IT496: Introduction to Data Mining

Lecture 31-32

Convolutional Neural Networks

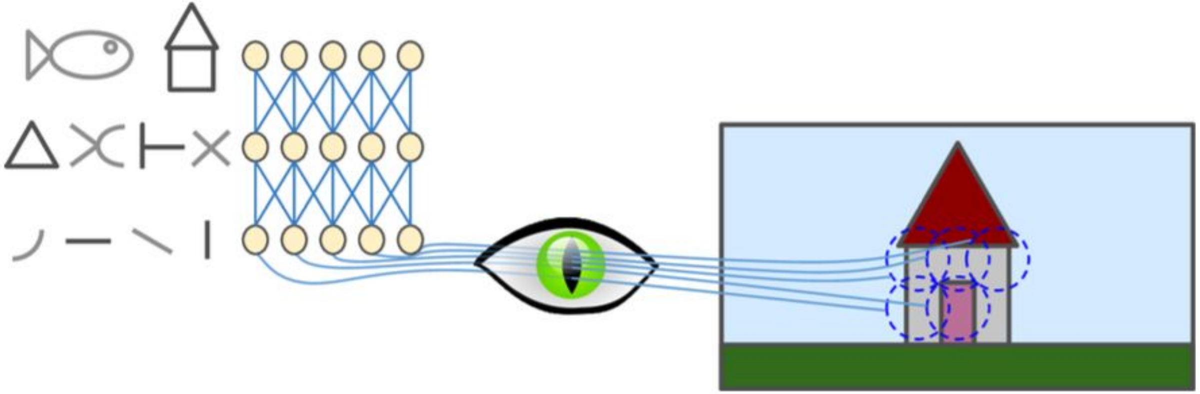
(Slides are created from the lecture notes of Dr. Derek Bridge, UCC, Ireland)

Arpit Rana

2^{nd} / 3^{rd} November 2023

Primate Vision

- David H. Hubel and Torsten Wiesel performed a series of experiments on cats in 1958 and 1959 and a few years later on monkeys.
- They showed that in the primate vision system, there seems to be a hierarchy of neurons within the visual cortex:



Primate Vision

- In the lowest layers,
  - neurons have small local ***receptive fields***, i.e. they respond to stimuli in a limited region of the visual field; and
  - they respond to, e.g., spots of light.
- In higher layers,
  - they combine the outputs of neurons in the lower layers;
  - they have larger receptive fields; and
  - they respond to, e.g., lines at particular orientations (two neurons may have the same receptive field but react to different line orientations).
- In the highest layers,
  - they respond to ever more complex combinations, such as shapes and objects.

Visual Cortex in Human Brain

- There are perhaps as many as 8 layers in the visual cortex alone.
- The image shows the feedforward circuits involved in rapid categorization tasks.
- Numbers for each cortical stage corresponds to the shortest latencies observed and the more typical mean latencies.

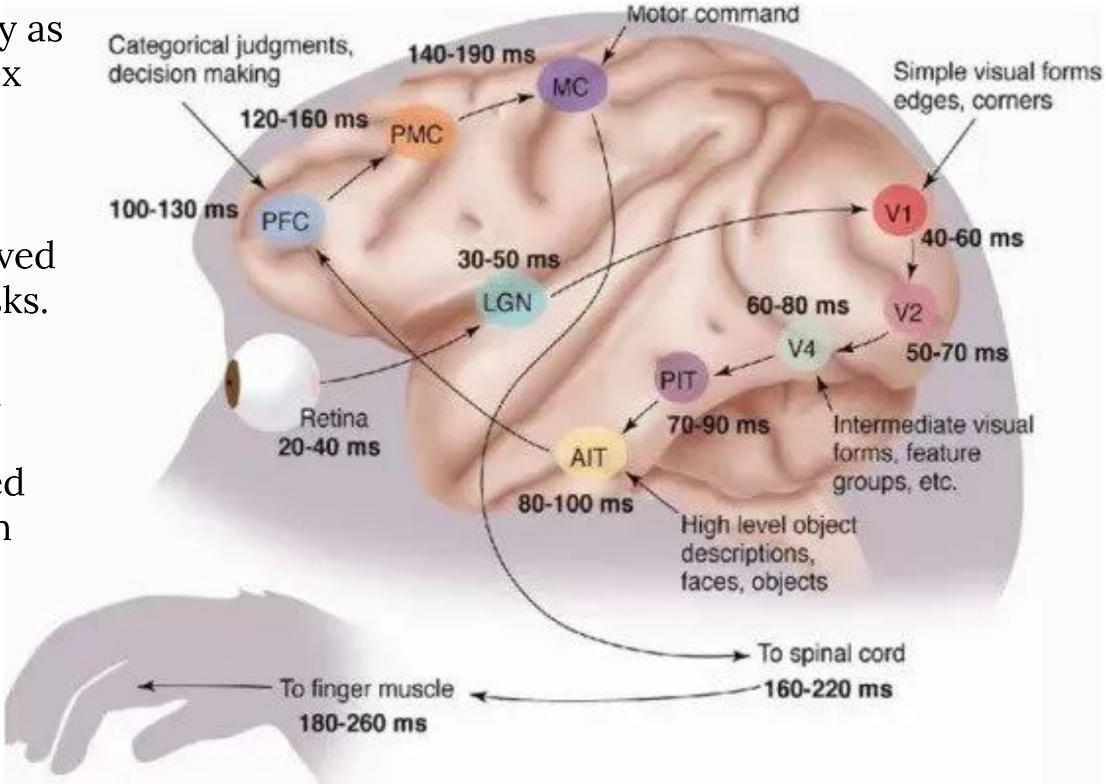


Image Source: Learning a Dictionary of Shape-Components in Visual Cortex: Comparison with Neurons, Humans and Machines by Thomas Serre (PhD Thesis)

Convolutional Neural Networks (CNNs)

- Yann LeCun et al. (1998) introduced the famous *LeNet*-5 architecture, widely used to recognize handwritten check numbers.

- It introduces two new building blocks: *convolutional layers* and *pooling layers*.

Motivation to Use CNNs

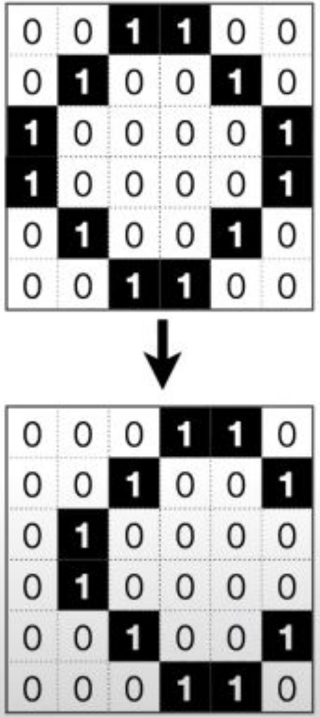
Why not simply use a regular deep neural network with fully connected layers for image recognition tasks?

- It breaks down for larger images because of the huge number of parameters it requires.
- For example,
  - A 100 x 100 image has 10,000 pixels, and if the first layer has just 1,000 neurons (which already severely restricts the amount of information transmitted to the next layer), this means a total of 10 million connections. And that's just the first layer.
- CNNs solve this problem using *partially connected layers* and *weight sharing*.

Motivation to Use CNNs

Why not simply use a regular deep neural network with fully connected layers for image recognition tasks?

- It breaks down even for the small shifts in the original image..
- CNNs learn features that are *translation invariant*:
    - a feature map in a convolutional layer will recognize that feature anywhere in the image: bottom-left, top-right, …



Shifted to the right 1 pixel.

Motivation to Use CNNs

Why not simply use a regular deep neural network with fully connected layers for image recognition tasks?

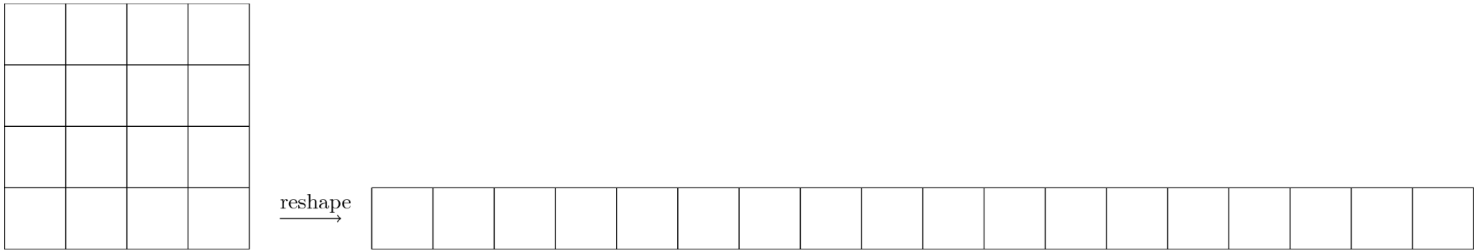
- It does not take advantage of spatial correlation in the original image..
- CNNs learn *spatial hierarchies* of features:
  - they take advantage of correlations that we observe in complex images.

Images are Rank-3 Tensors

Grayscale images:

- A grayscale image has a certain height h and width w. Therefore, it makes sense to represent them as rank 2 tensors (matrices) of integers in [0, 255].
- So far, however, we have reshaped them into rank 1 tensors (vectors):

```
mnist_x_train = mnist_x_train.reshape((60000, 28 * 28))
```



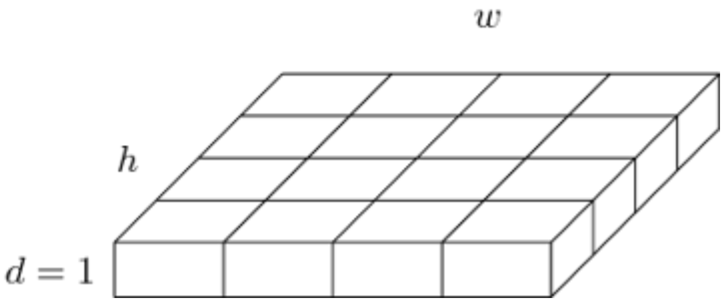
What is the disadvantage of this: what information gets destroyed?

Image Source: Dr. Derek Bridge's Class Notes

Images are Rank-3 Tensors

- Henceforth, we will not flatten them in this way.
- In fact, for consistency with colour images, we will treat grayscale images as rank 3 tensors of shape:

```
mnist_x_train = mnist_x_train.reshape((60000, 28, 28, 1))
```



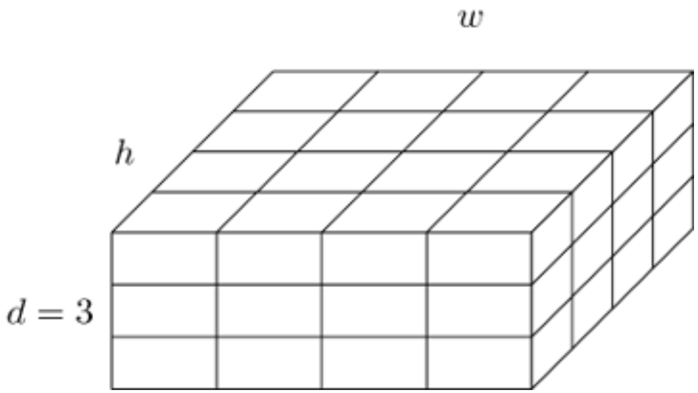
- Unlike the flattened representation, this shape makes it easier to match neurons with their corresponding inputs.

Image Source: Dr. Derek Bridge's Class Notes

Images are Rank-3 Tensors

Colour images:

- These will be rank 3 tensors: height h, width w, and channels (or depth) d.
- d = 3. why?



- Datasets of images:
  - Datasets of images (or mini-batches) will be rank 4 tensors: (m, h, w, d).
- Why will datasets of videos be rank 5 tensors?

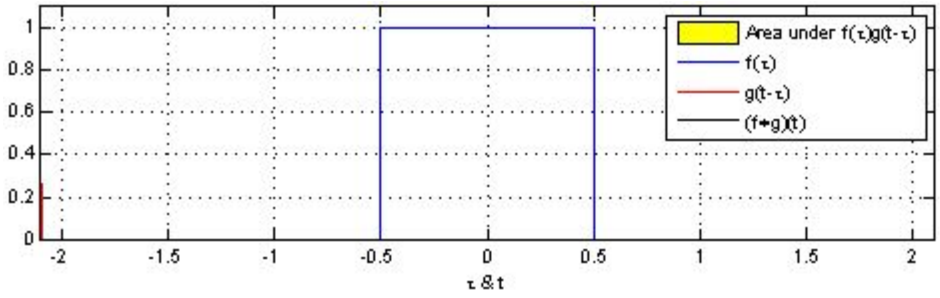
Image Source: Dr. Derek Bridge's Class Notes

Convolution: A Mathematical Operation

A convolution is a mathematical operation that slides one function over another and measures the integral of their pointwise multiplication.

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\, d\tau.$$

At each t, the convolution of f and g can be described as the area under the function $f(\tau)$ weighted by the function $g(-\tau)$ shifted by the amount t.



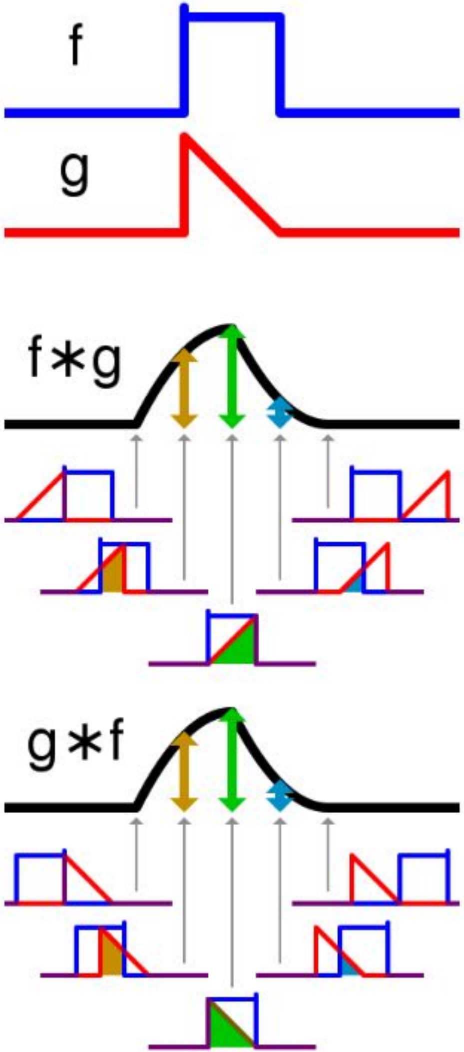
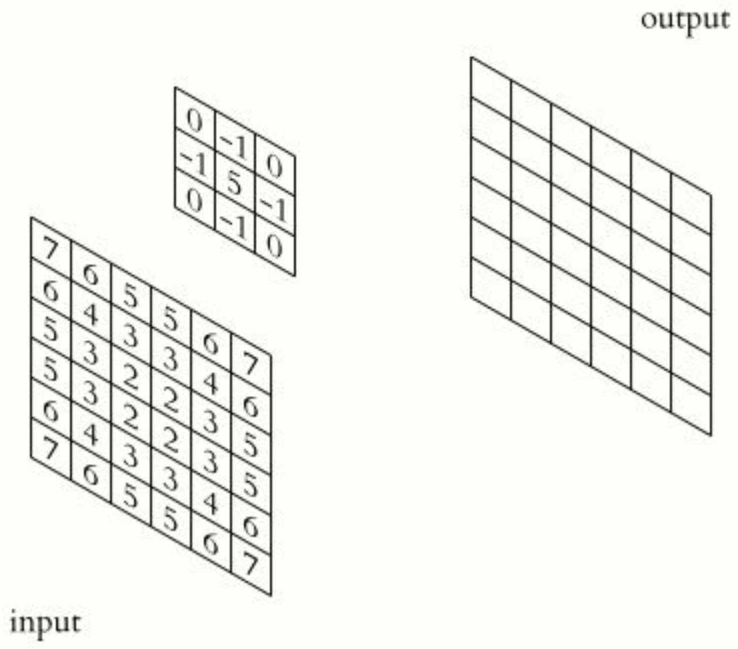


Image Source: Wikipedia - https://en.wikipedia.org/wiki/Convolution

Convolution: A Mathematical Operation

For complex-valued functions f, g defined on the set Z of integers, the discrete convolution of f and g is given by

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m],$$



Discrete Finite 2D Convolution

Image Source: Wikipedia - https://en.wikipedia.org/wiki/Convolution

Convolution Layer

- Consider a neural network whose inputs are images (each is a rank 3 tensor).
- A 2D convolutional layer is a rank 3 tensor of neurons, whose shape is (h, w, d):
  - where d, the depth, is the number of ***feature maps***
- For simplicity to begin with, let's assume d=1.

Convolution Layer

- Connections:
  - In the case of a dense layer, we saw that every neuron in a layer has connections from every neuron in the preceding layer.
  - But in the case of a convolutional layer, every neuron in a layer has connections from only a small rectangular window of neurons in the preceding layer, typically 3 x 3 or 5 x 5 or 7 x 7.

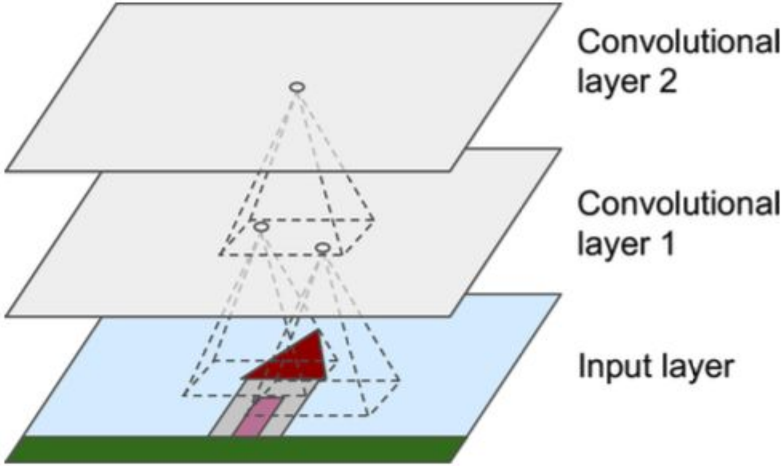


Image Source: Hands-on ML by Aurelien Geron

Connections between Convolution Layers

- Suppose the shape of the preceding layer is (28, 28, 1) and the windows (*receptive field*) in the convolutional layer are 3 x 3.

- This gives a convolutional layer whose height is 26 and whose width is 26. Why?

- A neuron located in row i, column j of a given layer is connected to the outputs of the neurons in the previous layer located in rows i to $i + f_h - 1$, columns j to $j + f_w - 1$, where f_h and f_w are the height and width of the *receptive field.*

Connections between Convolution Layers

- In order for a layer to have the same height and width as the previous layer, it is common to add zeros around the inputs. This is called *zero padding*.

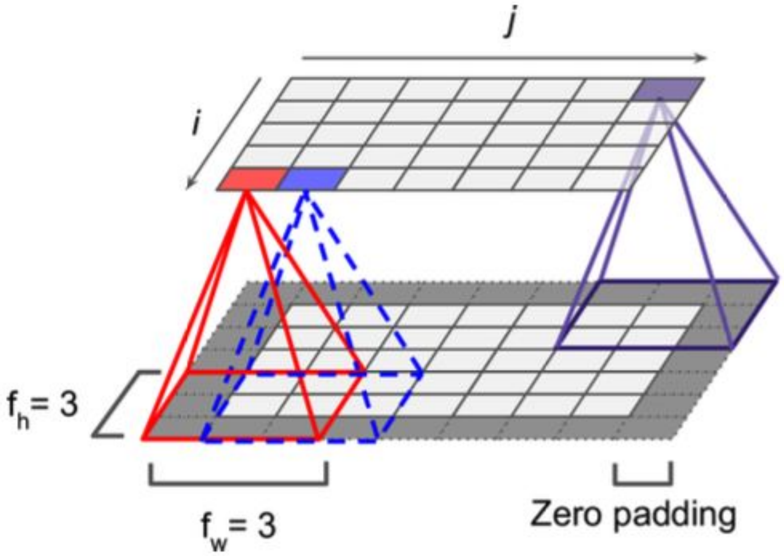
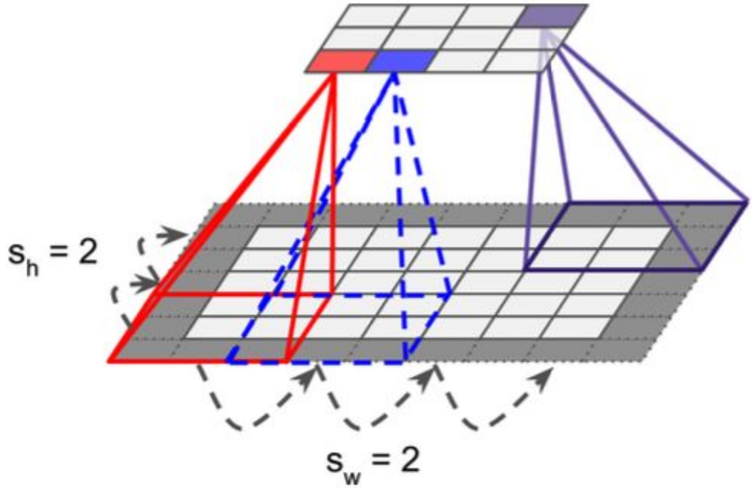


Image Source: Hands-on ML by Aurelien Geron

Reducing dimensionality using a Stride

- It is also possible to connect a large input layer to a much smaller layer by spacing out the receptive fields. The shift from one receptive field to the next is called the *stride*.



- A neuron located in row i, column j in the upper layer is connected to the outputs of the neurons in the previous layer located in rows $i \times s_h$ to $i \times s_h + f_h - 1$, columns $j \times s_w$ to $j \times s_w + f_w - 1$, where s_h and s_w are the vertical and horizontal strides.

Image Source: Hands-on ML by Aurelien Geron

Filters

- A neuron's weights can be represented as a small image of the size of the receptive field They are called *filters* or *convolution kernels*.
  - After initializing (through `kernel_initializer`), during training, the convolutional layer will *automatically* learn the most useful filters for its task.

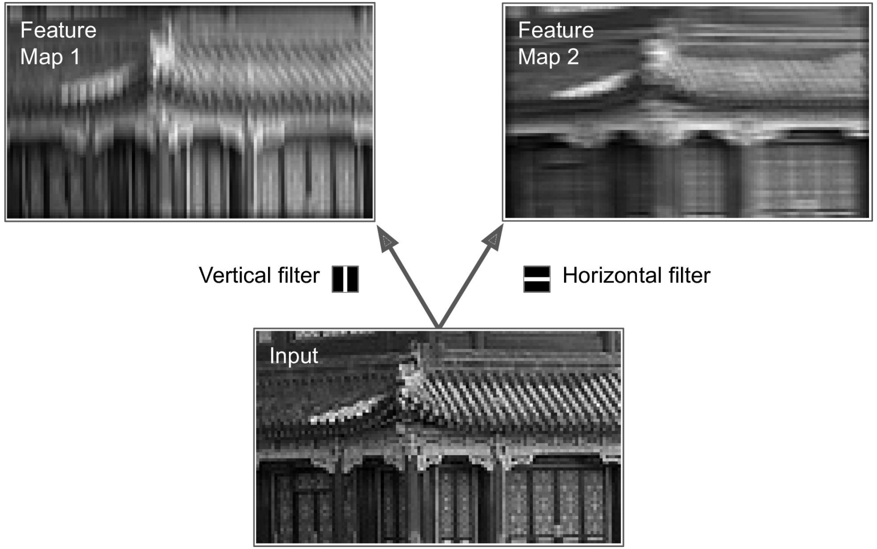


Image Source: Hands-on ML by Aurelien Geron

Filters

- A layer full of neurons using the same filter outputs a ***feature map***, i.e., within one feature map, all neurons share the same weights and bias term!

- The idea of a feature map is that it will learn a specific aspect (feature) of its input:

  - e.g. the presence of a vertical line;

  - e.g.. the presence of a pair of eyes.

Image Source: Dr. Derek Bridge’s Class Notes

CNNs: Input Image to Feature Map

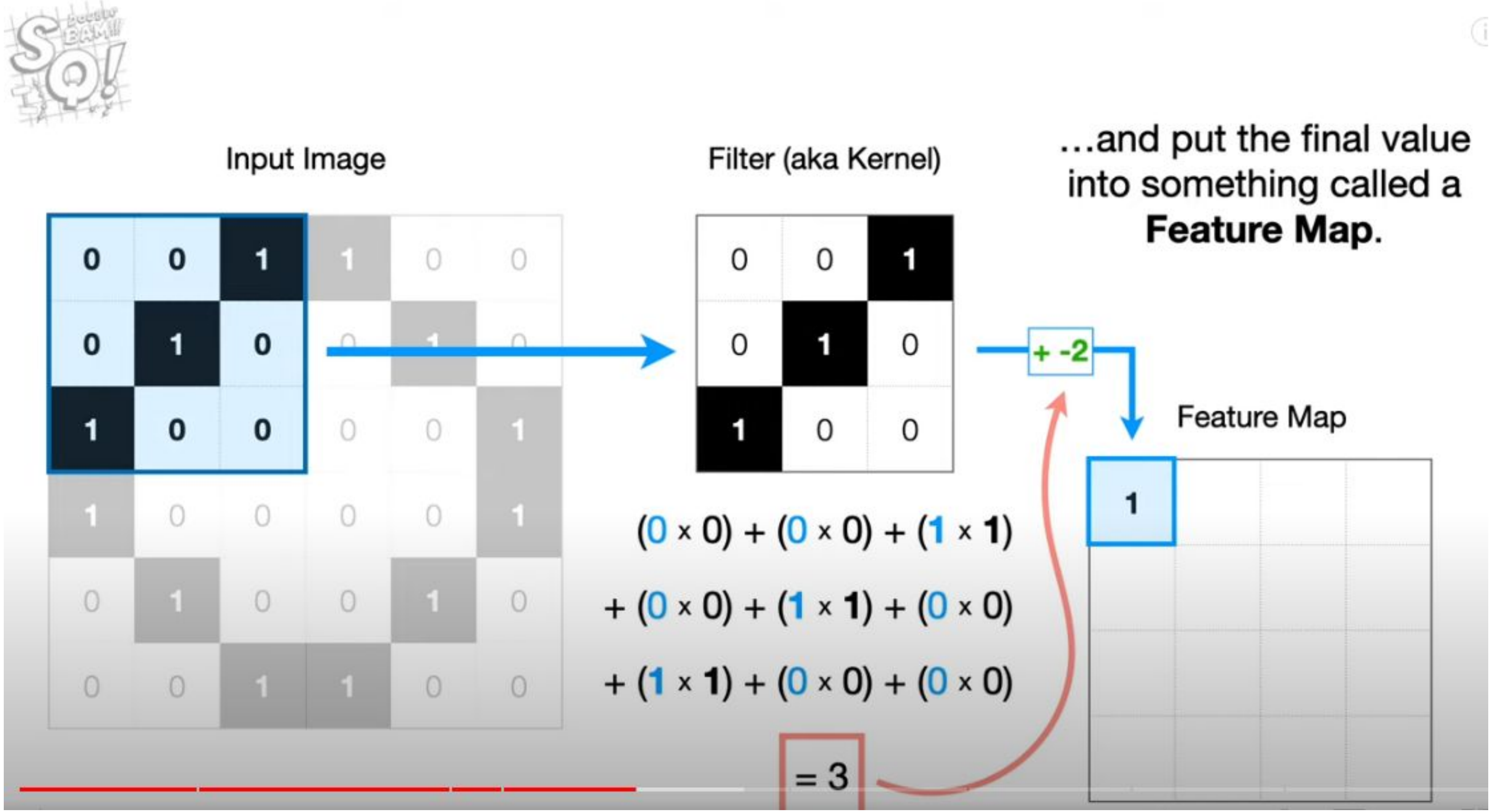


Image Source: StatQuest with Josh Stammer (on YouTube)

Stacking Multiple Feature Maps

- Now consider the case where $d > 1$: the convolutional layer comprises a stack of d feature maps.

- A neuron in a feature map in a convolutional layer is connected to a window of neurons in each of the feature maps of the previous layer
  - in the case of the first layer, in each of the channels of the input.

- This means that a feature map in one layer combines several feature maps (or channels) of the previous layer (the *spatial hierarchy*, mentioned earlier).

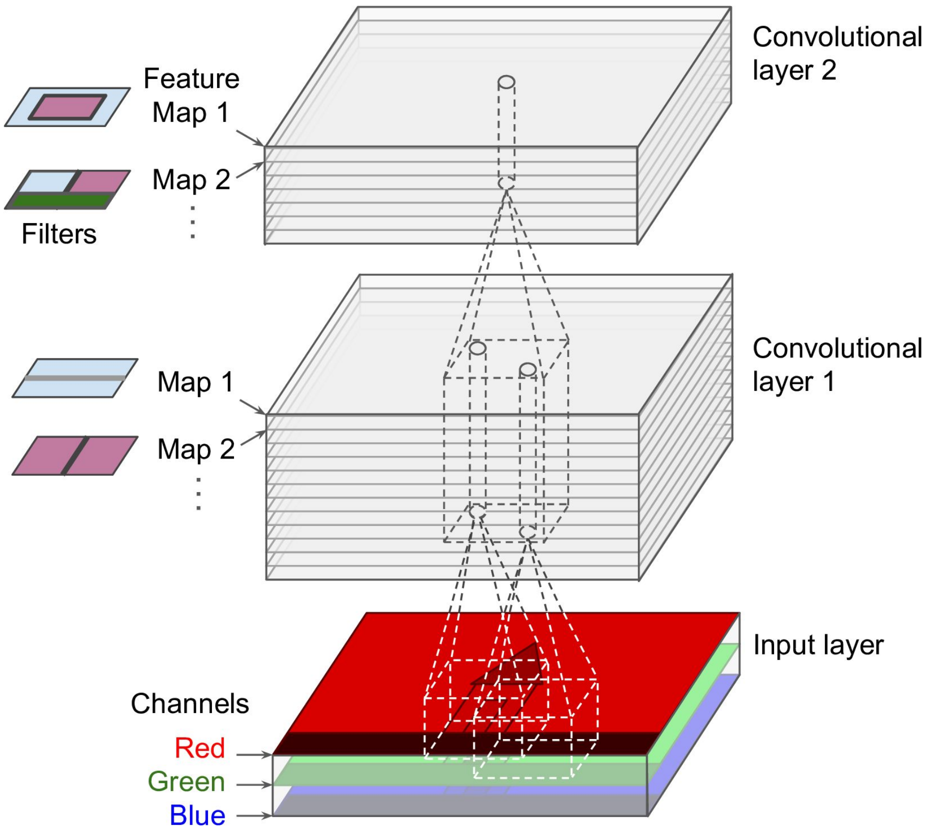


Image Source: Hands-on ML by Aurelien Geron

Stacking Multiple Feature Maps

- A neuron located in row i, column j of the feature map k in a given convolutional layer l is connected to the outputs of the neurons in the previous layer $l - 1$,
    - located in rows $i \times s_h$ to $i \times s_h + f_h - 1$ and
    - columns $j \times s_w$ *to* $j \times s_w + f_w - 1$, across all feature maps (in layer $l - 1$).
- **Note that all neurons located in the same row i and column j but in different feature maps are connected to the outputs of the exact same neurons in the previous layer.**

Stacking Multiple Feature Maps

- Computing the output of a neuron in a convolutional layer

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

- $z_{i,j,k}$ is the output of the neuron located in row i, column j in feature map k of the convolutional layer (layer l).
- As explained earlier, s_h and s_w are the vertical and horizontal strides, f_h and f_w are the height and width of the receptive field, and $f_{n'}$ is the number of feature maps in the previous layer (layer l – 1).
- $x_{i',j',k'}$ is the output of the neuron located in layer l – 1, row i', column j', feature map k' (or channel k' if the previous layer is the input layer).
- b_k is the bias term for feature map k (in layer l). You can think of it as a knob that tweaks the overall brightness of the feature map k.
- $w_{u,v,k',k}$ is the connection weight between any neuron in feature map k of the layer l and its input located at row u, column v (relative to the neuron's receptive field), and feature map k'.

Convolution Layer in Keras

- The following code creates a Conv2D layer in keras with
  - 32 filters (i.e., 32 feature maps),
  - each 3 × 3,
  - using a stride of 1 (both horizontally and vertically),
  - `SAME` padding (another padding type is `VALID`), and
  - applying the `ReLU` activation function to its outputs.

```
conv = keras.layers.Conv2D(filters=32, kernel_size=3, strides=1,
                           padding="SAME", activation="relu")
```

Convolution Layer in Keras

- **`kernel_size`** can be an integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window.
  - A single integer means the same value for all spatial dimensions.

- **`strides`** is equal to 1, however, it could also be a 1D array with 4 elements

  - batch stride (to skip some instances)
  - *vertical stride* (s_h)
  - *horizontal stride* (s_w)
  - channel stride (to skip some of the previous layer feature maps or channels)

- **`activation`** specifies the activation function to use. If we don't specify anything, no activation is applied on the feature maps.

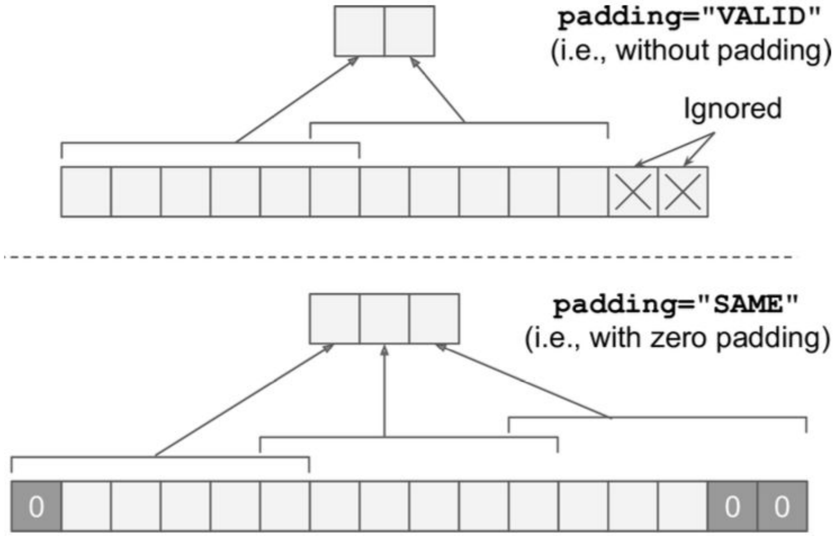
Convolution Layer in Keras

`padding` may be of two types:

- If set to "**`valid`**", the convolutional layer does not use zero padding, and may ignore some rows and columns at the bottom and right of the input image, depending on the stride.

- If set to "**`same`**", the convolutional layer uses zero padding if necessary. In this case, the number of output neurons is equal to the number of input neurons divided by the stride, rounded up.

- When `padding="same"` and `strides=1`, the output has the same size as the input.



input width: 13, filter width: 6, stride: 5

Image Source: Hands-on ML by Aurelien Geron

Pooling Layers

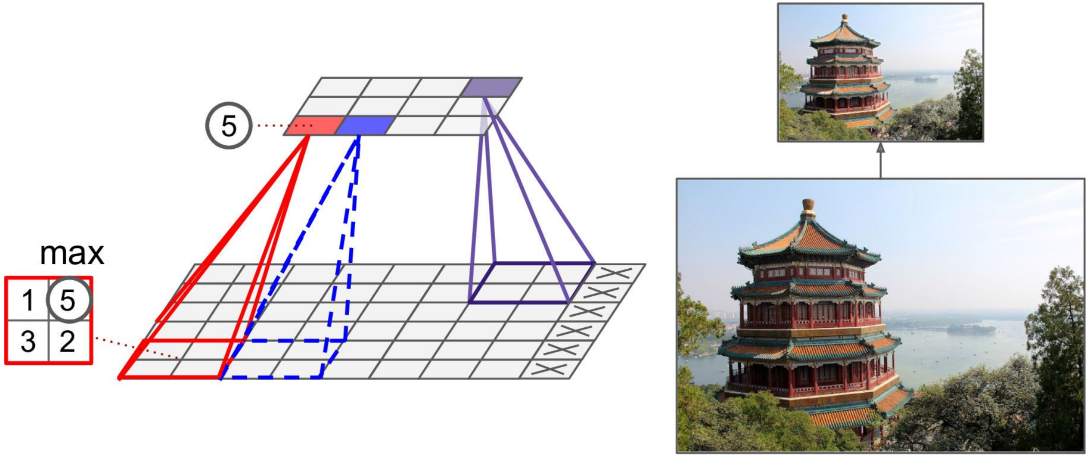
- The goal is to have a layer that shrinks the number of neurons in higher layers:
  - to reduce the amount of computation;
  - to reduce memory usage;
  - to reduce the number of parameters to be learned, thus reducing the risk of overfitting; and
  - to create a hierarchy in which higher convolutional layers contain information about the totality of the original input image.

Pooling Layers

- Again, it works on rectangular windows: neurons in the pooling layer are connected to *windows* of neurons in the previous layer
  - typically 2 x 2;
  - typically *adjacent* rather than overlapping.
- For example,
  - If the previous layer has height h and width w, and the pooling layer uses adjacent 2 x 2 pooling windows, then the pooling layer will have height $h/2$ and width $w/2$.
  - A pooling layer typically works on every input channel independently, so the output depth is the same as the input depth.

Types of Pooling Layers

- Pooling layers have no weights: nothing to learn.
- In a *max pooling* layer,
  - a neuron in the pooling layer receives the outputs of the neurons in the window in the previous layer and outputs only the largest of them.



Max pooling layer (2 × 2 pooling kernel, stride 2, no padding)

Types of Pooling Layers

- The following code creates a max pooling layer.

```
max_pool = keras.layers.MaxPool2D(pool_size=2)
```

  - To create an average pooling layer, just use `AvgPool2D` instead of `MaxPool2D`.
  - `AvgPool2D` works exactly like a max pooling layer, except it computes the *mean* rather than the *max*.

A Typical CNN Architecture

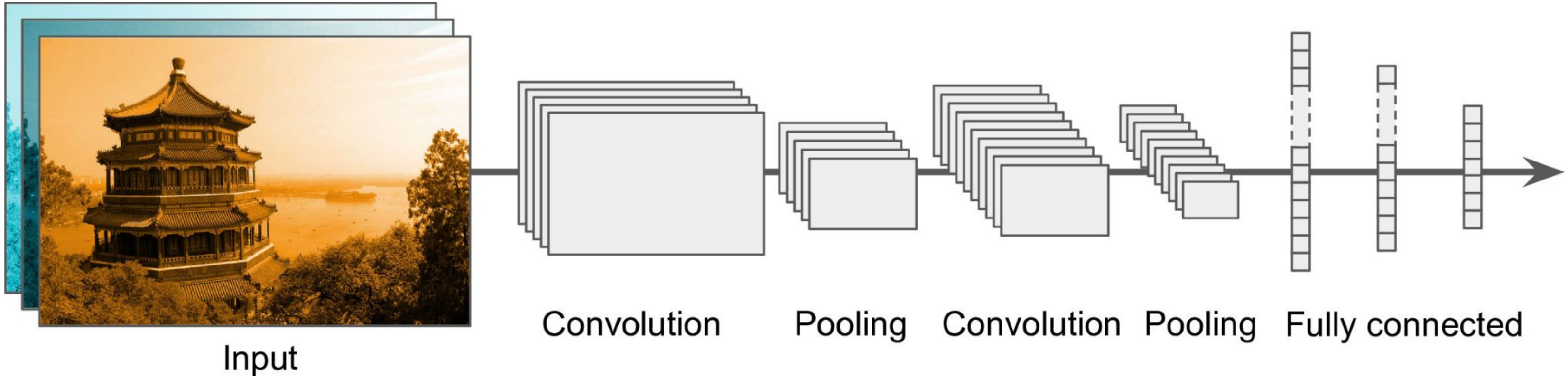


Image Source: Hands-on ML by Aurelien Geron

Typical Working of CNNs

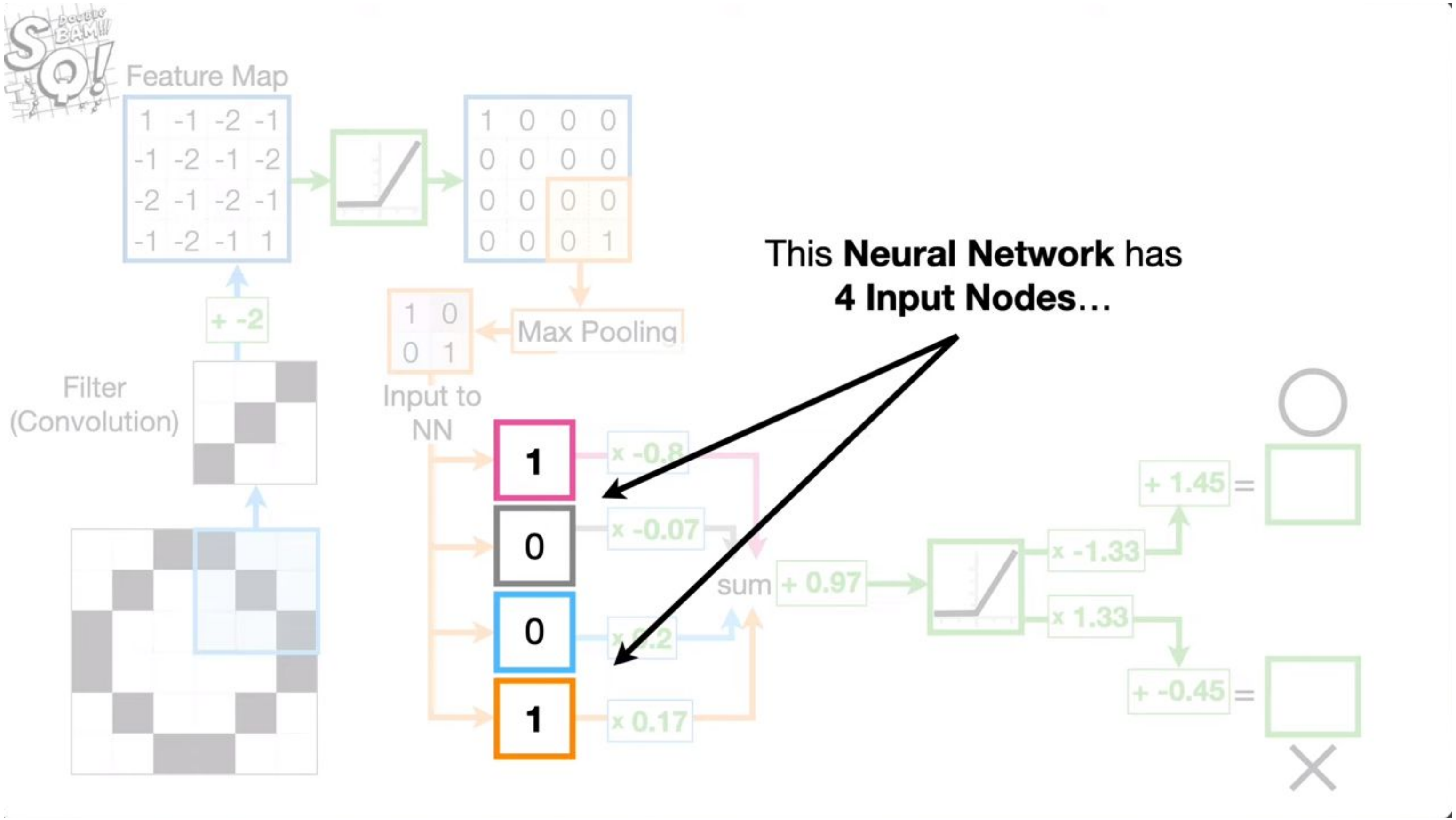


Image Source: StatQuest with Josh Stammer (on YouTube)

Implementation in Keras

- Here is how we can implement a simple CNN to tackle the fashion MNIST dataset

```python
from functools import partial

DefaultConv2D = partial(keras.layers.Conv2D, kernel_size=3, activation='relu',
                        padding="SAME")

convnet = keras.models.Sequential([
    DefaultConv2D(filters=64, kernel_size=7, input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(pool_size=2),
    DefaultConv2D(filters=128),
    DefaultConv2D(filters=128),
    keras.layers.MaxPooling2D(pool_size=2),
    DefaultConv2D(filters=256),
    DefaultConv2D(filters=256),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Flatten(),
    keras.layers.Dense(units=128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=10, activation='softmax')
])
```

```python
convnet.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_5 (Conv2D)           (None, 28, 28, 64)        3200      
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 64)        0         
 g2D)                                                            
                                                                 
 conv2d_6 (Conv2D)           (None, 14, 14, 128)       73856     
                                                                 
 conv2d_7 (Conv2D)           (None, 14, 14, 128)       147584    
                                                                 
 max_pooling2d_4 (MaxPoolin  (None, 7, 7, 128)         0         
 g2D)                                                            
                                                                 
 conv2d_8 (Conv2D)           (None, 7, 7, 256)         295168    
                                                                 
 conv2d_9 (Conv2D)           (None, 7, 7, 256)         590080    
                                                                 
 max_pooling2d_5 (MaxPoolin  (None, 3, 3, 256)         0         
 g2D)                                                            
                                                                 
 flatten_2 (Flatten)         (None, 2304)              0         
                                                                 
 dense_6 (Dense)             (None, 128)               295040    
                                                                 
 dropout_2 (Dropout)         (None, 128)               0         
                                                                 
 dense_7 (Dense)             (None, 64)                8256      
                                                                 
 dropout_3 (Dropout)         (None, 64)                0         
                                                                 
 dense_8 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 1413834 (5.39 MB)
Trainable params: 1413834 (5.39 MB)
Non-trainable params: 0 (0.00 Byte)
```

Implementation in Keras

- Here is how we can implement a simple CNN to tackle the fashion MNIST dataset

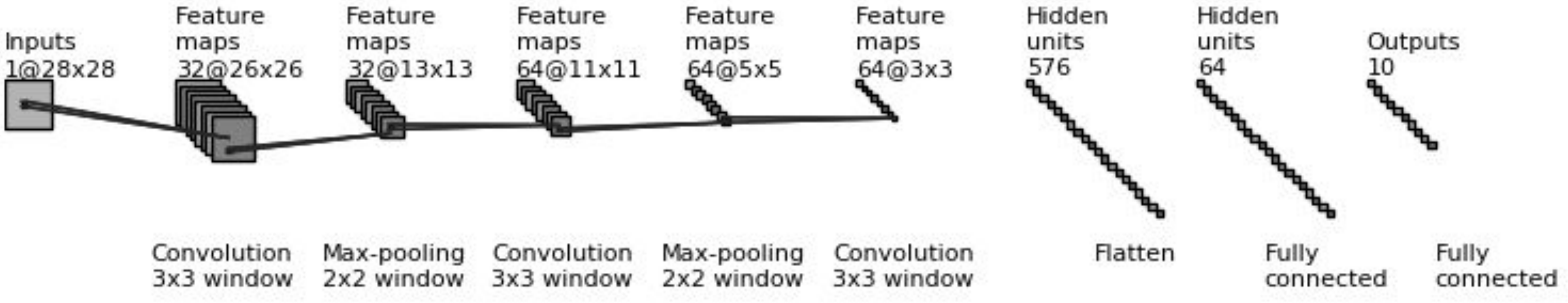
```
convnet.fit(X_train_full, Y_train_full, epochs=20, batch_size=32,
            verbose=0, validation_split=0.2,
            callbacks=[keras.callbacks.EarlyStopping(monitor="val_loss", patience=2, restore_best_weights=True)])
```

```
test_loss, test_acc = convnet.evaluate(X_test, Y_test)
test_acc
```

- This CNN reaches over 92% accuracy on the test set. Not the best, however, much better than the dense networks.

Check Your Understanding

- Do you understand the numbers in the code?
- Do you understand the numbers in the output of `convnet.summary()` ?
- Do you understand the diagram below?



Final Remarks on Convolution Layer

- Note how convolutional layers are computationally efficient:
    - They have fewer parameters than dense layers (although, care here, because each one is involved in a more multiplications).
    - They can be easily parallelised.
- This is one reason for their popularity.

Next lecture

Training CNNs

7th November 2023