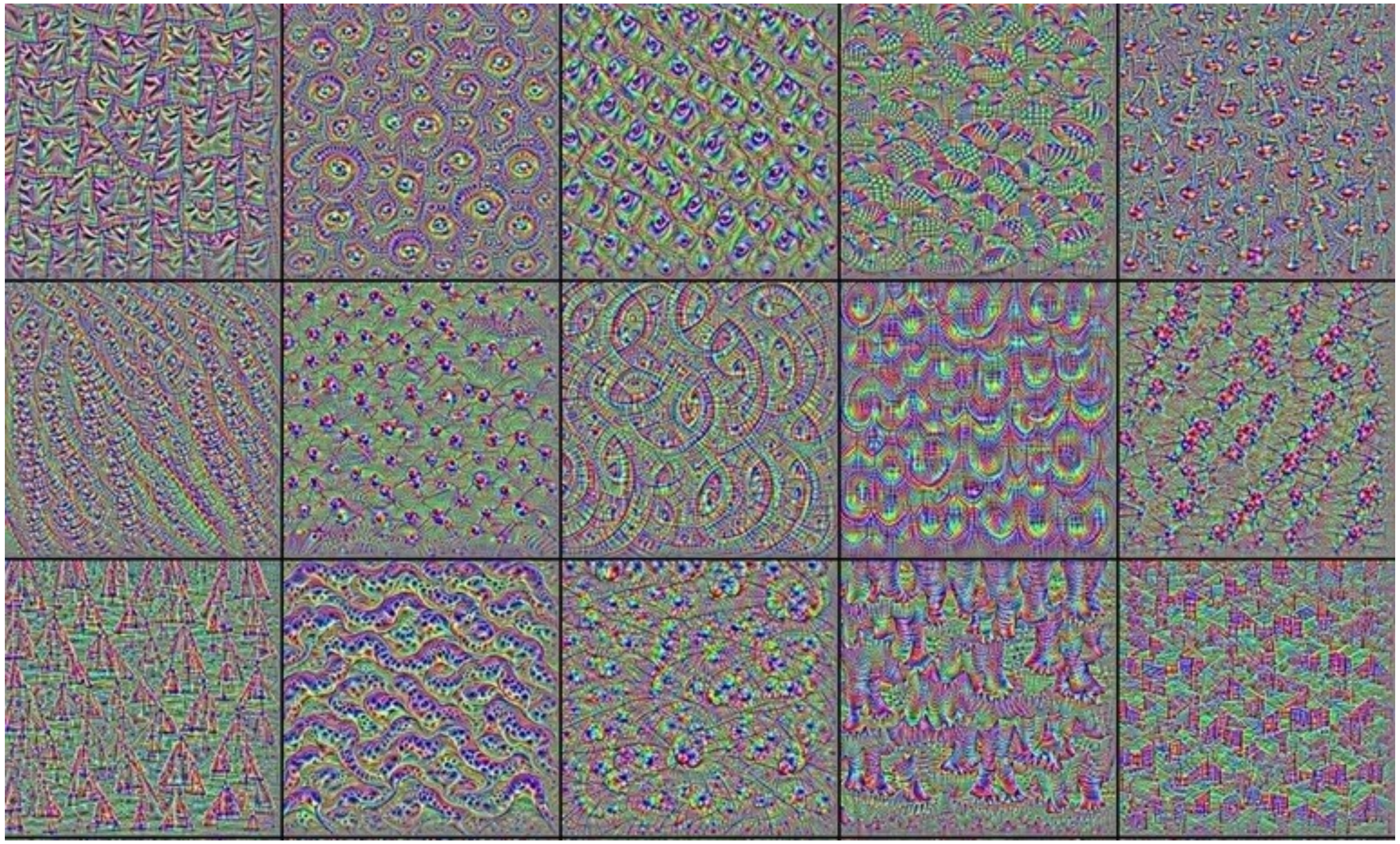
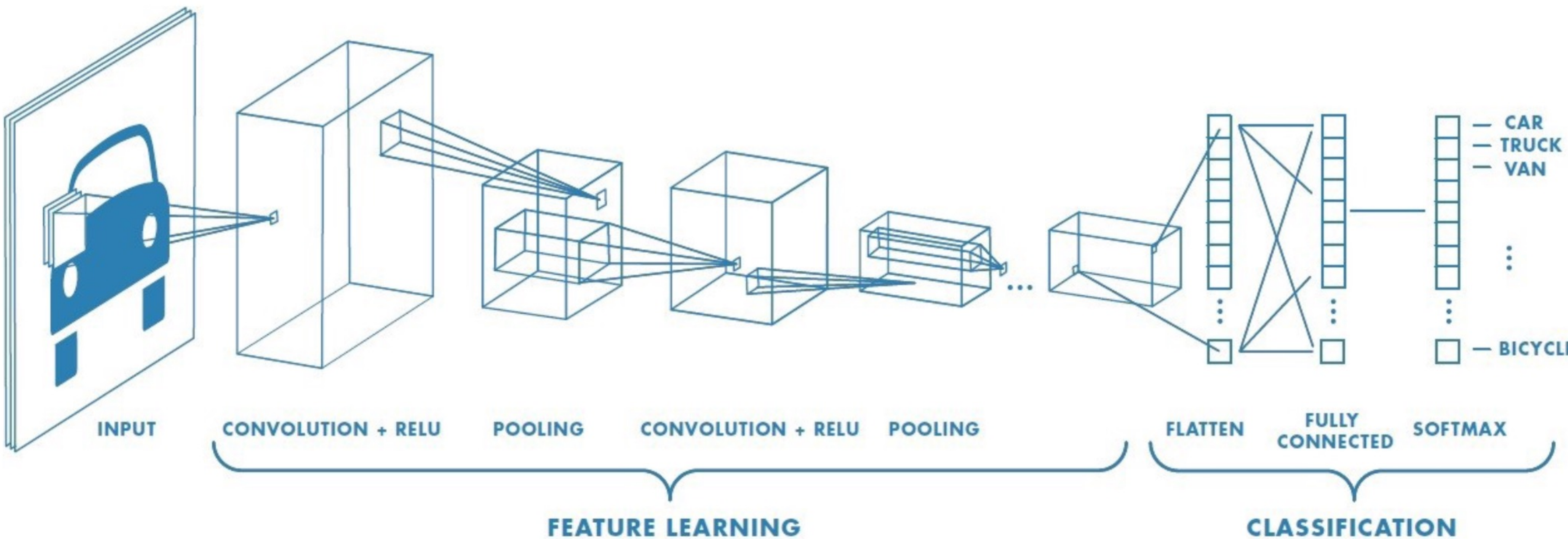
Learning with neural networks: Analysing images



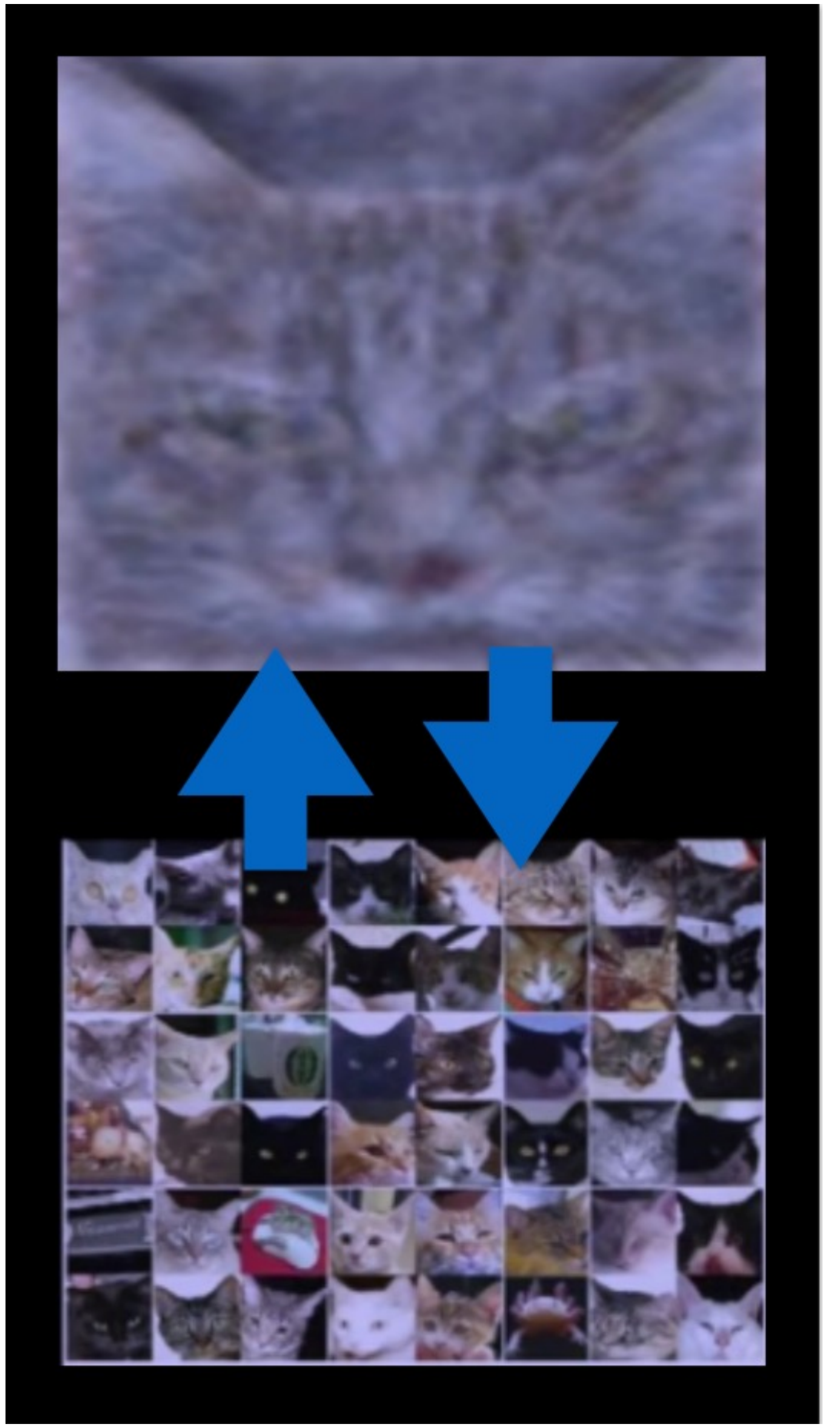
Applied Artificial Intelligence

Convolutional neural nets

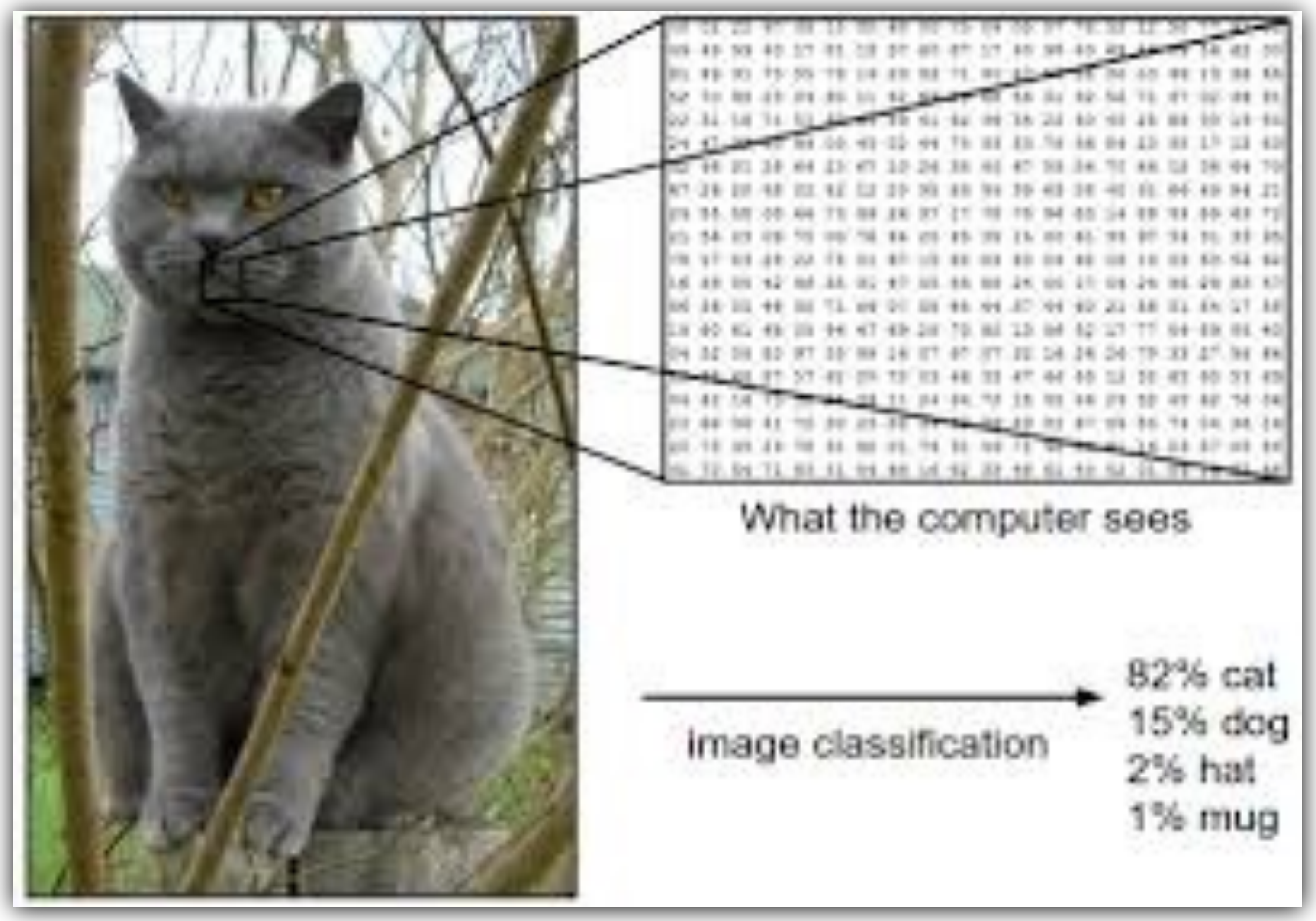


Reduce images into a set of features that makes them easier to process without losing vital information for classification.

Back to cats….



the “cat neuron”



What makes a cat a cat?

Convolutional neural nets

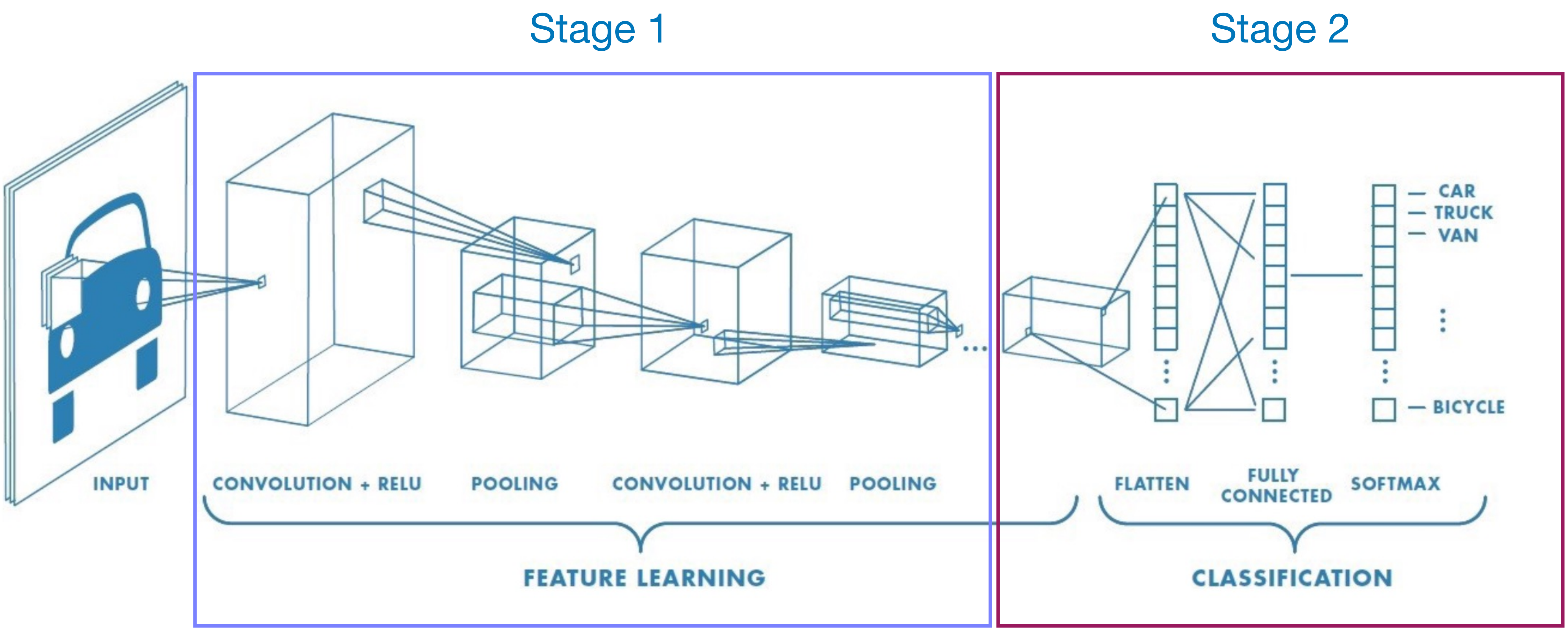
Deep learning models…

- that take an input image, assign importance to different regions and make a prediction
- Separation of *feature learning* (first stage) and *prediction* (second stage - often a “normal” neural net)

Inspired by the human visual cortex:

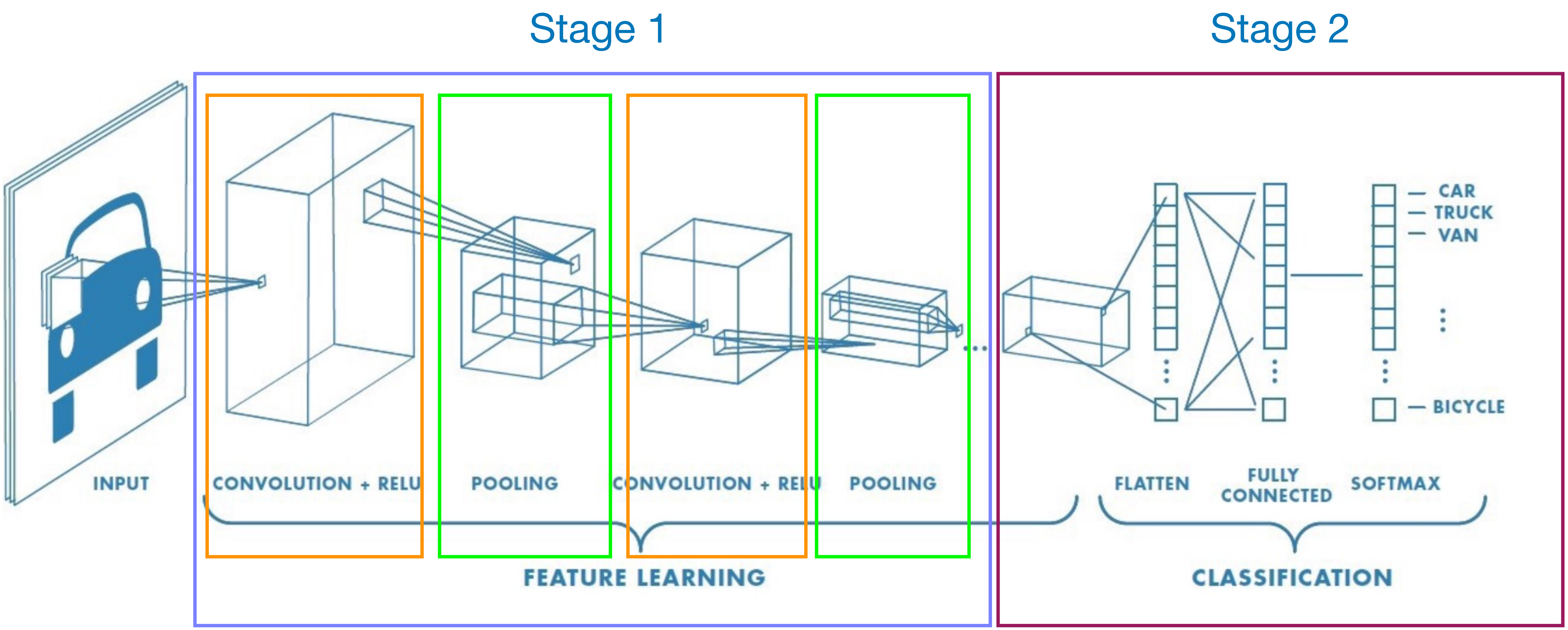
- individual neurons respond to stimuli in a particular region of relevance only (i.e. in the receptive field)

Convolutional neural nets



Reduce images into a set of features that makes them easier to process without losing vital information for classification.

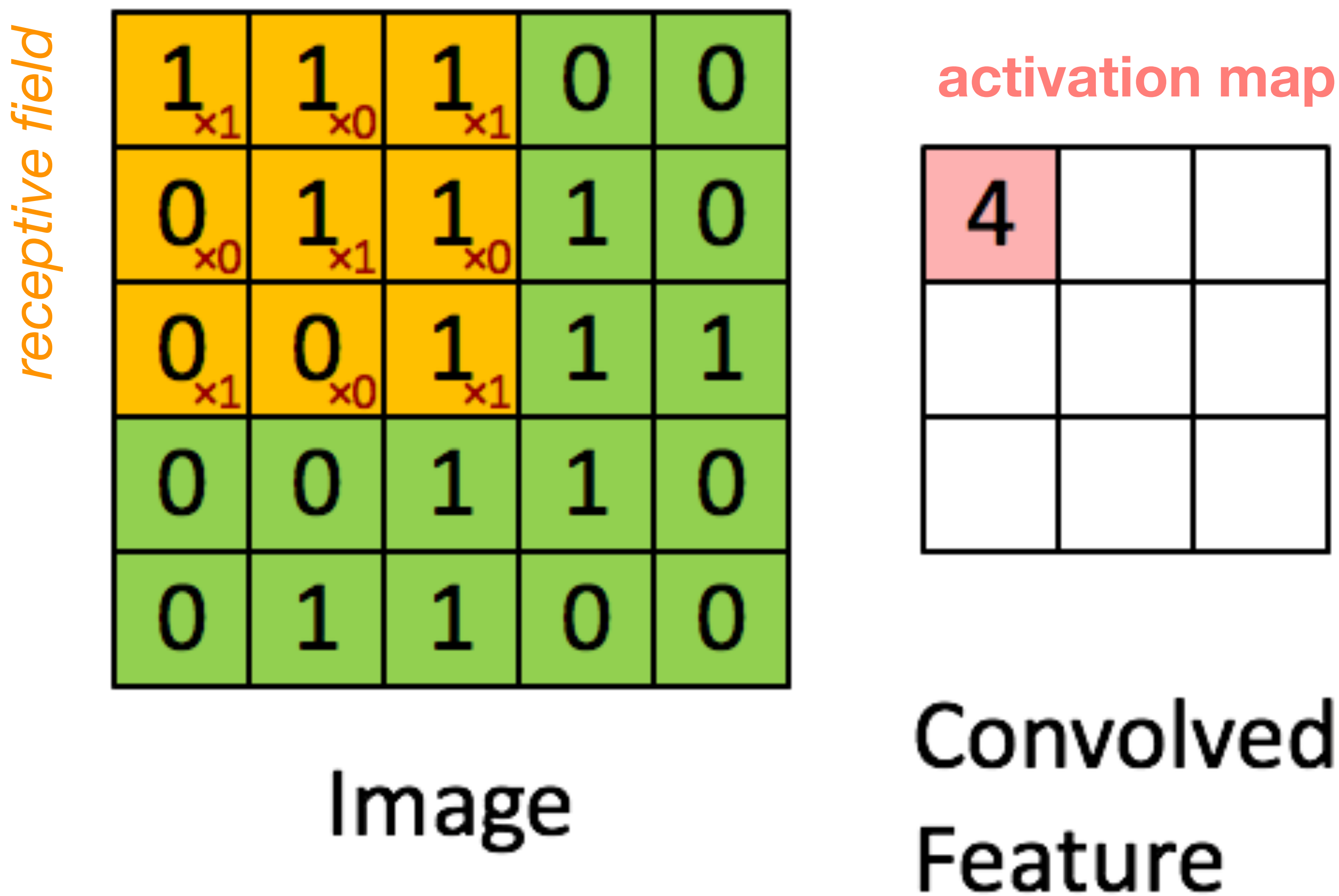
Convolutional neural nets



Reduce images into a set of features that makes them easier to process without losing vital information for classification.

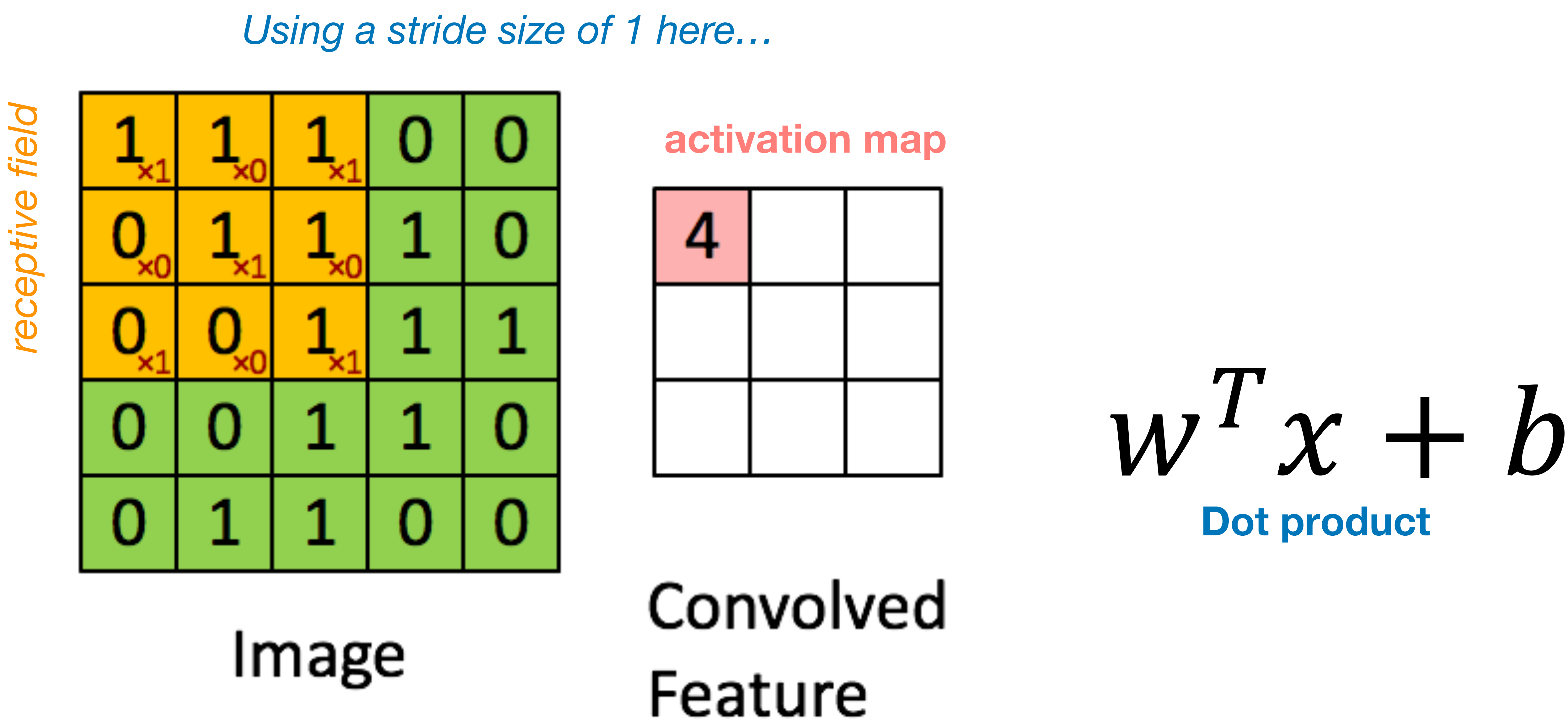
Convolutional layer

Using a stride size of 1 here…



A kernel (or filter) slides over the image spatially and computes dot products between inputs and a weight matrix. The purpose is to extract high-level features in images, such as edges, curves, colour etc.

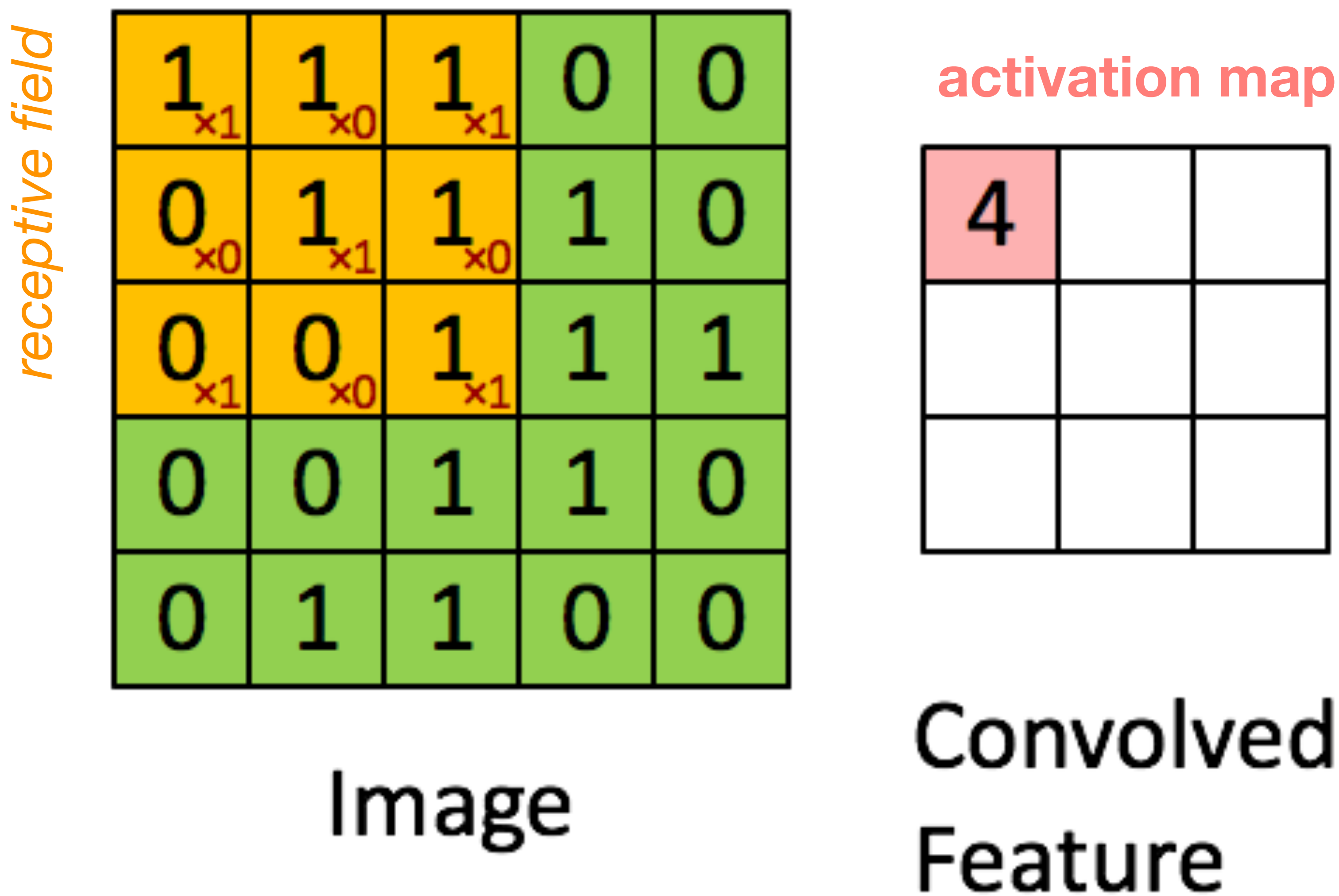
Convolutional layer



A kernel (or filter) slides over the image spatially and computes dot products between inputs and a weight matrix. The purpose is to extract high-level features in images, such as edges, curves, colour etc.

Convolutional layer

Using a stride size of 1 here…

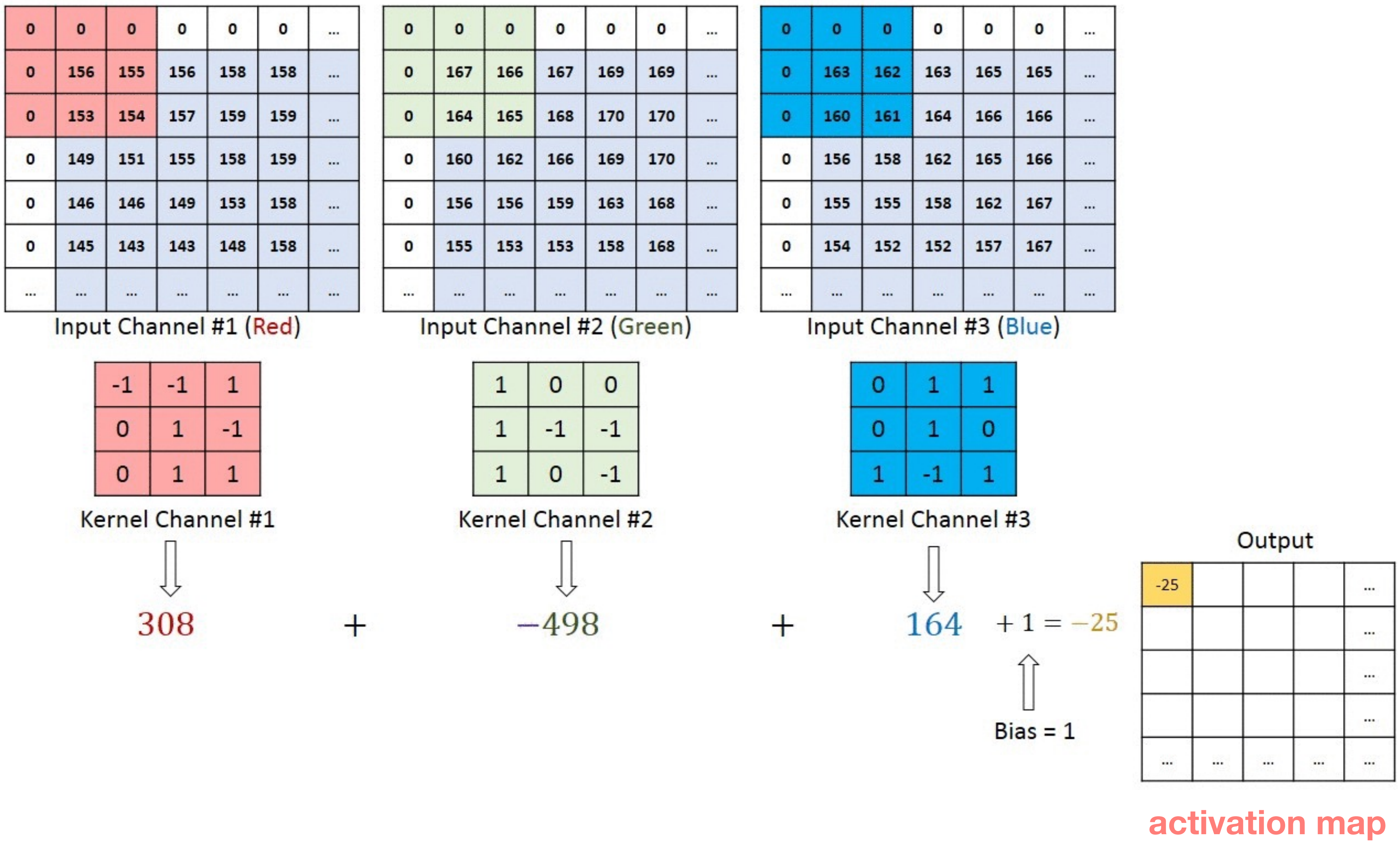



```python
import numpy as np

a = [[1, 1, 1], [0, 1, 1], [0, 0, 1]]
b = [[1, 0, 1], [0, 1, 0], [1, 0, 1]]

c = []

for i, j in enumerate(a):
    for k, l in enumerate(j):
        c.append(l * b[i][k])

print(np.sum(c))
```

Dot product

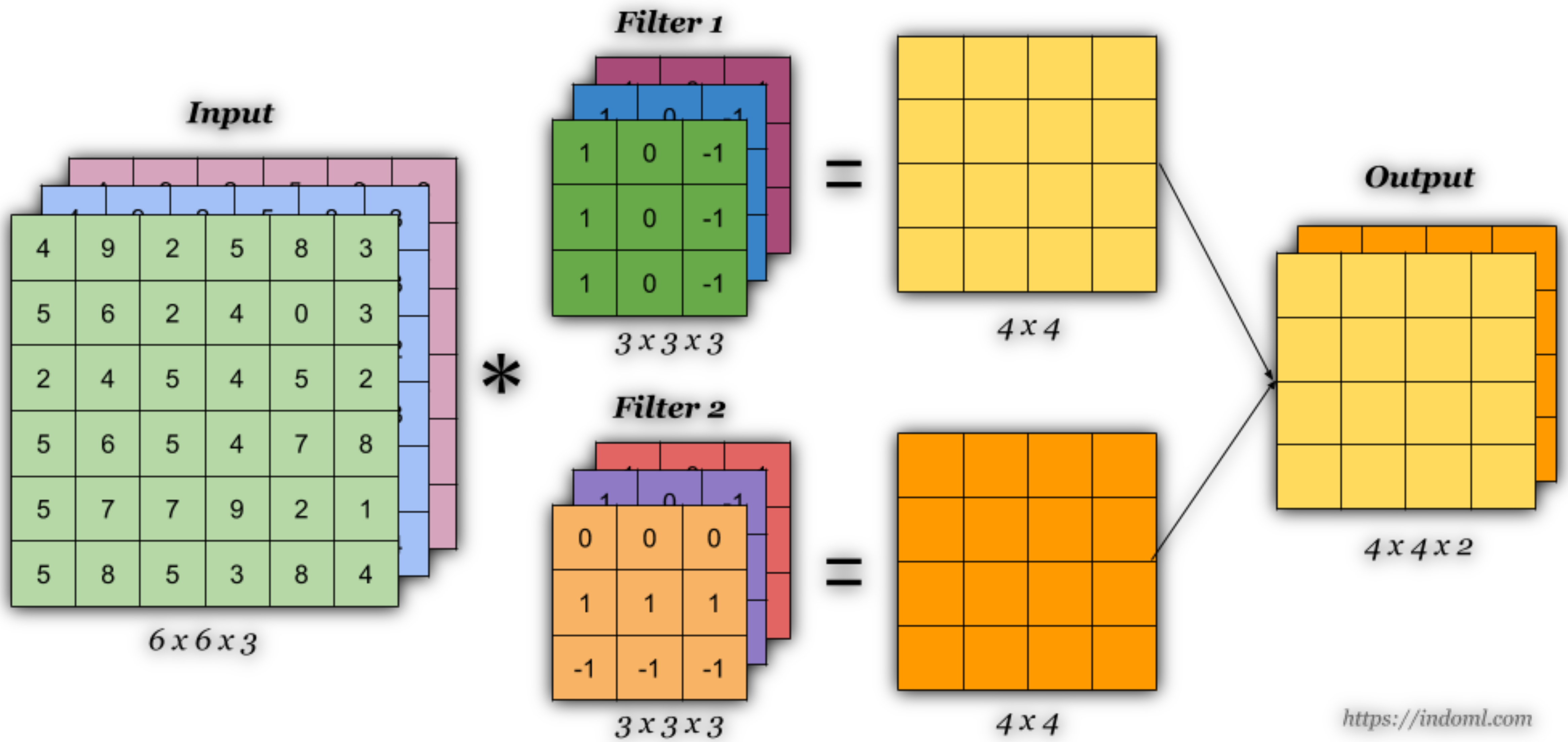
A kernel (or filter) slides over the image spatially and computes dot products between inputs and a weight matrix. The purpose is to extract high-level features in images, such as edges, curves, colour etc.

Convolutional layer



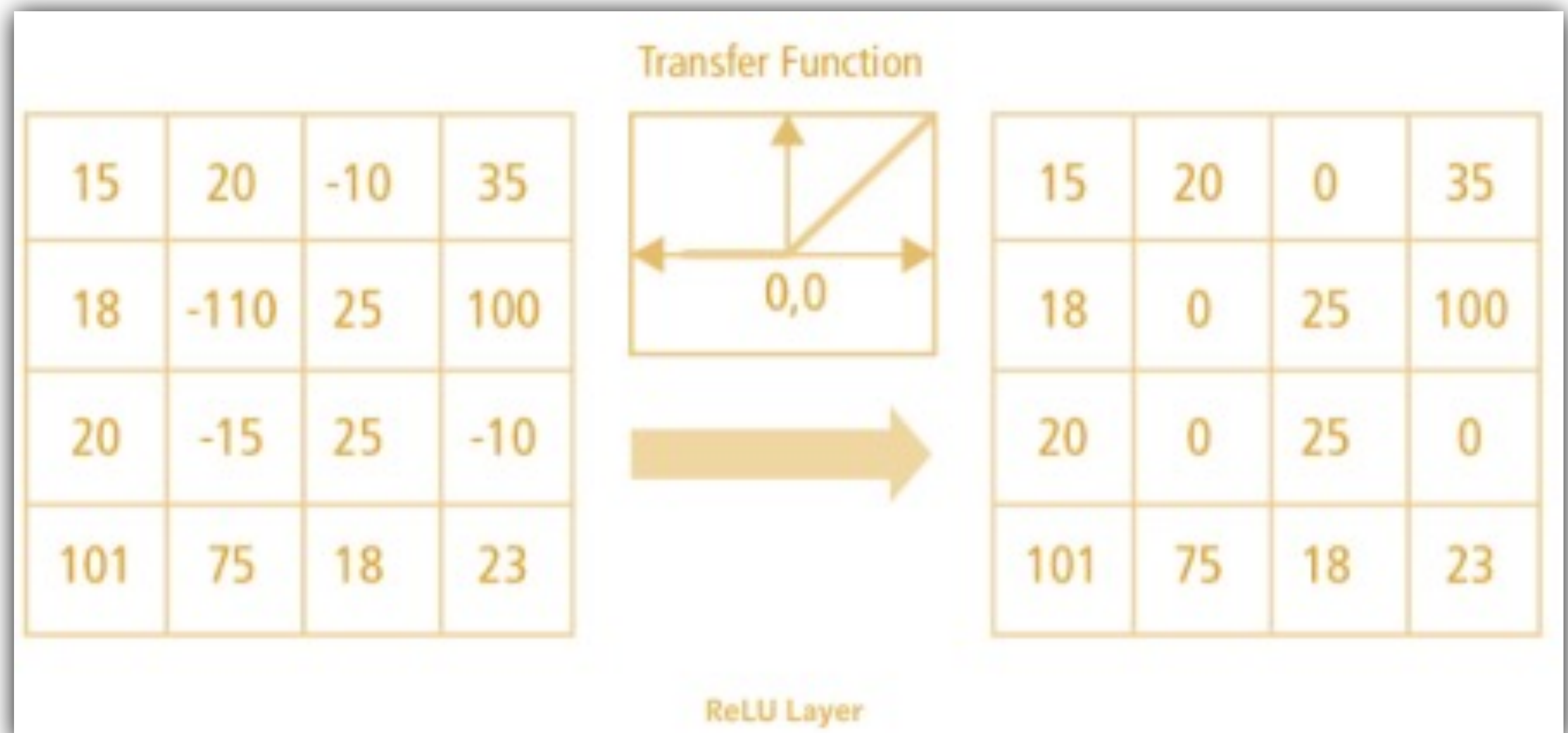
This is done for each colour channel (e.g. RGB) until a joint matrix is obtained (the activation map) and results are combined.

Convolutional layer



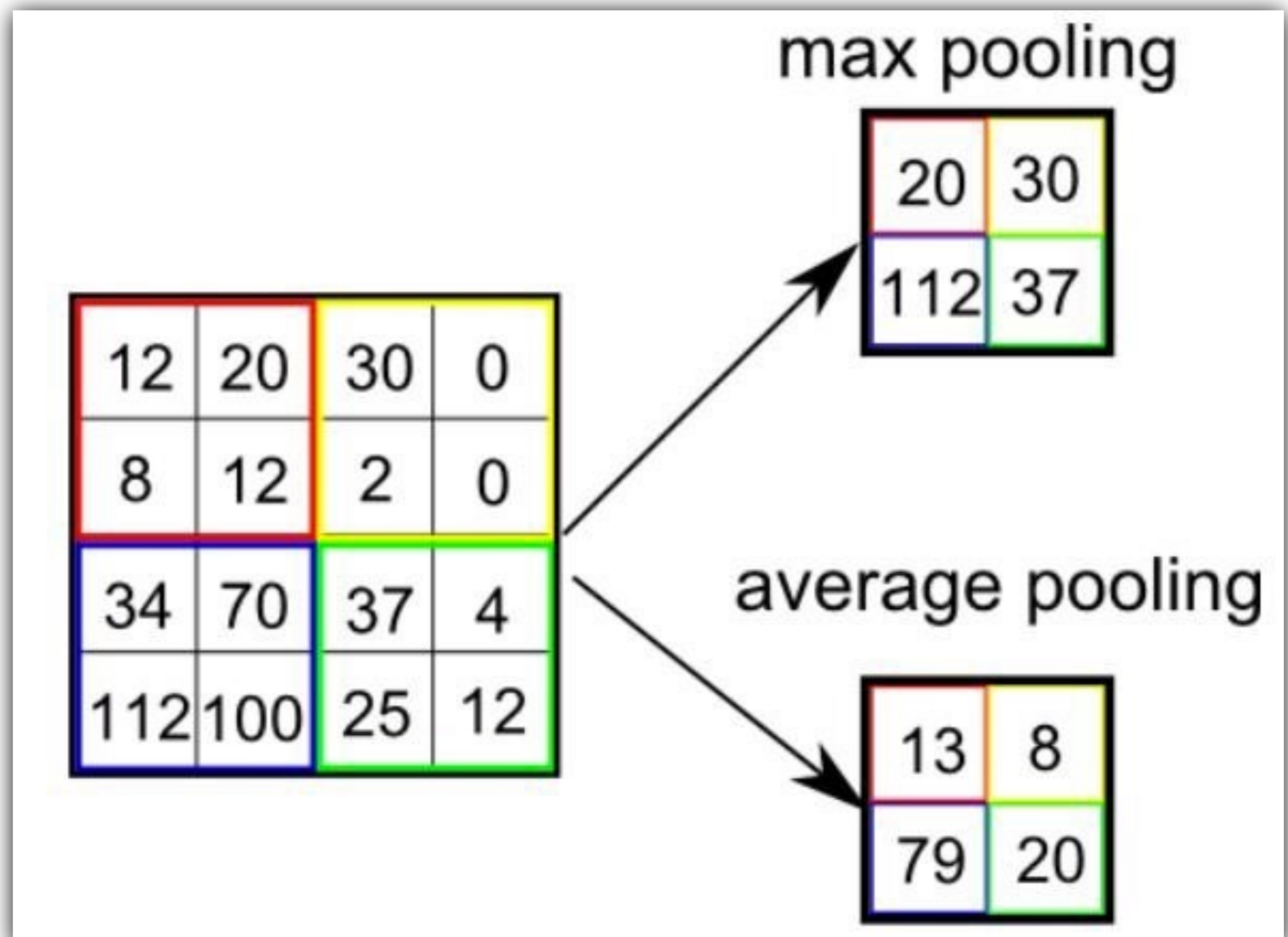
When computing convolutions, we will normally have more than one filter (with different weights). This leads to multiple activation maps that get combined later on (in the fully-connected layer).

ReLU



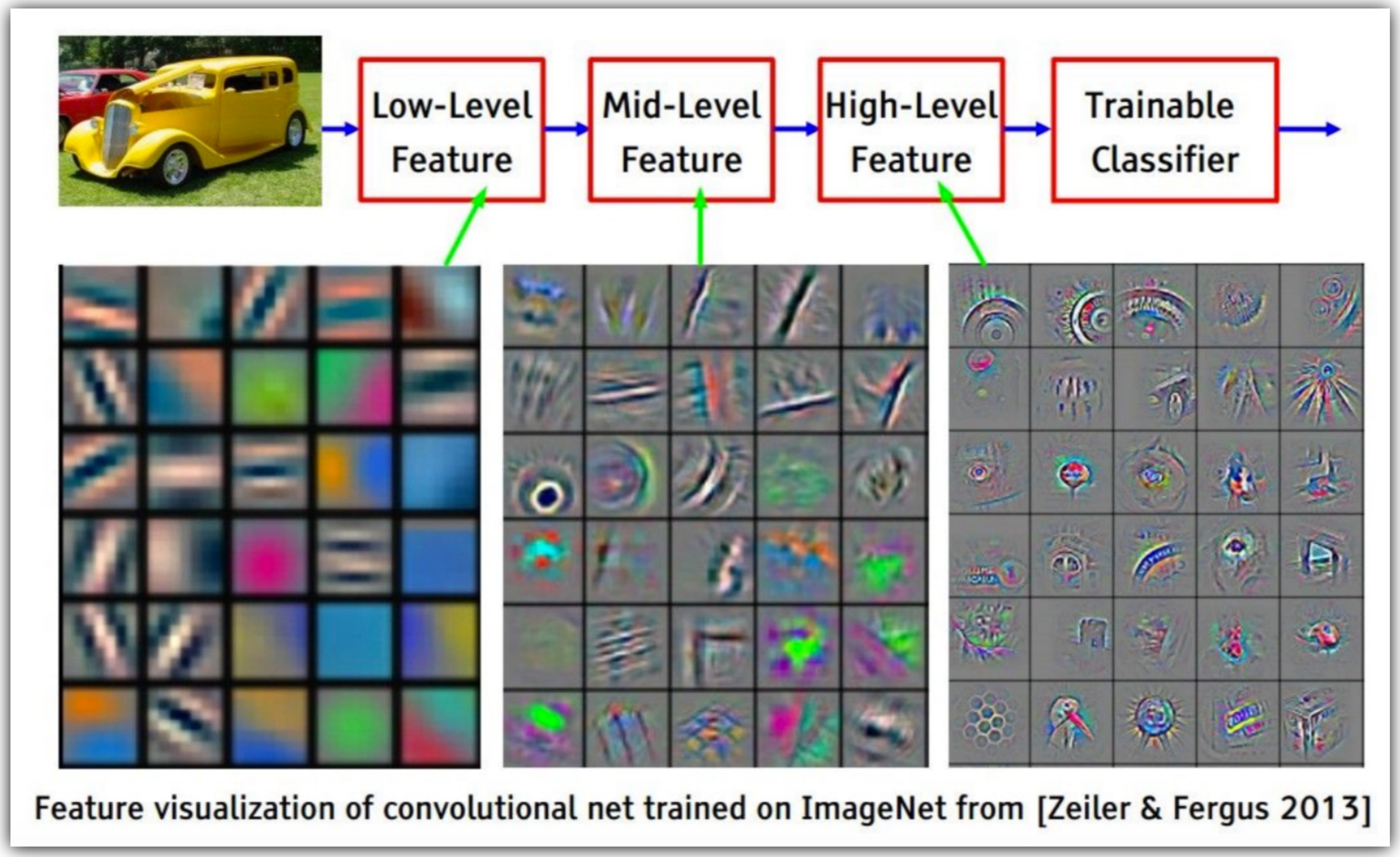
Activation maps are then passed through an activation function to account for non-linearities in the data. Performance-wise ReLU is often preferred over alternatives such as tanh or sigmoid.

Pooling Layer



The pooling layer reduces the size of the “convolved feature” and extracts dominant features. It can do this via max pooling or via average pooling (normally).

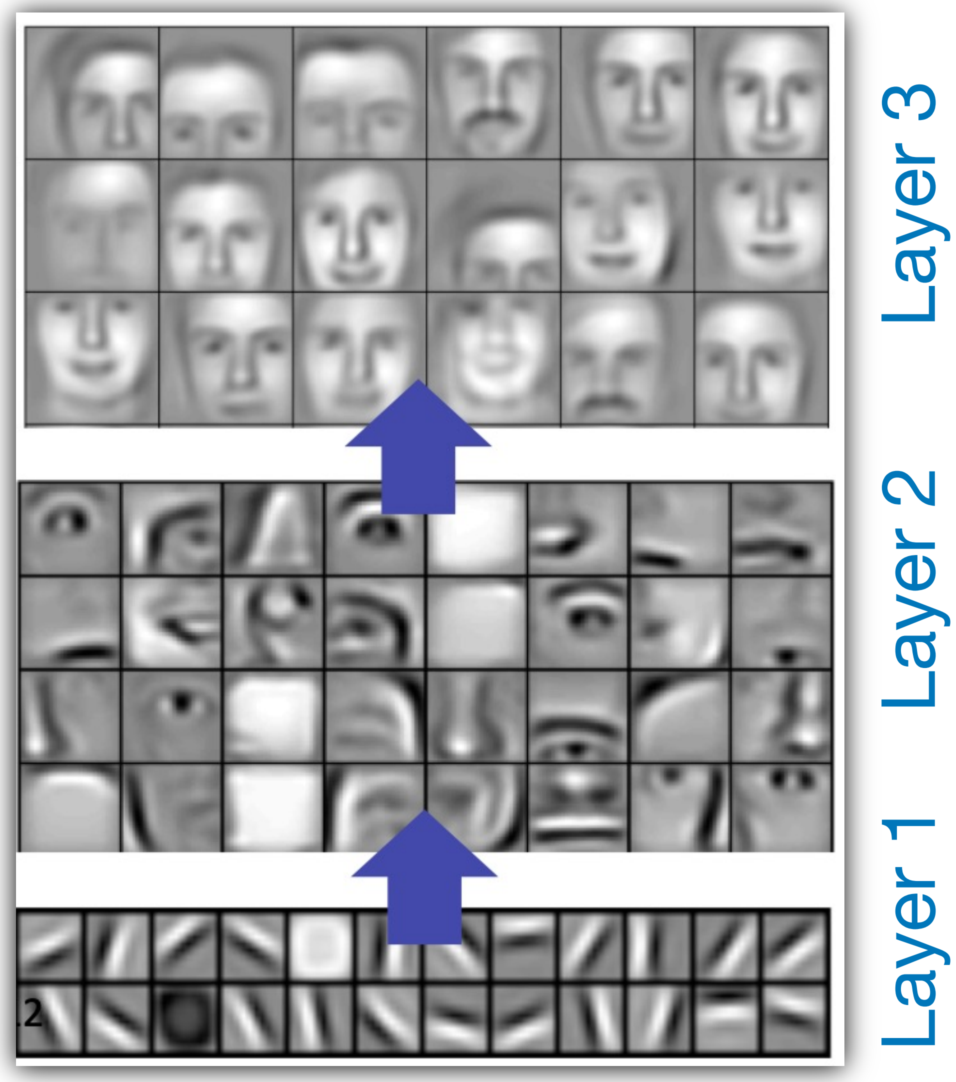
Feature learning



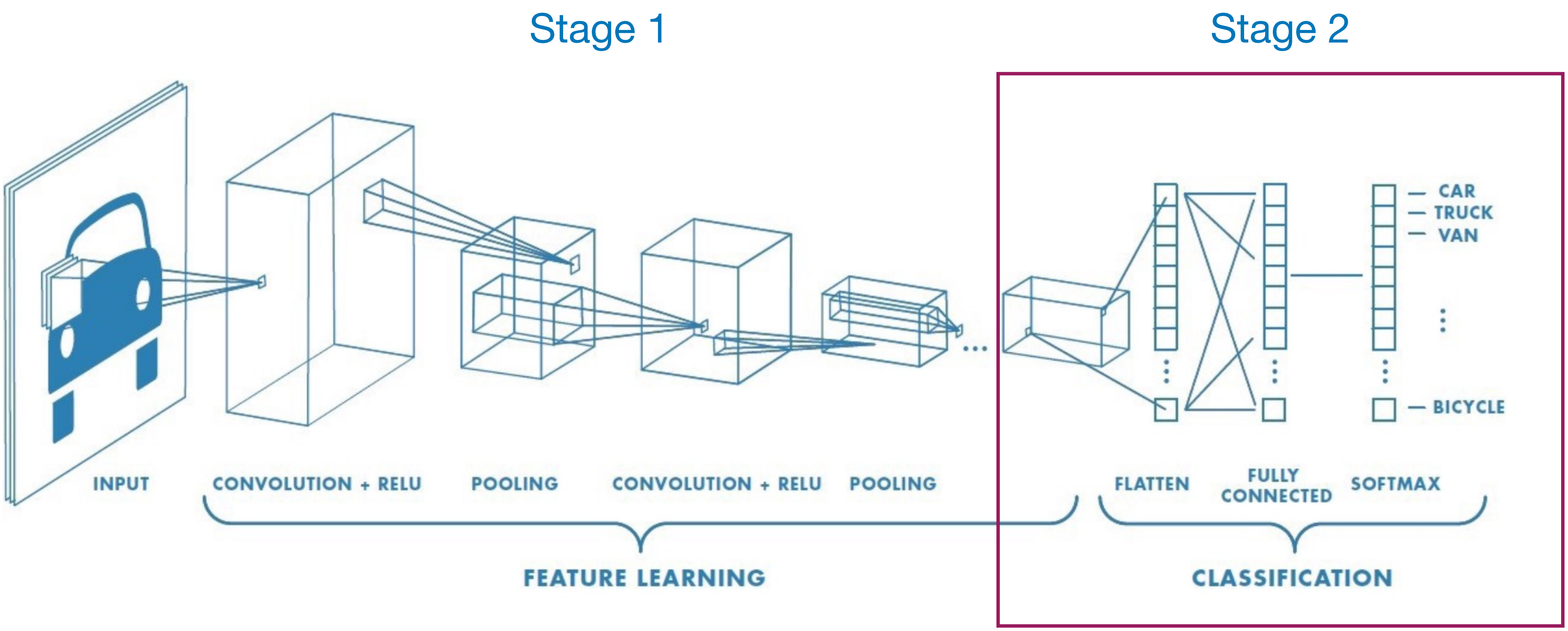
We then have an internal feature representation…

Feature learning

The feature learning stage is important as it extracts detail from the images at different level of abstraction and granularity.

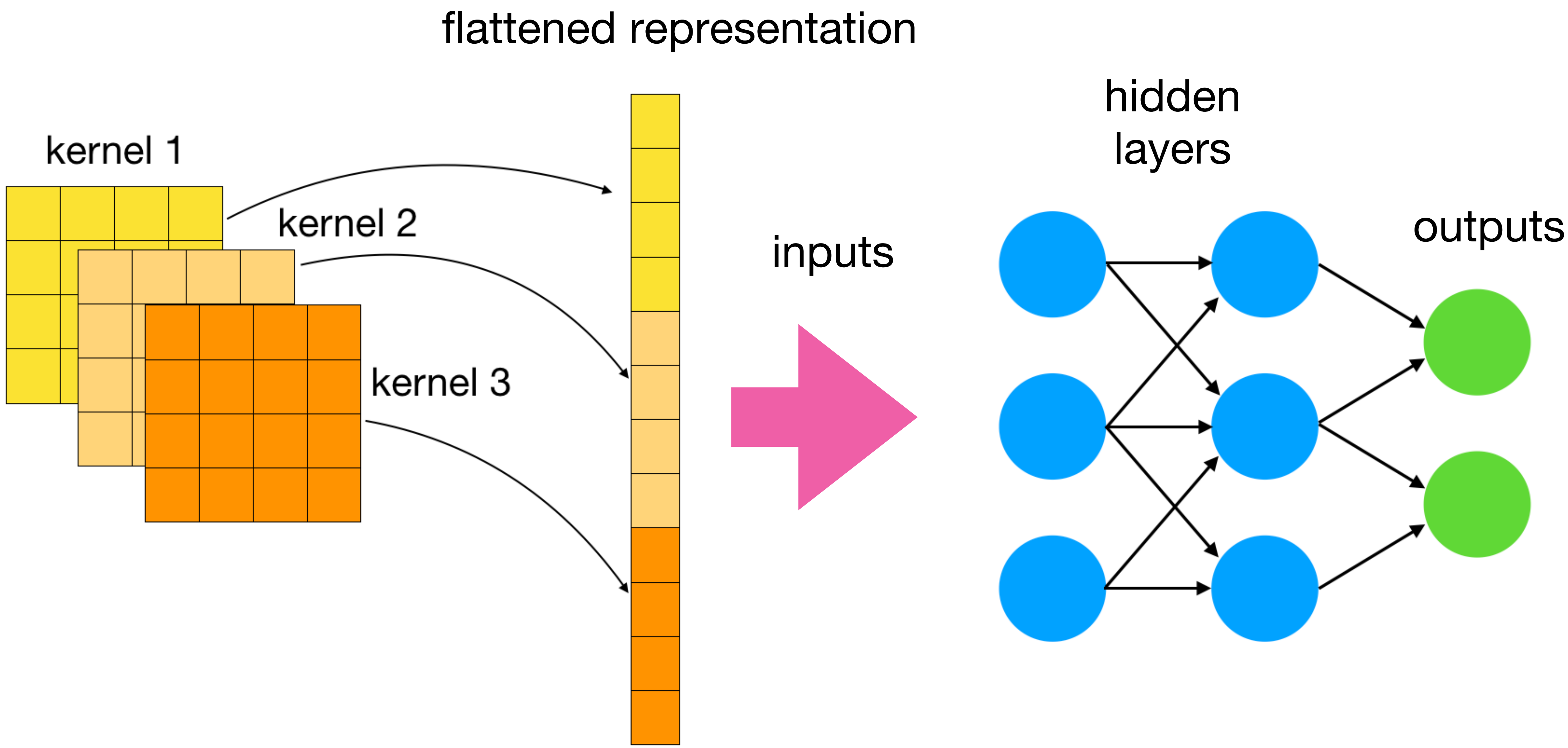


Convolutional neural nets



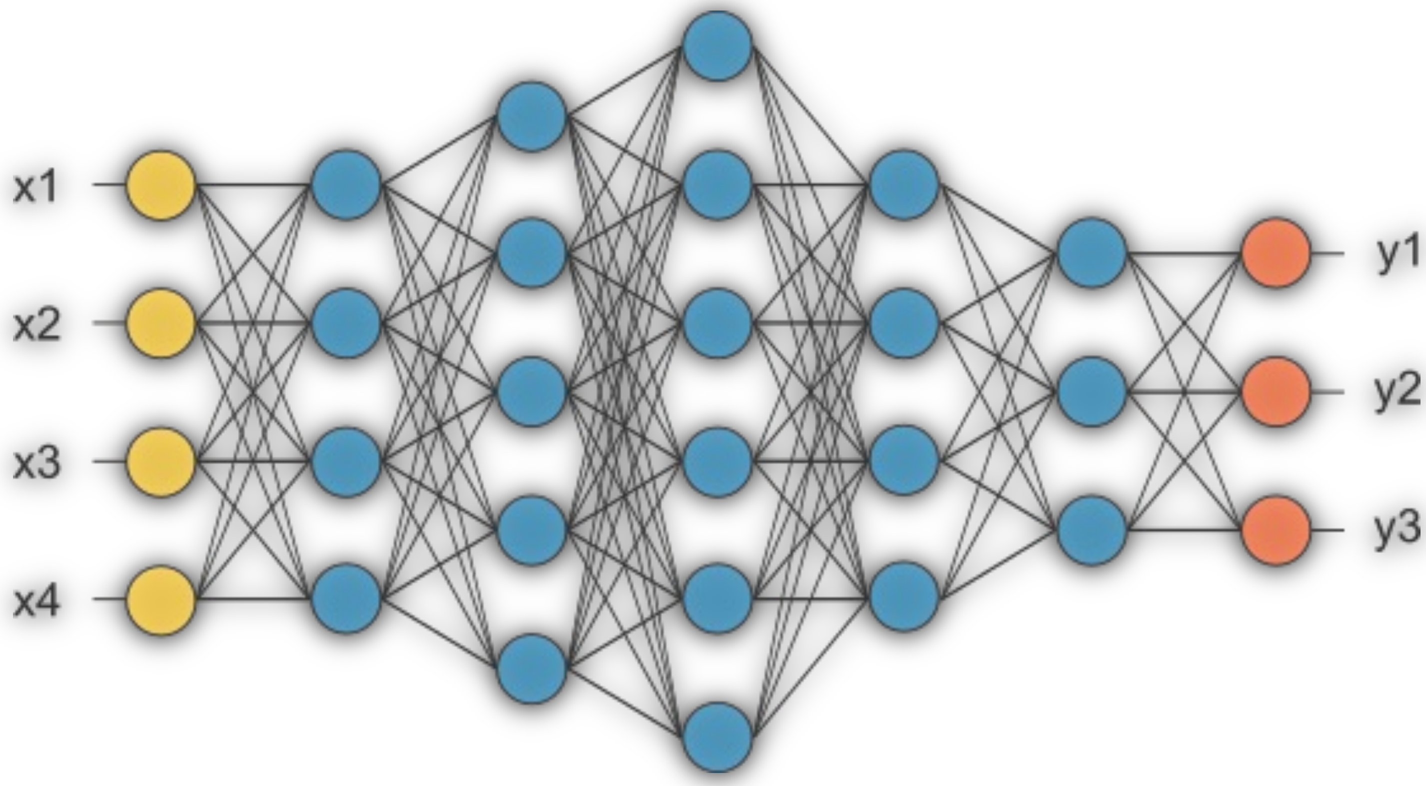
The fully-connected layer takes as input our set of learnt activation maps, flattens them into a single vector and learns to predict outputs from it.

Fully-connected Layer



Flatten the inputs (multiple activation maps, one per kernel/filter) and feed them into the final layer/s.

Fully-connected Layer



Trained using backpropagation (or variant) as discussed for artificial neural networks last week.

Algorithm 1 Backpropagation algorithm (from Wikipedia).

```
function ComputeWeights
    initialise network weights (often small random values)
    for each training exampled named ex do
        prediction = neural-net-output (network, ex) // forward pass
        actual = teacher-output (ex)
        compute error (prediction - actual) at the output units
        compute Δ_{w_h} for all weights from hidden layer to output layer // backward pass
        compute Δ_{w_i} for all weights from input layer to hidden layer // backward pass continued
        update network weights // input layer not modified by error estimate
    end for
    until all examples classified correctly or another stopping criterion is satisfied
    return the network
end function
```

CNNs with Keras

Training (60,000 examples)

CNNs with Keras

Training (60,000 examples)

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

CNNs with Keras

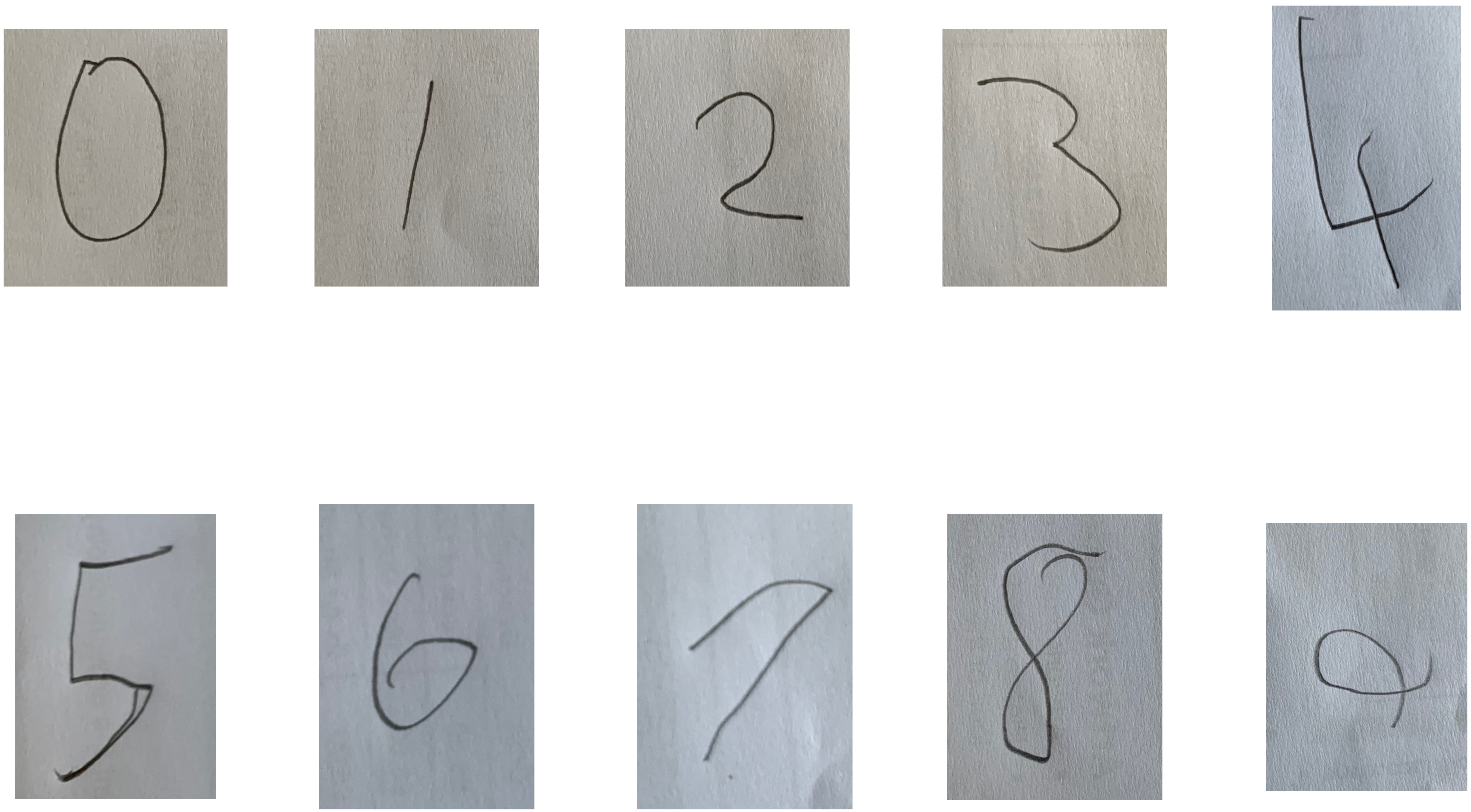
Training (60,000 examples)

99.24% after 10 epochs

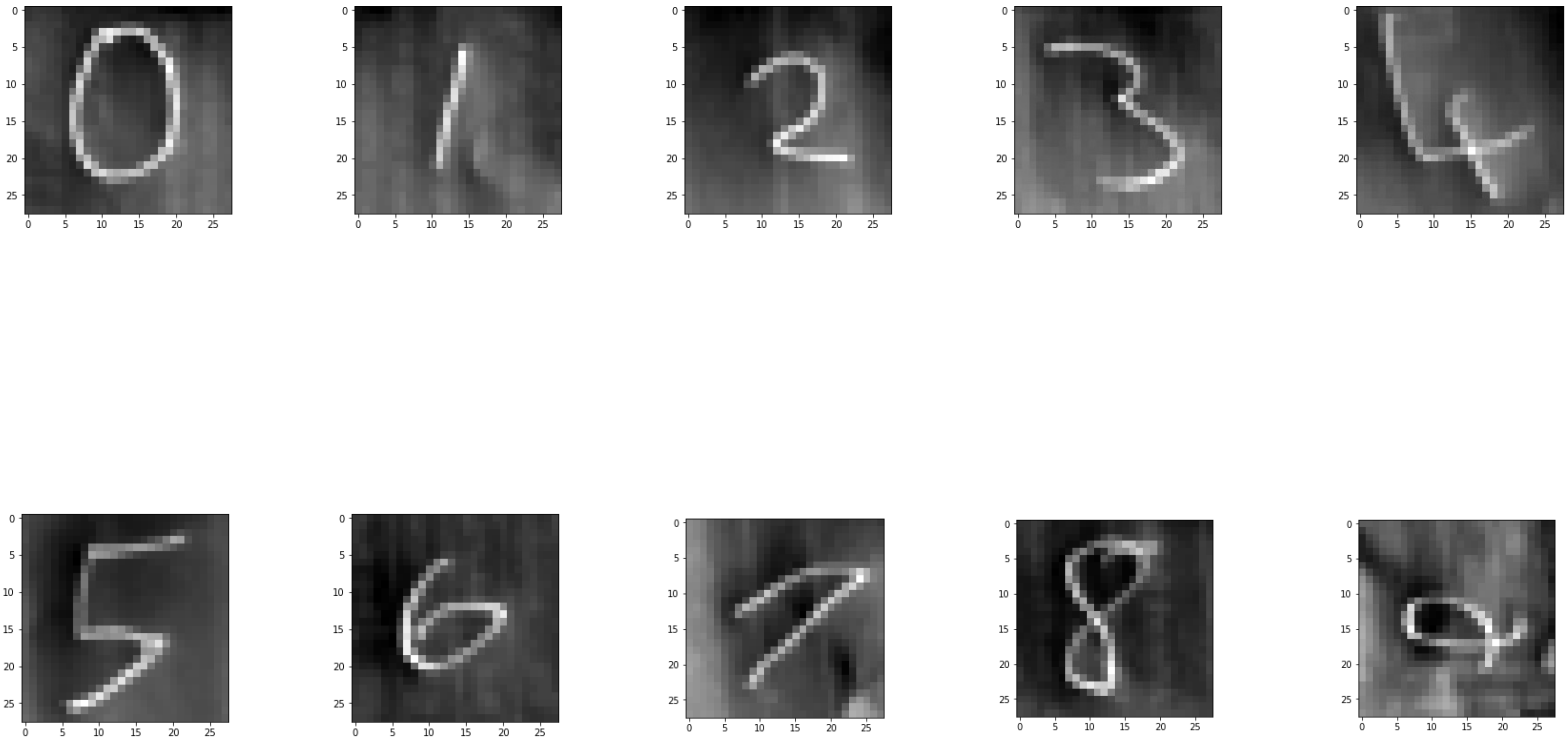
```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
```

But what happens if we use our own test data?

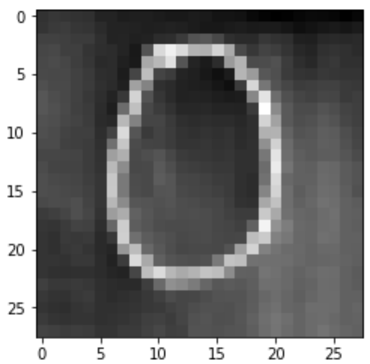
CNNs with Keras - Tom’s digits



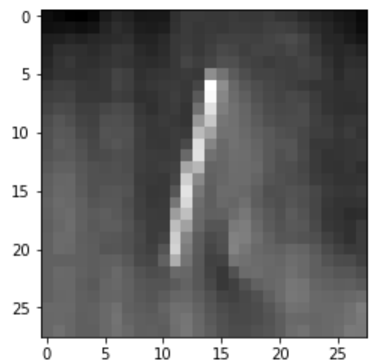
CNNs with Keras - Tom's digits



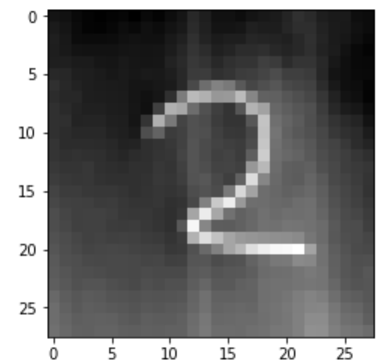
CNNs with Keras - Tom’s digits



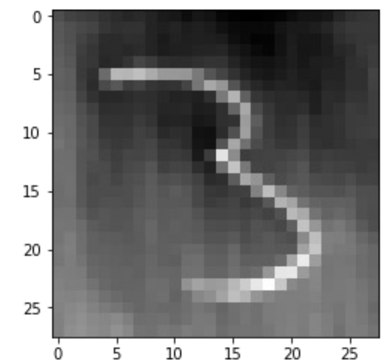
0	0.20110358
2	0.14562511
3	0.11874397
8	0.11115906



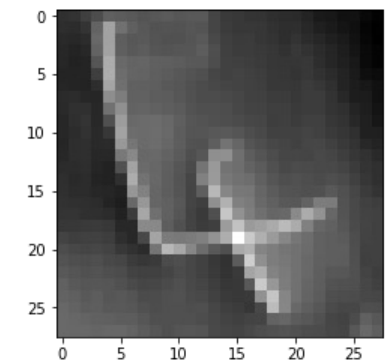
0	0.12581724
1	0.10913211
3	0.12886539
5	0.11482595
9	0.10597344



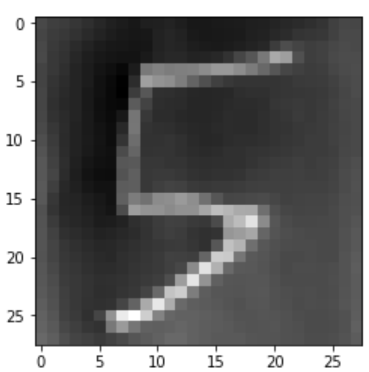
0	0.10848554
2	0.17623025
3	0.1374027
8	0.12013535



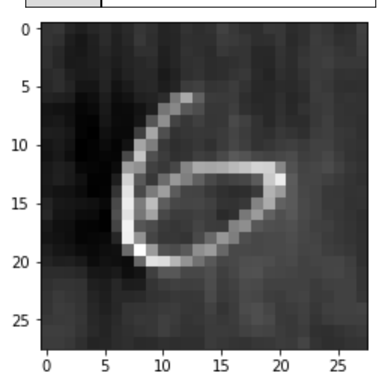
2	0.14474702
3	0.18350975
8	0.11873348



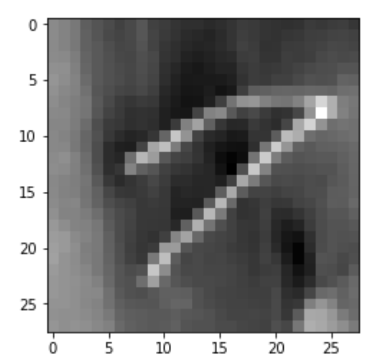
1	0.1035312
2	0.14548248
4	0.13108253
6	0.2559362
8	0.10085365



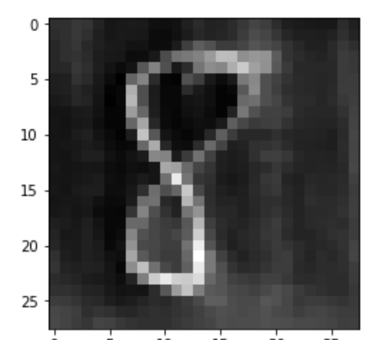
2	0.13071154
3	0.2254976
5	0.17201158
6	0.13599095
8	0.14935587



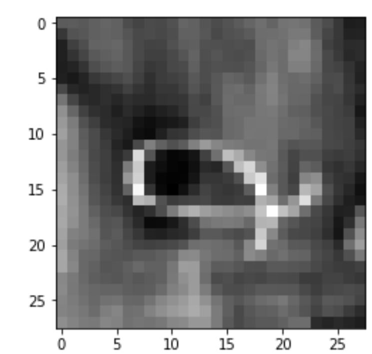
0	0.12649727
5	0.16573837
6	0.19378884
9	0.11177155



1	0.10536686
3	0.11079046
7	0.10666512
8	0.10597533
9	0.10386112



5	0.12932828
6	0.12932828
8	0.2376436



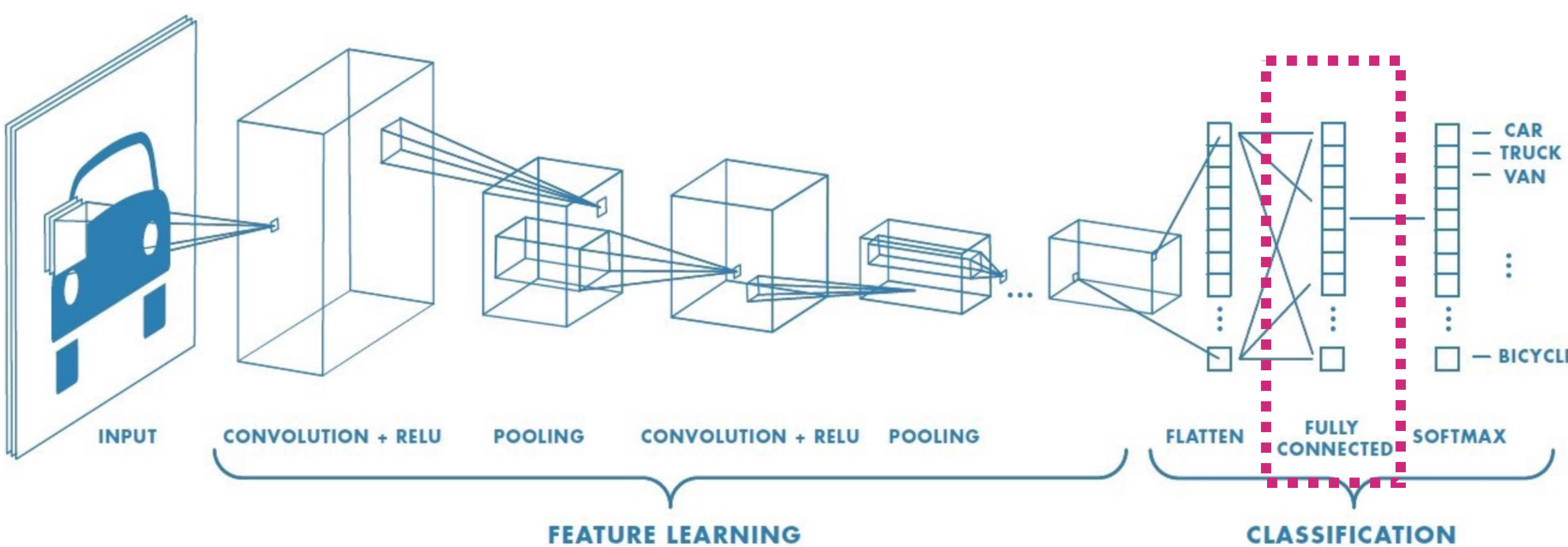
3	0.12017954
4	0.1000435
5	0.11944872
6	0.10832778
9	0.10199869

Note on Pre-training

Pre-training uses existing (pre-trained) weights from a larger dataset…. e.g. *VGG16/19*, *ResNet*, *MobileNet*, *InceptionNet*, etc. Often containing hundreds+ of trained layers.

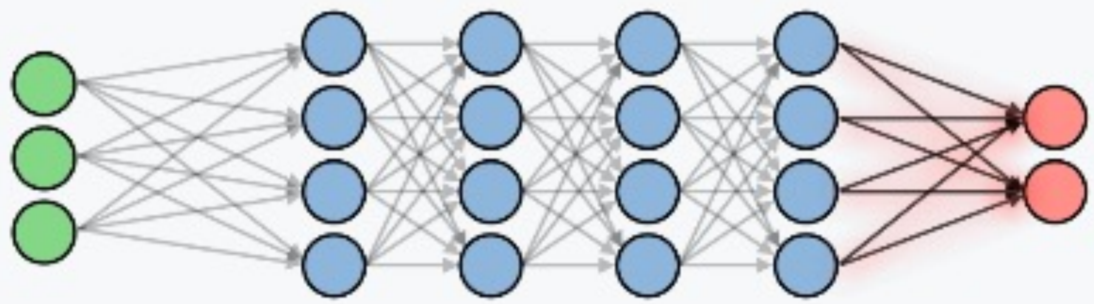
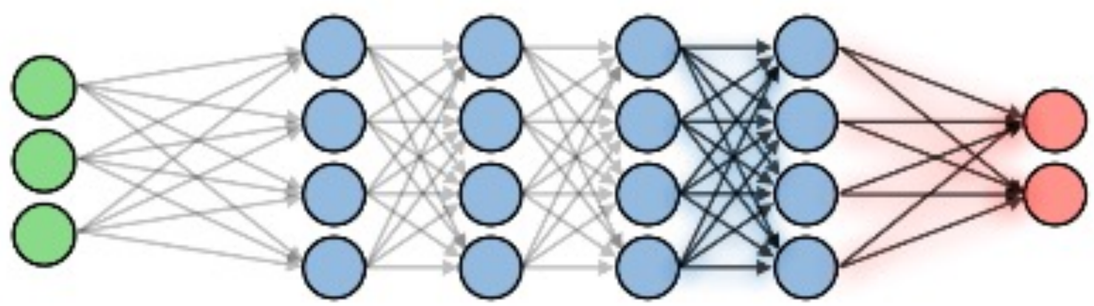
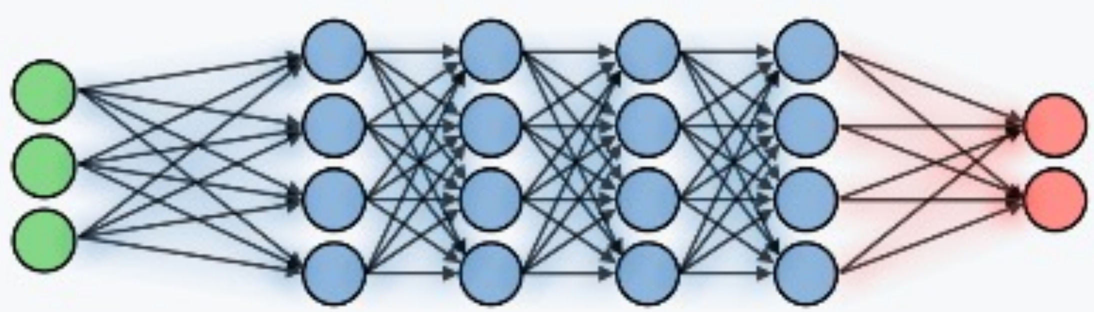
An active area of research - generally a good idea to use as prior knowledge can be injected in a computer vision model.

Can be used as is, or fine-tuned for specific domains…



Use last layer before prediction layer…

Note on Pre-training

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones

Advantages and Disadvantages:

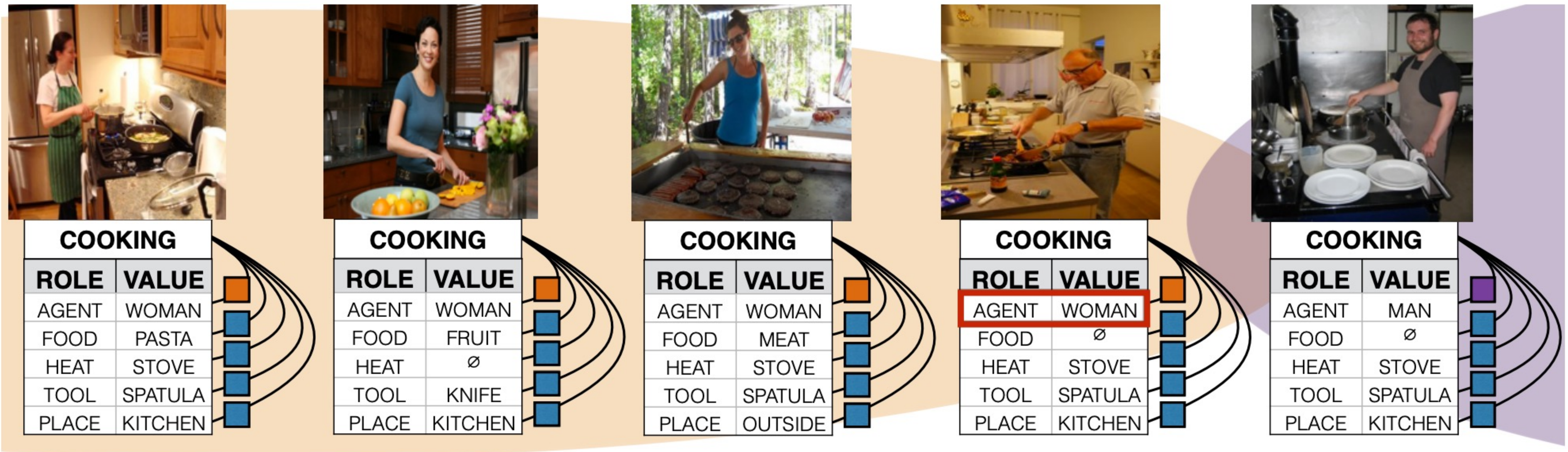
Pros:

- Superior accuracy over any other image analysis algorithm (mostly!)
- Automatic feature learning from images
- Can be combined with other architectures to gain flexibility for different classification problems and data shapes

Cons:

- High computational cost, slow to train
- Usually requires a GPU for decent problem size (cf. Viper though)
- Need lots of data to learn good model
- Some bias (as in any deep learner) depending on data …

Examples of Predictive Bias

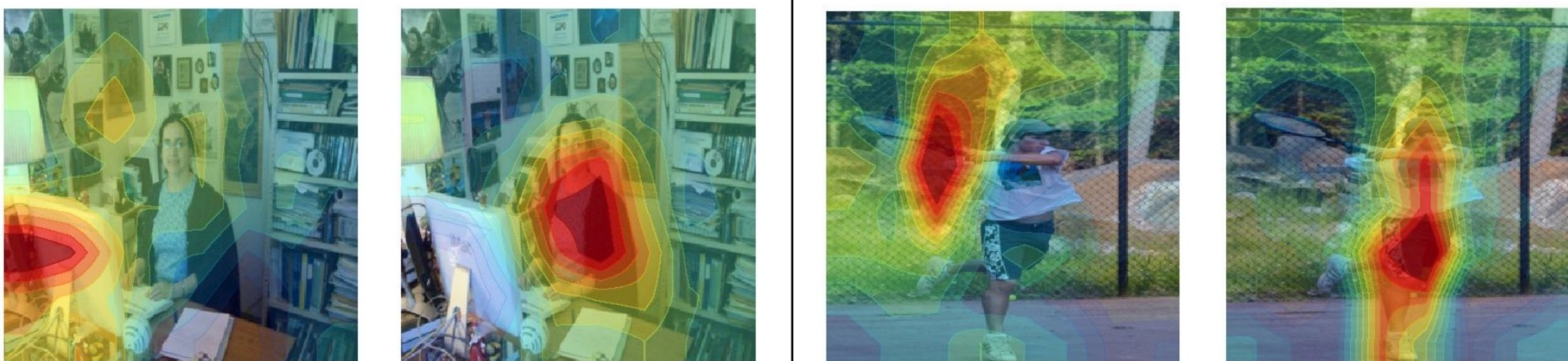


Over-amplification — occurs when a learning model "exaggerates" certain features in the data with undesirable consequences.

https://arxiv.org/pdf/1707.09457.pdf

Mitigations of Predictive Bias

Wrong | Right for the Right Reasons | Right for the Wrong Reasons | Right for the Right Reasons



Baseline:
*A **man** sitting at a desk with a laptop computer.*

Our Model:
*A **woman** sitting in front of a laptop computer.*

Baseline:
*A **man** holding a tennis racquet on a tennis court.*

Our Model:
*A **man** holding a tennis racquet on a tennis court.*

Research forcing an equal gender probability across samples to force a model to look at a person (rather than contextual cues).

https://arxiv.org/pdf/1803.09797.pdf

Possible other applications

Many applications in surveillance, entertainment, automation…

- Object recognition
- Video captioning
- CCTV analysis
- Face recognition
- Autonomous vehicles
- Legal cases
- …

Further study: Minecraft Fake Worlds

Can you add a CNN to the [“Preliminary Revision Lab?”]

▾ Preliminary Revision: Introduction to Neural networks (Please Review at Home Before Lectures Start) Prerequisites: Expected Behaviour (Complete this module first!)

Preliminary Revision

[Slides] neural_nets.pdf

Summary and Resources

We looked at *deep learning*, and:

- convolutional neural networks
- some of their applications

Reading:

Goodfellow et al. (2016) Deep Learning. MIT Press. Chapter 9.

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