Machine Learning II

Prof. Dr. Tim Downie

Week 13 Lecture – 5th January 2023

Neural networks for image data, convolutional neural networks

BHT Berliner Hochschule für Technik

Contents

- Data as images
- Image classification
- Locally connected neural networks
- Filters
- Convolutional neural networks

Image data

When analysing images, the data is usually represented as a set of values of pixels on a rectangular grid.

The values can be

- one dimensional: Grey scale, typically integer values between 0 and 255
- three dimensional: RGB, the intensity of red, green and blue.

In the RGB format then usually each of the 3 colours is a value between 0 and 255, and very often represented as three hexadecimal numbers concatenated.

Eg. #lc7f33 is a combination of 1c red (28 in decimal),
7f green (127) and 33 blue (51).

Each pixel is given a location in 2 dimensions. A grey scale image is a matrix (2-dim array). An RGB image is stored as either separate 3 matrices or as a 3-dim array.

This is equivalent to a basic bitmap format.

jpeg, gif, png etc. are all formats for storing and displaying images, the aim being to require less bytes to store the image. It would be unusual to feed data from a jpeg image directly into a NN, without first unpacking the data.

Image classification

Is this image a cat or a mouse?

There are many different ways of analysing an image, but the most common application for neural networks on image data is image recognition. Suppose we have several classes, including cat and mouse, and want to predict what class the image belongs to.

Is this image a cat or a mouse?

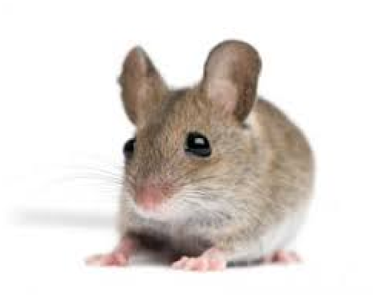


Image classification

We have a data set consisting of images, and for each image we know what class it belongs to (the label).

Labelling images is a bigger problem than at first sight. The ideal is thatthe labels should be assigned by humans, which takes a long time and is not an interesting job.

Using captions scraped from a website is often problematic.
Eg. A newspaper photo of a face has the caption
“Acourt has always denied involvement in the death of Lawrence”.
How do we scrape the correct label from this caption?

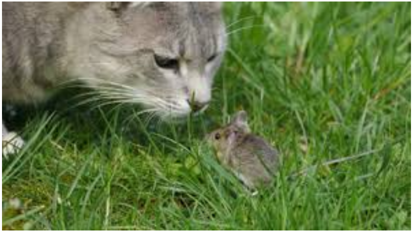
The image might be of Acourt or of Lawrence.

We also need to be careful how real images will be defined.

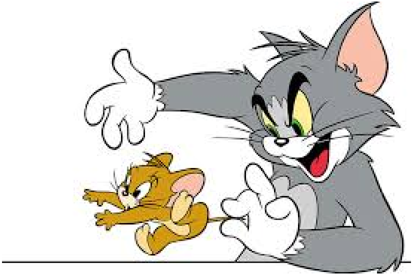
In the classification problems we have so far considered, we have always assumed that all the observations belong to exactly one of the available classes.

With image data this is not always the case.

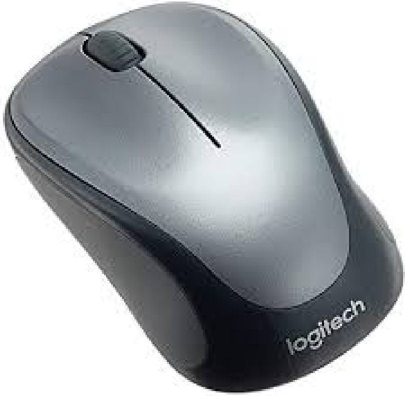
Is this image a cat or a mouse?



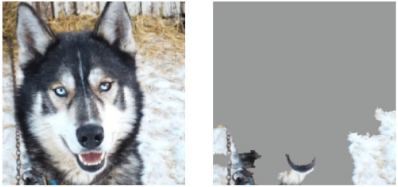
Is this image of a cat or a mouse?



Is this image a cat or a mouse?



In a famous example, a deep learning image classifier falsely identified an image of a husky as a wolf. When investigating the reason behind this error, it was noticed that all the images of wolves had snow in the background and the neural network was using the background not the foreground to identify the wolf.

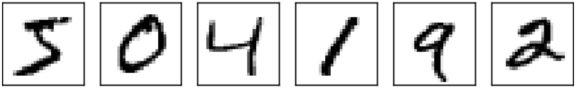


Source: "Why Should I Trust You?": Explaining the Predictions of Any Classifier, Ribeiro, Singh & Guestrin (2015)

Image data sets

There are a number of standard image data sets used to benchmark an image classifier. A very famous one is the MNIST data set of handwritten digits,[1] taken from ZIP codes.

The training data consist of 60 000 images of digits, which have been size-normalized and centered in a fixed-size image.



The test data consists of 10 000 labelled images.

[1] http://yann.lecun.com/exdb/mnist/

Locally connected neural networks

So far we have represented the input and hidden layers as row of nodes.

For a standard data matrix the input nodes could be permuted, with no change in the model.

Now our input data has a spatial structure, which we don't want to lose.

A very small grey scale image with 32^2 pixels has 1028 input nodes.[2] Even a small MLP with just one fully connected hidden layer with 10 nodes, we have 10280 weights plus 10 biases to estimate.

In addition a fully connected model does not take into account the spatial nature of an image, two pixels in opposite corners have the same connectivity as two neighbouring pixels.

[2] We will mainly consider grey scale images and cover colour images quickly later.

An alternative approach:

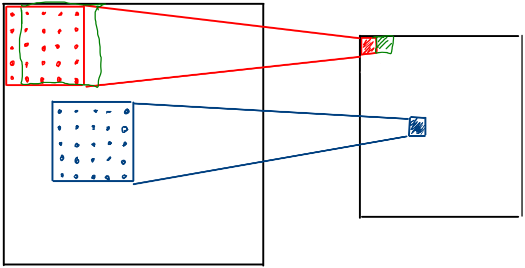
The input nodes are structured as a matrix one node per pixel.

Each node in the first hidden layer is only connected to 5×5 input nodes (pixels).

We also structure the first hidden layer as a matrix.

The 5×5 square region of input nodes is called a *local receptive field*. We can consider that the local receptive field is a square which sweeps over the input matrix.

Graphical representation: Top left 25 input nodes connect to the top left hidden node, then shift the local receptive field one to the right giving the second hidden node in the top row etc.



With a 32^2 image and a 5^2 local receptive field, the first hidden layer is a 28^2 matrix, with $5^2 28^2 = 19600$ connections.

With an n^2 image and a k^2 local receptive field, the first hidden layer is a $(n-k+1)^2$ matrix, with $k^2(n-k+1)^2$ connections.

This is a hige number of parameters for just the first layer and a low resolution image.

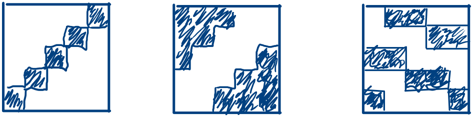
The trick is to use the **same weights** leading to each node in the hidden layer. Although there are a large number of connections there are only 25 weights and one bias.

We could use (an adapted form) of back propagation to find the optimal shared weights and bias, but usually a different approach is used.

We can think of a combination of weights as identifying a pattern in the 5 × 5 local receptive field. It is called a *feature detector*.

Example:

3 Examples of feature detectors



Gives a

High response

Low response

Average response

when the input has the form of a diagonal line with a positive gradient

This property is very useful for images. It allows us to identify small regions with a certain property.

The next step is to predefine the 25 weights and bias for a particular type of feature.

This leads us to the concept of filters, we filter the image to find regions which matches our predefined pattern.

Filters and convolutions: 1-dimensional data

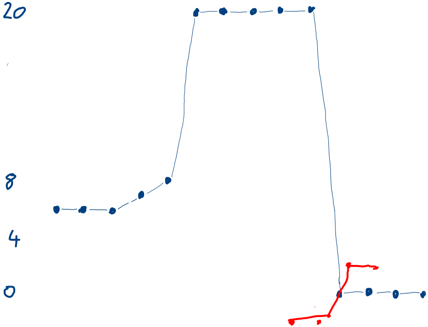
Suppose we have a sequence of numeric data $x_1, x_2, x_3, \ldots, x_n$.
Data of this type is called a signal.

A filter is defined by a sequence of J values. A simple example is:
$a_1 = -1$, $a_2 = -1$, $a_3 = 1$, $a_4 = 1$ and $J = 4$

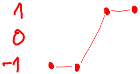
The filter coefficients are multiplied with data points x_1 to x_J and added up to give z_1. The filter is then shifted one step to the right and the filter coefficients are multiplied with x_2 to x_{J+1} and added up to give z_2, etc.

The above example is a filter which gives a large positive value when the signal increases and a large negative value when signal decreases.

The signal is



The filter is:



$$20(-1)+20(-1)+20\cdot$$
$$=0$$

E.g. The signal is

$$6,\ 6,\ 6,\ 7,\ 8,\ 20,\ 20,\ 20,\ 20,\ 20,\ 0,\ 0,\ 0,\ 0$$

and $a_1 = -1$, $a_2 = -1$, $a_3 = 1$, $a_4 = 1$.

$$\begin{aligned}
\text{Applying the filter to the first position gives} \quad z_1 &= -6 - 6 + 6 + 7 && = 1 \\
\text{Applying the filter to the second position gives} \quad z_2 &= -6 - 6 + 7 + 8 && = 3 \\
\text{etc...} \quad z_3 &= -6 - 7 + 8 + 20 && = 15 \\
z_4 &= -7 - 8 + 20 + 20 && = 25 \\
z_5 &= -8 - 20 + 20 + 20 && = 12 \\
z_6 &= -20 - 20 + 20 + 20 && = 0 \\
z_7 &= -20 - 20 + 20 + 20 && = 0 \\
z_8 &= -20 - 20 + 20 + 0 && = -20 \\
z_9 &= -20 - 20 + 0 + 0 && = -40 \\
z_{10} &= -20 - 0 + 0 + 0 && = -20 \\
z_{11} &= -0 - 0 + 0 + 0 && = 0
\end{aligned}$$

Large positive values overlap the jump from 8 to 20 and large negative values overlap the jump from 20 to 0.

The mathematical formula for applying the filter is

$$z_i = \sum_{j=1}^{J} a_j x_{i+j}$$

The process of applying the filter to the signal is called a **convolution**, the properties of the filter are determined by the coefficients $a = \{a_1, \ldots, a_J\}$

The convolution of filter a with signal x is written using the $*$ notation

$$z = a * x$$

The filter in the example above is a type of *finite impulse response filter* because the length of the filter is short, and responds to what is occurring in the signal in a finite region.

The idea of a filter comes from a field in electrical engineering called *signal processing*. An electrical signal e.g. the electrical impulses in a telephone call can be transformed by applying a filter, which have specific properties.

A high pass filter responds to the high frequencies in the signal[3]
A mid pass filter responds to the middle frequencies in the signal
A low pass filter responds to the low frequencies in the signal.

To understand speech we only need to hear a narrow band of mid-frequencies.

Using filters, three telephone conversation can easily be simultaneously sent along one wire by compressing the speech into a narrow band of frequencies and separating them at the receiving end.

Another application of filtering a signal is to reduce noise in a signal which has background noise. This has certain similarities with lowess smoothing.

[3] the filter $a_1, \ldots a_J$ in the last example is a type of high pass filter

An image is a 2-dimensional signal and much of the signal processing research is focussed on image processing.

Back to Neural Networks:
A feature detector is a two dimensional filter applied to an image.
The filter coefficients are in the form of a matrix.

The result of applying a feature detector to an image is a **feature map**, which is the convolution of the feature detector filter and the image.

Roughly speaking a feature map identifies which parts of the image is good fit to the feature defined by the filter.

In an image classification NN, typically 20 to 40 feature detectors are used. Giving the same number of feature maps, each with a square matrix layout.

There are libraries for these feature maps LeNet-5[4] is a famous example of a feature map library.

[4] http://yann.lecun.com/exdb/lenet/

Pooling

After obtaining a feature map, it is usual to reduce the dimension, by using a pooling layer.

Max-pooling means take 4 nodes in a square and send the maximum activation of those nodes to the pooled layer.

An alternative is L_2*-pooling*: the root sum of squares of the four activations.

The pooled layer again has a square structure, but contains a quarter of the nodes.

The activations for each node in the pooled layer are then pushed forward to the next NN-layer.

A simple configuration for an image classifier is

Input layer → feature maps → pooled layer → output

The pooled layer and output are fully connected: each "pixel" is connected to a node for each of the possible outcomes.

Common extensions to this structure are:

- To have one or more standard hidden layers between the pooled layer and the output layer, which are fully connected.
- To add another layer of filtering and pooling, maintaining the matrix structure. This time the filter coefficients are not feature detectors, but weights to be optimised while training the NN.

Colour images

The above assumes on square matrix of input nodes, which corresponds to a grey scale image. For a RGB color image, we have 3 square matrices as the inputs.

If colour is not important to the classes e.g. a red 3 and a blue 3 both belong to the “3” class, then simply add the values of three colours together, which results in a grey scale image. Other image-processing procedures are also possible such as applying a filter which outputs “saturation” of the pixel.

Often colour is important for the classes, e.g. distinguishing red, yellow and green traffic signals. For a colour image NN:

- The input layer consists of 3 matrices
- Each colour input will be filtered independently giving 3 feature maps
- Followed by a Pooling layer, etc.
- At some stage before the output layer, the three colour channels will be merged, using weights which are optimised in the NN training.

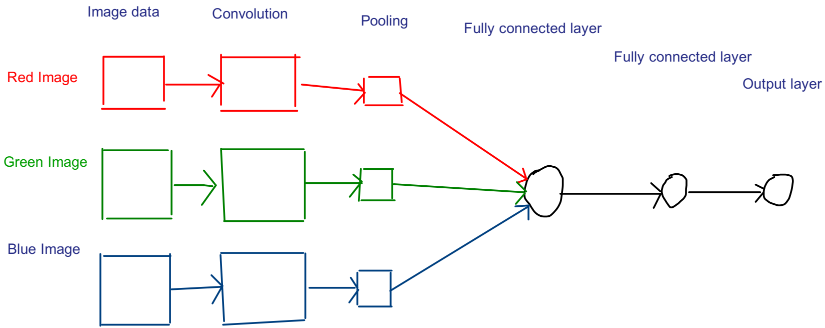


Image data
Convolution
Pooling
Fully connected layer
Fully connected layer
Output layer
Red Image
Green Image
Blue Image

Summary of convolutional NNs for image data

- The input layer consists of nodes corresponding to the image pixel values
- A library of feature detector filters is chosen, and applied using a locally connected layer.
- The resulting feature maps form the first hidden layer. There is one feature map for each feature detector each with a matrix structure.
- The dimension of each feature map is reduced by a pooling layer.
- The nodes in the pooling layer then feed into the next hidden layer or the output layer of the NN.

Comparison of NN configurations for the MNIST digits

Michael Nielsen in his (online) book *Neural Networks and Deep Learning* investigates the recent performances of neural networks on this benchmark data set
http://neuralnetworksanddeeplearning.com/chap6.html.

NN1: One fully connected hidden layer with 100 nodes.
Correct classification rate 97.80% error rate 2.2%.

NN2: Convolutional layer with 5×5 local receptive fields, 20 feature maps and max-pooling layer, followed by one fully connected hidden layer with 100 nodes.
Correct classification rate 98.78% error rate 1.22%.

NN3: Like NN2 with a second convolutional-pooling layer between the first convolutional-pooling layer and the fully-connected hidden layer.
Correct classification rate 99.06% error rate 0.94%.

NN4: The same architecture as NN3 but using ReLu activation instead of the standard sigmoid activation. Correct classification rate 99.23% error rate 0.77%.

Nielsen remarks:
This is a small improvement but “across all my experiments I found that networks based on rectified linear units consistently out-performed networks based on sigmoid activation functions.”
At the moment it is not understood why there is an improvement using ReLu, but it gives consistently good empirical results.

Further procedures which led to improved results were

- *Expanding the data set* by shifting and rotating the original images by a small amount.
- Using an ensemble method eg boosting: fit multiple NNs with different architectures and hyperparameters, and taking a majority vote.

Summary of neural networks topics covered in this course

- MLP: Fully Connected Neural Network with weights and biasses for
  - Regression (one output node with identity output activation)
  - Classification (K or $K-1$ output nodes)
- Numerical minimisation in one and two dimensions, steepest descent methods, the back propagation method of computing the gradient vector in a Neural Network
- Algorithmic aspects of training a NN: learning rate, batch optimisation (stochastic gradient descent), regularisation.
- Time delay and recurrent neural networks, for time series data.
- Convolutional neural networks for image data, using feature detectors, feature maps and pooling layers.