Vudoku - Visual Sudoku Solver

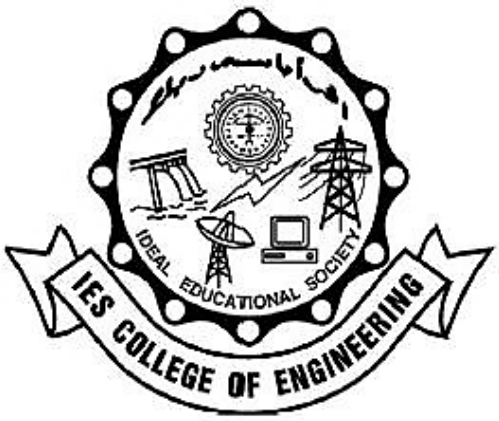
A project report submitted to APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering

by

Jovial Joe Jayarson (IES17CS016)

7th December 2020



Department of Computer Science and Engineering
IES College of Engineering, Chittilappilly - 680551

IES College of Engineering

Department of Computer Science and Engineering

Certificate

This is to certify that the project report entitled

Vudoku - Visual Sudoku Solver

submitted by

Jovial Joe Jayarson

in partial fulfilment of the requirements of the degree of *Bachelor of Technology in Computer Science and Engineering*, to APJ Abdul Kalam Technological University is a record of *bona fide* work done by him under my supervision and guidance and this work has not been submitted elsewhere for any degree or diploma.

Place _____	Mr. Ebin P M	Dr. Kiruthiga G
	Asst. Professor, CSE	Head of the Department
Date _____	(Guide)	CSE

Acknowledgements

I gladly present this report on *Vudoku - Visual Sudoku Solver* as a part of the final year B.Tech Computer Science and Engineering seminar. Let me take opportunity to first thank God the Almighty for providing His grace and guidance in this dispensation. I express my sincere thanks to Dr. Brilly S Sangeetha, principal for providing us with all the facilities we required to make this happen. I also acknowledge the ever encouraging presence of Dr. Kriuthiga G, head of the department. Heartfelt gratitude to my guide Mr. Ebin P M, Assistant Professor, for his undivided attention, support and coaching. Last but not the least I convey my regards to all the well wishers, family and friends who have helped me during the needed times.

May God bless us all.

Abstract

It is no secret that AI is an upcoming titan. Even though people are stunned to hear that AI has been here for around a century, due to the advancement in computational methods and resources AI peaks like never before. As a tiny glimpse into the field of Digit Recognition, this project aims to understand the underlying cogs and wheels on which the neural networks spin. Ranging from core mathematical functions to flowery programming, this project will include all that is essential to recognize a digit from an image. Even though the project is primarily research oriented, on the application part of it, this can be used to identify licence plate, read bank cheques, grade primary class mathematics homework. Currently the project tries to solve Sudoku drawn and written by hand The paraphernalia for this project includes programming language: Python3/Nim; libraries: opencv, pandas, numpy, keras/tensorflow, matplotlib; datasets: MNIST handwritten digit database. Digit recognition is a classical problem which will introduce neurons, neural networks, connections hidden layers, weights, biases, activation functions like sigmoid and ReLU, back-propagation and other topics as well.

Contents

List of Figures

Chapter 1

Introduction

Digit Recognition is a classical machine learning problem. The objective is implicitly evident, it is to identify and classify (usually) handwritten digits. Machines are inherently incapable to understand, text, images and audio. These formats which carry information, needs to be converted into numbers, matrices or vectors which is a significant step, in any procedure, to solve problems using computers. Another vital stage for digit recognition is *machine learning*. With classical problem solving techniques, humans provide computers with both data and rules as input. In contrast, we provides data and presumed result, as input to a machine learning system. And then we let it find out some *rules*. A quick example would be to give three numbers x, y and z and tell a machine that - certain operation was performed on x and y to obtain z. Once a machine is fed with lots and lots of similar examples, it will be equipped find out some pattern behind it. At a later stage, it can very well predict, what operation(s) might have been performed on x and y to result in z.

This report delves into a century long history of digit recognition, all the way from perceptron networks to recently released transformers. The means and ways, in which this problem was approached converges to SOTA (state of the art) models like in 1. This report explores *The Transformer*, released in 2017. *Attention* mechanism in the Transformer 2 has sparked a lot of interest in the research community, not to mention the buzz associated with GPT-3, due to its mind-boggling capabilities flaunted all across the media. The very next year a preprint on *Image Transformer* 3 was released. Interestingly Google research just released another preprint which dubs the whole idea of manipulating images with this new technique as *Vision Transformer* 4. Now at the core of this project, the digit recognition makes use of the Vision Transformer.

The project is named *Vudoku*, so the second part of the project involves solving classical computational problem. Sudoku is a mathematical problem and by default has 9×9 grids. There are other variant to it but to keep things simple, the project moves along with the default one. Sudoku is classified as Exact Cover or Hitting Set problem 5. Various algorithms has been developed over the years to solve it, like Backtracking, Stochastic algorithms, Constraint programming and so on. Given that the output of the digit recognition system is as *value*, *position* pairs, this project will consider some of these algorithms to solve Sudoku and possibly visualize the steps.

Chapter 2

Overview and Prerequisites

2.1 Overview of the Topic

Complex problems exists all around us. Some of them are stumbled upon and other have be thoughtfully crafted. Solutions to these problems at times are intuitive and yet difficult to explain. Structured mathematical approach to problems has engineered some of the finest solutions to some of the toughest problem. In the era of computing, appetite to solve problems using machines has become a hobby. With technology at fingertips, it's no-brainer that upcoming generations have a choice to be way smarter.

Speaking of generations, there are puzzles and trivia question that people like to solve. These are not always driven by any particular motive. Sudoku is a famous mathematical puzzle. The rules to solve this can be summarized as: *All grids must be filled with numbers from 1 to 9 without repetition in either row, column or box.* Today any computer can solve this within seconds - if given with proper parameters. But computers by default do not possess visual perception. This is where image recognition steps in. This project can be thought of as an amalgamation of machine learning and classical computing. Instead of feeding a system with direct Sudoku *clues*, an image of Sudoku is provided. The computer has to derive the clues out of an image and then pass them along with correct positions to the second modules which then tries to solve Sudoku.

2.2 Prerequisites for the Project

Since this project strives to reach minimum research standard, reader is advised to have knowledge about related topics. This section tries to cover some of them but the rest is left at reader's discretion to follow up.

2.2.1 Mathematics to solve Sudoku

NP-Complete

In computational complexity theory, a problem is NP-complete when:

- A nondeterministic Turing machine can solve it in polynomial-time.

- A deterministic Turing machine can solve it in large time complexity classes and can verify its solutions in polynomial time.
- It can be used to simulate any other problem with similar solvability. 6

Combinatorics

Combinatorics is an area of mathematics primarily concerned with counting, both as a means and an end in obtaining results 7.

Group Theory

In mathematics and abstract algebra, group theory studies the algebraic structures known as groups. In mathematics, a group is a set equipped with a binary operation that combines any two elements to form a third element in such a way that four conditions called group axioms are satisfied, namely 'closure', 'associativity', 'identity' and 'invertibility' 8 9.

2.2.2 Machine Learning and related Terminology

Gradient Descent

Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent. Conversely, stepping in the direction of the gradient will lead to a local maximum of that function; the procedure is then known as gradient ascent. 10.

Fourier Transform

In mathematics, a Fourier transform (FT) is a mathematical transform that decomposes a function (often a function of time, or a signal) into its constituent frequencies, such as the expression of a musical chord in terms of the volumes and frequencies of its constituent notes. The term Fourier transform refers to both the frequency domain representation and the mathematical operation that associates the frequency domain representation to a function of time 11 This is used in image preprocessing step in CNN 5.2.1.

Activation Function

In artificial neural networks, the activation function of a node defines the output of that node given an input or set of inputs. A standard integrated circuit can be seen as a digital network of activation functions that can be "ON" or "OFF", depending on input. 12

Chapter 3

Literature Review

3.1 Handwritten Digit Recognition Using Machine Learning : A Review

Abstract (as is): The task for handwritten digit recognition has been troublesome due to various variations in writing styles. Therefore, we have tried to create a base for future researches in the area so that the researchers can overcome the existing problems. The existing methods and techniques for handwritten digit recognition were reviewed and understood to analyze the most suitable and best method for digit recognition. A number of 60,000 images were used as training sets of images with pixel size of 28×28. The images/training sets were matched with original image. It was found out after complete analysis and review that classifier ensemble system has the least error rate of just 0.32%. In this paper, review of different methods handwritten digit recognition were observed and analyzed.

3.2 An ensemble of simple convolutional neural network models for MNIST digit recognition

Abstract (as is): We report that a very high accuracy on the MNIST test set can be achieved by using simple convolutional neural network (CNN) models. We use three different models with 3×3, 5×5, and 7×7 kernel size in the convolution layers. Each model consists of a set of convolution layers followed by a single fully connected layer. Every convolution layer uses batch normalization and ReLU activation, and pooling is not used. Rotation and translation is used to augment training data, which is frequently used in most image classification tasks. A majority voting using the three models independently trained on the training data set can achieve up to 99.87% accuracy on the test set, which is one of the state-of-the-art results. A two-layer ensemble, a heterogeneous ensemble of three homogeneous ensemble networks, can achieve up to 99.91% test accuracy. The results can be reproduced by using the code at https://github.com/ansh941/MnistSimpleCNN.

3.3 An Image is Worth 16x16 Words : Transformers For Image Recognition at Scale

Abstract (as is): While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.

Chapter 4

Evolution of Neural Networks

It's of no surprise that the field of artificial intelligence has come a long way. This section deals with some of the famous models and how they are related to digit recognition. Starting from the ones released in 80's like *Recurrent Neural Network (RNN)* to the latest 2017 model *The Transformer*, much study has been conducted.

4.1 Recurrent Neural Network

Year: 1986

Recurrent neural networks are derived from feed-forward networks. The connections in a feed-forward network do not form a cycle. RNN is a class of neural network where the connections between nodes form a directed graph. This is always along a temporal sequence, so it exhibits a temporal dynamic behaviour. Dynamic systems are those that have a function which describes time dependence of a point in geometrical space. RNNs can use their internal state (or memory) to process variable length sequences of inputs 13. Section 5.1 discusses more about vanilla networks.

Basic RNNs are a network of neuron-like nodes organized into successive layers. Each node in a given layer is connected with a directed (one-way) connection to every other node in the next successive layer. Each node (neuron) has a time-varying real-valued activation. Each connection (synapse) has a modifiable real-valued weight. Nodes are either input nodes (receiving data from outside of the network), output nodes (yielding results), or hidden nodes (that modify the data en route from input to output). Each sequence produces an error as the sum of the deviations of all target signals from the corresponding activations computed by the network. For a training set of numerous sequences, the total error is the sum of the errors of all individual sequences. The following equation 4.1 was proposed by **Michael Irwin Jordan** as part of his research on *Simple Recurrent Neural Network* 13:

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h y_{t-1} + b_h) \\ y_t &= \sigma_y(W_y h_t + b_y) \end{aligned} \tag{4.1}$$

Where, x_t is input vector, h_t is hidden layer vector, y_t is output vector, W, U and b are parameter matrices and vector, respectively σ_h and σ_y are activation functions

The following figure 4.1 gives the internal schematics of Recurrent Neural Networks.

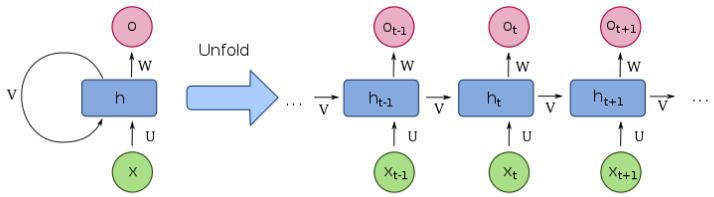


Figure 4.1: Recurrent Neural Networks Unfolded 13

Recurrent neural networks models contain a self-connected hidden layer. One benefit of the recurrent connection is that a 'memory' of previous inputs remains in the network's internal state, allowing it to make use of past context. This is significant for handwriting recognition as mentioned in 14. A major problem with gradient descent for standard RNN architectures is that error gradients vanish exponentially quickly with the size of the time lag between important events 13.

4.2 Long Short Term Memory

Year: 1997

Long short-term memory (LSTM) is a deep learning system that avoids the vanishing gradient problem. LSTM is normally augmented by recurrent gates called "forget gates". LSTM prevents backpropagated errors from vanishing or exploding. Instead, errors can flow backwards through unlimited numbers of virtual layers unfolded in space. That is, LSTM can learn tasks that require memories of events that happened thousands or even millions of discrete time steps earlier 13.

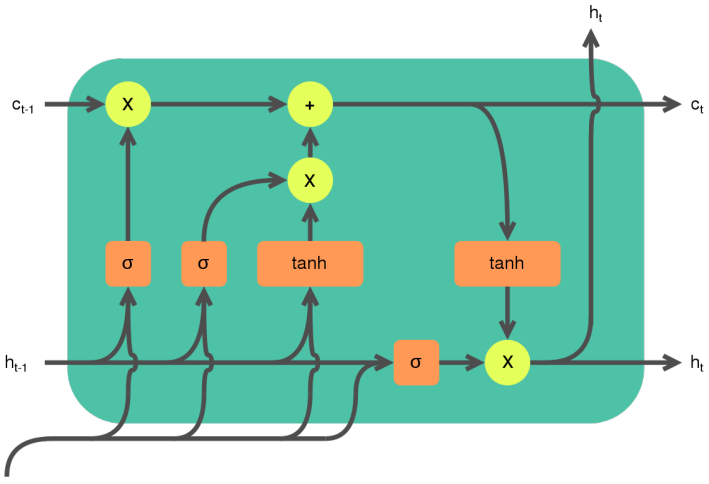


Figure 4.2: Long Short Term Memory Cell 15

A common LSTM unit, as shown in figure 4.2, is composed of a cell, an *input gate*, an *output gate* and a *forget gate*. The cell remembers values over arbitrary

time intervals and the three gates regulate the flow of information into and out of the cell. Some variations of the LSTM unit do not have one or more of these gates or maybe have other gates. For example, *Gated Recurrent Units* (GRUs) do not have an output gate. LSTM networks are hence well-suited for classifying time series data.

The compact forms of the equations for the forward pass of an LSTM unit with a forget gate are:

$$\begin{aligned} f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\ \tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\ h_t &= \sigma_t \circ \sigma_h(c_t) \end{aligned} \tag{4.2}$$

The initial values are $c_0 = 0$ and $h_0 = 0$ and the operator $\circ$ denotes the Hadamard element-wise product. Activations are σ_g - a sigmoid function, σ_c and σ_h - a hyperbolic tangent functions. The variable are 15:

x_t : input vector to the LSTM unit
f_t : forget gate's activation vector
i_t : input or update gate's activation vector
o_t : output gate's activation vector
h_t : hidden state vector
$\tilde{c}_t$: cell input activation vector
c_t : cell state vector

4.3 Convolution Neural Networks

Year: 1998

A convolutional neural network (CNN), is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or *space invariant artificial neural networks* (SIANN). CNNs have a wide variety of applications including, image and video recognition, recommender system, brain-computer interface etc.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Figure 4.3 shows a common CNN architecture. CNNs use relatively less preprocessing compared to other image classification algorithms. Independence from prior knowledge and human effort in feature design is a major advantage 16. More about CNN in section 5.2.

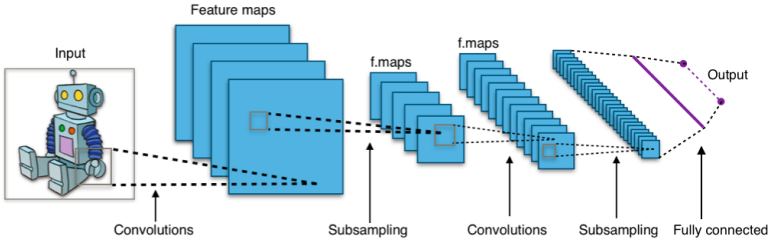


Figure 4.3: Typical CNN Architecture 16

4.4 The Transformer Network

Year: 2017

The Transformer is a deep learning model primarily used in the field of natural language processing (NLP). Transformers are designed to handle sequential data, such as natural language unlike RNNs, Transformers do not require that the sequential data be processed in order. Thus Transformer allows for much more parallelization than RNNs and therefore reduced training hours. This has led to the development of pre-trained systems such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer), which have been trained with huge general language datasets, and can be fine-tuned to specific language tasks.

Transformer is an encoder-decoder architecture. The *encoder* consists of a set of encoding layers that processes the input iteratively one layer after another and the *decoder* consists of a set of decoding layers that does the same thing to the output of the encoder. Each encoder and decoder layer makes use of an *attention mechanism*, which for each input, weighs the relevance of every other input and draws information from them accordingly to produce the output. Each decoder layer also has an additional attention mechanism which draws information from the outputs of previous decoders, before the decoder layer draws information from the encodings. Both the encoder and decoder layers have a feed-forward neural network for additional processing of the outputs, and contain residual connections and layer normalization steps 17.

The basic building blocks of the Transformer are scaled dot-product attention units. When a sentence is passed into a Transformer model, attention weights are calculated between every token simultaneously. The attention unit produces embeddings for every token in context that contain information not only about the token itself, but also a weighted combination of other relevant tokens weighted by the attention weights. More about Transformer and Attention in section 5.3.

Chapter 5

Concurrent Systems

There are several existing systems or models which can perform digit recognition. Some of these even have state of the art accuracy claims. This section delves into two significant artificial neural network models, namely *The Perceptron* and *Convolutional Neural Network*. Even though these networks can perform a variety of tasks, the focus will be upon digit classification.

5.1 Vanilla ANN - The perceptron

Vanilla Artificial Neural Network (ANN) is a primitive model. They are computing systems vaguely inspired by the biological neural networks. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons.

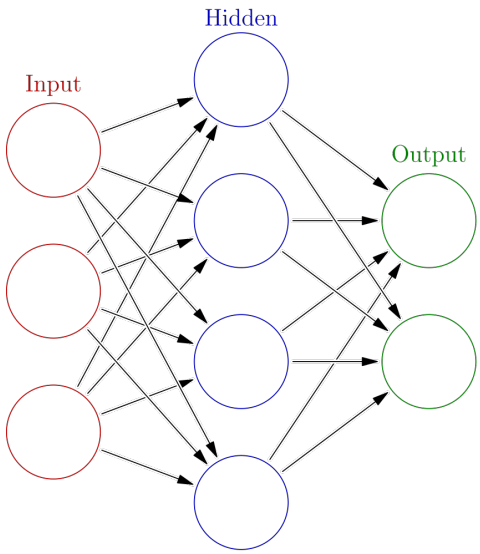


Figure 5.1: Simple Artificial Neural Network 18

An artificial neuron that receives a signal then processes it and can signal neurons connected to it. The 'signal' at a connection is a real number, and the output of

each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold.

Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times. 18

5.1.1 Neurons

ANNs are composed of artificial neurons, each artificial neuron has inputs and produce a single output which can be sent to multiple other neurons. To find the output of the neuron, first we take the weighted sum of all the inputs, weighted by the weights of the connections from the inputs to the neuron. We add a bias term to this sum. This weighted sum is then passed through a (usually nonlinear) activation function to produce the output 18.

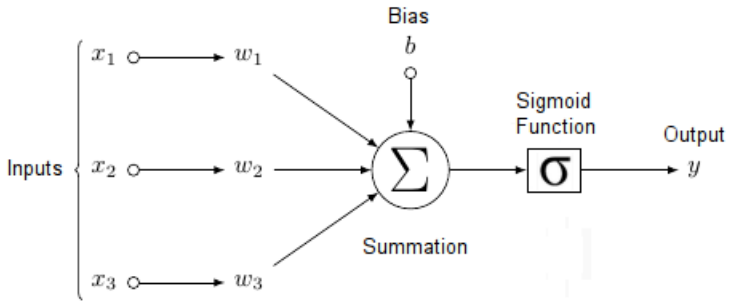


Figure 5.2: Artificial Neuron

5.1.2 Weights and Biases

The network consists of connections, each connection providing the output of one neuron as an input to another neuron. Each connection is assigned a *weight* w_i, that represents its relative importance. A given neuron can have multiple input and output connections. The *activation* function σ, computes the input to a neuron from the outputs of its predecessor neurons and their connections as a weighted sum as shown in figure 5.2 A *bias* b, term can be added to the result of the forward propagation 18.

$$a_i^{(j)} = \sigma(w_{i0}a_0^{(j)} + w_{i1}a_1^{(j)} + \ldots + w_{in}a_n^{(j)} + b_0) \tag{5.1}$$

The equation 5.1 is an example of an output from a single layer, it can be generalized and expressed neatly as in 5.2.

$$a^{(L)} = \sigma(W^{(L)}a^{(L-1)} + b^{(L)}) \tag{5.2}$$

5.1.3 Hyperparameters

A hyperparameter is a constant parameter whose value is set before the learning process begins. The values of parameters are derived via learning. Examples of hyperparameters include learning rate, the number of hidden layers and batch size. The values of some hyperparameters can be dependent on those of other hyperparameters. For example, the size of some layers can depend on the overall number of layers 18.

5.1.4 Learning or Training

Learning is the adaptation of the network to better handle a task by considering sample observations. Learning involves adjusting the weights (and optional thresholds) of the network to improve the accuracy of the result. This is done by minimizing the observed errors. Learning is complete when examining additional observations does not usefully reduce the error rate 18.

5.1.5 Backpropagation

Backpropagation is a method to adjust the connection weights to compensate for each error found during learning. The error amount is effectively divided among the connections. Technically, Backpropagation calculates the gradient (the derivative) of the cost function associated with a given state with respect to the weights. The weight updates can be done via stochastic gradient descent or other methods 18.

5.1.6 Digit Recognition

Since supervised model is used to create a a digit classifier, during the training dispensation the models is provided with correctly labeled outputs (say y). If errors arise, they need to be minimized. The cost function is calculated as the square of the difference between the expected and actual output.

$$C_0 = (a^{(L)} - y)^2 \tag{5.3}$$

The RHS of 5.2 can re-written as $\sigma(z^{(L)})$. To minimize cost function the derivative needs to be found, and here chain rule can be applied:

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C_0}{\partial a^{(L)}} \tag{5.4}$$

Therefore for a single training sample the minimized cost function will be as in 5.5 and the net cost ∇C will be as shown in 5.7.

$$\frac{\partial C_0}{\partial w^{(L)}} = a^{(L-1)} \cdot \sigma'(z^{(L)}) \cdot 2(a^{(L)} - y) \tag{5.5}$$

$$\frac{\partial C}{\partial w^{(L)}} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}} \tag{5.6}$$

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(0)}} \\ \frac{\partial C}{\partial w^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \end{bmatrix} \tag{5.7}$$

This is propagated backwards, errors are minimised after each sample and hence the digits are recognized correctly 19.

5.2 Digit Recognition with CNN

As stated in section 4.3, digit recognition is a powerful deep learning model mostly used with digital image manipulation tasks. It is unsurprisingly complex, yet beautiful in construct. The accuracy with which CNNs classify object is at par. No wonder why the current SOTA holder is a CNN based model. This section tries to delve a little more deeper into what exactly CNNs does under the hood.

5.2.1 Object Recognition Fundamentals

General flow of digit recognition includes 20:

1. **Image Acquisition :** This stage receives an input image which can from in any format from any source. They can be of image formats like: .png, .jpeg or capture from a camera or scanner.

2. **Preprocessing :** Here the input image undergoes a number of procedures which tries to minimize those factors which may cause the accuracy to drop. It usually works upon images to reduce and if possible remove distortions.

3. **Segmentation :** It is used to isolate focused parts of the input. Histogram process is use retrieve the entities from the image. Contents are processed in a tree like structure.

4. **Feature Extraction :** Converts an image into *n-dimensional array tensor*. It extracts those features from the sample which are most importance for classification. Various techniques include Principle Component Analysis (PCA), Chain Code (CC) etc.

5. **Classification :** Extracted characteristics is used to make decisions. Different classifier algorithms are used and they compare the input with the training sample. *Softmax* regressions is applied to rate the accuracy in terms of probability.

6. **Post Processing :** To correct misclassified results, shape recognition is used. Then along with the language knowledge, the errors can be minimized in case of handwriting recognition.

5.2.2 Demystifying CNNs

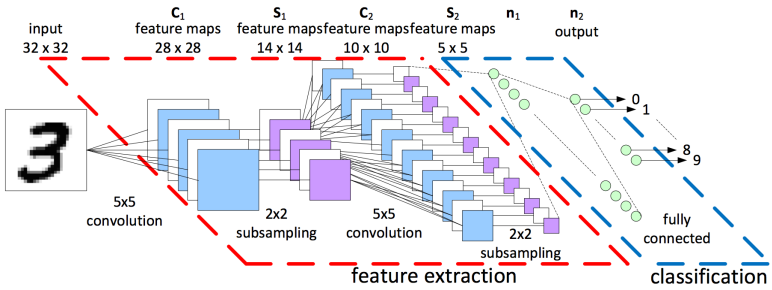


Figure 5.3: Layers in Convolutional Neural Network

Tensor

In mathematics, a tensor is an algebraic object that describes a (multilinear) relationship between sets of algebraic objects related to a vector space. Objects that tensors may map between, include vectors and scalars, and even other tensors. Tensors can take several different forms – for example: scalars and vectors etc 21.

Convolution

Convolutional layers convolve the input and pass its result to the next layer. The input is a tensor with shape (number of images) x (image height) x (image width) x (image depth). It becomes abstracted to a feature map, with shape (number of images) x (feature map height) x (feature map width) x (feature map channels) 16.

Pooling

Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2×2. They are applied with a stride S, of 2 downsamples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations 16:

$$f_{X,Y}(S) = \max_{a,b=0}^{1} S_{2X+a,2Y+b} \tag{5.8}$$

Fully Connected Layer

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images 16 and thus recognize handwritten digits aptly.

5.3 Vision Transformer

On October 22, 2020 Google Research, Brain Team released an arXiv preprint on *Vision Transformer* 4. A paper on *Image Transformer* was released in 2018 3 which performed tasks like generative image modelling and super-resolution scaling. The latter model as expected diverted to fulfill different purpose. Inspired by the Transformer scaling successes in NLP, 4 experiments with applying a standard Transformer 2 directly to images, with the fewest possible modifications.

5.3.1 ViT Architecture

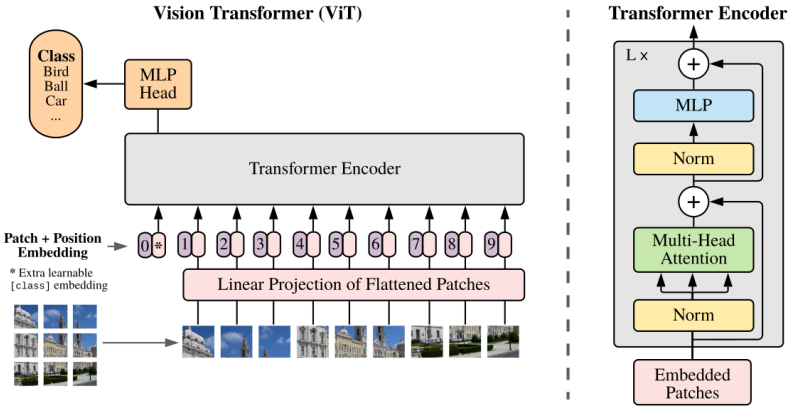


Figure 5.4: Vision Transformer Architecture 4

The input image is split into fixed-size patches and each one of them is linearly embedded and position embeddings are added. The resulting sequence of vectors is fed to a standard Transformer encoder 2. In order to perform classification, ViT uses the standard approach of adding an extra learnable 'classification token' to the sequence.

The standard Transformer receives as input a 1D sequence of token embeddings. To handle 2D images, image $x \in \mathbb{R}^{H \times W \times C}$ is reshaped into a sequence of flattened 2D patches $x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$ where:

- (H, W) is the resolution of original image,
- C is the number of channels
- (P, P) is the resolution of each image patch and
- $N = HW/P^2$ is the resulting number of patches.

The Transformer uses constant latent vector size D throughout all of it layers fo the flatten patches are mapped to D dimensions, with a trainable linear projection. This is termed as *patch embeddings*. A learnable embedded is pre-appended to the sequence embedding, whose state at the output of the Transformer encoder $(\mathbf{z}_L^0)$ serves as the image representation y. Both during pre-training and fine-tuning, a classification head is attached to $(\mathbf{z}_L^0)$. The classification head is implemented by a

Multi-layer Perceptron (MLP) with one hidden layer at pre-training time and by a single linear layer at fine-tuning time 4.

5.3.2 Hybrid Architecture

As an alternative to raw image patches, the input sequence can be formed from feature maps of a CNN 22. In this hybrid model, the patch embedding projection is applied to patches extracted from a CNN feature map. As a special case, the patches can have spatial size 1×1, which means that the input sequence is obtained by simply flattening the spatial dimensions of the feature map and projecting to the Transformer dimension. The classification input embedding and position embeddings are added as described above 4.

5.3.3 Fine Tuning ViT

Typically, ViT is pre-trained on large datasets, and fine-tuned to (smaller) downstream tasks. For this, the pre-trained prediction head is removed and a zero-initialized $D \times K$ feed-forward layer is attached, where K is the number of downstream classes. It is often beneficial to fine-tune at higher resolution than pre-training. When feeding images of higher resolution, the patch size is kept the same, which results in a larger effective sequence length. The Vision Transformer can handle arbitrary sequence lengths (up to memory constraints), however, the pre-trained position embeddings may no longer be meaningful. Therefore 2D interpolation of the pre-trained position embeddings is performed, according to their location in the original image 4.

Chapter 6

Proposed System

We've so far approached the project from machine learning perspective. Being a research oriented project, it focuses on the field of digit recognition. Yet to remind the reader of another part which is as relevant is Sudoku. Solving Sudoku might sound simple initially and that is quite natural. Our cognitive thinking skills tackles a problem with vigorous intuitiveness. But a machine needs to be told to solve a task in the sequence of steps, which is called an algorithm.

Various algorithms exists to solve Sudoku, some of which are 'backtracking', 'stochastic search', 'exact cover' and 'constrain programming' 23. This project can be said to have two broad parts - *Artificial Intelligence* and *Sudoku Solver*. Figure 6.1 showcases an overly simplified project logic.

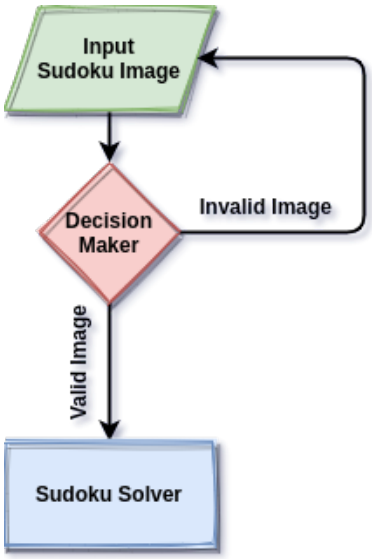


Figure 6.1: Vudoku: Simplified Architecture

The fundamental idea is to capture Sudoku pattern from images (even hand-written ones) using the latest Vision Transformer and solve them by an effective or optimal algorithm.

6.1 Project Modules

Applying divide and conquer method, this project can be broken in to different segments based on their functionality. Broadly speaking this project has a AI and non-AI module. The AI comprises of the *decision maker* and non-AI that of *Sudoku solver* as in figure 6.1.

6.1.1 Image Input Module

A little more expanded in architecture is shown in figure 6.2. The input image is split into patches, which is depicted by those yellow blocks, as discussed in 5.3.1. They are encoded and positional embeddings are provided. As it reaches the Transformer encoder, they are converted into computer readable vectors. A hypothesis is that the output in desired format can be extrapolated from the decoder.

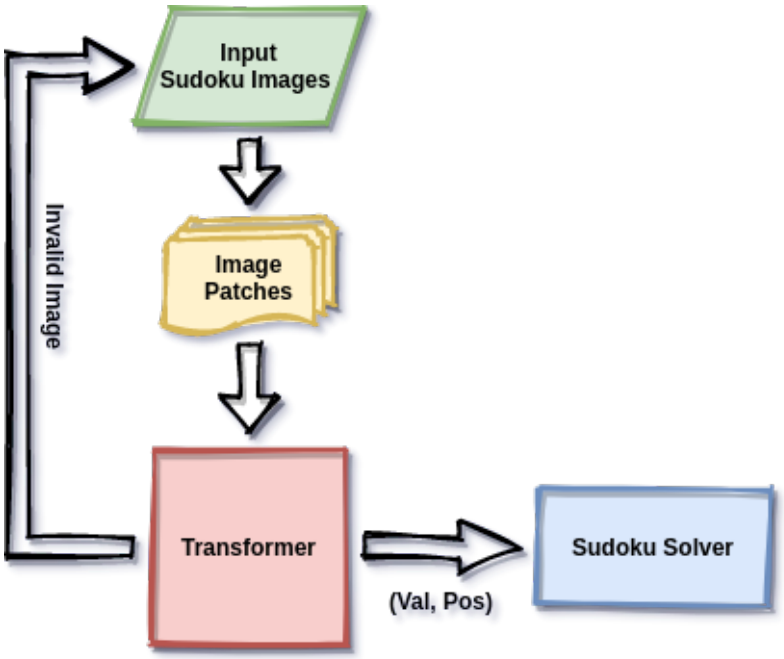


Figure 6.2: Vudoku: Expanded Architecture

6.1.2 Encoder and Decoder

The Vision Transformer (ViT) 5.3 tires to keep to the original Transformer architecture, hence similar encoder and decoder are used.

Encoder

The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. A residual connection 24 around each of the two sub-layers, followed by layer normalization 25. That is,

the output of each sub-layer is $LayerNorm(x + Sublayer(x))$, where $Sublayer(x)$ is the function implemented by the sub-layer itself 2.

Decoder

The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, residual connections are employed around each of the sub-layers, followed by layer normalization. The self-attention sublayer is also modified in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i 2.

6.1.3 Attention Mechanism

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

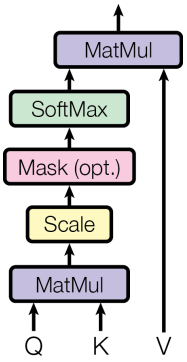


Figure 6.3: Scaled Dot-Product Attention 2

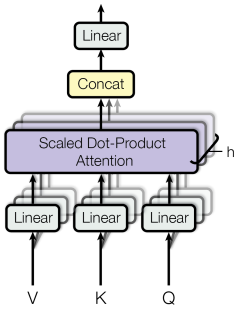


Figure 6.4: Masked Multi-head Attention 2

The input consists of queries and keys of dimension d_k, and values of dimension d_v. The dot products of the query with all keys is computed then each one divided by $\sqrt{d_k}$. Finally a *softmax* function is applied to obtain the weights on the values. In practice, the attention function on a set of queries is computed simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V see figures 6.2 and 6.3. The matrix of outputs is computed as 2:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{(QK^T)}{\sqrt{d_k}})V \tag{6.1}$$

6.1.4 Sudoku Solvability Checker

Since inputs can be noisy, constrains are applied during training to avoid garbage values from the Sudoku solver. Some of the basic conditions to solve Sudoku are hardcoded into this module. One of the simplest constrain is a puzzle should have at the minimum 17 clues and there is a solvable puzzle with at most 21 clues for every solved grid. rows, columns or boxes with repeated values is an invalid input. Digits with positions beyond the 9×9 gird automatically disqualifies a Sudoku.

6.1.5 Solve Sudoku

Once a correct set of inputs are received from the Transformer, this module works to solve them as effectively as possible. As stated in the beginning of 6, there are multiple algorithms available to solve Sudoku. Theoretically, this section tries to implement constrain programming and stochastic search approach or *algorithm* to arrive at a solutions. But in practice, the code will be more convoluted and inspired by 26. Instead strictly following a single algorithm, it could be a mixture so as to get a optimal solution. The figure 6.5 displays the final proposed architecture of this mini dissertation.

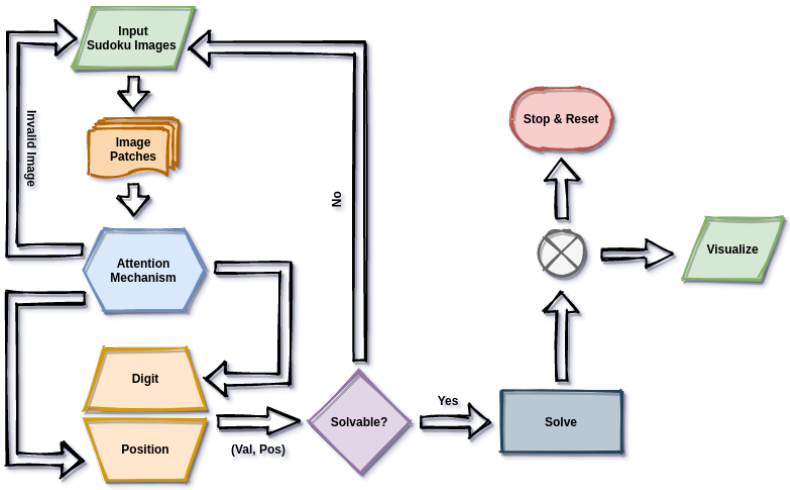


Figure 6.5: Vudoku: Final Architecture

6.1.6 Visualize Solution

Solving Sudoku is one thing and visualizing it is another. Software libraries like matplotlib 27 and engines like manim 28 can be used to showcase how the Sudoku is being solved step by step before stopping and resetting the model state as shown in figure 6.5. This part of the project is educational as it gives a clearer picture of the process that takes places within the model. Note visualization is not quite easy as it sounds and hence this module will be considered as peripheral bonus.

6.2 Project Requirements

This section spotlights paraphernalia required to create Vudoku. The followed tables provides a compact list of necessary items.

Hardware	Software
CPU: 2+ Cores, 2.46+ GHz	**Code Editor:** VSCode/OSS
RAM: 6 - 8 GB	**Language(s):** Python/Nim
Disk: 5+ GB	**Libraries:** OpenCV, Tensorflow etc.
Capture Device: Camera or Scanner	**Dataset:** MNIST

Some of the components mentioned are expensive, so those who desire to test the code, but hardware is not feasible, alternatives exist. Google provides and online platform called Colaboratory (Colabs for short) which is basically Jupyter Notebooks on a remote system. Another one Kaggle, which was later acquired by Google, is also a similar but far more powerful platform, commonly for data science and related tasks. High performance Tensor Processing Units (TPUs) are available to tinker with. Both of the platforms provide free access to massive computational resources.

For exhaustive development a version control system is inevitable. This project will be versioned using *Git* and the code is hosted on a GitHub repository at `https://github.com/joe733/vudoku`. The source code is distributed under MIT licence.

Chapter 7

Conclusion

Bibliography

[1] Yellapragada SS Bharadwaj, Rajaram P, Sriram V.P, Sudhakar S, Kolla Bhanu Prakash. Effective handwritten digit recognition using deep convolution neural network. *ijatcse*, 9(2), April 2020.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv e-prints*, page arXiv:1706.03762, June 2017.

[3] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image Transformer. *arXiv e-prints*, page arXiv:1802.05751, February 2018.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv e-prints*, page arXiv:2010.11929, October 2020.

[5] Wikipedia contributors. Exact cover — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Exact_cover#Sudoku, 2020. [Online; accessed 5-December-2020].

[6] Wikipedia contributors. Np-completeness — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=NP-completeness&oldid=991332663, 2020. [Online; accessed 6-December-2020].

[7] Wikipedia contributors. Combinatorics — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Combinatorics&oldid=991295523, 2020. [Online; accessed 6-December-2020].

[8] Wikipedia contributors. Group theory — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Group_theory&oldid=988930238, 2020. [Online; accessed 6-December-2020].

[9] Wikipedia contributors. Group (mathematics) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Group_(mathematics)&oldid=992525951, 2020. [Online; accessed 6-December-2020].

[10] Wikipedia contributors. Gradient descent — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Gradient_descent&oldid=992165304, 2020. [Online; accessed 6-December-2020].

[11] Wikipedia contributors. Fourier transform — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Fourier_transform&oldid=992485222, 2020. [Online; accessed 6-December-2020].

[12] Wikipedia contributors. Activation function — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Activation_function&oldid=991291159, 2020. [Online; accessed 6-December-2020].

[13] Wikipedia contributors. Recurrent neural network — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Recurrent_neural_network&oldid=991832590, 2020. [Online; accessed 5-December-2020].

[14] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009.

[15] Wikipedia contributors. Long short-term memory — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Long_short-term_memory&oldid=991684445, 2020. [Online; accessed 5-December-2020].

[16] Wikipedia contributors. Convolutional neural network — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=992073762, 2020. [Online; accessed 5-December-2020].

[17] Wikipedia contributors. Transformer (machine learning model) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Transformer_(machine_learning_model)&oldid=992450951, 2020. [Online; accessed 5-December-2020].

[18] Wikipedia contributors. Artificial neural network — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Artificial_neural_network&oldid=992177465, 2020. [Online; accessed 5-December-2020].

[19] Grant Sanderson. But what is a neural network? | deep learning, chapter 1, 2017.

[20] Anmol Adhikari. Hand-written digit recognition using cnn classification(process explanation). *Medium - Analytics Vidhya*, August 2020.

[21] Wikipedia contributors. Tensor — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Tensor&oldid=991219127, 2020. [Online; accessed 6-December-2020].

[22] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[23] Wikipedia contributors. Sudoku solving algorithms — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Sudoku_solving_algorithms&oldid=984975619, 2020. [Online; accessed 6-December-2020].

[24] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the Limits of Language Modeling. *arXiv e-prints*, page arXiv:1602.02410, February 2016.

[25] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv e-prints*, page arXiv:1607.06450, July 2016.

[26] Ali Shazly. Sudoku-py. https://github.com/AliShazly/sudoku-py, 2019.

[27] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[28] Grant Sanderson. manim. https://github.com/3b1b/manim, 2018.