Report
Coin Detection Using Neural Networks

(Computer Graphics Project, D2 Slot)

Zeeshan Vohra (11BCE1106), Divyansh Jain (11BCE1085)
School of Computer Science and Engineering
VIT University, Chennai Campus

1. Introduction

Neural networks (more accurately, Artificial Neural Networks, or ANNs) are computational models inspired by animals' brain, which are capable of machine learning and pattern recognition. They are usually presented as systems of interconnected 'neurons' arranged as layers (n-partite graphs) that can compute values from inputs by feeding information through the network. A sample neural network is shown in Fig 1.1 below.

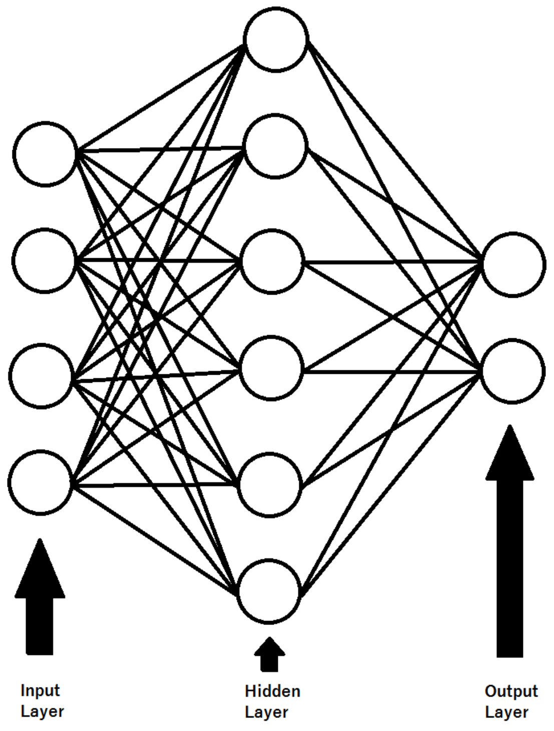

Fig 1.1 – Sample neural network

In the given problem, the aim is to recognize coins present in an image over a stable background, and, by using pattern recognition techniques, classify them as 1 Rupee, 2 Rupee, or 5 Rupee coins, and finally compute the total amount of money present in the image as coins.

2. Methodology

a) Working of a single neuron

To understand how an ANN works, we look at the working of one single neuron. The architecture of an artificial neuron (inspired by a biological neuron) is shown in Fig 2.1 below.

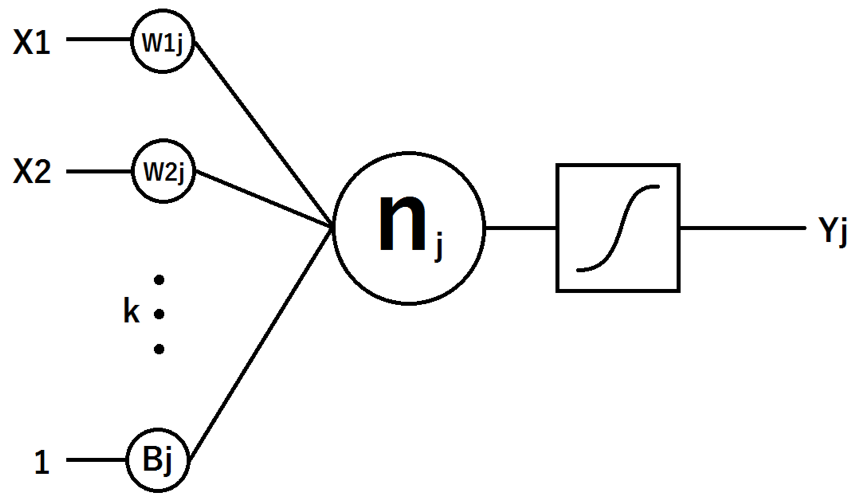

Fig 2.1 – Architecture of a single neuron

There are k inputs coming in, as X_1, X_2 … X_k, and one output Y_j, to the neuron n_j.The weights of each input are given as W1j, W2j … Wkj, and there is a constant input (known as bias), denoted here by Bj. The output is defined as:

$$\mathrm{Yj} = sigmoid(Bj + \sum_{i=1}^{k}(\mathrm{Wij.Xi}))$$

Where the sigmoid function (which essentially ‘squeezes’ the input value between -1 and +1) is defined as:

sigmoid(x) = (2/(1+exp(-2*x))-1)

There can be many such layers of neurons between the actual inputs and outputs. Fig 1.1 has one such layer (called the hidden layer) of 6 neurons. By changing the weights and the biases of each neuron, the entire network can be made to recognize certain types of inputs, hence they are useful in pattern recognition.

b) Recognizing circular objects in an image

To run a pattern recognition algorithm over an image of coin, first we would need to separate the image of the each coin from the given input image. Assuming that the background is stable, and no coin overlap each other, detecting circles in the image can give us the locations of the coins in the image, which can then be fed to a trained neural network. A very commonly used technique for detecting circles in an image is the Hough Transform [1]. Applying Hough Transform to the image, using suitable parameters, returns the centers and radii of the circles present in the image, using which we can extract only the coins present in the input image and save them as separate images for further processing. A resulting image after applying the Hough transform is shown in Fig 2.2 below.

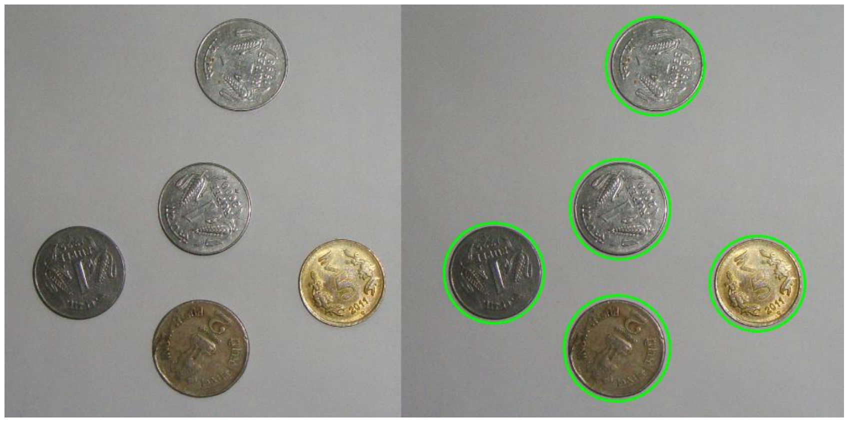

Fig 2.2 – Hough transform result for detecting circles

c) Preprocessing in training and simulation of the neural network

The image of a coin separated from the original input image cannot be given as an input the neural network as such. There is a lot of noise associated with a raw image, which will hinder the process of pattern recognition in the image. Moreover, a raw image has 3 channels of RGB, with values ranging from 0 to 255 in each pixel of each channel, making even a small image of size 32x32 pixels large enough to cause out of memory errors with a neural network of suitable size. To avoid all this, a lot of preprocessing has to be done on the image of the coin to make it a suitable input to a neural network.

To train the neural network with sample images, the first step is enhancing the contrast and brightness of the images by using suitable values of alpha and beta (gain and bias parameters, respectively) [2] to bring out prominent features in the image. Then the image is converted to a grayscale image, and then resized to the size of the input layer of the neural network. For example, if there are 128 neurons in the input layer, the image is resized to 128x128 pixels. A larger size preserves more detail, but is slower to process. Although grayscale images have a single channel of values ranging from 0 to 255, they still contain a lot of extra details which are not very useful for the task at hand, and unnecessarily slow down the neural network. All that is needed for the network to correctly work, is the information about the edges in the image, which give us the shape of the coin and the features inside the coin very clearly, while keeping the value of each pixel either as 0 or 1 (a binary image), which is much smaller in size than a grayscale image, hence faster to process. Applying edge detection directly to the image results in a lot of unnecessary information, most of which is noise, generated because of the changing light on different portions of the coin, and not actual edges on the coin. A slight blurring of the image before detecting the edges can solve this problem, as it smoothens the image, and only actual physical edges are left detectable. Gaussian blur [3] has been used with a suitable kernel size to slightly blur the images. After blurring, the edges have been detected by using the canny edge detection algorithm with suitable parameters [4].

This entire preprocessing on the images of coins already labelled as 1 Rupee, 2 Rupee, or 5 Rupee coins for training the neural network (the process of adjusting the weights of neurons in all the layers to make the network work for test images) can be summarized by Fig 2.3 below.

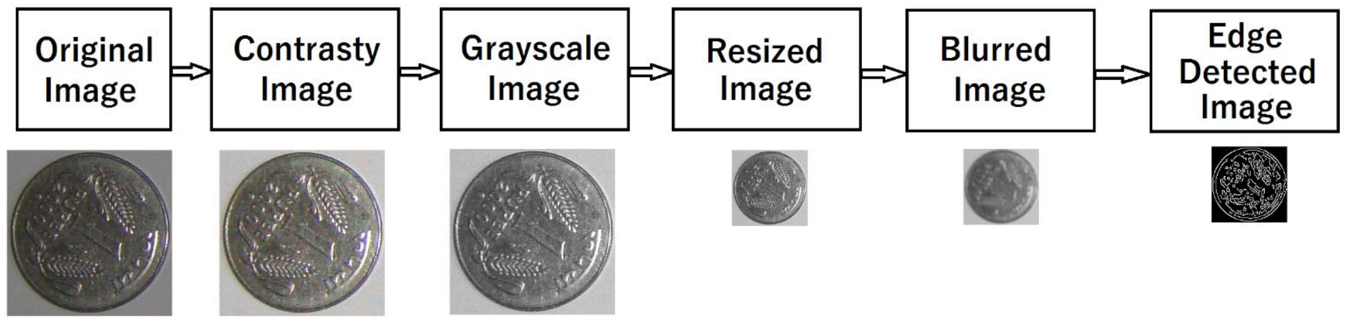

Fig 2.3 Preprocessing for training the neural network

The edge detected image is then fed to the designed neural network for training. For the process of simulation of the neural network, i.e., to detect which coins are of what value in the image, a similar preprocessing is required. If the preprocessing steps or the parameters of the functions used during training differ from those used during the simulation of the network, the results will not be correct, as the network is trained and adjusted for different input values and is being tested on different ones. The only extra step at the beginning of preprocessing of images for simulating the network, is the Hough transform, which will give us separate images of coins present in the input image.

d) Training and simulating the neural network

Neural networks take a lot of time and processing power for training. Reducing the size of input, and giving only the relevant information to the network is very important, which means a lot of preprocessing has to be done before an image is given as input to a network. For the current problem, if an input image of size 128x128 pixels is to be given as input to a network, there would have to be 128*128 = 16384 neurons in the input layer, corresponding to each pixel of the image. Also, the subsequent layers' size would also be of a similar size, to ensure the details of the image are not lost. This would make a very large network, and training such a colossal ANN is not feasible for a small dataset of images of coins.

Instead of giving the whole image as input, only a single row (the central row) of the image matrix can be given as input to the neural network, as that row itself would be different for coins of different denominations, and enough for the network to differentiate between the coins. This reduces the input layer size to 128 neurons.

Another problem to consider here, is the rotation of coins. We expect the network to recognize coins rotated to any degree. To solve this problem, the input images are rotated from 0 to 359 degrees, creating a set of 360 images from one single input image. From all these images, the central vector is extracted, and given as input to the neural network.

The neural network architecture used in the problem in shown in Fig 2.4. The network was trained with 8 images of coins of each denomination. The same type of coin was used for each denomination. Each image, after rotating, generated a set of 360 images. This made the total dataset size for training as 3*8*360 = 8640 vectors of size 128 each, with binary values. The target dataset consisted of 8640 vectors corresponding to the input dataset, of size 3 each for the 3 denominations of coins.

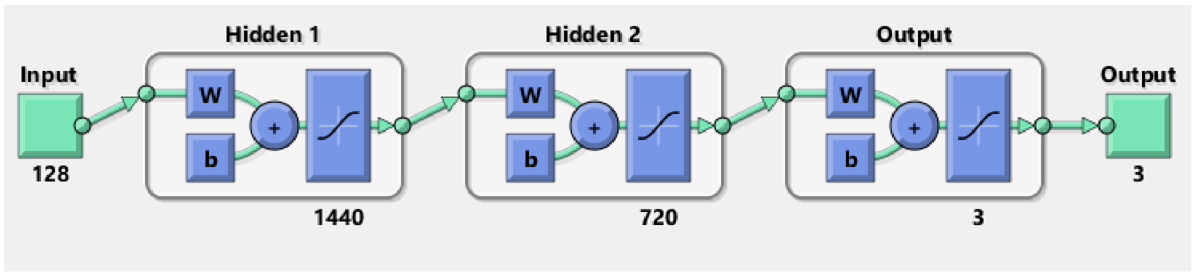

Fig 2.4 – Architecture of the neural network used

The preprocessing of images was done in C++ using *opencv library* [5] [6] [7] on Visual Studio 2010. This neural network was trained in MATLAB. The training was done for 1024 epochs using the Scaled Conjugate Gradient [8] training function, using MATLAB's *nprtool* toolbox. The trained network can be saved as the weight matrix of each neuron in each layer, and can be used later for simulating the network on any input image.

Simulation of the network is programmed in C++ using opencv. The program reads the text files generated by MATLAB having the weight matrices of the neural network, and the input image. Preprocessing is done on the input image, the read network is simulated on the preprocessed image, and the results are displayed. To consider rotation changes while simulating, a similar approach is used as the one used for training. The images of separate coins are rotated for 360 degrees generating 360 images. The network is simulated for these 360 images, and the final result for that coin's image is shown as the mean of these values.

3. References

[1] - http://en.wikipedia.org/wiki/Hough_transform

[2] - http://docs.opencv.org/doc/tutorials/core/basic_linear_transform/basic_linear_transform.html

[3] - http://en.wikipedia.org/wiki/Gaussian_blur

[4] - http://en.wikipedia.org/wiki/Canny_edge_detector

[5] - http://opencv.org/

[6] - http://en.wikipedia.org/wiki/OpenCV

[7] - http://docs.opencv.org/index.html

[8] - http://www.mathworks.in/help/nnet/ref/trainscg.html

4. Attachments

a) GenerateDataSet.cpp – Program to generate the preprocessed images from the given set of images.
b) TrainNetwork.m – MATLAB m file to read the preprocessed images from a folder named 'Generated' and train a neural network, saving the weight matrices as txt files.
c) Simulate.cpp – Program to read the weight matrices from the txt files and simulate a given input image on it named as 'Input.png'.