Advance Software Engineering Project

Recognizing Hand Written Digits and Characters

Software Requirement Specification

Group 5394_SM_2

Ravula, Balachandar
Kongara, Jyothesh
Pavuluri, Nitindra Chowdary
Sabah, Ayesha Fatima
Singh, Anurag

1. INTRODUCTION

Optical character recognition (OCR) is an important problem in computer vision, with applications ranging from efficiently scanning books to recognizing cheques in an ATM. We aim to give a simple solution to the problem of OCR via training a neural network. In particular, we approximately model the human brain as a weighted directed graph and use training samples to tune the weights, enabling the network to compute the character corresponding to the representation in an image. This gives a simple version of a general purpose linear classifier, more sophisticated version of which are used in industry to perform OCR. We propose to design and train a feed-forward, multilayer neural network with back propagation to perform optical character recognition.

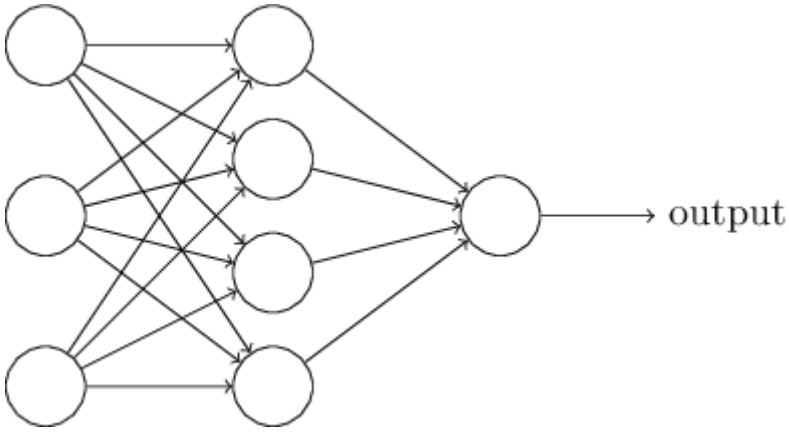
2. BACKGROUND

The human visual system is one of the wonders of the world. In each hemisphere of our brain, humans have a primary visual cortex, also known as V1, containing 140 million neurons, with tens of billions of connections between them. And yet human vision involves not just V1, but an entire series of visual cortices - V2, V3, V4, and V5 - doing progressively more complex image processing. We carry in our heads a supercomputer, tuned by evolution over hundreds of millions of years, and superbly adapted to understand the visual world. Recognizing handwritten digits isn't easy. Rather, we humans are stupendously, astoundingly good at making sense of what our eyes show us. But nearly all that work is done unconsciously. And so we don't usually appreciate how tough a problem our visual systems solve. The difficulty of visual pattern recognition becomes apparent if you attempt to write a computer program to recognize digits like those above. What seems easy when we do it ourselves suddenly becomes extremely difficult and turn out to be not so simple to express algorithmically.

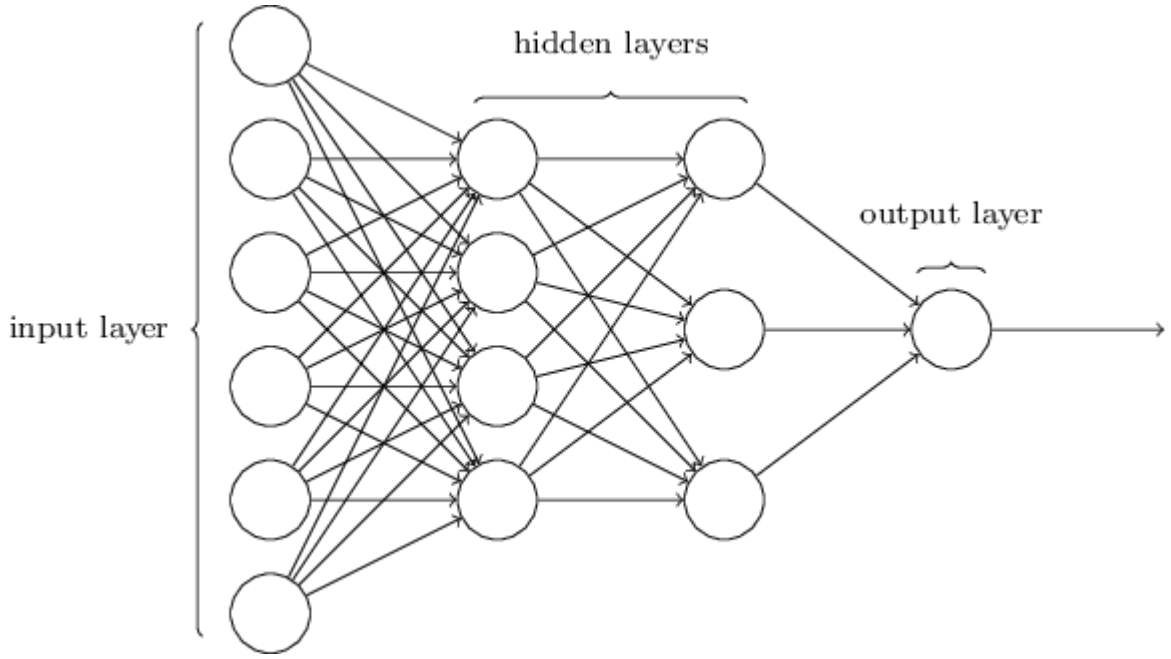
Neural networks approach the problem in a different way. The idea is to take a large number of handwritten digits, known as training examples, and then develop a system which can learn from those training examples. In other words, the neural network uses the examples to automatically infer rules for recognizing handwritten digits. Furthermore, by increasing the number of training examples, the network can learn more about handwriting, and so improve its accuracy.

3. ARCHITECTURE

Suppose we have the network



The leftmost layer in this network is called the input layer, and the neurons within the layer are called *input neurons*. The rightmost or *output* layer contains the *output neurons*, or, as in this case, a single output neuron. The middle layer is called a *hidden layer*, since the neurons in this layer are neither inputs nor outputs. The term "hidden" perhaps sounds a little mysterious - the first time I heard the term I thought it must have some deep philosophical or mathematical significance - but it really means nothing more than "not an input or an output". The network above has just a single hidden layer, but some networks have multiple hidden layers. For example, the following four-layer network has two hidden layers:



Somewhat confusingly, and for historical reasons, such multiple layer networks are sometimes called *multilayer perceptrons* or *MLPs*, despite being made up of sigmoid neurons, not perceptrons. I'm not going to use the MLP terminology in this book, since I think it's confusing, but wanted to warn you of its existence.

The design of the input and output layers in a network is often straightforward. For example, suppose we're trying to determine whether a handwritten image depicts a "9" or not. A natural way to design the network is to encode the intensities of the image pixels into the input neurons. If the image is a 64 by 64 greyscale image, then we'd have 4,096=64×64 input neurons, with the intensities scaled appropriately between 0 and 1. The output layer will

contain just a single neuron, with output values of less than 0.5 indicating "input image is not a 9", and values greater than 0.5 indicating "input image is a 9 ". While the design of the input and output layers of a neural network is often straightforward, there can be quite an art to the design of the hidden layers. In particular, it's not possible to sum up the design process for the hidden layers with a few simple rules of thumb. Instead, neural networks researchers have developed many design heuristics for the hidden layers, which help people get the behavior they want out of their nets. For example, such heuristics can be used to help determine how to trade off the number of hidden layers against the time required to train the network. We'll meet several such design heuristics later in this book.

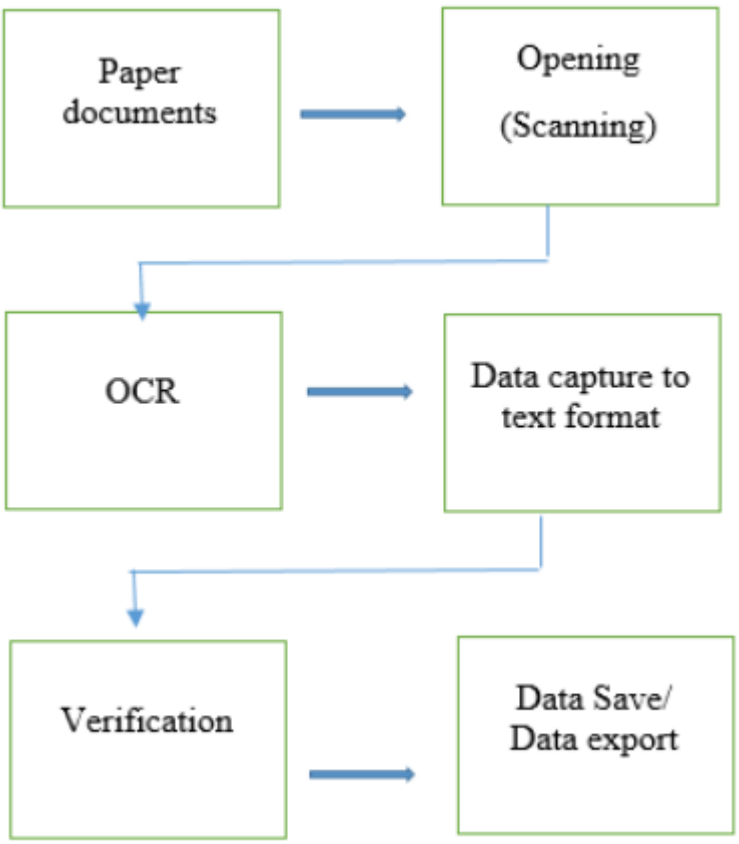


Figure 1. OCR process

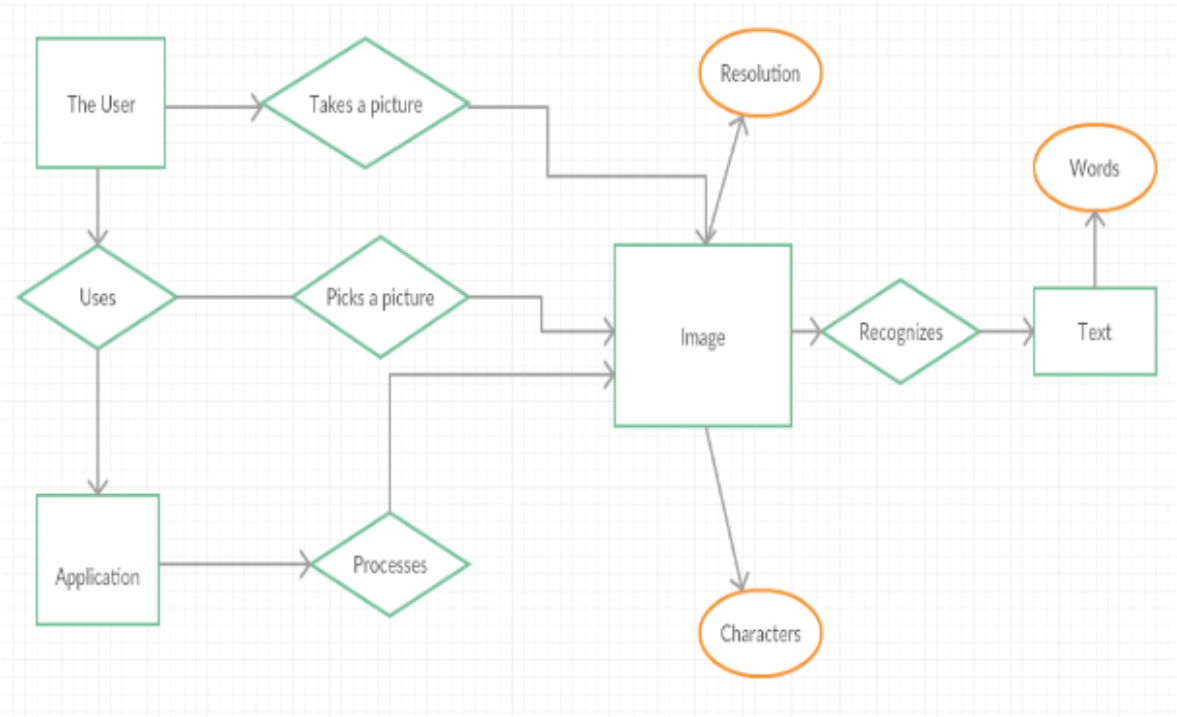


Figure 2. Architecture of proposed System

Up to now, we've been discussing neural networks where the output from one layer is used as input to the next layer. Such networks are called *feedforward* neural networks. This means there are no loops in the network - information is always fed forward, never fed back. If we did have loops, we'd end up with situations where the input to the σ function depended on the output. That'd be hard to make sense of, and so we don't allow such loops.

However, there are other models of artificial neural networks in which feedback loops are possible. These models are called recurrent neural networks. The idea in these models is to have neurons which fire for some limited duration of time, before becoming quiescent. That firing can stimulate other neurons, which may fire a little while later, also for a limited duration. That causes still more neurons to fire, and so over time we get a cascade of neurons firing. Loops don't cause problems in such a model, since a neuron's output only affects its input at some later time, not instantaneously.

Recurrent neural nets have been less influential than feedforward networks, in part because the learning algorithms for recurrent nets are (at least to date) less powerful. But recurrent networks are still extremely interesting. They're much closer in spirit to how our brains work than feedforward networks. And it's possible that recurrent networks can solve important problems which can only be solved with great difficulty by feedforward networks. However, to limit our scope, in this book we're going to concentrate on the more widely-used feedforward networks.

4. REQUIREMENTS

Hardware Requirements:

Operating System : Ubuntu, Windows 7/10

RAM : 4 GB

Disk : 80 GB

Software Requirements:

Installation : Java Development kit, Java Runtime Environment, Eclipse

4.1 Functional Requirements

We have classified these functional requirements as follow:

1. Draw the characters using mouse
2. Recognition of character
3. Displaying the character

4.1.1 Draw the character using mouse

Description

The User will draw a character or digit using mouse on a canvas. Once the character is drawn it is converted to image at the back end

4.1.2 Recognition of character

Description

The text will be recognized from the image picked up in the earlier step. The text will be recognized and ready to used.

1. Recognition of the text from the image
2. Ready to be used

4.1.3 Displaying the character

Description

Once the text is recognized and ready, the user will be able to see what character or digit it is.

4.2 Non Functional Requirements

After the functional requirements, we have been able to classify the non-functional requirements as follows:

4.2.1Product requirements:

Usability Requirements

The application shall be user friendly and doesn't require any guidance to use. In other words, the application has to be simple as possible, so its users shall use it easily. The interface is quite simple and straight forward so that anyone can understand and use it easily.

Reliability Requirements

The application should not have any unexpected failure. In order to avoid any failure occurrences, the specifications have been respected and followed correctly. The only problem that may occur in some cases is that the application does not get 100% of the characters or digits.

4.2.2 Efficiency Requirements

Performance

The application response time shall be adequate and sufficient enough, that's why the time required for this application to response to its user's actions has to be managed and controlled. But, in order to maintain the performance of the application, the user has to follow the required steps to get the desired result.

Portability

The application should be compatible with different operating systems like Windows 10 and Linux systems.

5. REFERENCES

- https://en.wikipedia.org/wiki/Optical_character_recognition
- https://en.wikipedia.org/wiki/Artificial_neural_network
- http://natureofcode.com/book/chapter-10-neural-networks/
- Pattern Classification by DUDA