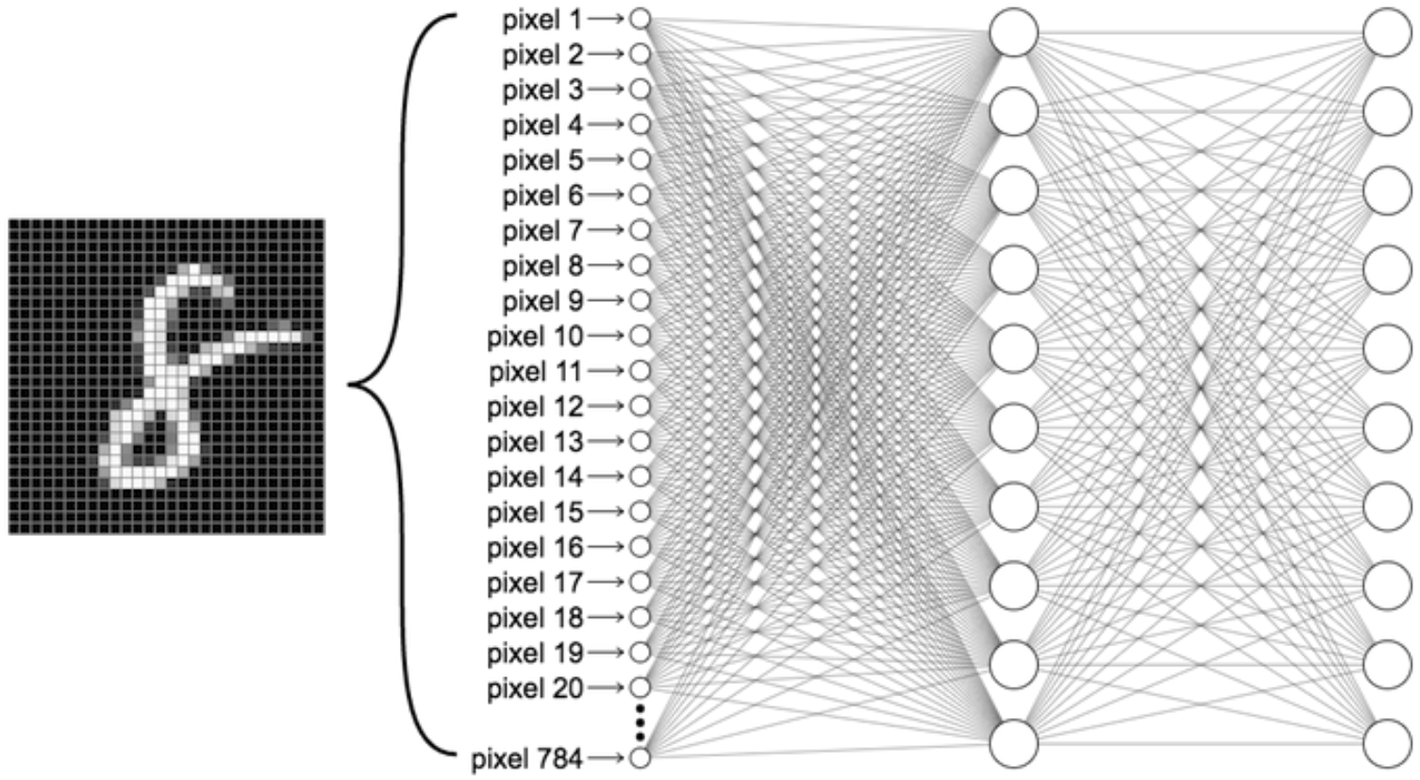
Images and FC neural networks



https://www.google.com/url?sa=i&url=https%3A%2F%2Fml4a.github.io%2Fml4a%2Flooking_inside_neural_nets%2F&psig=AOvVaw0UknUai0JapuIndBrY_BpX&ust=1604588386364000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCOiB-NmT6ewCFQAAAAAdAAAAABAD

Say we have a dataset of one-megapixel photographs we need to classify. This means that each input to the network has one million dimensions.
Even an aggressive reduction to one thousand hidden dimensions would require a fully-connected layer characterized by $10^6 \times 10^3 = 10^9$ parameters.
Moreover, learning a classifier by fitting so many parameters might require collecting an enormous dataset.

Images and FC neural networks



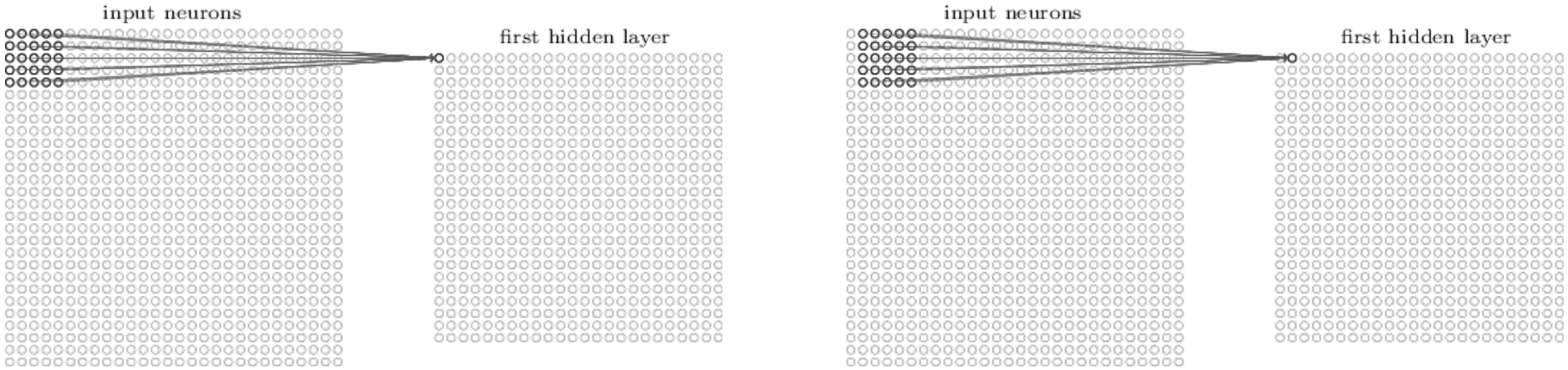
https://www.google.com/url?sa=i&url=https%3A%2F%2Fml4a.github.io%2Fml4a%2Flooking_inside_neural_nets%2F&psig=AOvVaw0UknUai0JapuIndBrY_BpX&ust=1604588386364000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCOiB-NmT6ewCFQAAAAAdAAAAABAD

Convolutional neural networks (CNNs) are one creative way that machine learning has embraced for exploiting some of the known structure in natural images in order to **minimize the number of parameters** and also to build interesting **invariances in the model**.

Minimizing the number of parameters: LRF's

One possible solution for this problem is to consider **local receptive fields** (LRF) (an inspiration that comes from neuroscience).

A local receptive field is a standard MLP with units that have a local view of the image: its input is a submatrix $M \times M$ of the original $N \times N$, where $M << N$.



No flattening operation!

MNIST image size= 28 x 28 pixels = 784 dimensions

A FC layer with 100 units has (784 x 100 + 100) parameters = 78500 parameters

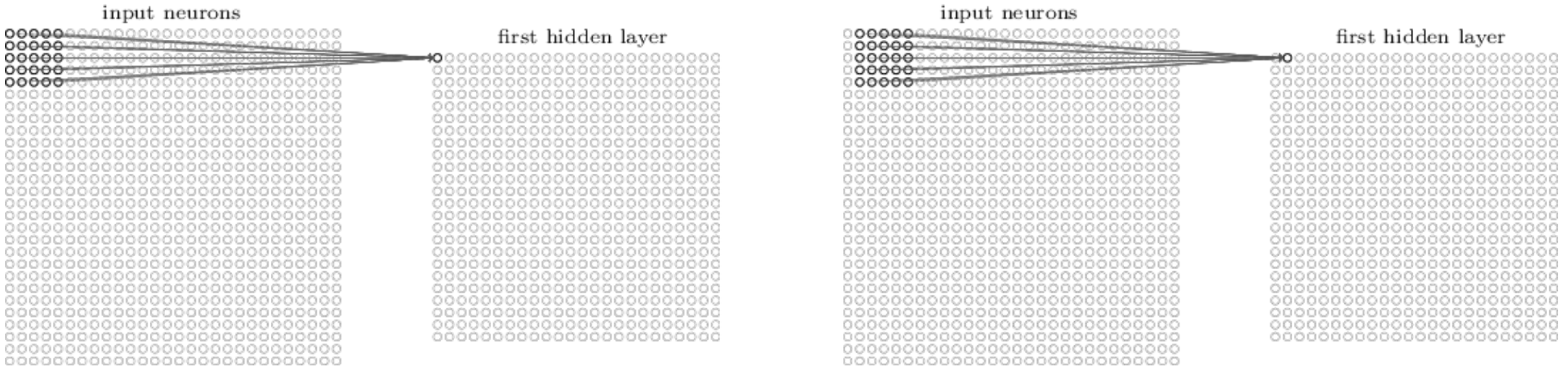
This LRF layer has 25 x 24 x 24 + (24 x 24) parameters = 14976 parameters

http://neuralnetworksanddeeplearning.com/chap6.html

Minimizing the number of parameters: LRF's

One possible solution for this problem is to consider **local receptive fields** (LRF) (an inspiration that comes from neuroscience).

A local receptive field is a standard MLP with a local view of the image: its input is a submatrix $M \times M$ of the original $N \times N$, where $M << N$.



Original image = 28 x 28 pixels

This LRF layer has 14976 parameters

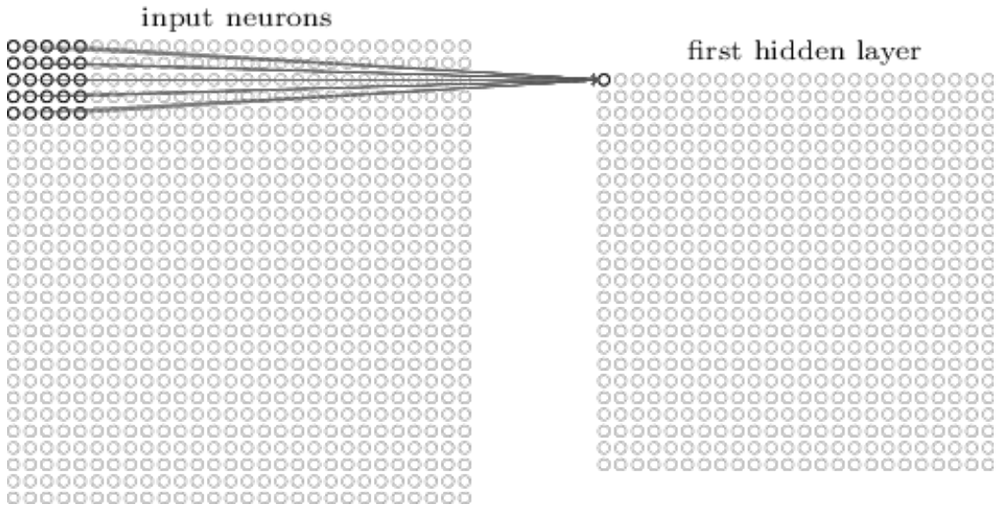
We have shown the local receptive field being moved by one pixel at a time, but this can be redundant in the case of natural images. In fact, sometimes a different **stride** length is used. For instance, we might move the local receptive field 2 pixels to the right (or down), in which case we'd say a stride length of 2 is used. This also **reduces the number units of the first layer** and consequently the number of parameters.

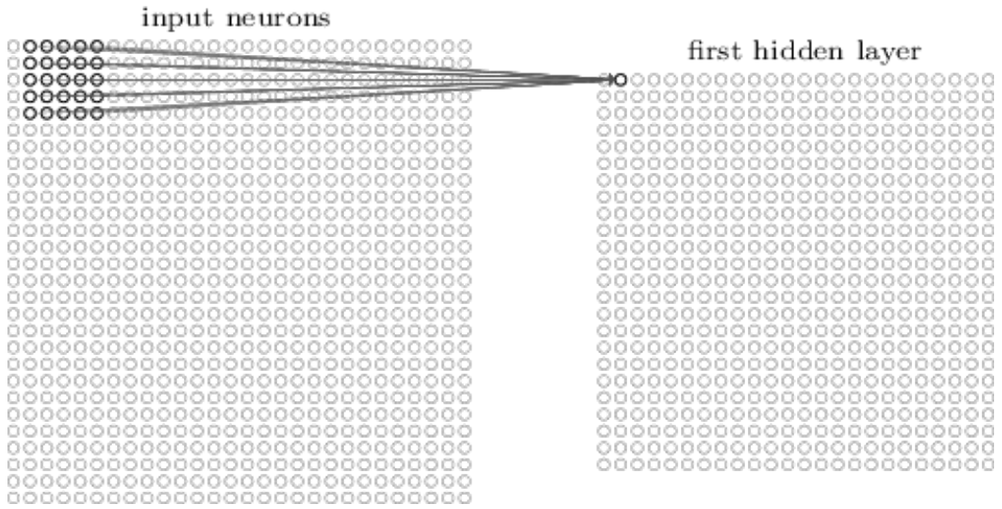
http://neuralnetworksanddeeplearning.com/chap6.html

Minimizing the number of parameters: LRF's

One possible solution for this problem is to consider **local receptive fields** (LRF) (an inspiration that comes from neuroscience).

A local receptive field is a standard MLP with a local view of the image: its input is a submatrix $M \times M$ of the original $N \times N$, where $M << N$.





Original image = 28 x 28 pixels

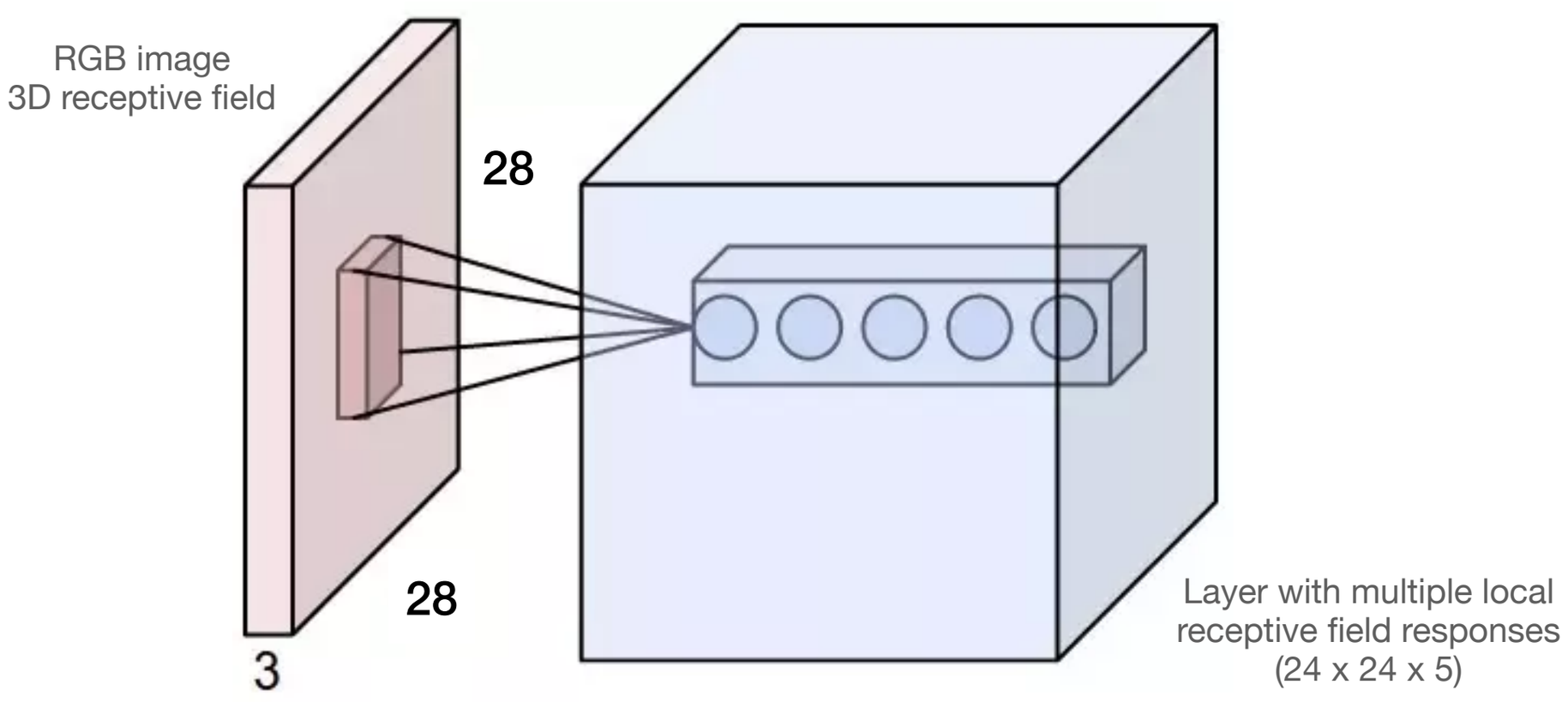
This LRF layer has 25 x 24 x 24 + (24 x 24) parameters = 14976 parameters

Note that if we have a 28×28 input image, and 5×5 local receptive fields, then there will be 24×24 neurons in the hidden layer. This is because we can only move the local receptive field 23 neurons across (or 23 neurons down), before colliding with the right-hand side (or bottom) of the input image.

http://neuralnetworksanddeeplearning.com/chap6.html

Minimizing the number of parameters: LRF's

Additionally, to have more capacity, we can compute **several local receptive fields** per pixel and also to consider **3D local receptive fields** for color images:



number of receptive fields per pixel

This LRF layer has $\underline{25 \times 3} \times \underline{5 \times 24 \times 24} + \underline{(24 \times 24 \times 5)} = 218880$ parameters

parameters per receptive field

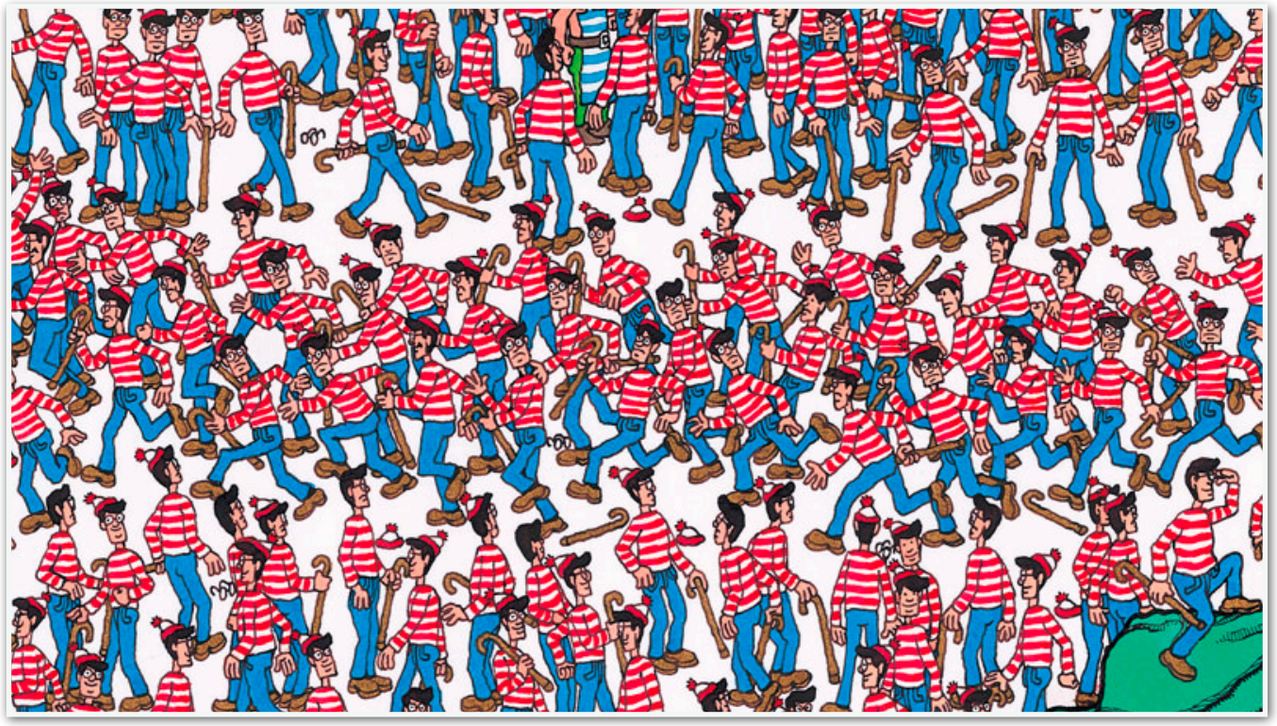
number of units in the first layer

biases

Froom LRF's to convolutions

Using LRF is like decomposing the problem of image analysis in a set of local experts that are specialized in certain parts of the image. The outcome of these specialists can be integrated at a high level of the model by **stacking** layers. This can be useful for some applications.

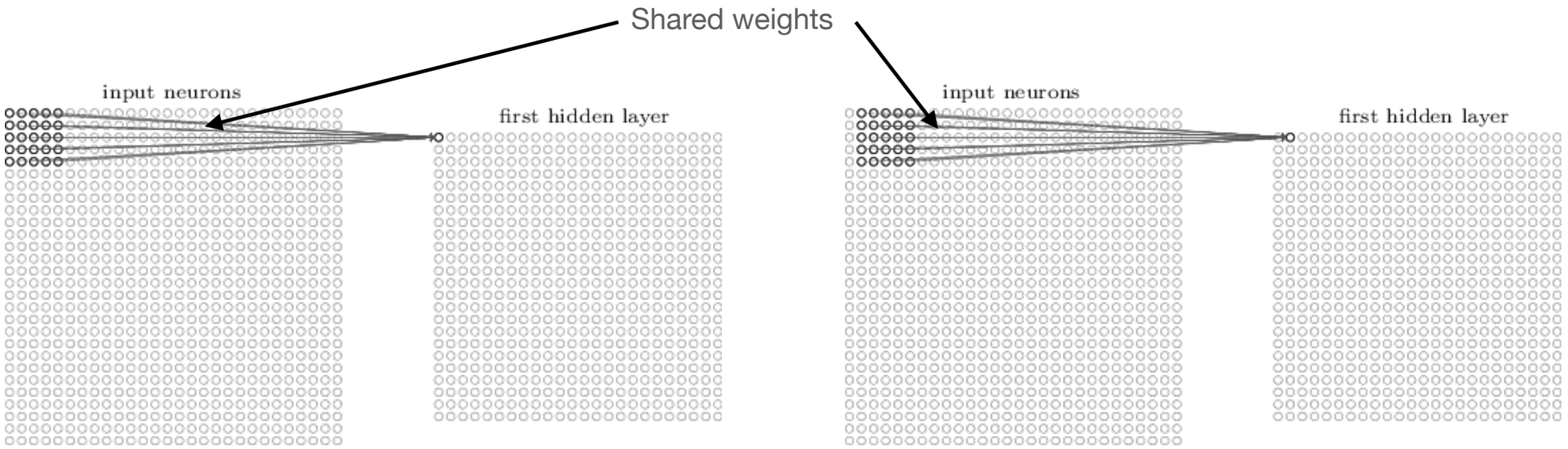
But in natural images, an object may appear at any image coordinates!



In this case LRF are not suited for this task because in order to learn a detector we should provide the model with a lot of examples in order to consider all possible object localizations!

Froom LRF's to convolutional layers

But, what about **sharing** LRF weights (and using several receptive fields per pixel)?



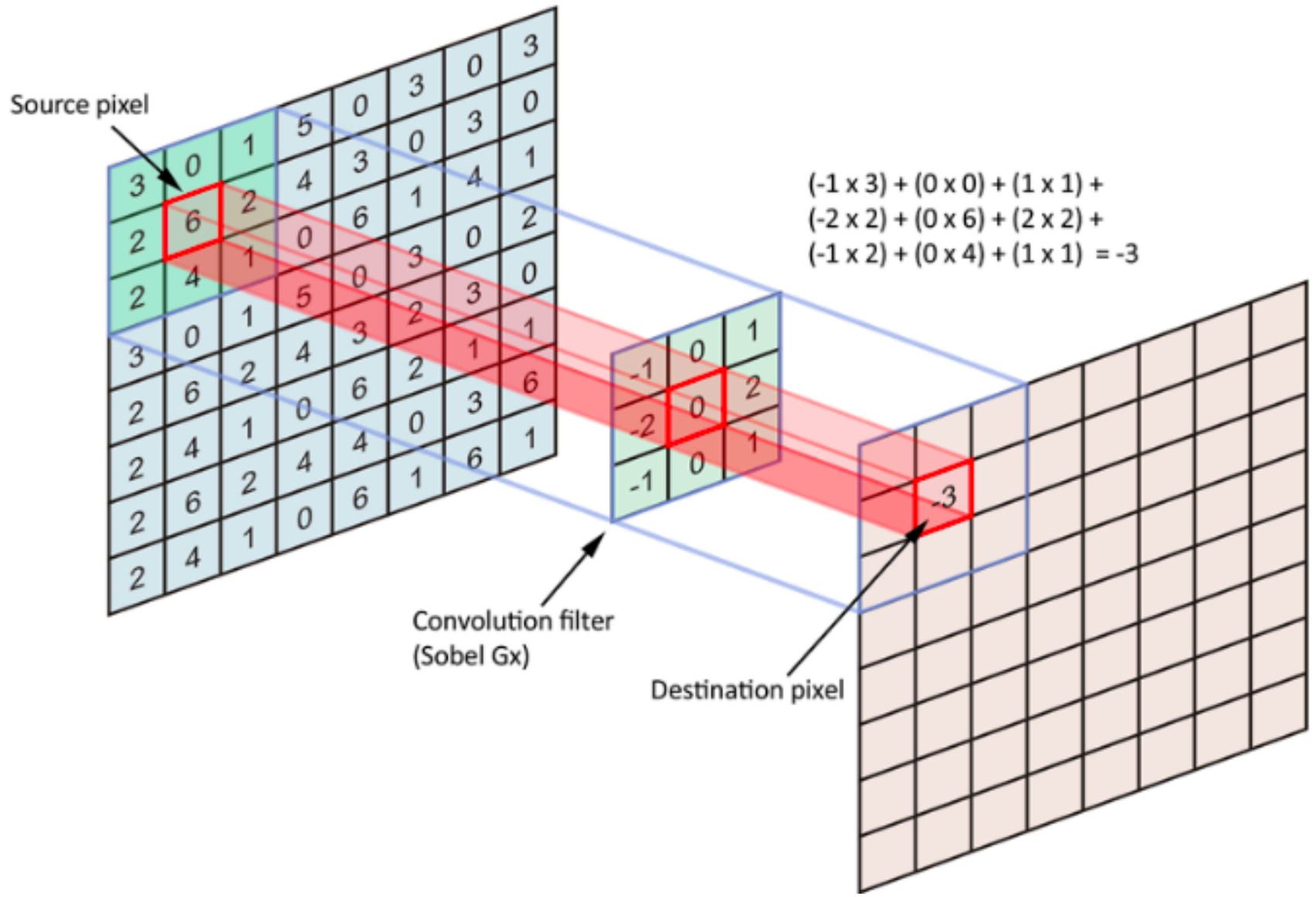
This can be expressed as a **convolution**. Now, for the (j, k) hidden unit, the output is of "shared" a receptive field can be expressed as:

$$\sigma\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}\right)$$

where b is the shared value for the bias, $w_{l,m}$ is a 5x5 array of shared weights and $a_{x,y}$ represents the input activation at position x, y .

Convolutional layers

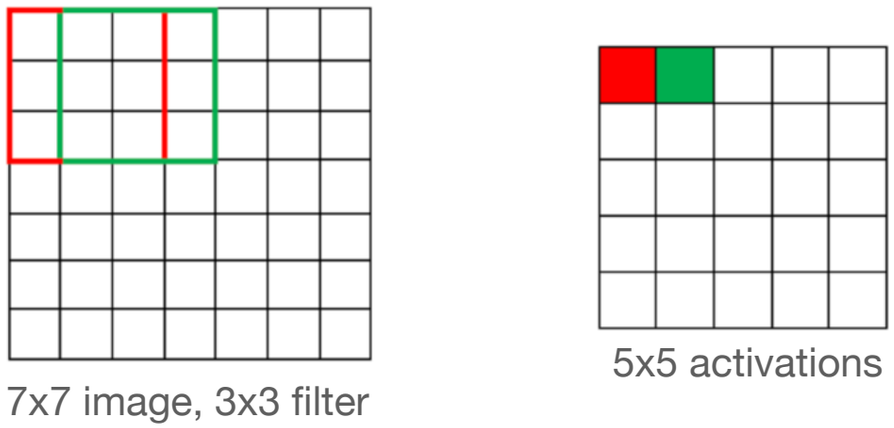
The term **convolution** refers to the mathematical combination of two functions to produce a third function. It merges two sets of information.



https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Convolutional layers

For a gray scale (n x n) image and (f x f) filter/kernel/LRF, the dimensions of the image resulting from a convolution operation is (n – f + 1) x (n – f + 1).



Thus, the image shrinks every time a convolution operation is performed. This places an upper **limit to the number of times such an operation could be performed before the image reduces to nothing** thereby precluding us from building deeper networks.

To overcome these problems, we use **padding**. Padding is simply a process of adding “dimensions” to our input images so as to avoid the problems mentioned above.

Convolutional layers

Input: 6x6 image

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

0-padding

*

3x3 Filter

1	0	-1
1	0	-1
1	0	-1

=

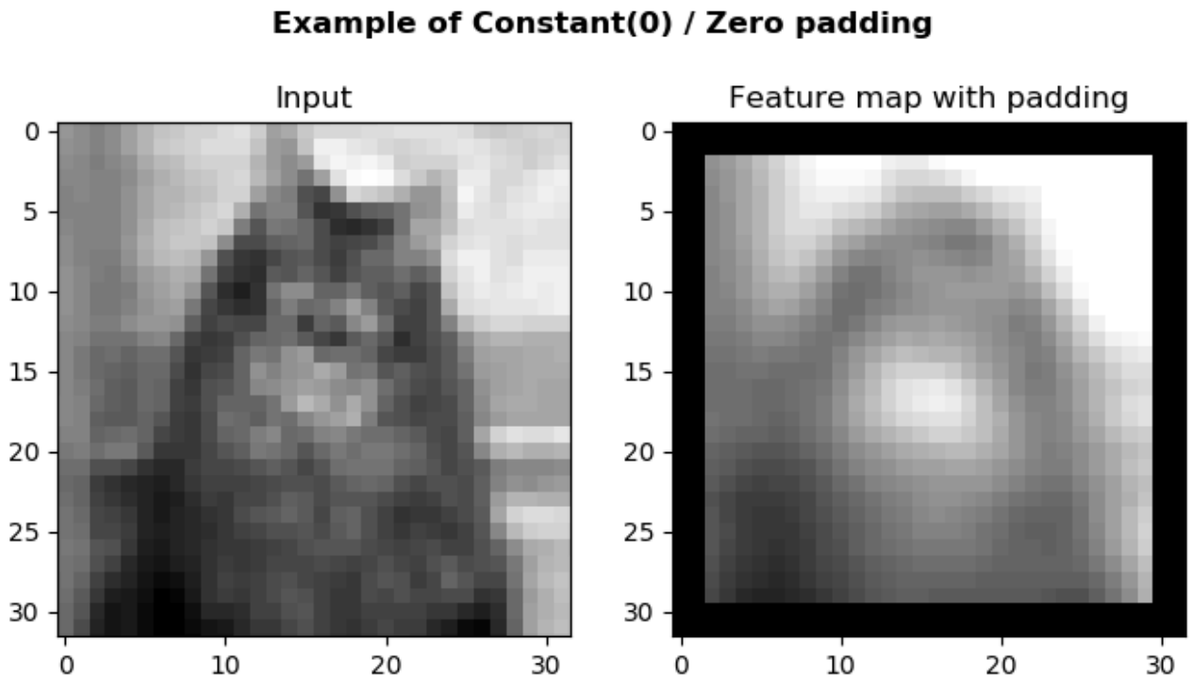
Output: 6x6 activations

-4	-5	-1	3	-5	5
-10	-14	-1	10	-1	7
-8	-11	-11	7	12	8
11	-7	-10	1	13	13
-6	5	-16	-4	10	12
-6	4	-7	-1	2	8

https://www.brilliantcode.net/1584/convolutional-neural-networks-1-convolution-layer-stride-padding-kernel/

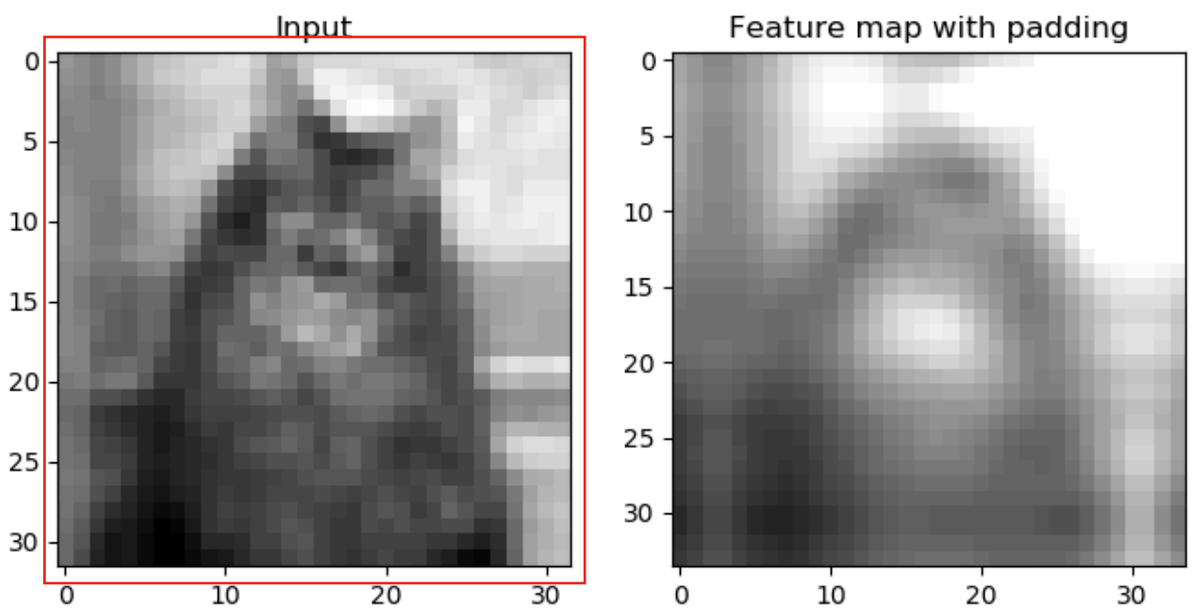
Convolutional layers

32 x 32 image



32 x 32 activation map

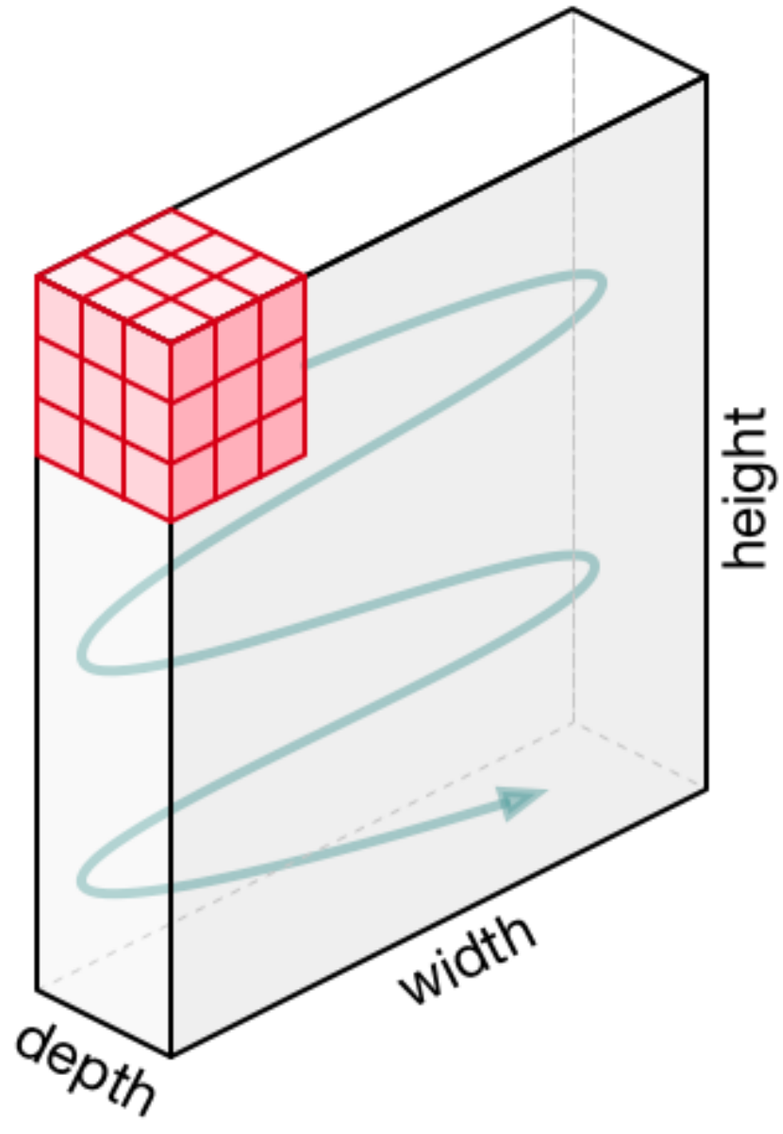
34 x 34 image



32 x 32 activation map

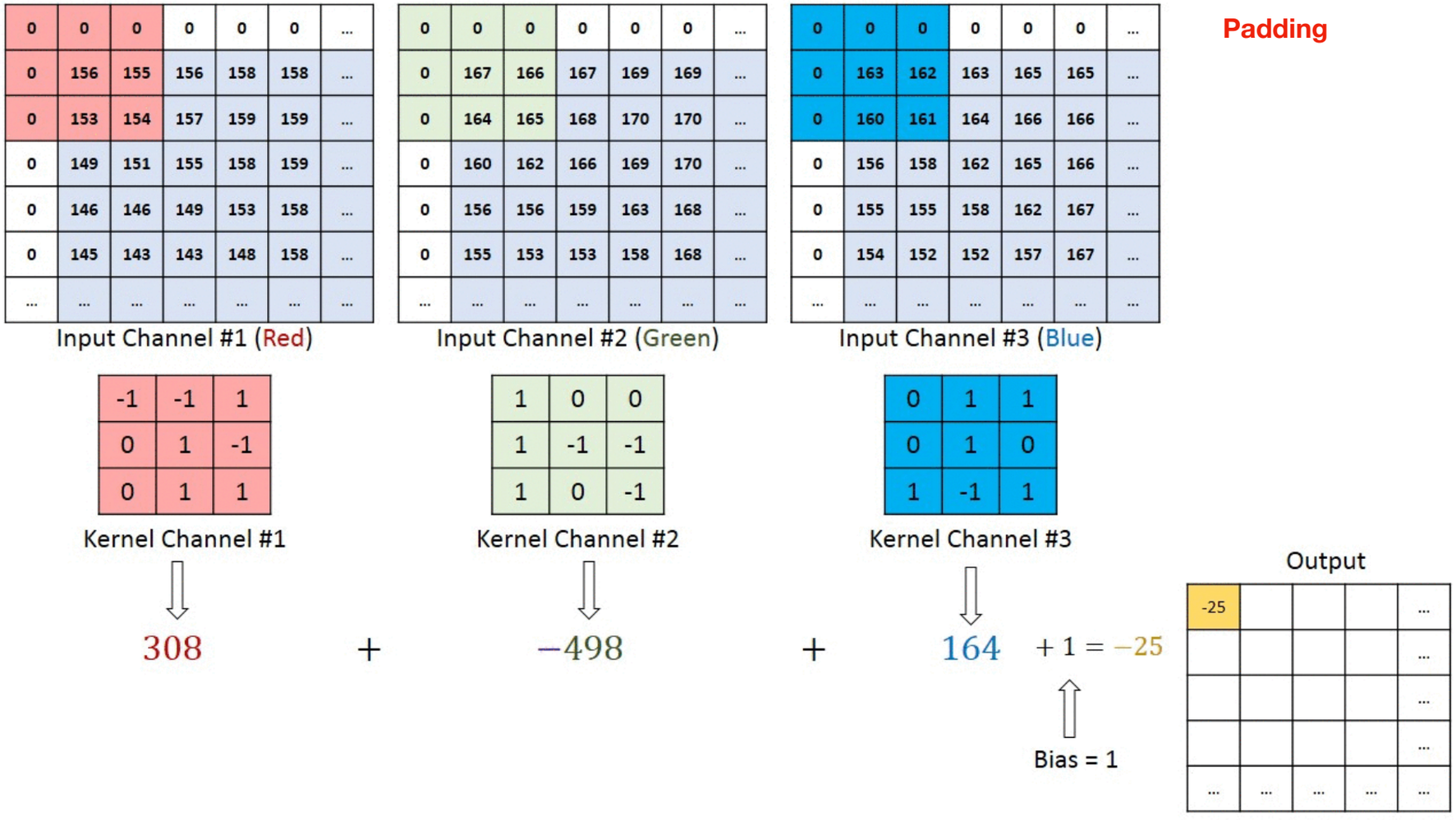
https://www.machinecurve.com/index.php/2020/02/07/what-is-padding-in-a-neural-network/

Convolutional layers (tensors)



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolutional layers

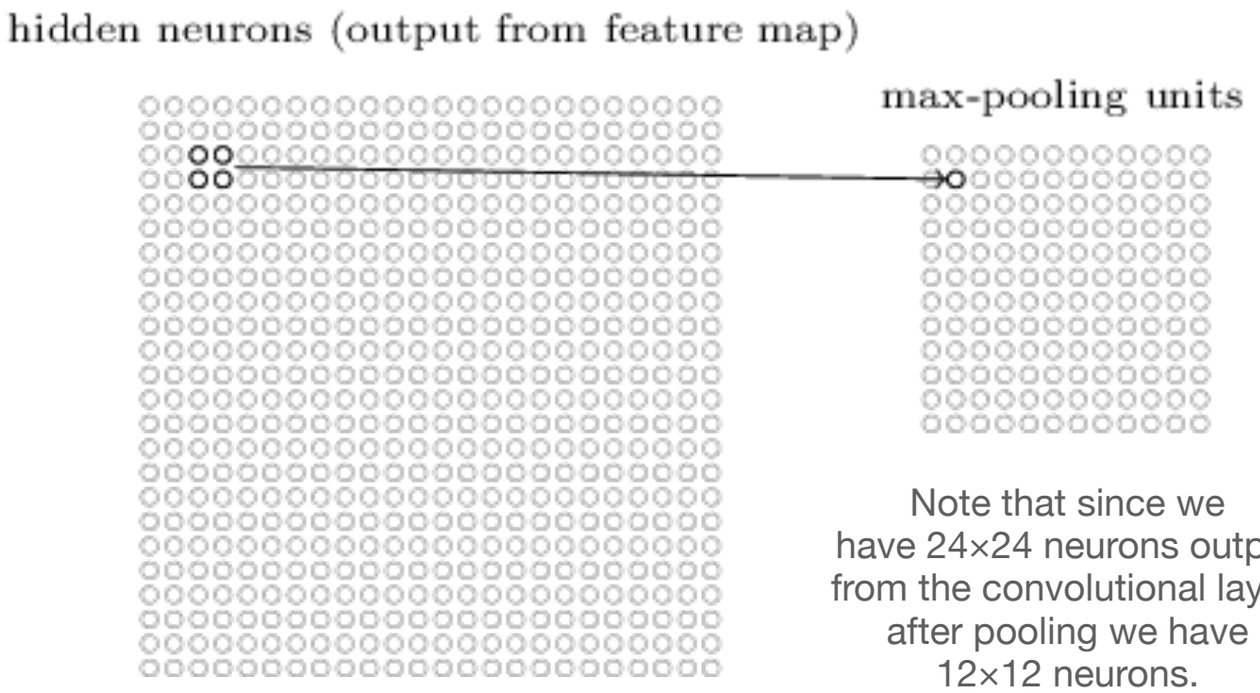


https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolutional layers

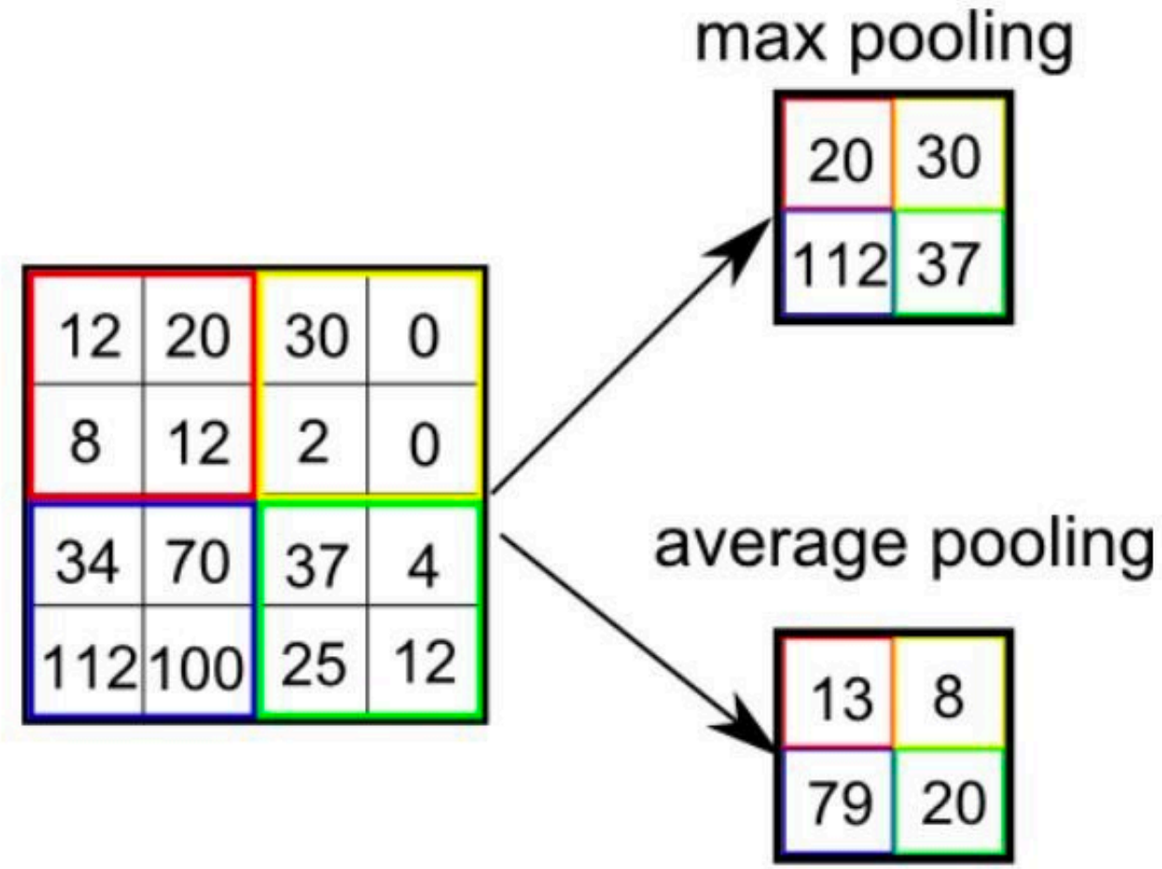
In addition to the convolutional layers just described, convolutional neural networks also contain **pooling layers**. Pooling layers are usually used immediately after convolutional layers. What the pooling layers do is **simplify the information in the output** from the convolutional layer.

As a concrete example, one common procedure for pooling is known as *max-pooling*. In max-pooling, a pooling unit simply outputs the maximum activation in the 2×2 input region, as illustrated in the following diagram:



https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

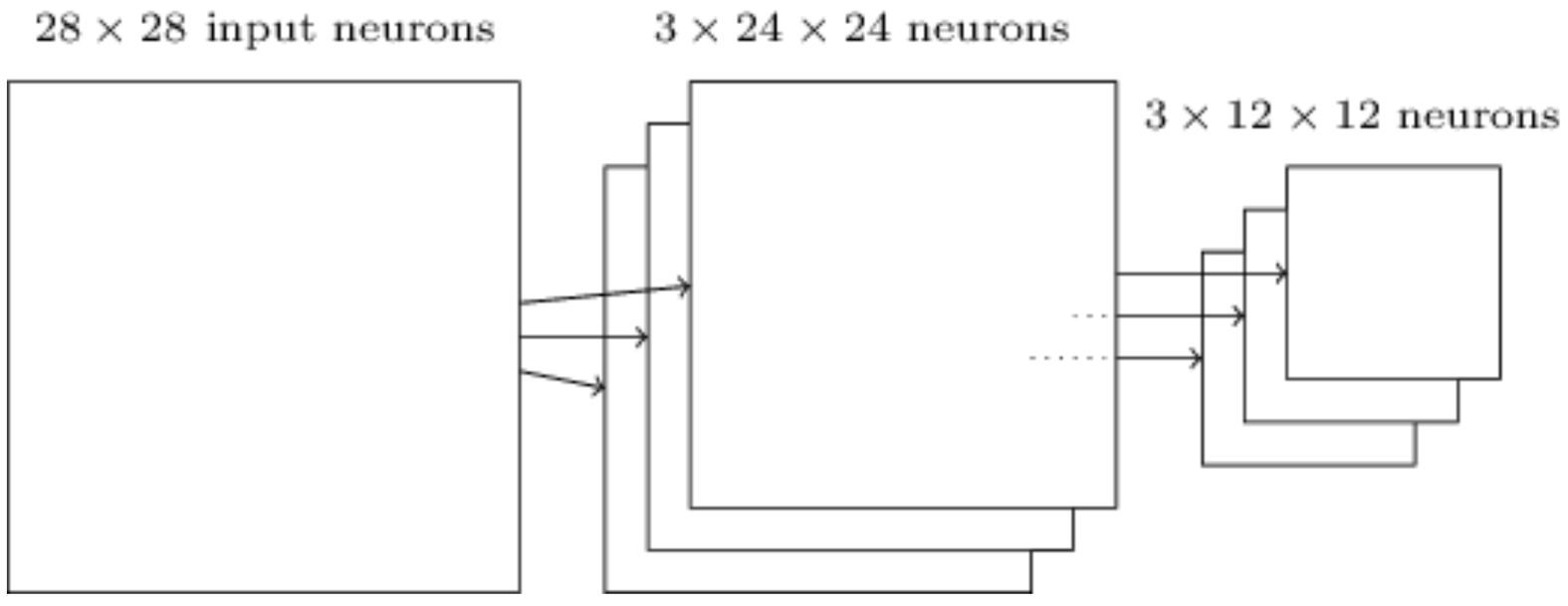
Convolutional layers



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolutional layers

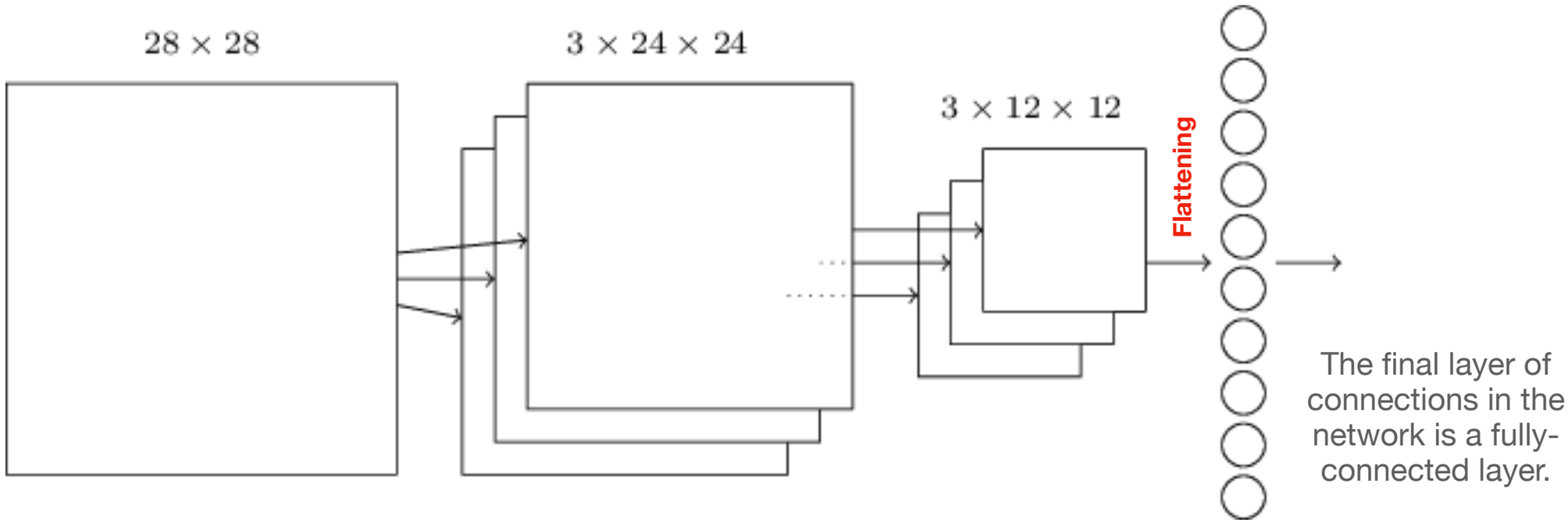
If there were three feature maps with no padding, the combined convolutional and max-pooling layers would look like:



https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Convolutional layers

We can now put all these ideas together to form a complete convolutional neural network:



https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

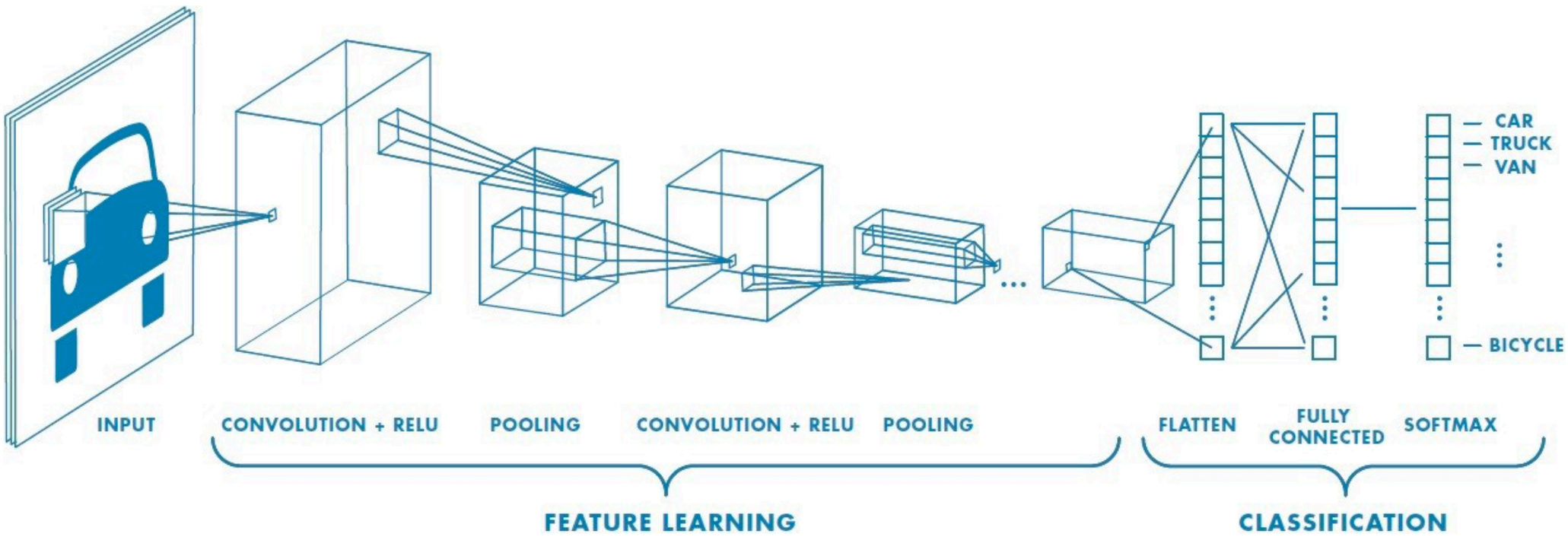
The network begins with 28×28 input neurons, which are used to encode the pixel intensities for the MNIST image. This is then followed by a convolutional layer using a 5×5 local receptive field and 3 feature maps. The result is a layer of 3×24×24 hidden feature neurons. The next step is a max-pooling layer, applied to 2×2 regions, across each of the 3 feature maps. The result is a layer of 3×12×12 hidden feature neurons.

Convolutional layers and stacking

The **stacking** of convolutional layers allows a **hierarchical decomposition** of the input.

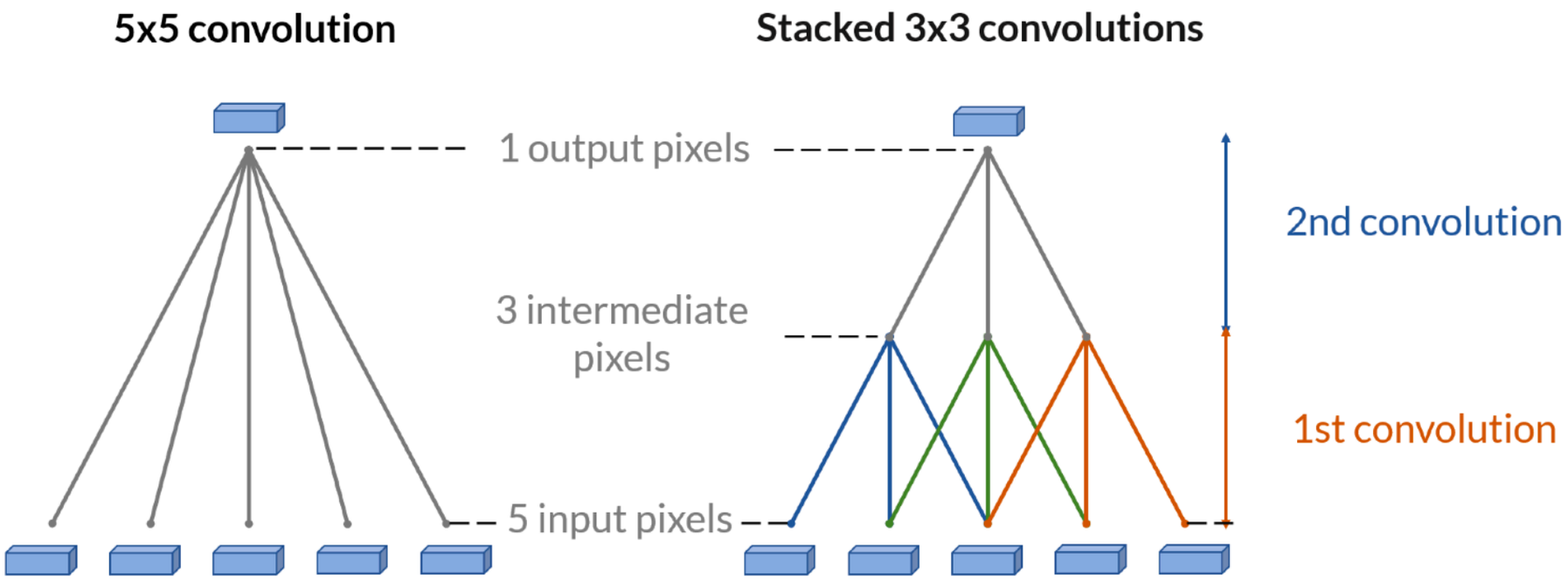
Consider that the filters that operate directly on the raw pixel values will learn to extract low-level features, such as lines and small patterns. The filters that operate on the output of the first line layers may extract features that are combinations of lower-level features, such as features that comprise multiple lines to express shapes.

This process continues until very deep layers are extracting faces, animals, houses, and so on.



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolutional layers and stacking



We can replace large convolution kernels with multiple 3x3 convolutions on top of one another.

This is good for two reasons: deeper is better and less computational cost.

Convolutional layers in Keras

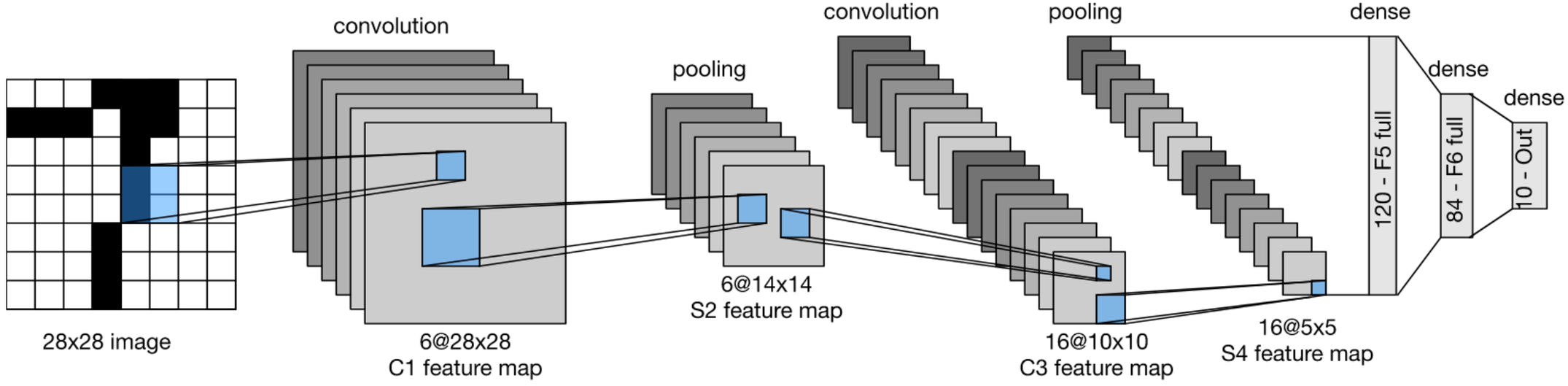
```
1
2 model = keras.Sequential(
3     [
4         keras.Input(shape=input_shape),
5         layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
6         layers.MaxPooling2D(pool_size=(2, 2)),
7         layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
8         layers.MaxPooling2D(pool_size=(2, 2)),
 9         layers.Flatten(),
10         layers.Dropout(0.5),
11         layers.Dense(num_classes, activation="softmax"),
12     ]
13 )
14
15 model.summary()
16
```

```
Test loss: 0.027600124478340 15
Test accuracy: 0.990000009536743 2
```

CNN architectures

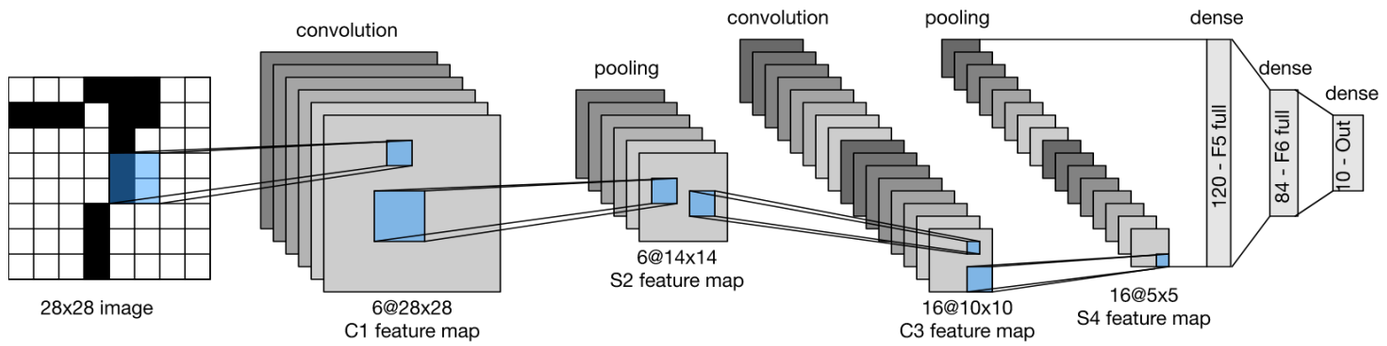
LeNET (1989)

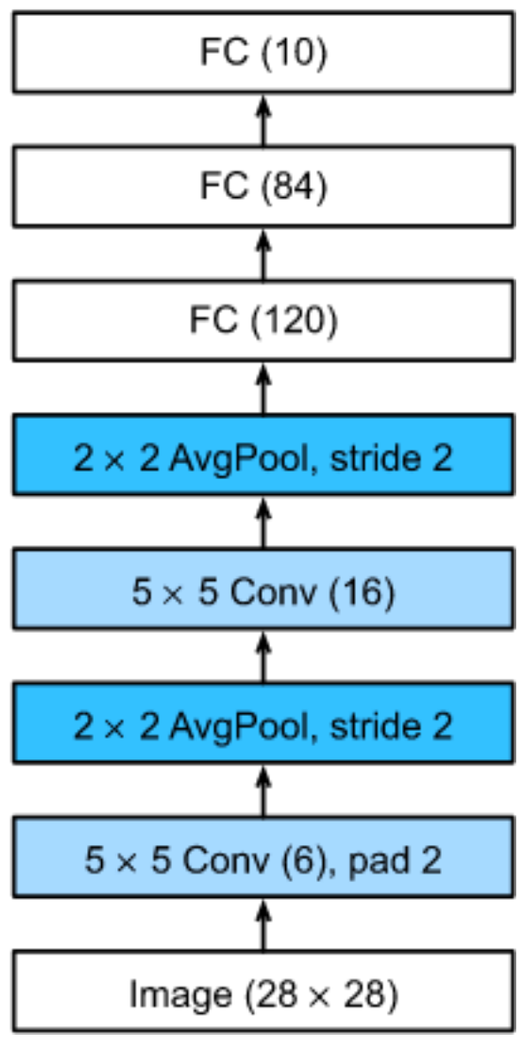
LeNet was the first published CNNs to capture wide attention for its performance on computer vision tasks. The model was introduced by (and named for) Yann LeCun, then a researcher at AT&T Bell Labs, for the purpose of recognizing handwritten digits in images.



https://d2l.ai/chapter_convolutional-neural-networks/lenet.html

LeNET (1989)

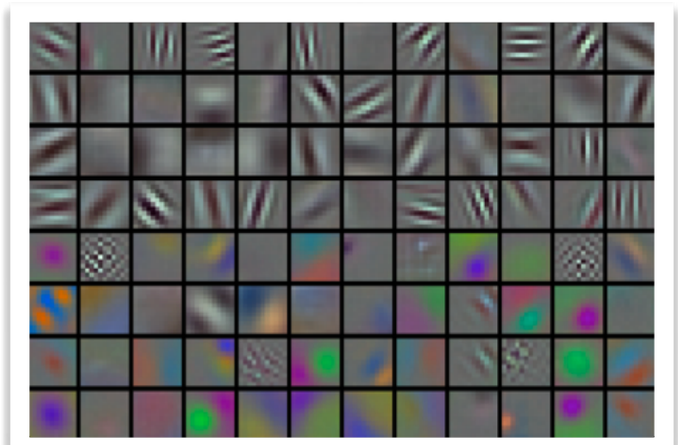


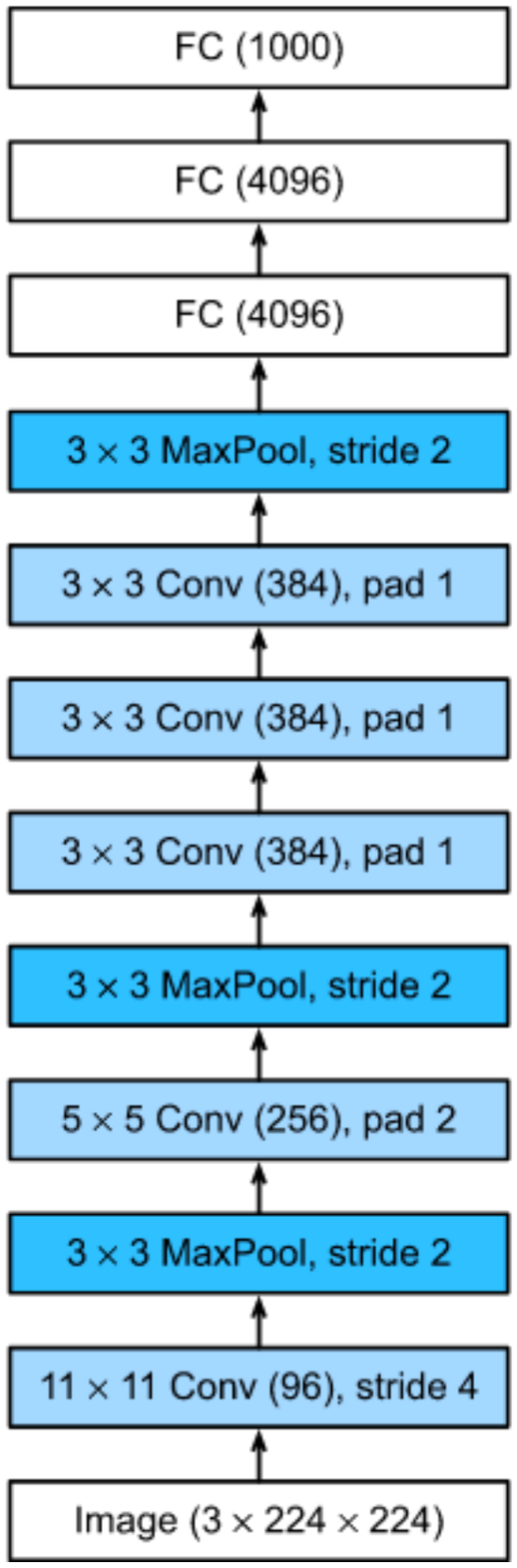


https://d2l.ai/chapter_convolutional-neural-networks/lenet.html

AlexNet (2012)

Filters that operate in raw pixels can be visualized because they are like images (**11 x 11 x 3**).





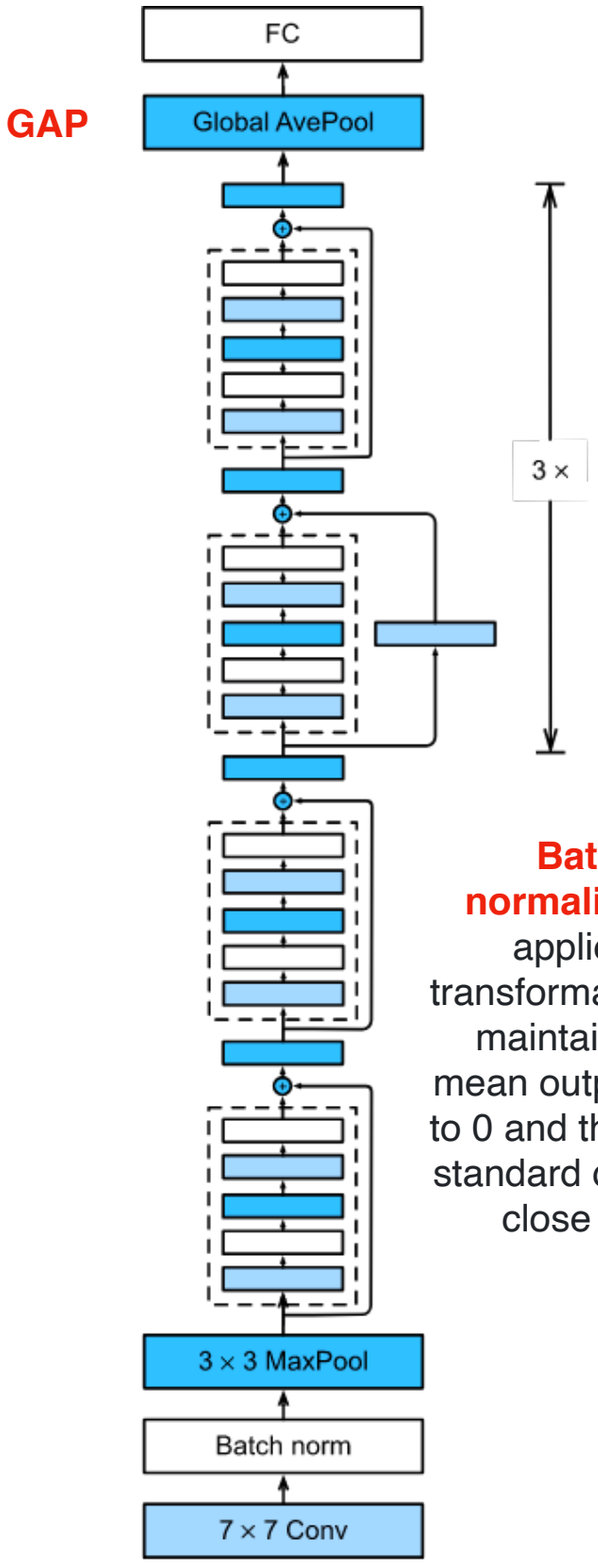
The lowest layers of the network, the model learned feature extractors that resembled some traditional filters.

Higher layers in the network might **build upon these representations to represent larger structures**, like eyes, noses, blades of grass, and so on. Even higher layers might represent whole objects like people, airplanes, dogs, or frisbees.

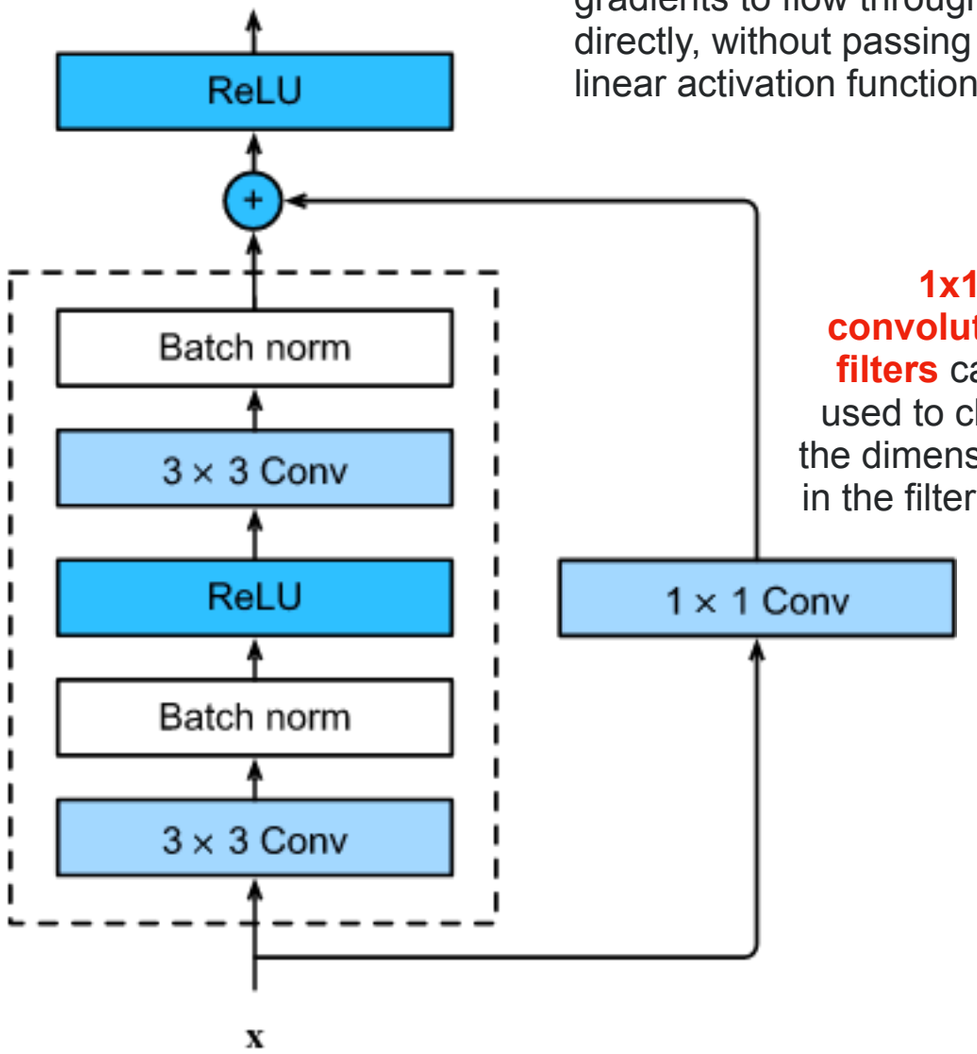
Ultimately, **the final hidden state learns a compact representation of the image that summarizes its contents** such that data belonging to different categories be separated easily.

https://d2l.ai/chapter_convolutional-neural-networks/lenet.html

ResNet (2016)



Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.

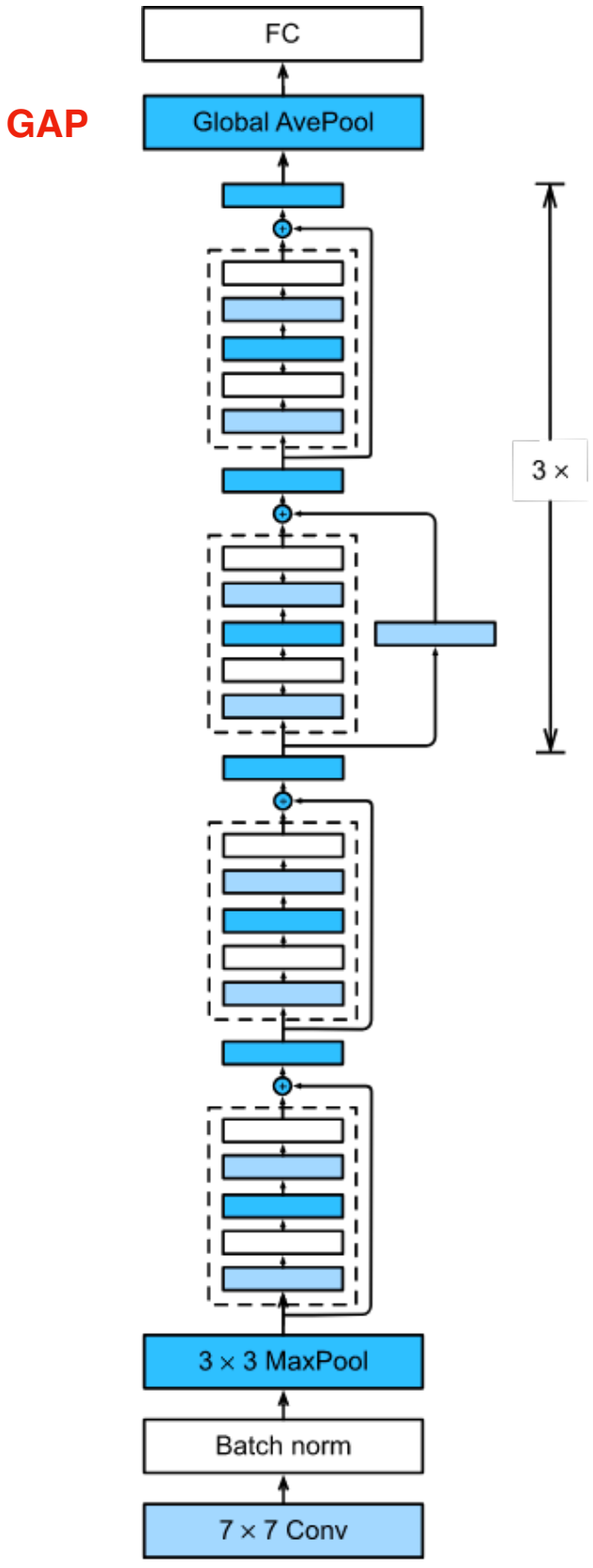


Residual connections are used to allow gradients to flow through a network directly, without passing through non-linear activation functions.

1x1 convolutional filters can be used to change the dimensionality in the filter space.

https://d2l.ai/chapter_convolutional-neural-networks/lenet.html

ResNet (2016)



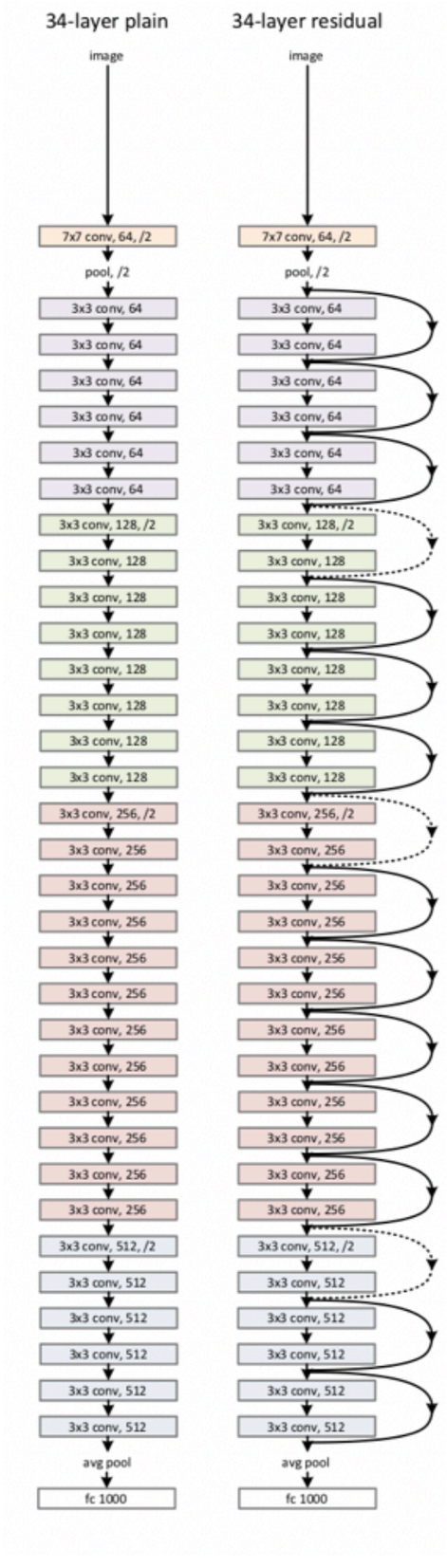
GAP layers are used to reduce the spatial dimensions of a three-dimensional tensor. **A tensor with dimensions h×w×d is reduced in size to have dimensions 1×1×*d*.**

GAP layers reduce each h×w feature map to a single number by simply taking the average of all h,w values.

In ResNet, the GAP layer is followed by one densely connected layer with a softmax activation function that yields the predicted object classes.

https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/

ResNet (2016)

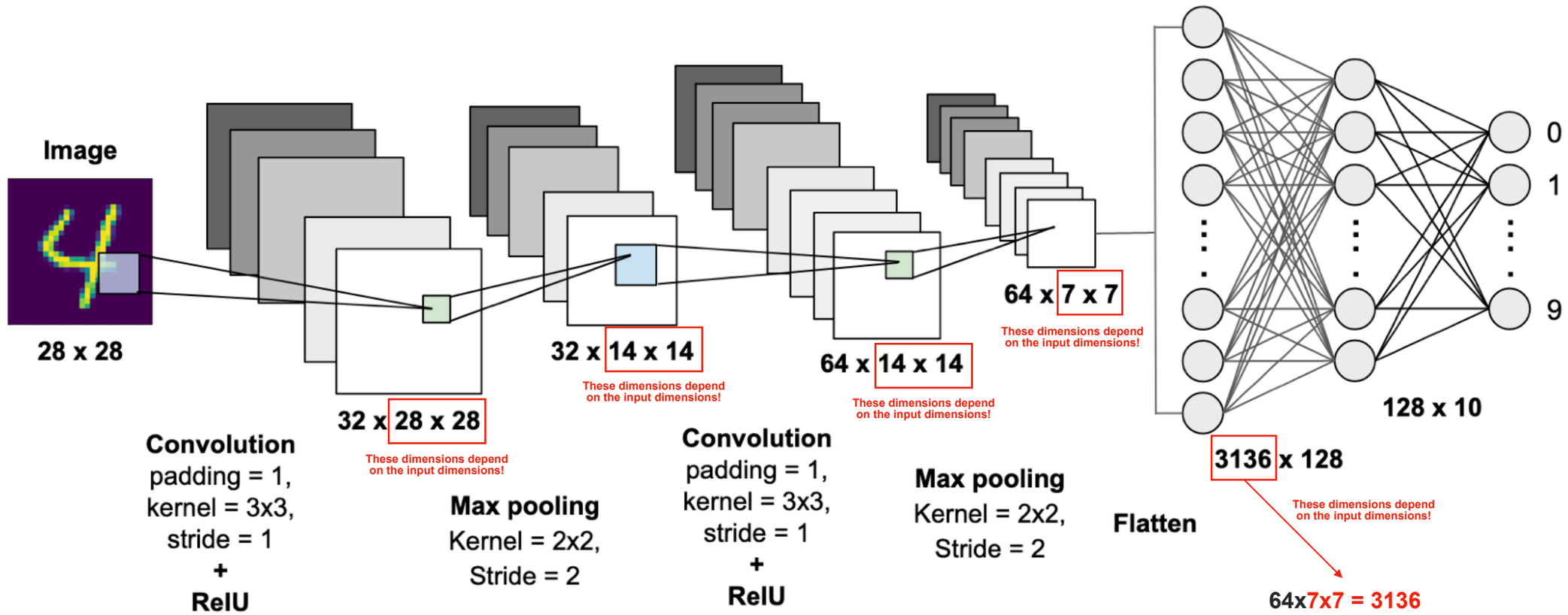


Gradient information can be lost as we pass through many layers, this is called vanishing gradient.

Advantages of skip connection are they pass feature information to lower layers so learning becomes easier.

Fully Convolutional Networks (2015)

The use of a fully connected layer at the end of a model represents a severe limitation!



https://becominghuman.ai/building-a-convolutional-neural-network-cnn-model-for-image-classification-116f77a7a236

We can only classify images of this size! What about larger images?

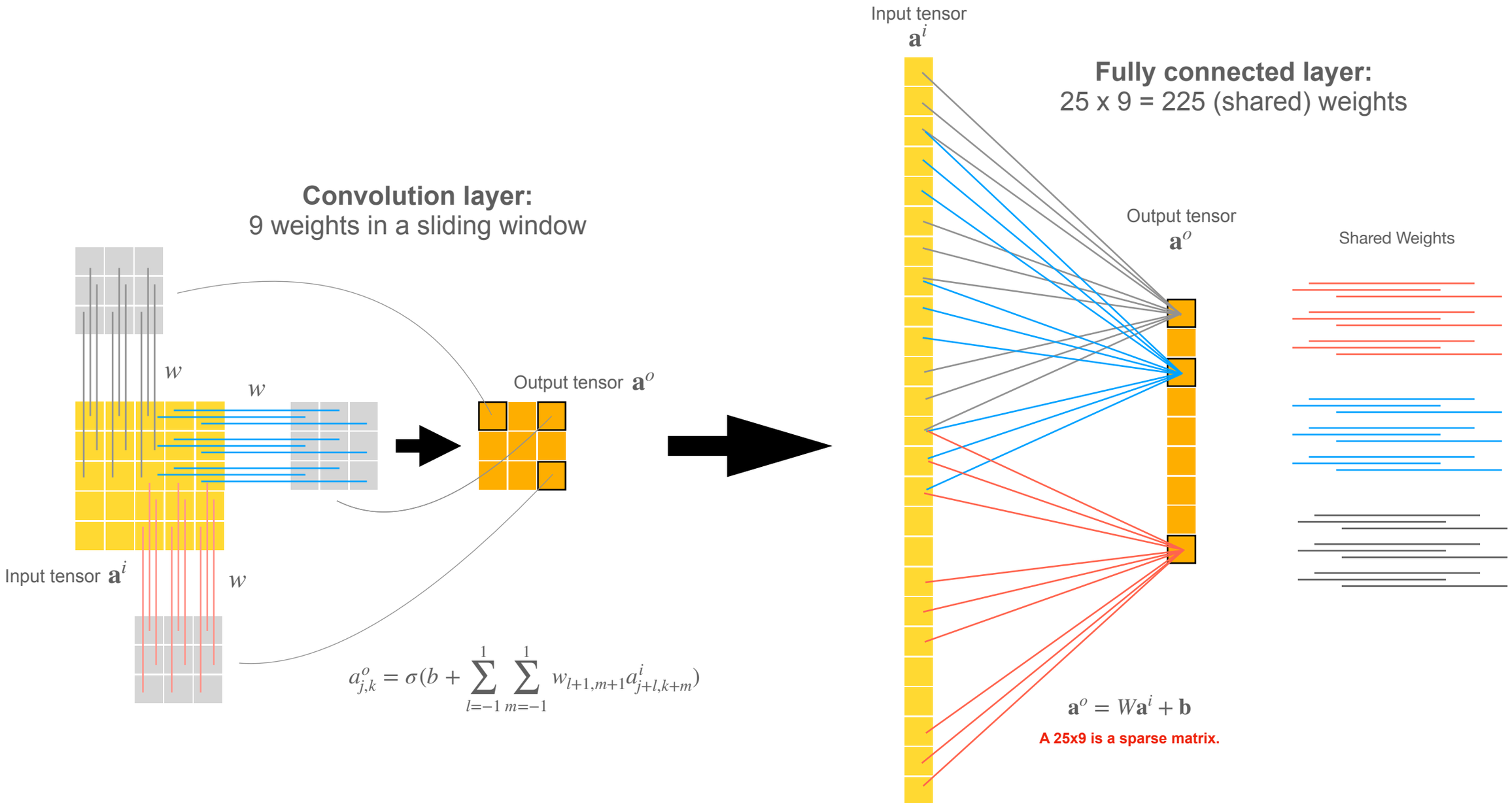
Fully Convolutional Networks (2015)

The only difference between Fully Connected (FC) and Convolutional (CONV) layers is that the neurons in the CONV layer are connected only to a local region in the input, and that many of the neurons in a CONV volume share parameters.

However, **the neurons in both layers still compute dot products, so their functional form is identical**.

Then, it is easy to see that **for any CONV layer there is an FC layer that implements the same forward function**. The weight matrix would be a large matrix that is mostly zero except for at certain blocks (due to local connectivity) where the weights in many of the blocks are equal (due to parameter sharing).

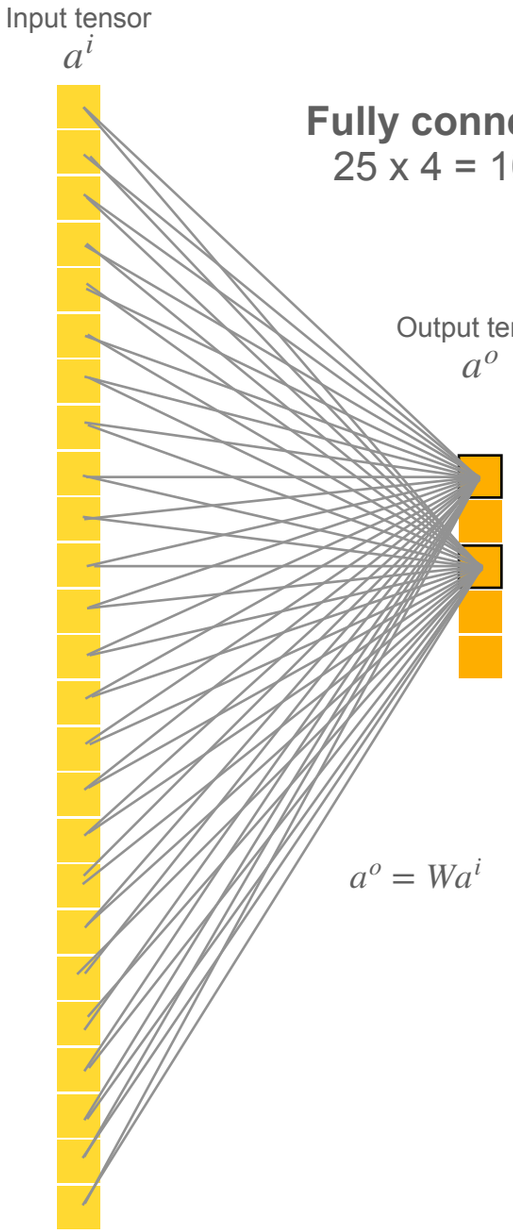
Fully Convolutional Networks (2015)



A convolutional layer can be expressed as a mutilated FC layer.

Fully Convolutional Networks (2015)

Conversely, any FC layer can be converted to a CONV layer.



Fully connected layer:
25 x 4 = 100 weights

Output tensor
a^o

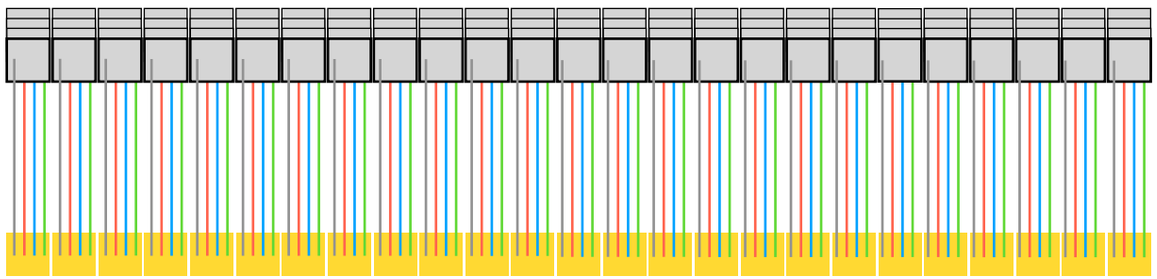
$a^o = Wa^i$

Convolution layer:
25 weights x 4 filters = 100 weights

Output tensor a^o

$a_k^o = \sigma(b + \sum_{j=0}^{25} w_j^k a_j^i)$

$\{w^k\}_{k=0,3}$



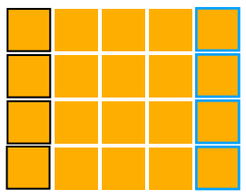
Input tensor a^i

Fully Convolutional Networks (2015)

And now we can process bigger input tensors!

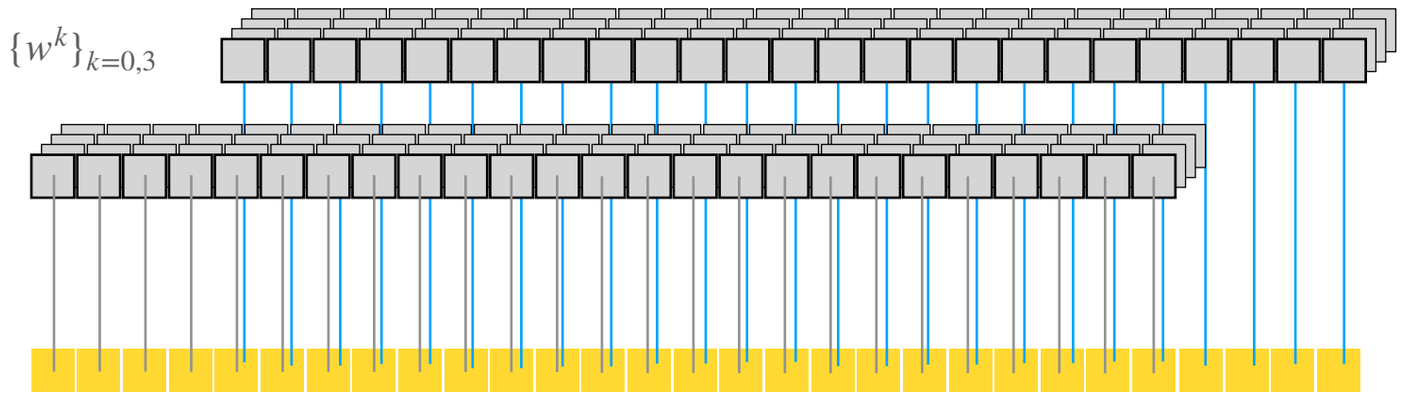
Convolution layer:
25 weights x 4 filters = 100 weights

Output tensor $\mathbf{a}^o$



We get a set of additional output dimensions that can be understood as the output of a sliding window classifier.

$$a^o_{k,l} = \sigma(b + \sum_{m=-12}^{12} w^k_{m+12} a^i_{k+m+l})$$

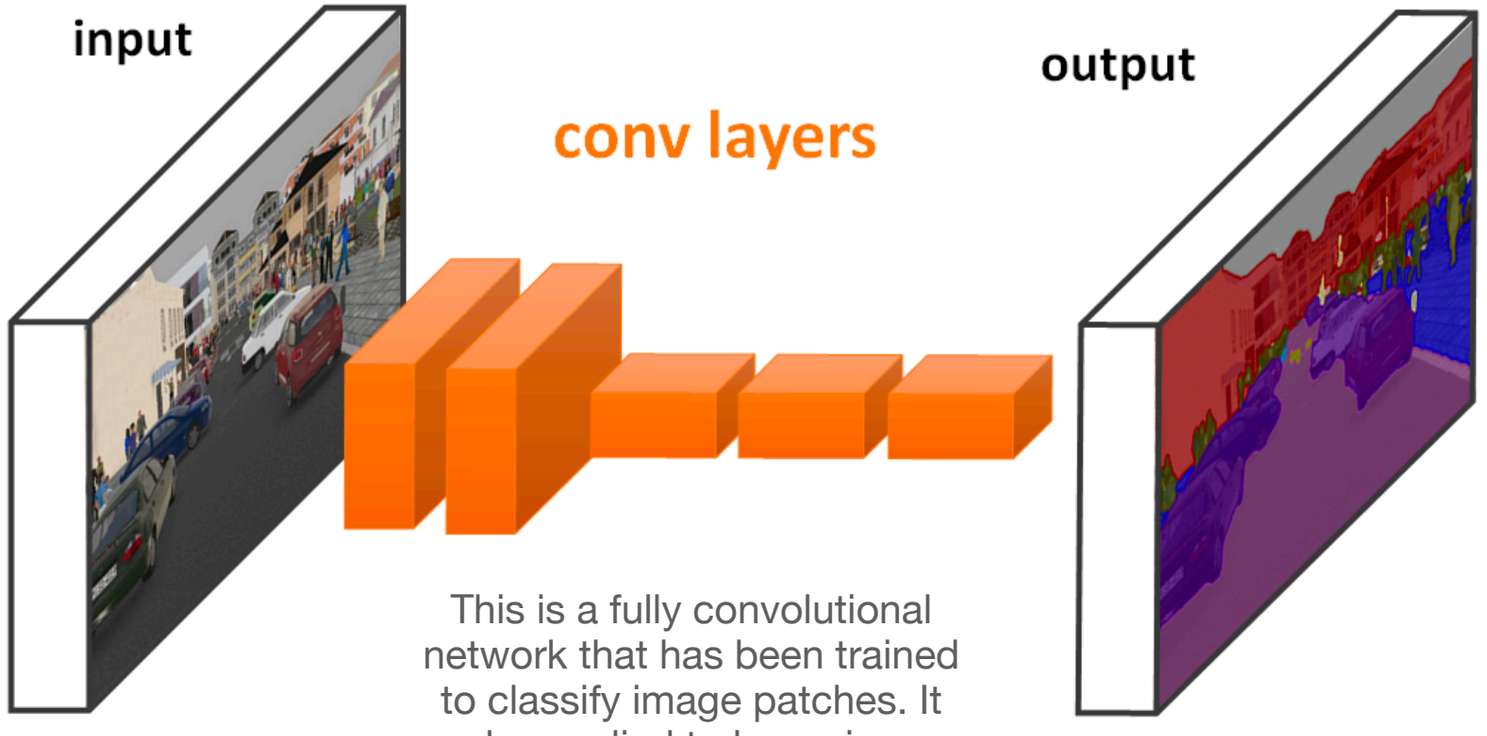


Input tensor a^i

Fully Convolutional Networks (2015)

It turns out that this conversion allows us to “slide” the original ConvNet very efficiently across many spatial positions in a larger image, in a single forward pass.

Now the model is independent of the input image size!



Fully Convolutional Networks

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(128, kernel_size=(3, 3), activation="relu"),
        layers.Conv2D(num_classes, kernel_size=(1, 1), activation="softmax"),
        layers.GlobalAveragePooling2D(),
    ]
)
```

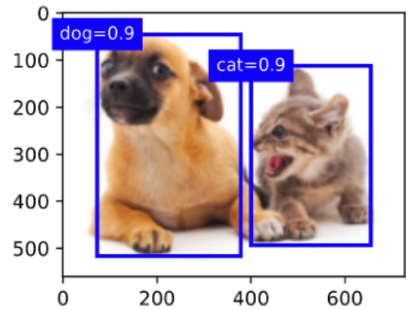
"Winner takes all"

```
Test loss: 0.050505142658948
Test accuracy: 0.9909999966621399
```

Classification and detection

We can deal with multiple scales by processing the image at multiple resolutions.

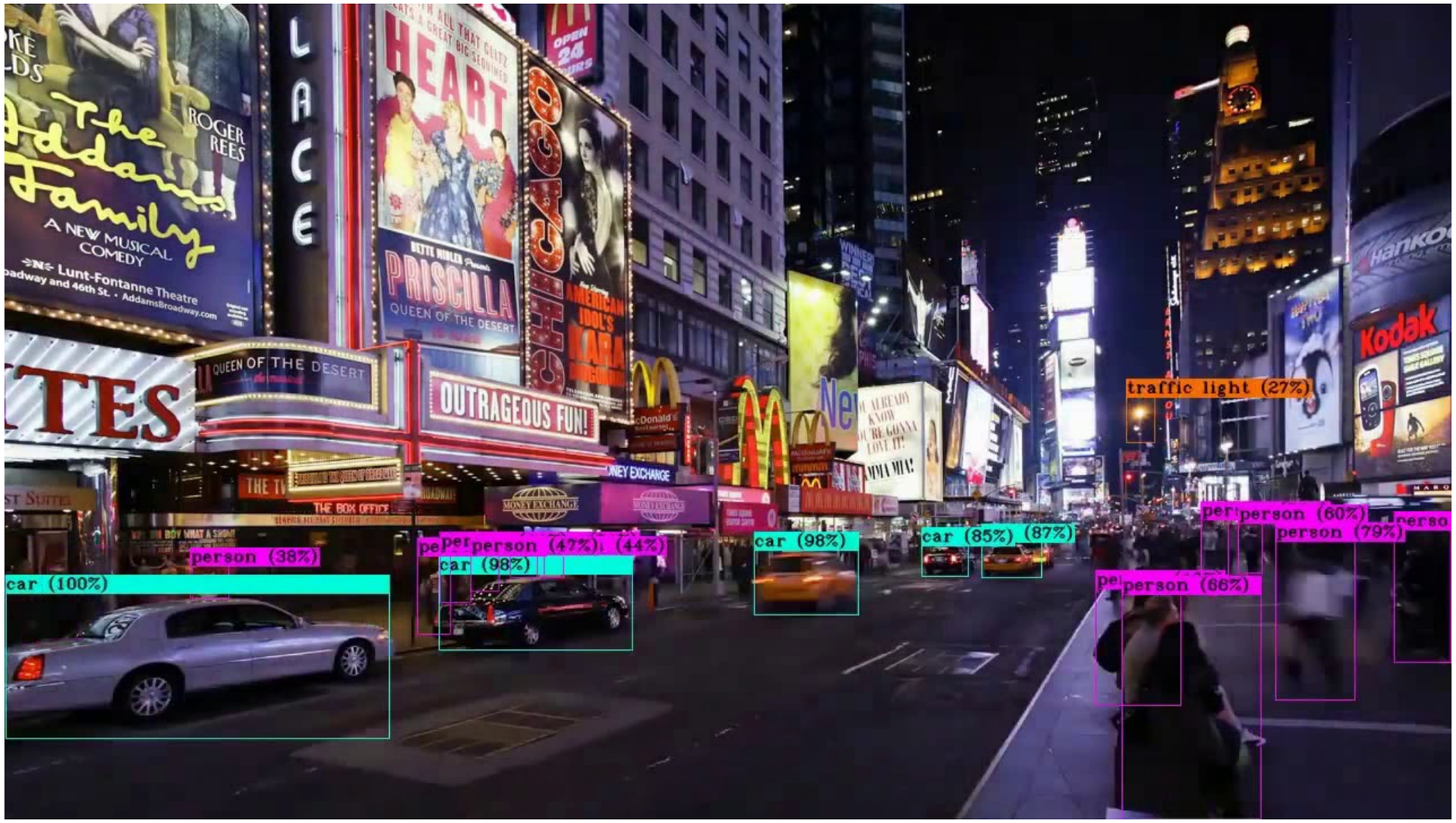
- **Image Classification**: Predict the type or class of an object in an image.
  - *Input*: An image with a single object, such as a photograph.
  - *Output*: A class label (e.g. one or more integers that are mapped to class labels).



Dealing with multiple scales is not so easy

- **Object Detection**: Locate the presence of objects with a bounding box and types or classes of the located objects in an image.
  - *Input*: An image with one or more objects, such as a photograph.
  - *Output*: One or more bounding boxes (e.g. defined by a point, width, and height), and a class label for each bounding box.

Object detection: Yolo4



Object detection algorithms usually **sample a large number of regions**, at multiple scales, in the input image, **determine whether these regions contain objects of interest**, and adjust the edges of the regions so as to predict the ground-truth bounding box of the target more accurately.

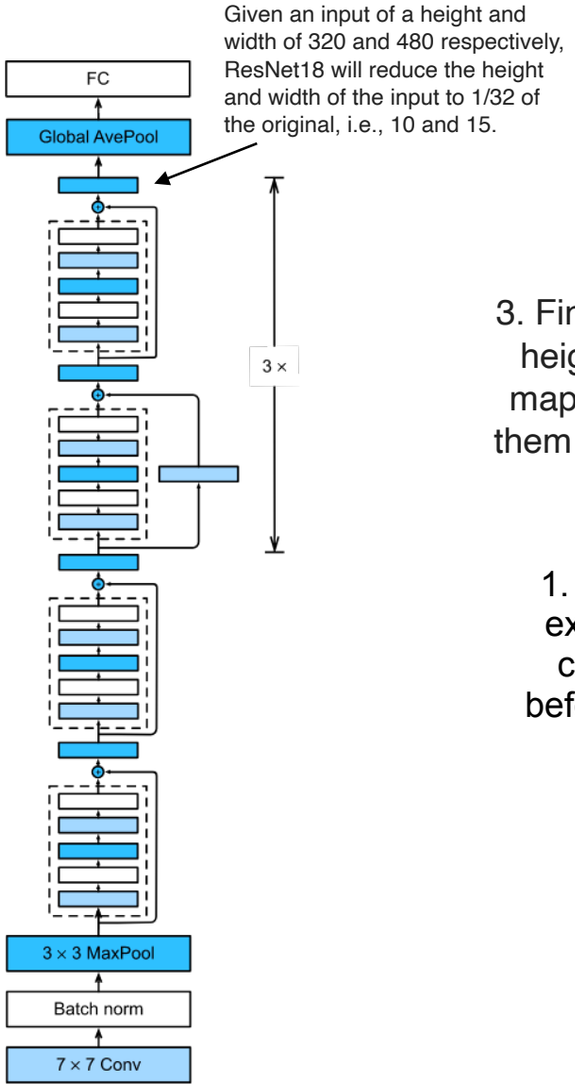
Semantic Segmentation

The layers we introduced so far for convolutional neural networks, including convolutional layers and pooling layers, often **reduce the input width and height**, or keep them unchanged.

Applications such as semantic segmentation, however, require to predict values for each pixel and therefore needs to increase input width and height. Transposed convolution, also named **deconvolution**, serves this purpose.

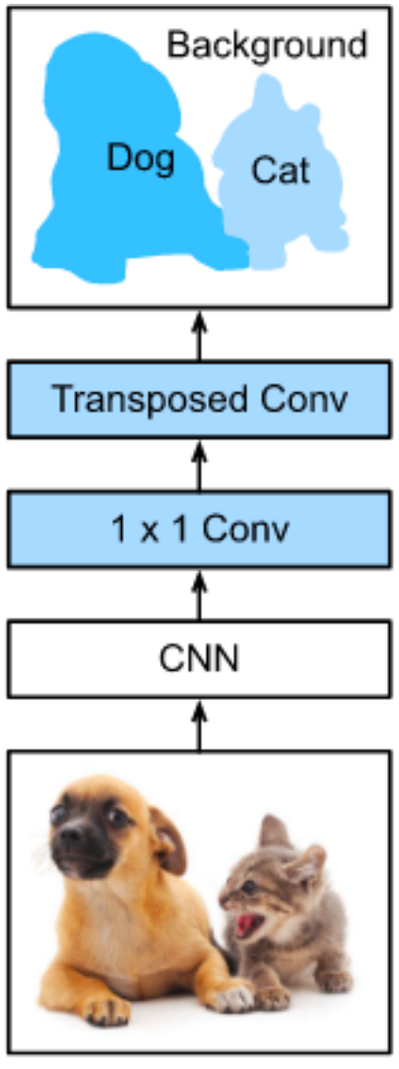
Semantic Segmentation

We can use a fully convolutional network. First we will use a convolutional neural network to extract **image features**, then we will transform the number of channels into the number of categories through the 1x1 convolution layer, and finally we will transfer the height and width of the feature map to the size of the input image by using a deconvolution layer.



3. Finally, we need to magnify the height and width of the feature map by a factor of 32 to change them back to the height and width of the input image.

1. We can use **ResNet18** to extract image features (last convolutional layer output before global average pooling layer).



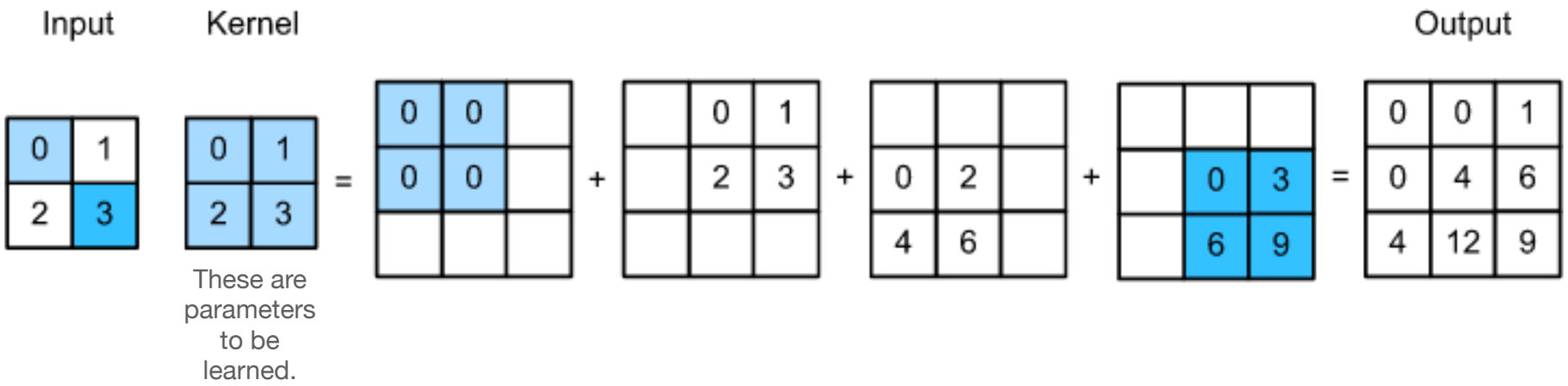
2. Next, we transform the number of output channels to the number of object categories of through the1×1convolution layer.

https://d2l.ai/chapter_computer-vision/fcn.html

Deconvolution

Let us consider a basic case that both input and output channels are 1, with 0 padding and 1 stride.

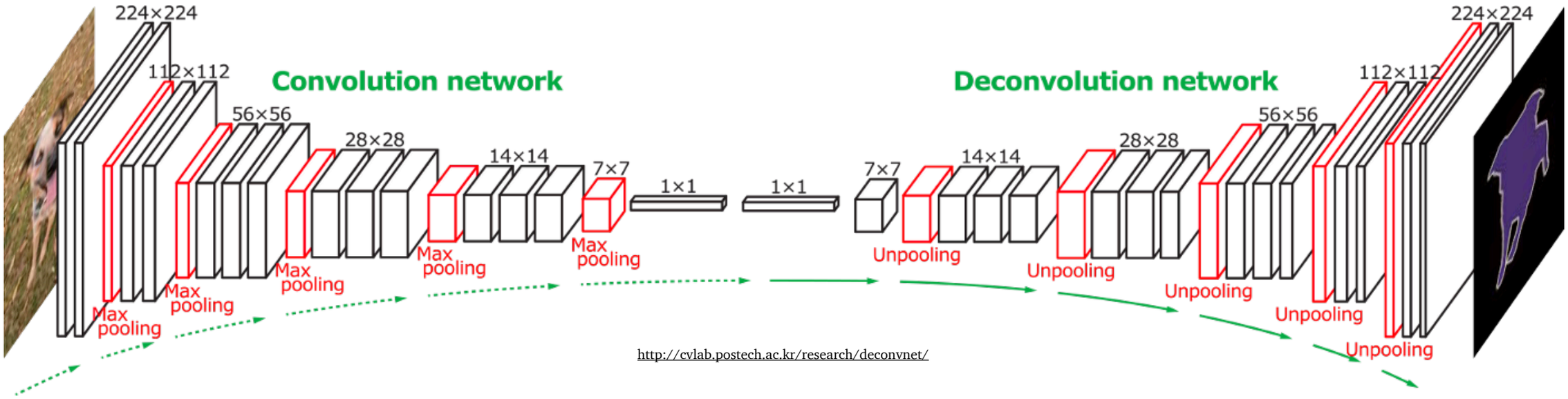
This is how transposed convolution with a 2×2 kernel is computed on the 2×2 input matrix:




```
def trans_conv(X, K):
    h, w = K.shape
    Y = np.zeros((X.shape[0] + h - 1, X.shape[1] + w - 1))
    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            Y[i: i + h, j: j + w] += X[i, j] * K
    return Y
```

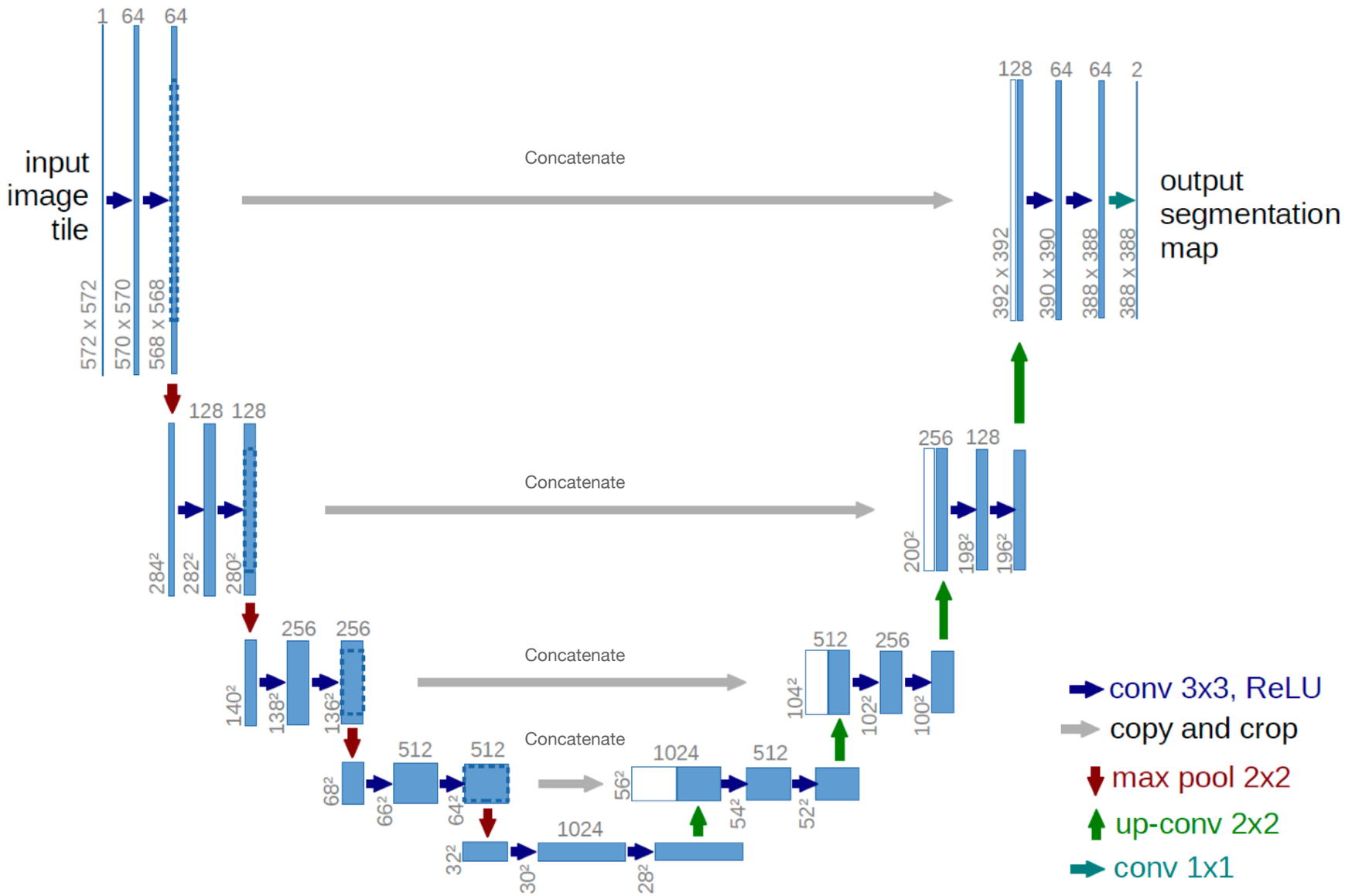
https://d2l.ai/chapter_computer-vision/fcn.html

Semantic Segmentation



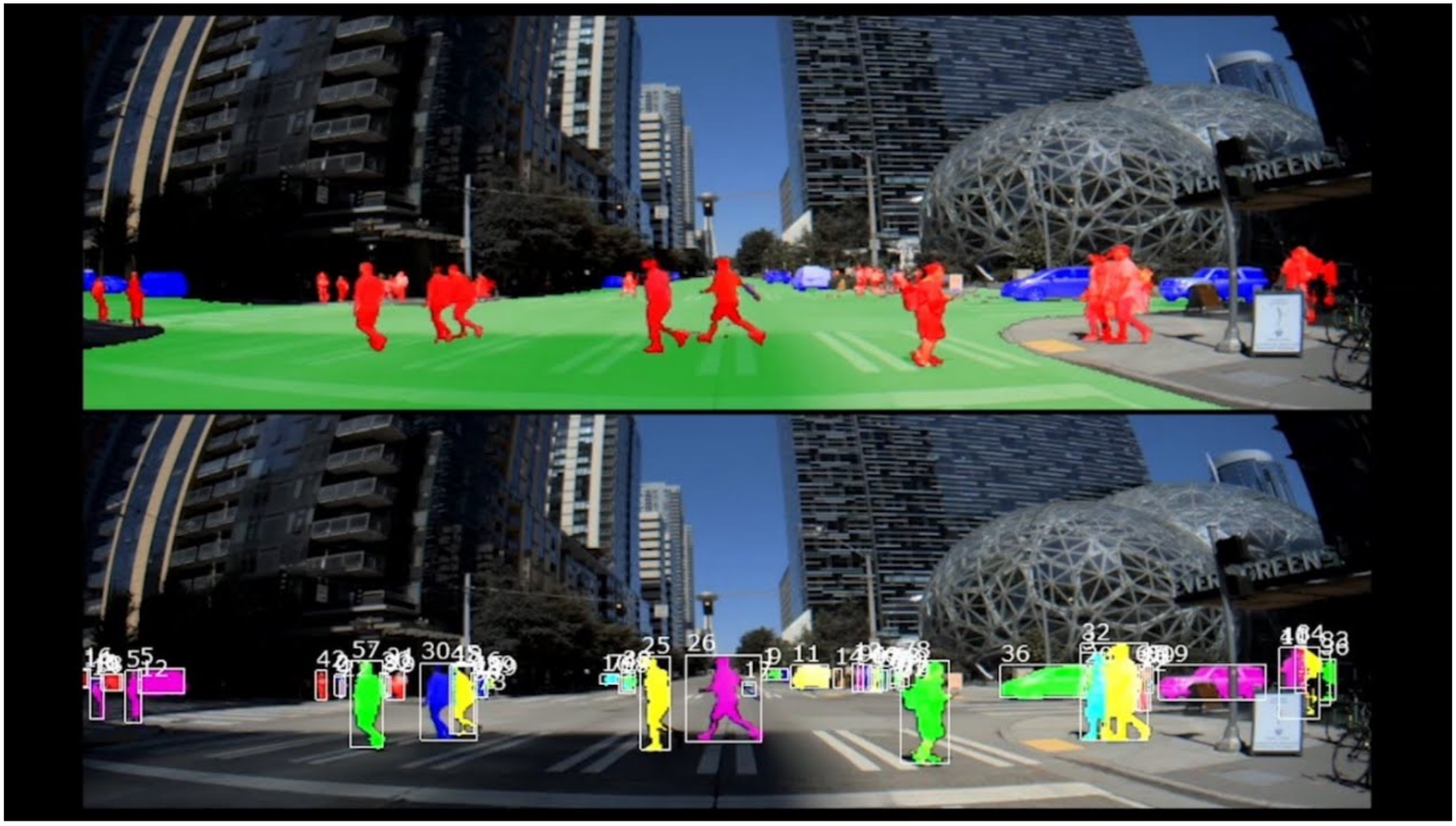
http://cvlab.postech.ac.kr/research/deconvnet/

Semantic Segmentation: U-Net



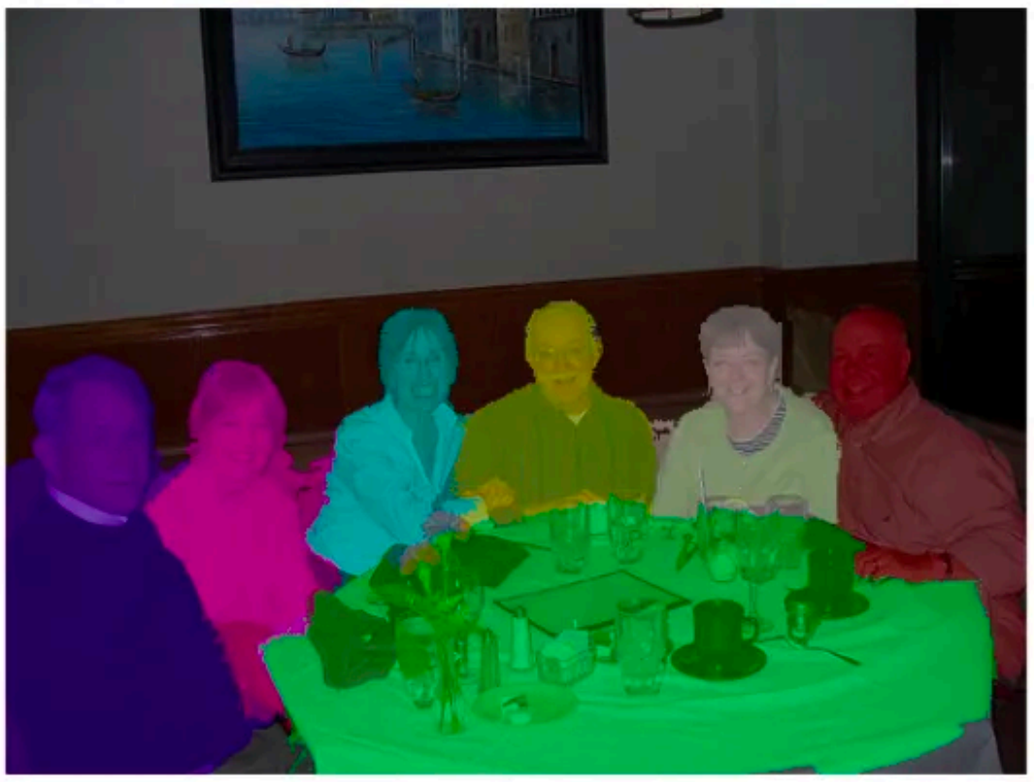
https://medium.com/@nishanksingla/unet-with-resblock-for-semantic-segmentation-dd1766b4ff66

Semantic Segmentation



Semantic Segmentation

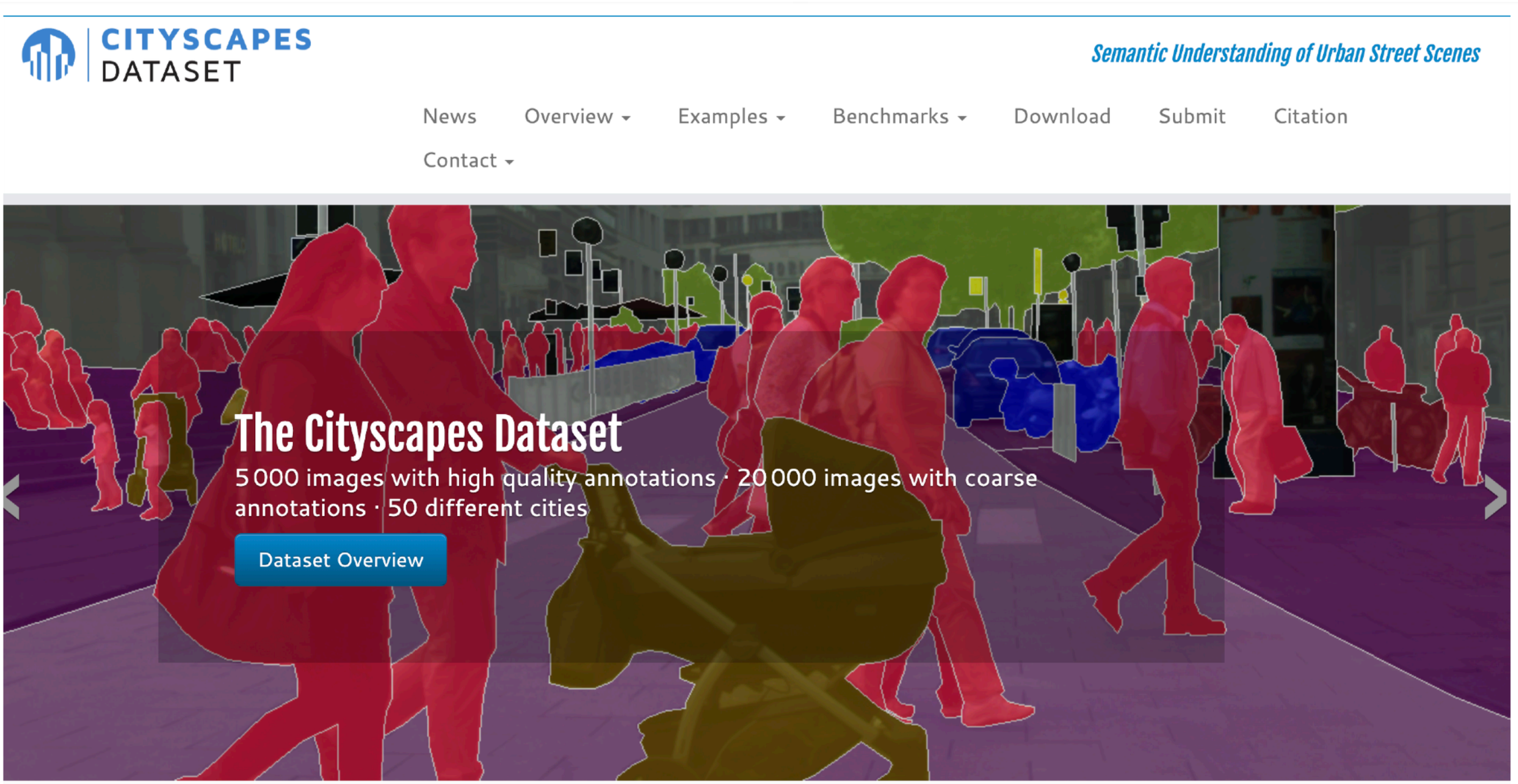
Semantic Segmentation



Instance Segmentation

http://www.robots.ox.ac.uk/~tvg/publications/2017/CRFMeetCNN4SemanticSegmentation.pdf

Semantic Segmentation: Databases

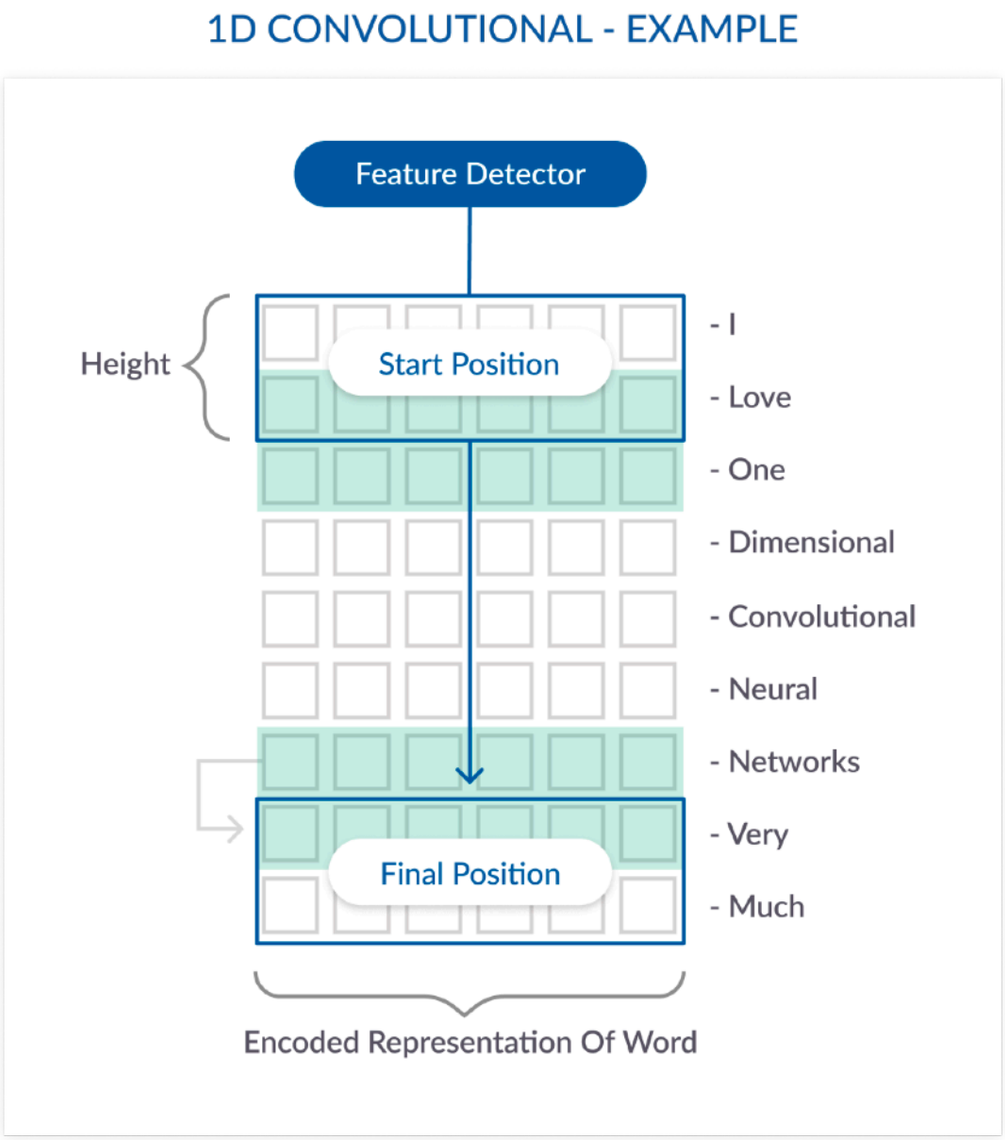


1D Convolutional Networks for text classification

IMDB Movie reviews sentiment classification: Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative). Reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data. This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words”.

The seminal research paper on this subject was published by Yoon Kim on 2014. In this paper Yoon Kim has laid the foundations for how to model and process text by convolutional neural networks for the purpose of sentiment analysis. He has shown that by simple one-dimentional convolutional networks, one can develops very simple neural networks that can reach 90% accuracy very quickly.

1D Convolutional Networks for text classification



This sentence is made up of 9 words. Each **word** is a **vector** that represents a word. A parameter specifies how many words the filter should consider at once. In this example the height is 2, meaning the filter moves 8 times to fully scan the data.

https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-networks-keras/

1D Convolutional Networks for text classification

```
model = Sequential()
model.add(Embedding(max_features, embedding_dims, input_length=maxlen))
model.add(Dropout(0.25))
model.add(Convolution1D(filters=nb_filter,
                        kernel_size=filter_length,
                        padding='valid',
                        activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Convolution1D(filters=nb_filter,
                        kernel_size=filter_length,
                        padding='valid',
                        activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(hidden_dims))
model.add(Dropout(0.25))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

2ond Assignment: Facial Point Detection

> You will need **GPU computation**. Consider the use of Colab!

2ond Assignment: Facial Point Detection

The objectives of this assignment are:

- To develop a better solution by changing the baseline model.
- To increase performance by using several tricks:
    - Data augmentation: flipped images, greylevel image editing, etc.
    - Changing learning rate and momentum over time.
    - Using regularization techniques such as Dropout.
    - Instead of training a single model, train a few specialist networks, with each one predicting a different subset of target values (f.e. eye features, mouth features, etc.).
    - Etc.

Reporting

Please, report the results of your experiments in this cell.

What is the best result you got when testing?

> Results must be evaluated by computing the **mean error in pixel units**. If you get a mean error of 1.6 pixels is ok. If your get 1.5 is very good. If you get 1.4 or less, it is an outstanding result!

Answer: The mean pixel error is

What architecture and tricks you used for that result?

Describe your design strategy as well all those tricks that contributed to your result.

Answer: