NUMBER RECOGNITION USING CONVOLUTION NEURAL NETWORKS

Karthik S[1], Vaibhav Sharma[2]

[1] *Student, Electronics and communication department, SRM Institute of Science and Technology, Tamil Nadu, India*
[2] *Student, Electronics and communication department, SRM Institute of Science and Technology, Tamil Nadu, India*

ABSTRACT

This project is aimed at clarifying the role of Number Recognition in accordance with today's maturing technologies. Most people effortlessly recognize those digits as 504192. That ease is deceptive. 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. 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. It tries to list and clarify the components that build number recognition and related technologies such as OCR (Optical Character Recognition) and Image Recognition using machine learning. Images of digits were taken from a variety of sources, normalized in size and centered. This makes it an excellent dataset for evaluating models, allowing the developer to focus on the machine learning with very little data cleaning or preparation required. Results are reported using prediction error, which is nothing more than the inverted classification accuracy. It also has diversified applications in multiple fields such as in automatic number plate recognition and has security applications. It can also be used in archaeological surveys where digitization of archaic handwritten characters needs to be stored in a database.This technique offers an offline machine learning based algorithm to do the same.

Keyword: - *Machine learning, Digital image processing, Convolution neural networks, Artificial intelligence*

1. INTRODUCTION TO MACHINE LEARNING

Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to progressively improve performance on a specific task with user data, without being explicitly programmed.
The name machine learning was coined in 1959 by Arthur Samuel. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include email filtering, detection of network intruders or malicious insiders working towards a data breach, optical character recognition (OCR), learning to rank, and computer vision.
Tom M. Mitchell provided a widely quoted, more formal definition of the algorithms studied in the machine learning field: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience .This definition of the tasks in which machine learning is concerned offers a fundamentally operational definition rather than defining the field in cognitive terms. This follows Alan Turing's proposal in his paper "Computing Machinery and Intelligence", in which the question "Can machines think?" is replaced with the question "Can machines do what we

(as thinking entities) can do?". In Turing's proposal the various characteristics that could be possessed by a thinking machine and the various implications in constructing one are exposed.

1.1 Machine learning tasks

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:

- Supervised learning: The computer is presented with example inputs and their desired outputs, given by programmer, and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal
- can be only partially available, or restricted to special feedback:
- Semi-supervised learning: the computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.
- Active learning: the computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labelling.
- Reinforcement learning: training data (in form of rewards and punishments) is given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent.
- Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

1.2 Artificial neural networks

Artificial neural networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" (i.e. progressively improve performance on) tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analysing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any a priori knowledge about cats, e.g., that they have fur, tails, whiskers and cat-like faces. Instead, they evolve their own set of relevant characteristics from the learning material that they process. An ANN is based on a collection of connected units or nodes called artificial neurons (a simplified version of biological neurons in an animal brain). Each connection (a simplified version of a synapse) between artificial neurons can transmit a signal from one to another. The artificial neuron that receives the signal can process it and then signal artificial neurons connected to it.

1.3 Convolution neural networks

In common ANN implementations, the signal at a connection between artificial neurons are a real number, and the output of each artificial neuron is calculated by a non-linear function of the sum of its inputs. Artificial neurons and connections typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that only if the aggregate signal crosses that threshold is the signal sent. Typically, artificial neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times.

In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery.CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a

major advantage. They have applications in image and video recognition, recommender systems and natural language processing

2. IMAGE PROCESSING

The human visual system is one of the wonders of the world. Consider the following sequence of handwritten digits:

Fig -1: Sequence of Handwritten digits

Most people effortlessly recognize those digits as 504192. That ease is deceptive. 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. Simple intuitions about how we recognize shapes - "a 9 has a loop at the top, and a vertical stroke in the bottom right" - turn out to be not so simple to express algorithmically. When you try to make such rules precise, you quickly get lost in a morass of exceptions and caveats and special cases. It seems hopeless.

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.

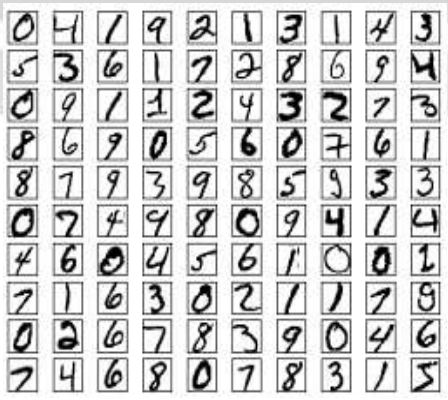

Fig -2: Pixels of Handwritten Digits

Furthermore, by increasing the number of training examples, the network can learn more about handwriting, and so improve its accuracy 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.

2.1 Preprocessing

In simple words, pre-processing refers to the transformations applied to your data before feeding it to the algorithm. In python, scikit-learn library has a pre-built functionality under sklearn .preprocessing.
Preprocessing stage has several tasks to be done:

- Binarization
- Noise filtering
- Smoothing
- normalization

Binary image is a digital image that has only two possible values for each pixel. Typically, the two colors used for a binary image are black and white. The color used for the object(s) in the image is the foreground color while the rest of the image is the background color.In the document-scanning industry, this is often referred to as "bi-tonal".

Binary images are produced from color images by segmentation. Segmentation is the process of assigning each pixel in the source image to two or more classes. If there are more than two classes then the usual result is several binary images. The simplest form of segmentation is probably Otsu's method which assigns pixels to foreground or background based on greyscale intensity. Another method is the watershed algorithm. Edge detection also often creates a binary image with some pixels assigned to edge pixels, and is also a first step in further segmentation.

Smooth a data set is to create an approximating function that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena. In smoothing, the data points of a signal are modified so individual points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased leading to a smoother signal. Smoothing may be used in two important ways that can aid in data analysis by being able to extract more information from the data as long as the assumption of smoothing is reasonable and by being able to provide analyses that are both flexible and robust.
In image processing, normalization is a process that changes the range of pixel intensity values. Applications include photographs with poor contrast due to glare, for example. Normalization is sometimes called contrast stretching or histogram stretching. In more general fields of data processing, such as digital signal processing, it is referred to as dynamic range expansion.
The purpose of dynamic range expansion in the various applications is usually to bring the image, or other type of signal, into a range that is more familiar or normal to the senses, hence the term normalization. Often, the motivation is to achieve consistency in dynamic range for a set of data, signals, or images to avoid mental distraction or fatigue. For example, a newspaper will strive to make all of the images in an issue share a similar range of gray scale.

2.2 Sub Image segmentation

Image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as super-pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.
The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image (see edge detection). Each of the pixels in a region is similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s).

2.3 Feature Extraction

Feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations. Feature extraction is related to dimensionality reduction.
When the input data to an algorithm is too large to be processed and it is suspected to be redundant (e.g. the same measurement in both feet and meters, or the repetitiveness of images presented as pixels), then it can be transformed

into a reduced set of features (also named a feature vector). Determining a subset of the initial features is called feature selection. The selected features are expected to contain the relevant information from the input data, so that the desired task can be performed by using this reduced representation instead of the complete initial data.

2.4 Pattern recognition

In machine learning, pattern recognition is the assignment of a label to a given input value. In statistics, discriminant analysis was introduced for this same purpose in 1936. An example of pattern recognition is classification, which attempts to assign each input value to one of a given set of *classes* (for example, determine whether a given email is "spam" or "non-spam"). However, pattern recognition is a more general problem that encompasses other types of output as well. Other examples are regression, which assigns a real-valued output to each input; sequence labeling, which assigns a class to each member of a sequence of values (for example, part of speech tagging, which assigns a part of speech to each word in an input sentence); and parsing, which assigns a parse tree to an input sentence, describing the syntactic structure of the sentence.

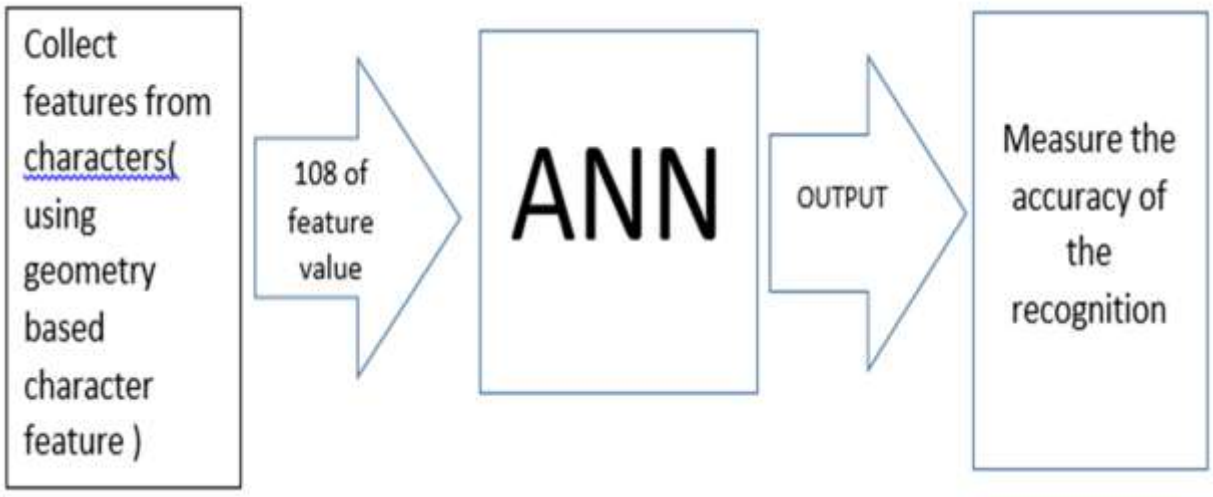

Fig -3: Preprocessing Characters

3. PERCEPTRONS

Perceptron's were developed in the 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts. Today, it's more common to use other models of artificial neurons - in this book, and in much modern work on neural networks, the main neuron model used is one called the sigmoid neuron.

So how do perceptrons work?
A perceptron takes several binary inputs, x1,x2,…, and produces a single binary output .The neuron's output, 0 or 1, is determined by whether the weighted sum ∑jwjxj is less than or greater than some threshold value. Just like the weights, the threshold is a real number which is a parameter of the neuron. To put it in more precise algebraic terms:

$$\text{output}=\begin{cases}0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold}\end{cases}$$

That's the basic mathematical model. A way you can think about the perceptron is that it's a device that makes decisions by weighing up evidence.
Obviously, the perceptron isn't a complete model of human decision-making! But what the example illustrates is how a perceptron can weigh up different kinds of evidence in order to make decisions. And it should seem plausible that a complex network of perceptron's could make quite subtle decisions:

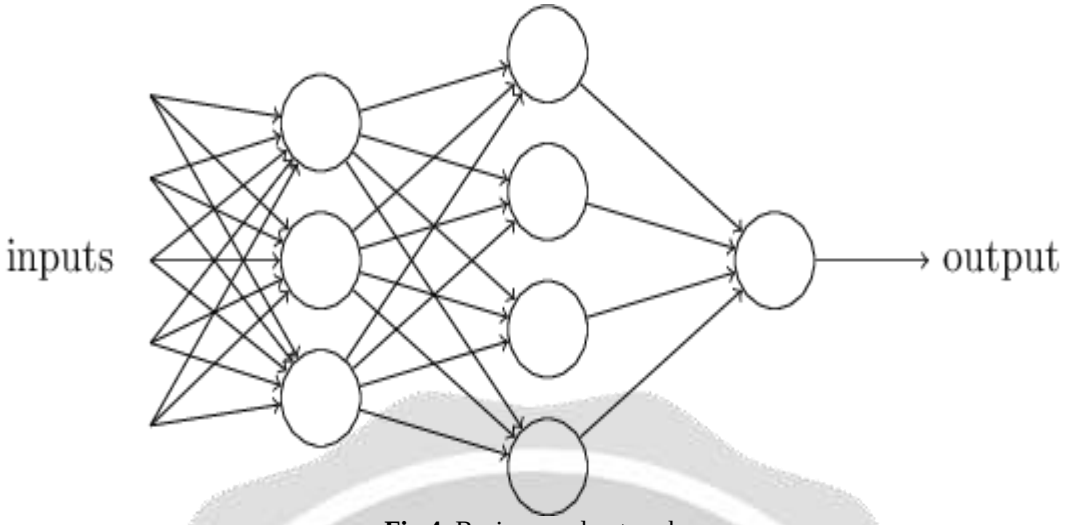

Fig 4: Basic neural network

In this network, the first column of perceptrons - what we'll call the first *layer* of perceptrons - is making three very simple decisions, by weighing the input evidence. What about the perceptrons in the second layer? Each of those perceptrons is making a decision by weighing up the results from the first layer of decision-making. In this way a perceptron in the second layer can make a decision at a more complex and more abstract level than perceptrons in the first layer. And even more complex decisions can be made by the perceptron in the third layer. In this way, a many-layer network of perceptrons can engage in sophisticated decision making.

The condition $\sum_j w_j x_j > \text{threshold}$ is cumbersome, and we can make two notational changes to simplify it. The first change is to write $\sum_j w_j x_j$ as a dot product, $w \cdot x \equiv \sum_j w_j x_j$, where w and x are vectors whose components are the weights and inputs, respectively. The second change is to move the threshold to the other side of the inequality, and to replace it by what's known as the perceptron's *bias*, $b \equiv -\text{threshold}$. Using the bias instead of the threshold, the perceptron rule can be rewritten:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Another way perceptron's can be used is to compute the elementary logical functions we usually think of as underlying computation, functions such as AND, OR, and NAND. For example, suppose we have a perceptron with two inputs, each with weight $-2-2$, and an overall bias of 33. Here's our perceptron:

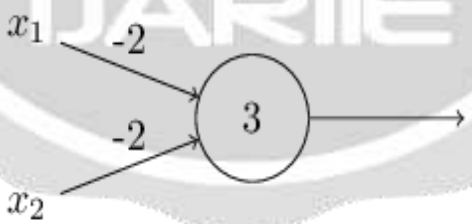

Fig 4: Example of perceptron

Then we see that input 0000 produces output 11, since $(-2)*0+(-2)*0+3=3(-2)*0+(-2)*0+3=3$ is positive. Here, I've introduced the $**$symbol to make the multiplications explicit. Similar calculations show that the inputs 0101 and 1010 produce output 11. But the input 1111produces output 00, since $(-2)*1+(-2)*1+3=-1(-2)*1+(-2)*1+3=-1$ is negative. And so our perceptron implements a NAND gate.

The inputs to the network might be the raw pixel data from a scanned, handwritten image of a digit. And we'd like the network to learn weights and biases so that the output from the network correctly classifies the digit. To see how learning might work, suppose we make a small change in some weight (or bias) in the network. What we'd like is for this small change in weight to cause only a small corresponding change in the output from the network.

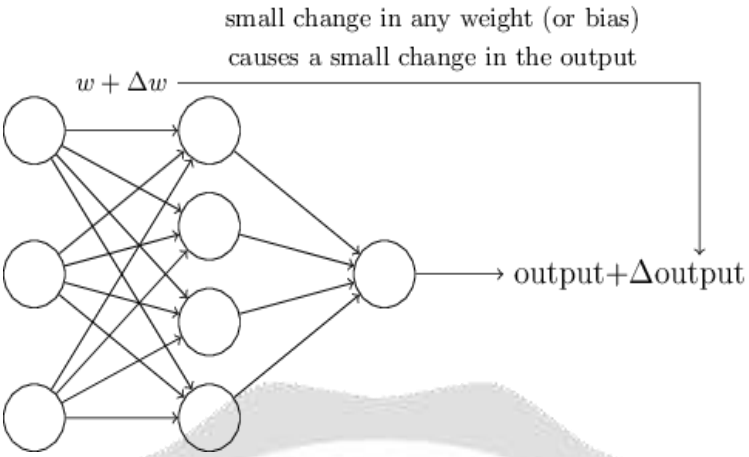

Fig 5: Operation of neural networks

If it were true that a small change in a weight (or bias) causes only a small change in output, then we could use this fact to modify the weights and biases to get our network to behave more in the manner we want. For example, suppose the network was mistakenly classifying an image as an "8" when it should be a "9". We could figure out how to make a small change in the weights and biases so the network gets a little closer to classifying the image as a "9". And then we'd repeat this, changing the weights and biases over and over to produce better and better output. The network would be learning. As mentioned earlier, 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.
For example, the following four-layer network has two hidden layers:

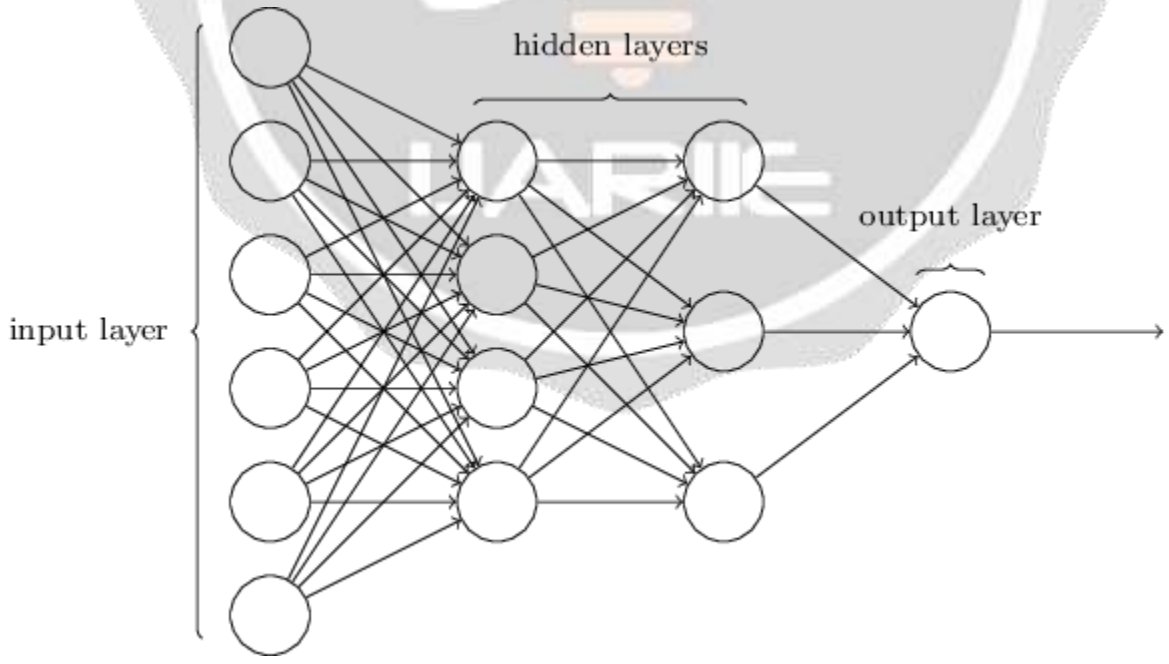

Fig 6: Layers in neural networks

Such multiple layer networks are sometimes called *multilayer perceptrons* or *MLPs*, despite being made up of sigmoid neurons, not perceptrons.

3.1 Handwriting recognition

We can split the problem of recognizing handwritten digits into two sub-problems. First, we'd like a way of breaking an image containing many digits into a sequence of separate images, each containing a single digit. For example, we'd like to break the image into six separate images,

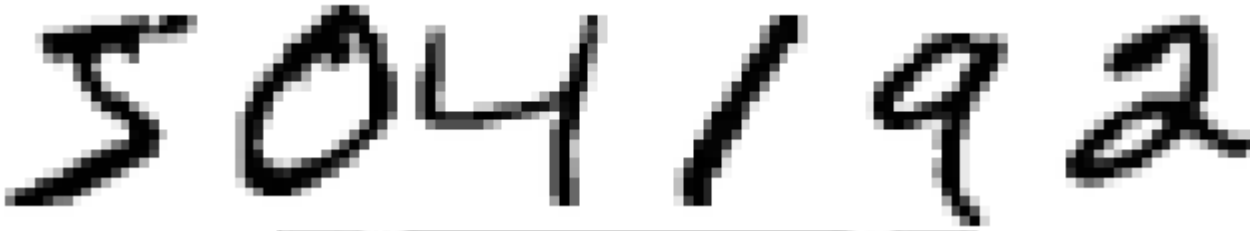

Fig 7: Handwritten characters

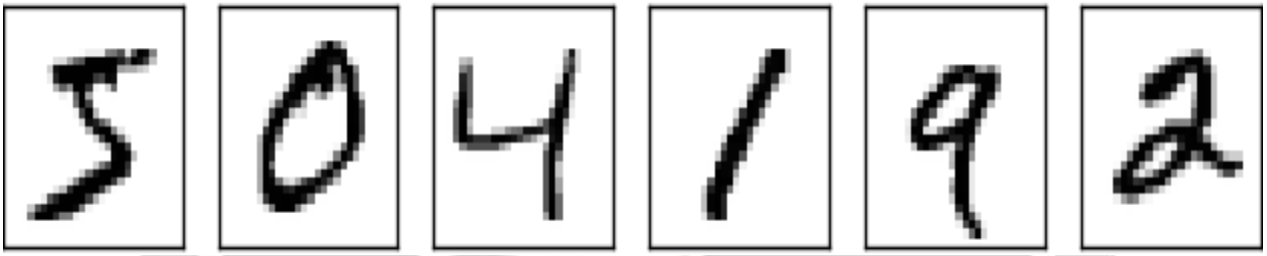

Fig 8: Segmentation Process

We humans solve this *segmentation problem* with ease, but it's challenging for a computer program to correctly break up the image. Once the image has been segmented, the program then needs to classify each individual digit. So, for instance, we'd like our program to recognize that the first digit above, is a 5.

Fig 9: Recognized output

We'll focus on writing a program to solve the second problem that is, classifying individual digits. We do this because it turns out that the segmentation problem is not so difficult to solve, once you have a good way of classifying individual digits. There are many approaches to solving the segmentation problem. One approach is to trial many different ways of segmenting the image, using the individual digit classifier to score each trial segmentation. A trial segmentation gets a high score if the individual digit classifier is confident of its classification in all segments, and a low score if the classifier is having a lot of trouble in one or more segments. The idea is that if the classifier is having trouble somewhere, then it's probably having trouble because the segmentation has been chosen incorrectly. This idea and other variations can be used to solve the segmentation problem quite well. So instead of worrying about segmentation we'll concentrate on developing a neural network which can solve the more interesting and difficult problem, namely, recognizing individual handwritten digits.
To recognize individual digits we will use a three-layer neural network:

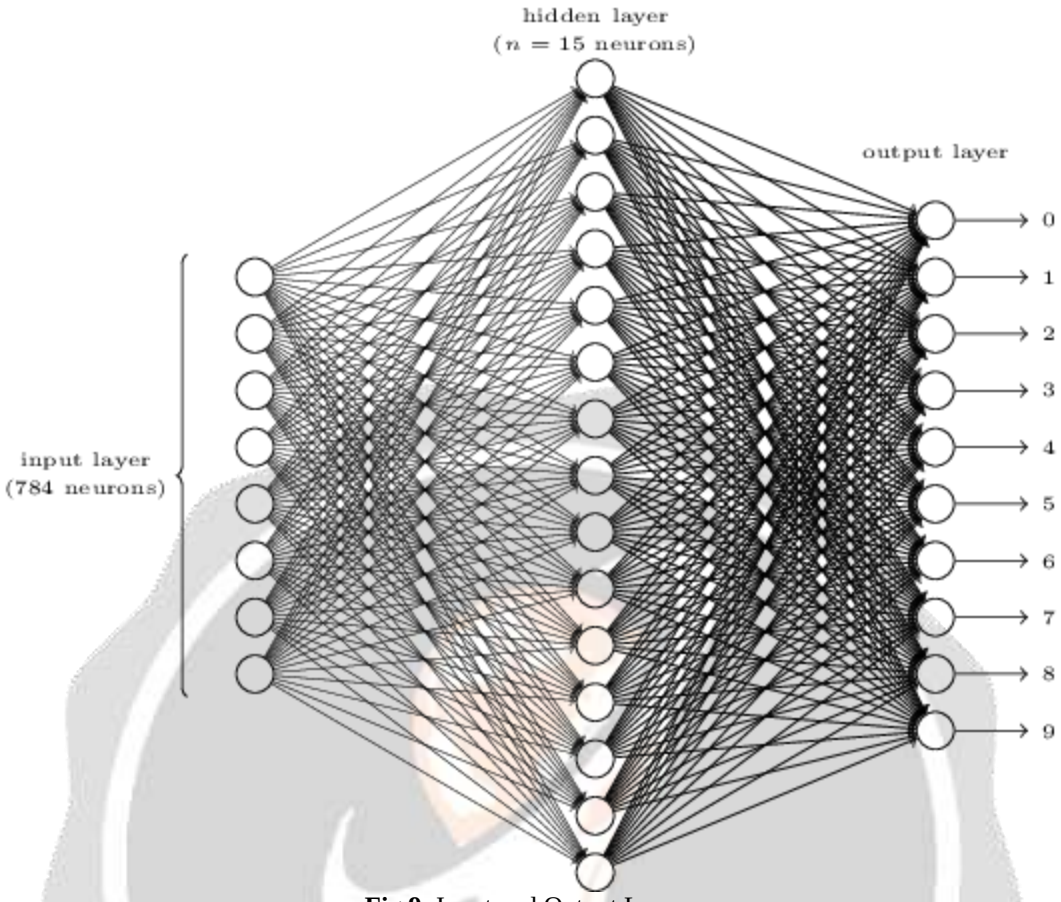

Fig 9: Input and Output Layers

The input layer of the network contains neurons encoding the values of the input pixels. As discussed in the next section, our training data for the network will consist of many 28 by 28 pixel images of scanned handwritten digits, and so the input layer contains 784=28×28 neurons. For simplicity we have omitted most of the 784 neurons in the diagram above. The input pixels are greyscale, with a value of 0 representing white, a value of 1representing black, and in between values representing gradually darkening shades of grey.

The second layer of the network is a hidden layer. We denote the number of neurons in this hidden layer by n, and we'll experiment with different values for n. The example shown illustrates a small hidden layer, containing just n=15 neurons.

The output layer of the network contains 10 neurons. If the first neuron fires, i.e., has an output ≈1, then that will indicate that the network thinks the digit is a 0. If the second neuron fires then that will indicate that the network thinks the digit is a 1. And so on. A little more precisely, we number the output neurons from 0 through 9, and figure out which neuron has the highest activation value. If that neuron is, say, neuron number 66, then our network will guess that the input digit was a 66. And so on for the other output neurons.

You might wonder why we use 1010 output neurons. After all, the goal of the network is to tell us which digit (0,1,2,…,9) corresponds to the input image. A seemingly natural way of doing that is to use just 4output neurons, treating each neuron as taking on a binary value, depending on whether the neuron's output is closer to 0 or to 1. Four neurons are enough to encode the answer, since 2^4=16 is more than the 10 possible values for the input digit. Why should our network use 10 neurons instead? Isn't that inefficient? The ultimate justification is empirical: we can try out both network designs, and it turns out that, for this particular problem, the network with 10 output neurons learns to recognize digits better than the network with 4 output neurons. But that leaves us wondering why using 10 output neurons works better. Is there some heuristic that would tell us in advance that we should use the 10s-output encoding instead of the 4-output encoding?

To understand why we do this, it helps to think about what the neural network is doing from first principles. Consider first the case where we use 10 output neurons. Let's concentrate on the first output neuron, the one that's trying to decide whether or not the digit is a 0. It does this by weighing up evidence from the hidden layer of

neurons. What are those hidden neurons doing? Well, just suppose for the sake of argument that the first neuron in the hidden layer detects whether or not an image like the following is present:

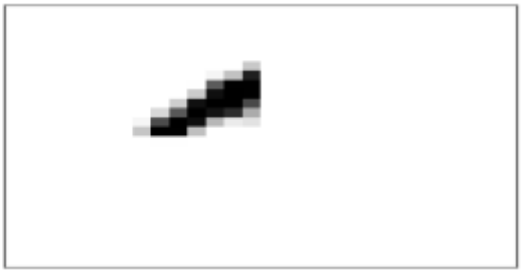

Fig 9: Input pixel

It can do this by heavily weighting input pixels which overlap with the image, and only lightly weighting the other inputs. In a similar way, let's suppose for the sake of argument that the second, third, and fourth neurons in the hidden layer detect whether or not the following images are present:

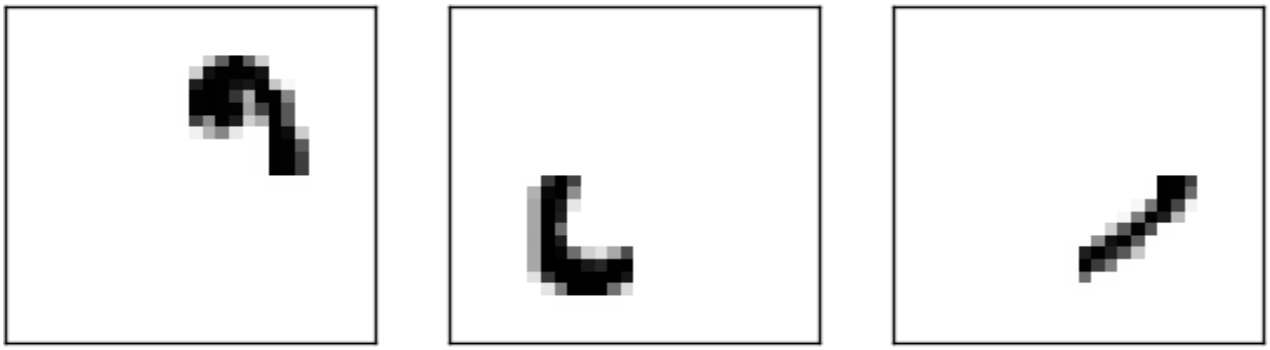

Fig 10: Segmentation of pixels

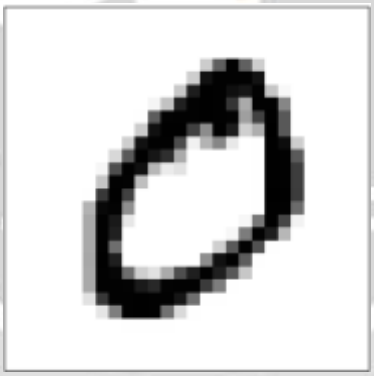

Fig 10: Hidden neurons

So if all four of these hidden neurons are firing then we can conclude that the digit is a 0. Of course, that's not the only sort of evidence we can use to conclude that the image was a 0, we could legitimately get a 0 in many other ways (say, through translations of the above images, or slight distortions). But it seems safe to say that at least in this case we'd conclude that the input was a 0.

Supposing the neural network functions in this way, we can give a plausible explanation for why it's better to have 10 outputs from the network, rather than 4. If we had 4 outputs, then the first output neuron would be trying to decide what the most significant bit of the digit was.

4. CONCLUSIONS

As per as technology is developing day by day the need of Artificial Intelligence is increasing because of only parallel processing. Parallel Processing is more needed in this present time because with only the help of parallel processing we can save more and more time and money in any work related to computers or robots. This paper presents a study of artificial neural networks for handwritten digit recognition. The traditional brute force methods for image comparison are too slow as well as inaccurate. The machine learning technique of logistic regression is also tested, in which the accuracy observed was about 94%. Finally, the technique using feed forward neural network proves to be the most efficient for a pattern recognition problem with an accuracy of about 97.5% in this case of handwritten digit recognition

Gradient Descent algorithm is an old but powerful algorithm for machine learning. And also, artificial neural networks prove to be a very effective solution for pattern matching problems. We plan to make this pattern matching neural network algorithms to solve medical problems like cancer detection with much more efficiency. The studies are going on, we have implemented that to some extent and we hope to see artificial neural network solve our daily life problems.

The benefit of this algorithm includes offline detection systems which are not provided as open source algorithms yet.

5. REFERENCES

[1]. R International Conference on Multidisciplinary Research & Practice Volume I Issue VIII IJRSI ISSN 2321-2705 Handwritten Digit Recognition using Neural Networks Mihir B. Thakkar Department of Computer Engineering, Nirma University, Ahmedabad, India.

[2]. Michael A Nielsen , "Neural Networks and Deep learning ",Determination Press ,2015