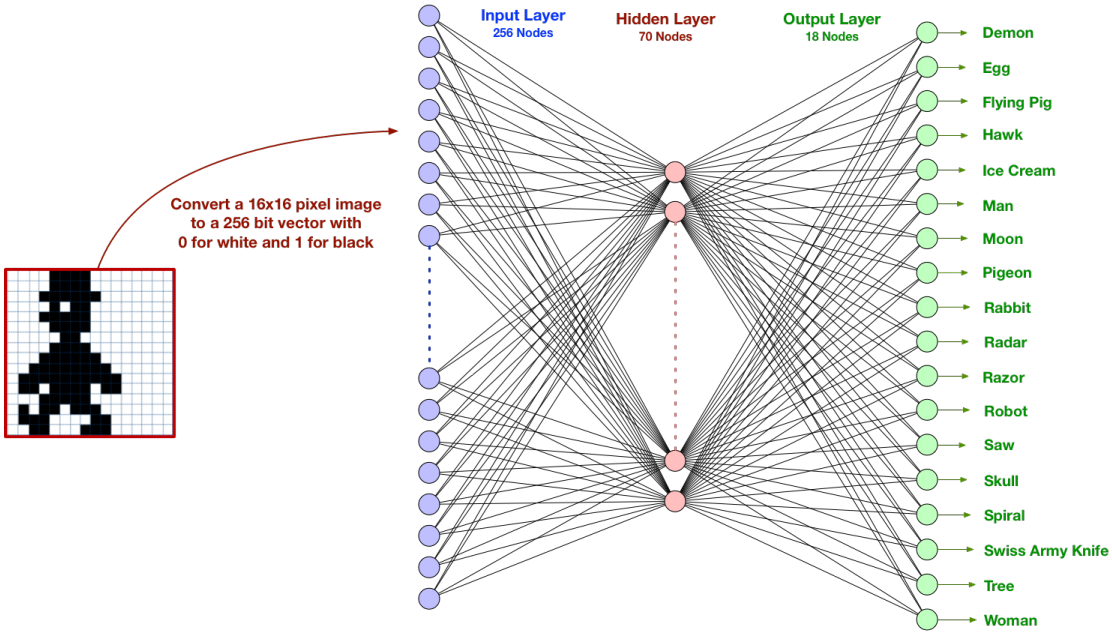
Department of Computer Science & Applied Physics

Using a Neural Network to Identify an Image

Overview

Image analysis is one of the most important applications of neural networks. In this practical we will create a neural network capable of classifying an image from a range of 18 different possibilities. The input data consists of a set of 72 black and white images with dimensions of 16x16 pixels.



A neural network can classify an image by re-encoding the image as a one-dimensional bit vector. For the 16x16 images used in this practical, there are a total of 16x16=256 pixels to encode. A separate node in the input layer is required for each pixel. Because the image is in black and white, a black pixel can be represented with a 1 and white area with 0. Note that for a coloured image, the total number of nodes will be the width x height x 4 (RGB + alpha channel). Thus, even a relatively modest size of image will require millions of input nodes. Note also that deep neural networks are required for more sophisticated image analysis. These consist of multiple hidden layers, where each hidden layer performs some processing / computation on the input bit vector.

Exercises

- Create a new project called *ImageAnalysis* and unpack the image resources correctly into the project. The directory "sprites" consist of 16 x 16 pixel images named as image-[number].png. There are four different versions of each image and 18 different characters, giving 72 images in total.

- Copy the class *ImageClassifier.java* into its correctly configured package and use the class *NetworkBuilder* to create a neural network with the following configuration:

  Input Layer: 256
  Hidden Layer: Use *LayerSize.GEOMETRIC_PYRAMID* and Sigmoid
  Output Layer: Use *names.length* and Sigmoid
  Alpha: 0.01
  Beta: 0.95
  Epochs: 10000
  Min Error: 0.00001
  Loss Function: Sum of Squares

- Execute the programme and examine the accuracy of the predicted image types. Run the programme over and over and examine the result. ***Is the network stable?***

- Progressively increase the value of the *percentage* variable in the method **test()** that is used to control the amount of noise to add to the training data for testing.

```
var percentage = 5;
```

  At what point does the accuracy of the prediction fall below 50%?

- Change the activation function to ***TanH*** and examine the training time and accuracy. Do the same for the other main activation functions. You can also experiment with changing the encoding from a binary [0,1] to something else by altering the following statement in the method *getPixels()*:

```
pixels[index] = colour > 0 ? 0 : 1;
```

  You should change the pixel value to match the activation function. The images that we are using are 16 x 16 black and white PNGs. What would be the overhead and topological changes if we used full RGB JPEG images in high resolution?

- What image processing techniques could be applied to facilitate learning in the network?