CS4619: Artificial Intelligence II

Convolutional Neural Networks

Derek Bridge
School of Computer Science and Information Technology
University College Cork

Initialization

In [1]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [10]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten

from keras.optimizers import RMSprop

from keras.datasets import mnist
```

In [17]:
```
# MNIST dataset

# Load MNIST into four Numpy arrays
(mnist_x_train, mnist_y_train), (mnist_x_test, mnist_y_test) = mnist.loa
d_data()
mnist_x_train = mnist_x_train.reshape((60000, 28 * 28))
mnist_x_train = mnist_x_train.astype("float32") / 255

#Normalize
mnist_x_test = mnist_x_test.reshape((10000, 28 * 28))
mnist_x_test = mnist_x_test.astype("float32") / 255
```

In [18]:
```
# Dense network

def build_mnist_network():
    network = Sequential()
    network.add(Dense(256, activation="relu", input_shape=(28 * 28,)))
    network.add(Dense(10, activation="softmax"))
    network.compile(optimizer=RMSprop(lr=0.003), loss="sparse_categorica
l_crossentropy", metrics=["accuracy"])
    return network

network = build_mnist_network()
network.fit(mnist_x_train, mnist_y_train, epochs=5, batch_size=128, verb
ose=0)

test_loss, test_acc = network.evaluate(mnist_x_test, mnist_y_test, verbo
se=0)
test_acc
```

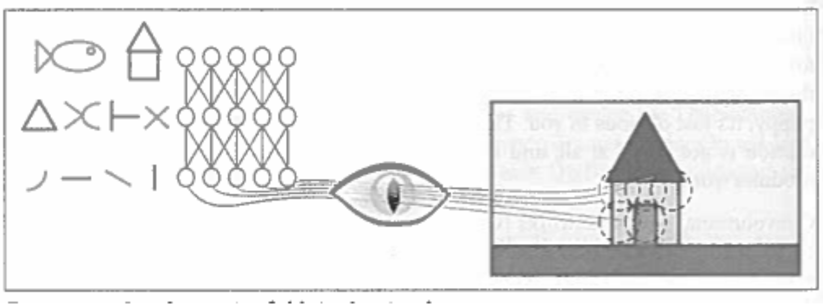
Out[18]:
```
0.9768999999999999
```

Acknowledgement

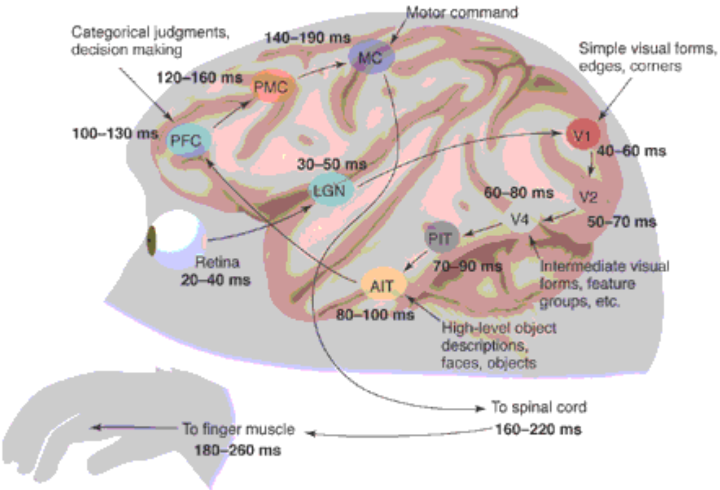
- The first image is scanned from Figure 13-1 in: A. Géron: *Hands-On Machine Learning with Scikit-Learn and TensorFlow*, O'Reilly, 2017
- The final image was produced by adapting the code from https://github.com/gwding/draw_convnet (https://github.com/gwding/draw_convnet)

Primate Vision

- In the primate vision system, there seems to be a hierarchy of neurons within the visual cortex



- In the lowest layers,
  - neurons have small local receptive fields, i.e. they respond to stimuli in a limited region of the visual field
  - 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
  - they respond to, e.g., lines at particular orientations
- In the highest layers,
  - they respond to ever more complex combinations, such as shapes and objects
- There are perhaps as many as 8 layers in the visual cortex alone



Convolutional Neural Networks

- Convolutional Neural Networks (convnets) are widely used in computer vision and in other perceptual problems including speech recognition and natural language processing
- They have nice properties, some of which resemble the visual cortex in primates:
  - They 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, …
  - They learn **spatial hierarchies** of features
    - from small local features such as lines in lower layers up to larger shapes in higher layers

MNIST Example

In [20]:
```
# MNIST dataset

# Load MNIST into four Numpy arrays
(mnist_x_train, mnist_y_train), (mnist_x_test, mnist_y_test) = mnist.loa
d_data()
mnist_x_train = mnist_x_train.reshape((60000, 28, 28, 1))
mnist_x_train = mnist_x_train.astype("float32") / 255

#Normalize
mnist_x_test = mnist_x_test.reshape((10000, 28, 28, 1))
mnist_x_test = mnist_x_test.astype("float32") / 255
```

In [24]:
```
def build_mnist_convnet():
    network = Sequential()
    network.add(Conv2D(32, (3, 3), activation="relu", input_shape=(28, 2
8, 1)))
    network.add(MaxPooling2D((2, 2)))
    network.add(Conv2D(64, (3, 3), activation="relu"))
    network.add(MaxPooling2D((2, 2)))
    network.add(Conv2D(64, (3, 3), activation="relu"))
    network.add(Flatten())
    network.add(Dense(64, activation="relu"))
    network.add(Dense(10, activation="softmax"))
    network.compile(optimizer=RMSprop(lr=0.003), loss="sparse_categorica
l_crossentropy", metrics=["accuracy"])
    return network

network = build_mnist_convnet()
network.fit(mnist_x_train, mnist_y_train, epochs=5, batch_size=64, verbo
se=0)

test_loss, test_acc = network.evaluate(mnist_x_test, mnist_y_test, verbo
se=0)
test_acc
```

Out[24]:
```
0.98699999999999999
```

- Note the reshaping
- Note the input shape
- Note the three numbers that configure convolutional layers (number of channels and height and width of windows) and two numbers for the max pooling layers (height and width)
- The last layers are densely connected, in the familiar way
- Training takes some time (unsurprising when we look at the number of parameters, below) but accuracy is now even higher
- Memory requirements for the network and for all the results that get stored during training are high, which is one reason to reduce mini-batch size

In [25]:
```
network.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_20 (Conv2D)           (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 3, 3, 64)          36928     
_________________________________________________________________
flatten_7 (Flatten)          (None, 576)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 64)                36928     
_________________________________________________________________
dense_16 (Dense)             (None, 10)                650       
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

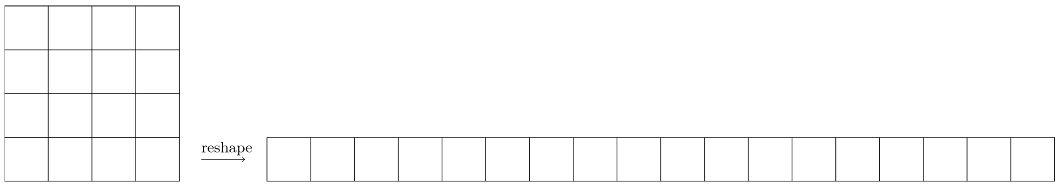
Tensors

- Tensors are multidimensional arrays of data
- A **scalar** is a 0D tensor
- A **vector** is a 1D tensor
- A **matrix** is a 2D tensor
- But we can have 3D tensors, 4D, …

Images are 3D tensors

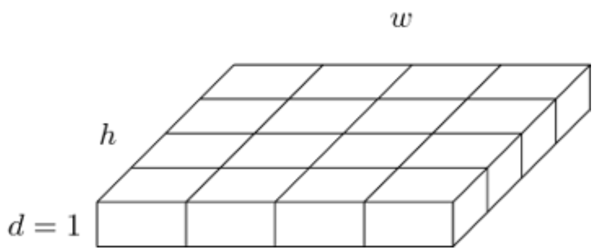
- Grayscale images
  - A 2D tensor (a certain height h and width w) of integers $[0, 255]$
  - Up to now, we have reshaped them into 1D tensors

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

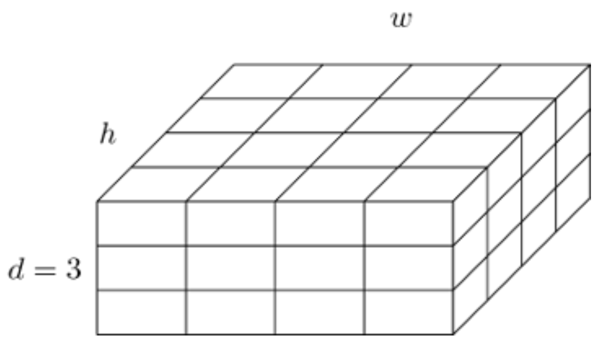


What is the disadvantage of this: what information gets destroyed?
  - So, henceforth, we will not flatten them in this way
  - In fact, for consistency with colour images, we will treat grayscale images as 3D tensors of shape $(h, w, 1)$

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



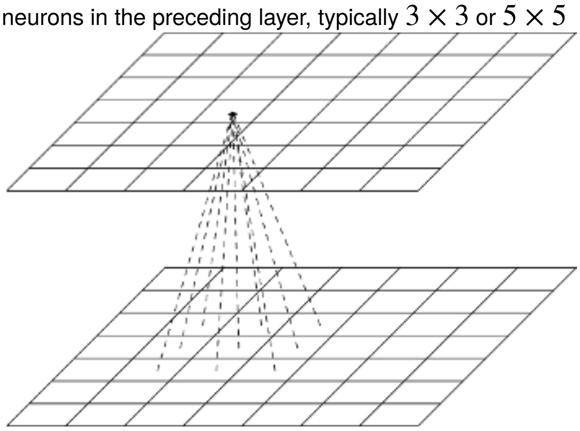
- Colour images
  - These will be 3D tensors: height h, width w, and channels (or depth) d
  - $d = 3$. Why?



- Datasets of images
  - Datasets of images will be 4D tensors: (m, h, w, d)
  - What is m?
- Why will datasets of videos be 5D tensors?

Convolutional Layers

- Consider a neural network whose inputs are images (each is a 3D tensor)
- A 2D convolutional layer is a 3D 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 $d = 1$
- Connections:
  - In the case of a dense layer, we saw that every neuron in that layer has connections from every neuron in the preceding layer
  - But in the case of a convolutional layer, every neuron in that layer has connections from only a small rectangular **window** of neurons in the preceding layer, typically 3×3 or 5×5

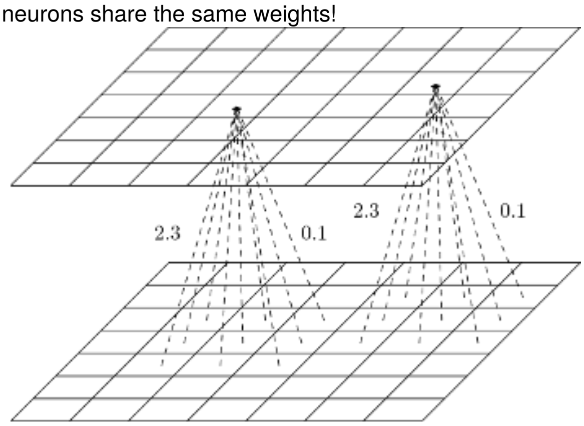


Convolutional layers: height and width

- Suppose the shape of the preceding layer is $(28, 28, 1)$ and the windows in the convolutional layer are 3×3
- This gives a convolutional layer whose height is 26 and whose width is 26. Why?
- Extra details that you can ignore in CS4619:
  - In fact, if we wish, we can make the convolutional layer have the same height and width as the preceding layer:
    - Padding: add a border of zeros around the previous layer
  - And, if we can wish we can make the convolutional layer have even smaller height and width than the preceding layer:
    - Strides: instead of contiguous windows, we can introduce a distance between successive windows

Convolutional layers: the weights of a feature map

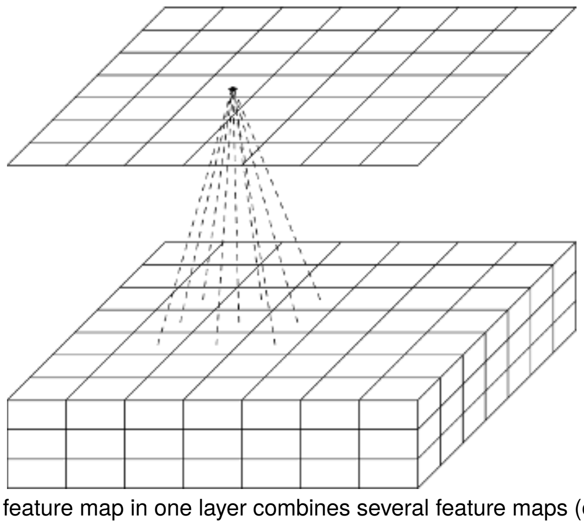
- Continue to assume $d = 1$, the convolutional layer consists of one feature map
- The idea of a feature map is that it will learn a specific aspect (feature) of its input
  - E.g. the presence of a vertical line
  - E.g.. the presence of a pair of eyes
- Within one feature map, all neurons share the same weights!



- Advantages
  - This reduces the number of parameters that must be learned
  - More importantly, it means that the feature map will respond to the presence of that feature *no matter where it is in the input* (the *translational invariance*, mentioned earlier)

Convolutional layers: stacks of feature maps

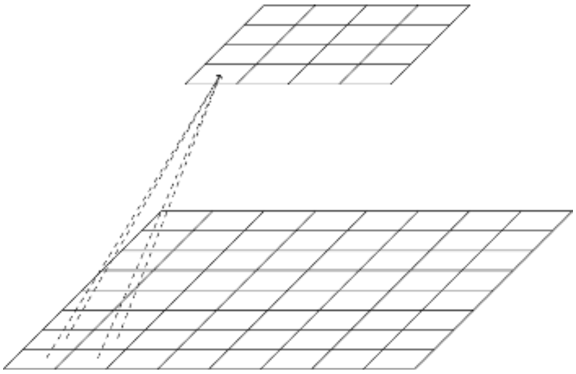
- Now consider the case where $d > 1$: the convolutional layer comprises a stack of d feature maps
- A neuron in a feature map in a convolutional layer is connected to a window of neurons in *each* of the feature maps of the previous layer



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

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
  - to create a hierarchy in which higher convolutional layers contain information about the totality of the original input image
- Again, it works on rectangular windows: neurons in the pooling layer are connected to windows of neurons in the previous layer
  - typically 2×2
  - typically adajcent rather than overlapping
- E.g. if the previous layer has height h and width w, and the pooling layer uses adjacent 2×2 windows, then the pooling layer will have height $h/2$ and width $w/2$



- The depth of the pooling layer is the same as the depth of the previous layer

Max 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
- Pooling layers work on the feature maps independently, which is why they have the same depth as the previous layer

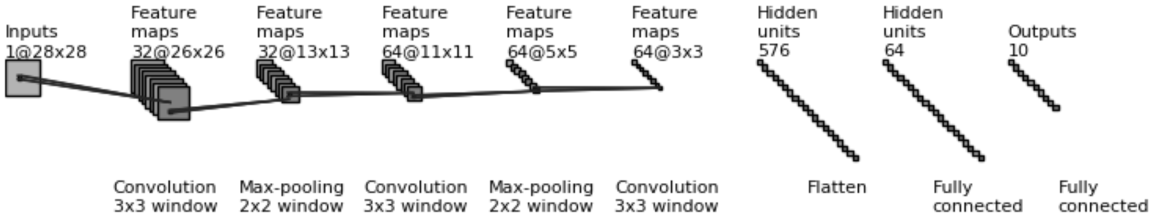
Check Your Understanding

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

In [28]:
```
def build_mnist_convnet():
    network = Sequential()
    network.add(Conv2D(32, (3, 3), activation="relu", input_shape=(28, 2
8, 1)))
    network.add(MaxPooling2D((2, 2)))
    network.add(Conv2D(64, (3, 3), activation="relu"))
    network.add(MaxPooling2D((2, 2)))
    network.add(Conv2D(64, (3, 3), activation="relu"))
    network.add(Flatten())
    network.add(Dense(64, activation="relu"))
    network.add(Dense(10, activation="softmax"))
    network.compile(optimizer=RMSprop(lr=0.003), loss="sparse_categorica
l_crossentropy", metrics=["accuracy"])
    return network
```

In [27]:
```
network.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_20 (Conv2D)           (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 3, 3, 64)          36928     
_________________________________________________________________
flatten_7 (Flatten)          (None, 576)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 64)                36928     
_________________________________________________________________
dense_16 (Dense)             (None, 10)                650       
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```



In [ ]: