Visual Sudoku Solver

Martina Kotseva

4th Year Project Report
Artificial Intelligence
School of Informatics
University of Edinburgh

2018

Abstract

In this report, we will discuss the work done for designing and implementing a Visual Sudoku Solver app for iOS. The app's capabilities include recognizing a sudoku puzzle on the phone's camera and overlaying the image with the correct solution. This process involves vision and machine learning which are both substantial parts of nowadays artificial intelligence. The report presents the chosen methods for implementation which is followed by critical evaluation along with suggestions for future improvements.

Acknowledgements

I would like to show my sincere appreciation to my supervisor, Professor Nigel Topham, for his help and guidance throughout the project.

I would also like to thank my family for the support and motivation.

Table of Contents

Chapter 1

Introduction

1.1 The Sudoku Puzzle

Sudoku is a logic-based combinatorial puzzle. When introduced in Japan, it receives its current name Sudoku, meaning ”single number” in Japanese. It is now popular all over the world. The game consists of a 9x9 grid divided into 9 square sub-grids (of size 3x3) which we shall refer to as blocks in this paper. Some of the squares in the grid contain a number from 1 to 9. The player is presented with a partially filled grid and their aim is to fill the rest of the squares with numbers from 1 to 9. The rules are fairly simple: each row, column and block must contain each of these numbers exactly once.

The capabilities of the app include being able to detect and recognise such a puzzle by getting feed from the rear camera of an iPhone. It should be able to solve the game in real time and fill the empty squares with the correct digits, displaying them over the feed from the camera on the screen in the corresponding positions.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 1.1: A sudoku puzzle

1.2 The Pipeline

The code for the app is written in Swift. In this report, we follow the pipeline of the whole program from getting the feed from the camera to displaying the solution on the screen. The process consists of:

- Getting feed from the device's camera.
- Image manipulation.
- Detecting the sudoku grid.
- Dividing the grid into separate squares.
- Locating and extracting the digits.
- Digit recognition.
- Solving the puzzle.
- Projecting the solution on the screen.

The algorithms implemented for each of the steps are described in a separate chapter. We also discuss alternative ways for solving the problems faced, explaining their pros and cons and justifying our decisions. At the end of this report, we have provided the results of a survey we carried to evaluate the app and we have suggested ideas for future improvements.

Chapter 2

Background

2.1 Image Transformations

A substantial part of computer vision and computer graphics concerns with image processing or, in particular, image transformations. There are different kinds of transformations in geometry.

An affine transformation is a transformation that preserves points, straight lines, planes, parallelism and ratios of distances between points lying on a straight line, while it does not necessarily preserve angles between lines or distances between points.

When an image is shot, different objects in it have different perspective distortion depending on their location in the scene relative to the position of the camera. This happens when objects from the real world space which have 3D coordinates are projected onto the 2D image space. Parallel projections only preserve straight lines. Parallel lines in objects, then, do not necessarily appear parallel in the images. In order to convert between the two spaces, we need to transform (warp) the images. In other words, we needed to find a suitable mapping between points from one plane to the other and apply it to the picture to obtain an image with the desired object properties, essentially looking at the scene from another point of view. Such mapping is called a homography. It has 8 degrees of freedom. To estimate homography, we need at least 4 points of reference.

A way to do this is via the direct linear method. The 3x3 homography matrix can be calculated using homogeneous coordinates to put the object in 3D space and introducing a virtual camera with a new image space to project onto. The projection matrix is $P_{13} = P_{23}(P_{21})^{-1}$ where P_{xy} is $\pi_x \rightarrow \pi_y$ (as can be seen in figure 2.1) [1]. The new coordinates that we get by multiplying the initial homogeneous coordinates by the estimated matrix are in 3D. To convert them back to 2D we simply need to divide the first two coordinates by the third one, eliminating the scaling factor. [1]

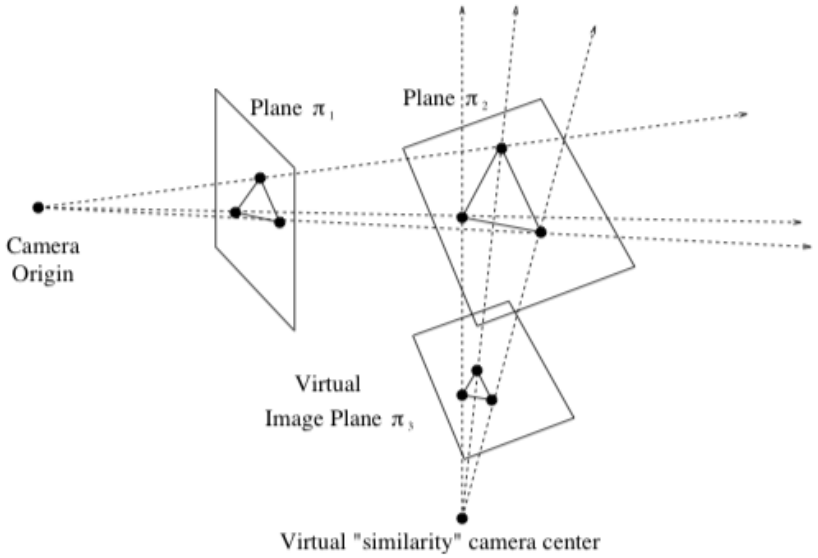


Figure 2.1: Plane projections [1]

Other approaches for removing perspective distortion require user input - the horizon line or a pair of lines in each plane that are parallel in the real world. [2]

Etienne Vincent and Robert Laganibre propose an algorithm for planar homography detection in pairs of images. It is based on the linear computation of the homography matrix. The RANSAC scheme uses four corresponding points in the two images to compute the matrix and then finds pairs that agree with the homography according to a threshold. This process is repeated until the number of such pairs is sufficiently big. The matrix is then recomputed using all consistent correspondancies. [3]

2.2 Object Recognition

In order to recognise a set of objects, we must identify features that describe them and differentiate the possible categories from one another. The next sections explain different descriptors that can be used for these purposes. [4]

2.2.1 Global Descriptors

Global descriptors such as convexity, compactness and elongation are simple and fast to compute. They are invariant to translation, rotation and scale but have little discriminative power. They are robust to shape deformations which is generally seen as an advantage, but for digit recognition, it might have a deleterious effect on recognising some objects - such as pairs of digit categories that have little difference in their overall shape.

- Convexity is the ratio of the objects's convex hull and its perimeter.

$$convexity = \frac{P_{hull}}{P_{shape}}$$

- Compactness is the ratio of the perimeter of the circle which area is equal to the one of the shape and the perimeter of the shape.

$$compactness = \frac{2\sqrt{A\pi}}{P}$$

- Elongation is the ratio of principle axis. Principle axis cross at the centroid. Their lengths are equal to the eigenvalues of the covariance matrix.

2.2.2 Shape Signatures

Shape signatures use the object's boundary points to represent its shape by a 1D function. To obtain such function we can use the distance from the centroid, curvature at the corresponding point, area, cumulative angles. The similarity between two shapes can be calculated by integrating the difference of the functions over the boundary points. Shape context divides the area around each point into sections and the count of points that fall into each section is used to derive the function. We can find point to point correspondences by solving for least cost assignment, using costs (e.g. by the Hungarian algorithm).

Shape signatures are quite informative and can be made invariant to scale, translation and rotation, they also handle some shape deformations, but are computationally rather expensive.

2.3 Machine Learning

Machine learning is a field in computer science, more specifically an application of artificial intelligence, concerned with building systems which are able to learn, that is, gradually improve their performance when presented with big amounts of labelled data without being explicitly programmed. The history of AI can be traced back to 1950, when Alan Turing creates the Turing Test, until the rise of machine learning nowadays. [5]

2.3.1 Feature Engineering

The essence of machine learning lies in performing metadata analysis. The very first step in doing machine learning is feature engineering. We can't use the raw information, which is, in our case images, as an input to the classifier. We need to process it first and extract the appropriate features from it. We use the combined set of features which is called a feature vector as the input. To identify objects in images, we can compile a vector from some of the object descriptors explained in the previous section. However, since we need to recognise digits, because of the properties of their shape, there is a more appropriate approach. That is, removing colour from the images and scaling them so that they all are of equal size and the number of pixels in each is small enough to make processing possible. The values of the pixels, then, form a feature vector.

2.3.2 Classification vs Regression

Machine learning tasks can be divided into two categories - classification and regression. In classification problems, the system is trying to predict a discrete class label. In regression problems, the output takes continuous values. There are various different methods for machine learning depending on the problem. In the app we are building, we are dealing with a classification task - the output values we are trying to predict are the digits. The method used in this project is a convolutional neural network compiled by Sri Raghu Malireddi (license: MIT).

2.3.3 Neural Networks

Neural networks are inspired by the way neurons work in the human brain. We take all the values of the input, multiply each by a corresponding weight, sum over those products and add a constant, called a bias. The bias acts as a threshold for separating the inputs into two groups - the ones that fall into a certain category, and those that don't. This combination is called a perceptron. Since it is a linear combination of the values, the boundary it produces is a straight line. Not all data is linearly separable, though. To cope with that, multilayer neural networks have multiple hidden layers between the input layer and the output layer. Each input from the input layer is fed to

the next layer until the output layer is reached. Convolutional neural networks are a type of deep neural networks designed specially to work on image input. The models of the objects that they learn are position invariant, in other words, they can detect features categorising a certain type of object, regardless of which part of the image this feature is located in. [6]

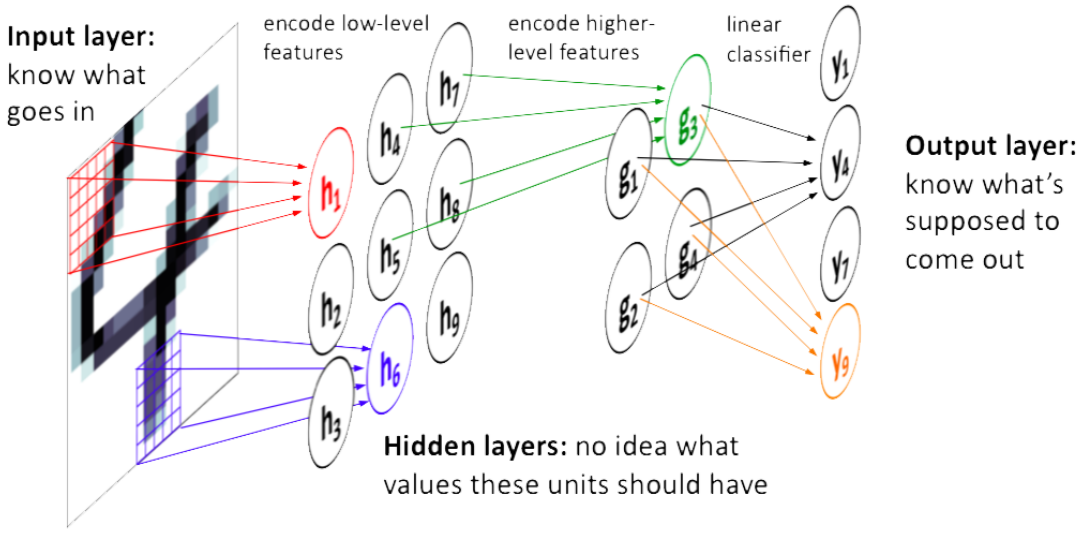


Figure 2.2: Using CNN for digit recognition [7]

2.4 Solving Sudoku Puzzles

2.4.1 Difficulty of the Puzzle

Contrary to popular belief, the difficulty of a sudoku puzzle is not related to the number of clues it has. The level of difficulty is defined by the number of steps required to solve it, as well as the complexity of the techniques needed for a person or an algorithm to solve the puzzle. A bottleneck is a situation in which there is only a single move which will make it possible to continue solving. Thus, the more bottlenecks in a puzzle, the harder it becomes, even if the required solving techniques are easy, as many squares have to be visited in order to find that specific move. [8]

Maria Ercsey-Ravasz at Babes-Bolyai University in Romania and Zoltan Toroczkai at the University of Notre Dame in Indiana have developed a Richter scale - a way to quantify the difficulty of a particular sudoku puzzle by using algorithmic complexity theory. The scale ranges from 1 to 4, with 1 being the easiest and 4 being the most difficult. A sudoku puzzle called platinum blond has a difficulty of 3.5789 and this is the hardest one according to the scale that has been found. [9] [10]

3	4	6	7	5	8	9	1	2
1	5	7	4	9	2	6	8	3
8	9	2	3	6	1	4	7	5
5	3	1	8	2	4	7	9	6
2	6	9	1	7	5	8	3	4
7	8	4	6	3	9	2	5	1
6	2	3	5	8	7	1	4	9
9	1	8	2	4	3	5	6	7
4	7	5	9	1	6	3	2	8

Figure 2.3: Blonde Platinum sudoku

2.4.2 Human Solving Techniques

In this section we turn our attention to some techniques that humans use for solving sudoku, starting from the basic ones and moving on to the more advanced. [11] [12] Apart from using them to determine a puzzle's difficulty, human strategies can also be implemented in computer algorithms. [13]

2.4.2.1 Open Singles

An open single is a square that is the only number missing from a row, a column or a block.

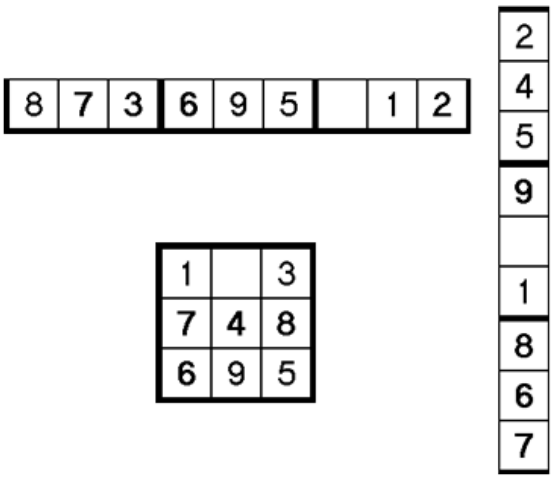


Figure 2.4: Open singles [11]

2.4.2.2 Searching for the Position of Each Number in Each Block

Since we know that all blocks have to contain every number, we can focus on a (number, block) combination. Then we look at the rows and columns intersecting with that block and use them to eliminate possible positions for that number in the block until we are left with one.

2.4.2.3 Searching for the Number for a Certain Square

Since no row, column or block may contain a number twice, we can focus on a square and look at the row, column and block it belongs to. We use the values in them to eliminate possible numbers for that square until we are left with one.

2.4.2.4 Pencil Marks and Lone Cells

Pencil marks are small numbers people put in squares to remember the possible values for that square (cell). A Lone Single is when a cell has only one pencil mark left. With solution progress, they can be eliminated and erased. The more advanced techniques that follow, mostly eliminate candidate numbers (pencil marks) rather than solving a cell.

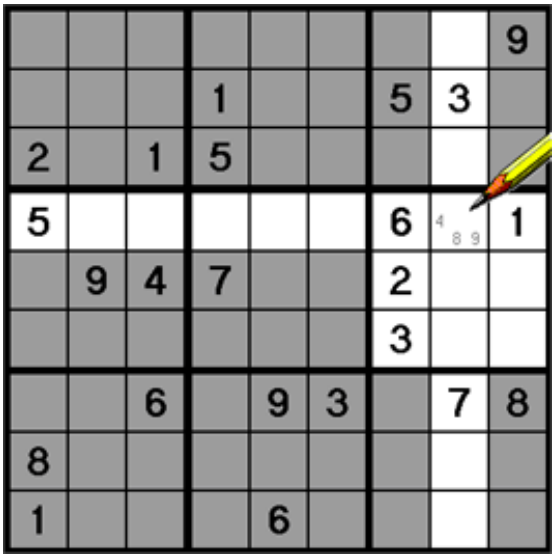


Figure 2.5: Pencil marks [11]

2.4.2.5 Hidden Singles

A hidden single is when one of the possible values for a cell is only possible for that cell in the row, column or block.

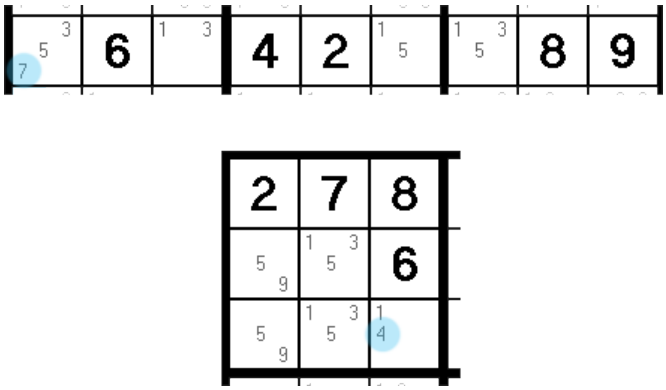
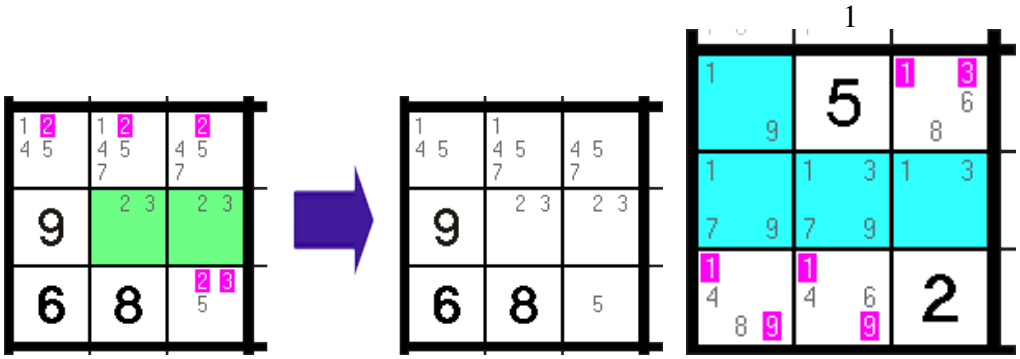


Figure 2.6: Hidden singles [11]

2.4.2.6 Naked Pairs, Triplets, Quadruples

A naked tuple is a group of N cells that belong to the same row, column or block, such that the number of unique possible values for all N cells is N. If a naked tuple exists, we can remove those values from all other cells in that row/column/block.



(a) Naked pair (b) Naked quadruple

Figure 2.7: Naked pairs, triplets, quadruples [11]

2.4.2.7 Hidden Pairs, Triplets, Quadruples

A hidden tuple is a group of N cells that belong to the same row, column or block, and containing a set of N possible values. A hidden tuple is, in essence, a naked tuple that might also have other values in its cells, making it harder to spot. If a hidden tuple exists, we can remove the values that are part of it from all other cells in that row/column/block.

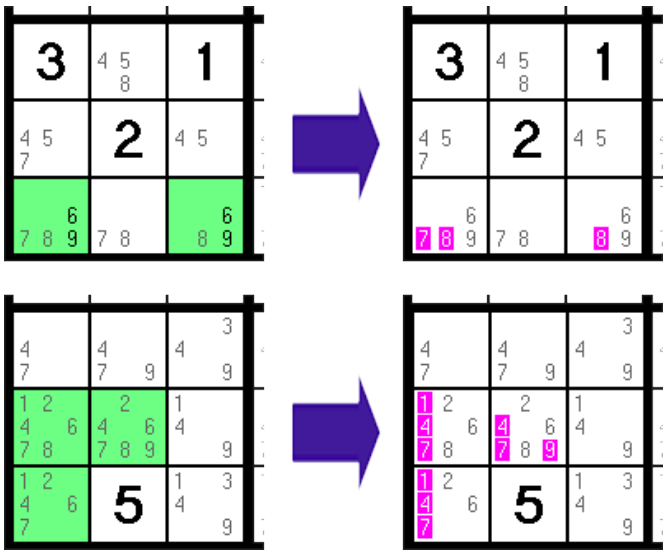


Figure 2.8: Hidden tuples [11]

2.4.2.8 Omission

When pencil marks of one kind in a row or column are contained inside a single block, pencil marks of that kind elsewhere in the block can be removed. Likewise, when pencil marks of one kind in a block are contained in a single row or column, we can remove the pencil marks of that kind elsewhere in the row/column.

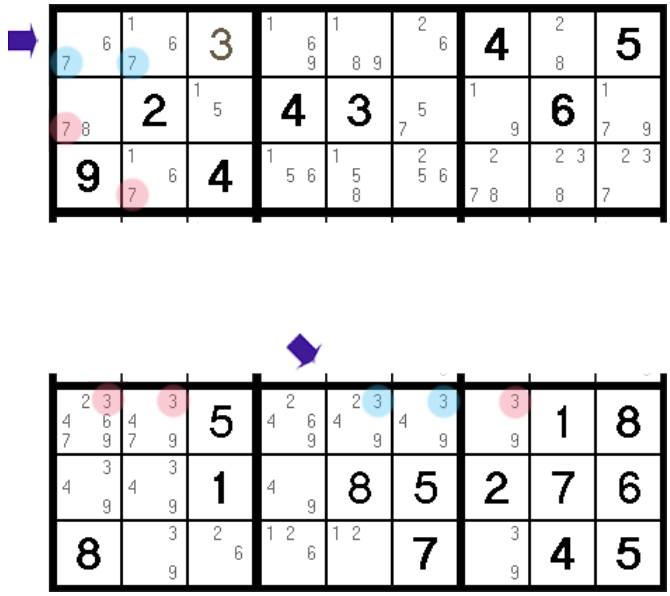


Figure 2.9: Omission [11]

2.4.2.9 X Wing

It is possible to find a pair of rows that have the same pencil mark exactly twice in cells with the same indices (forming a rectangle). Knowing that each of the columns connecting the cells must contain that value we can deduce that one of the pairs of values in the diagonals of the rectangle will be in the solution and therefore in both of the columns. Since the columns cannot contain a value more than once, we can erase that pencil mark from the rest of the cells in the columns. Analogically, if we find a pair of columns that have the same pencil mark exactly twice in cells with the same indices, we can remove the pencil mark from the rest of the cells in the rows intersecting these positions.

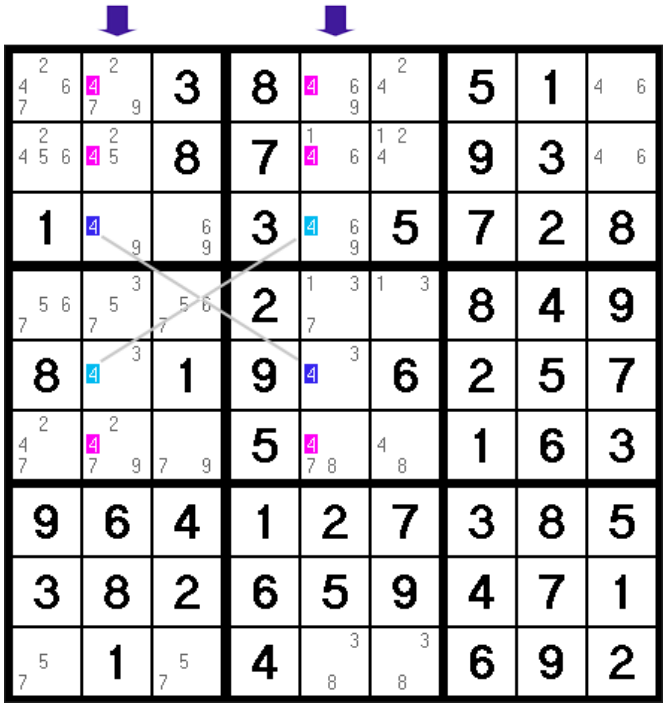


Figure 2.10: X wing [11]

2.4.2.10 Swordfish

Swordfish is a variation of X Wing but instead of two, it involves three pencil marks in three columns or rows.

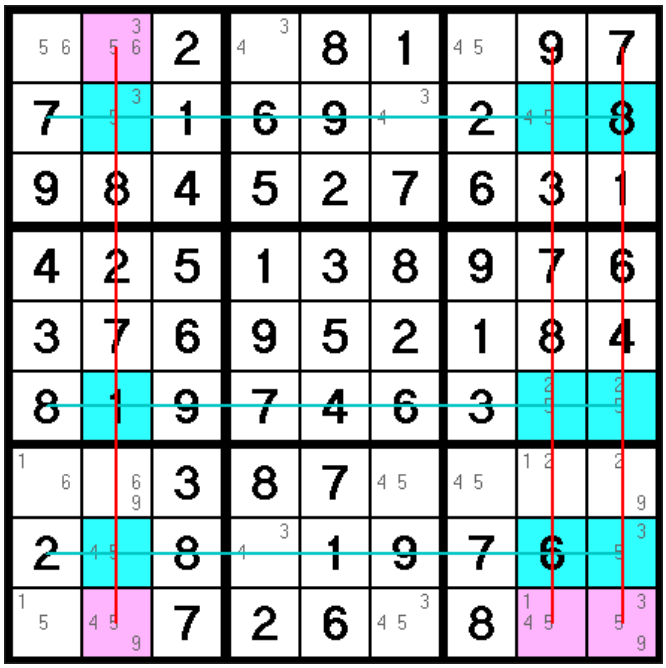


Figure 2.11: Swordfish [11]

2.4.3 Solving Sudoku Using Graph Colouring

A possible approach for solving sudoku puzzles is by using graph theory. Vertex graph colouring for a graph G with a set of vertices $V(G)$ and a set of edges $E(G)$ is a projection $c : V(G) \rightarrow S,\ S \subseteq N$ such that if $u \in V(G),\ v \in V(G),\ \{u,v\} \in E(G) \Rightarrow c(u) \neq c(v)$ where S is a set of colours.

The chromatic number $\chi(G)$ of a graph is the minimum number of colours necessary to colour it. Brook's theorem states that $\chi(G) \leq$ the highest vertex degree in the graph + 1. An optimal colouring of a graph G is one with $\chi(G)$ colours.

We can represent sudoku puzzles as graphs with the cells of the game at the graph's vertices. Two vertices are then connected by an edge if they belong to the same row, column or block. Hence, edges represent the constraint that the two cells cannot contain the same digit. Our aim is to find a colouring with 9 colours (corresponding to the numbers 1-9). The number of vertices in the graph is 81 and each of them belongs to the same group (row, column or block) with exactly 20 distinct other vertices and so has a degree of 20. This makes the total number of edges in the graph 810 ($\frac{81\times20}{2} = 810$). [14]

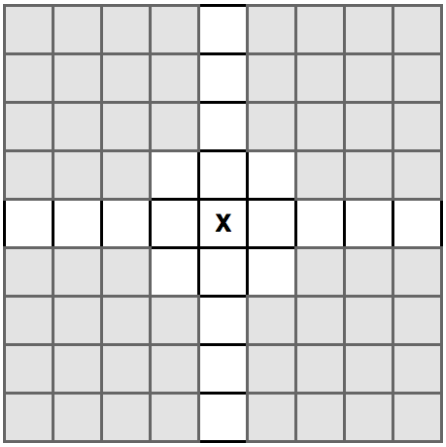


Figure 2.12: Squares that cannot have the same value as X.

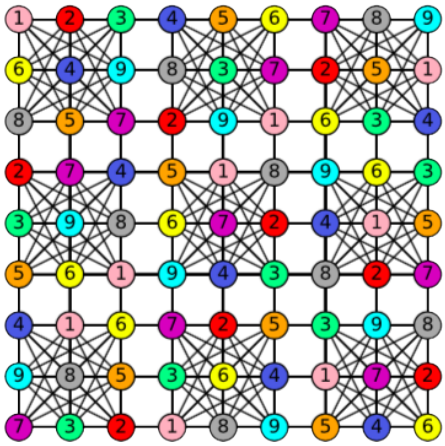


Figure 2.13: A sudoku coloured graph.

2.4.3.1 Greedy Algorithm

Greedy algorithms for graph colouring focus on the choice of a vertex to be coloured next. A coloured vertex's colour is never changed. A famous algorithm under this category is the WelshPowell algorithm. It works as follows:

1. Find the degree of each vertex.
2. Order the vertices in descending order according to their degree.
3. Go through the list, colour each vertex not connected to coloured vertices with the same colour.
4. Remove coloured vertices from the list and repeat the process until all vertices are coloured.

However, greedy algorithms are not guaranteed to find a colouring that uses only 9 colours and so the colouring is not always a valid solution to the puzzle. [15]

2.4.3.2 Contraction

A contraction algorithm works by finding two vertices that can be assigned the same colour and combining them in a single vertex which is connected to all of their edges:

1. Select the vertex of maximum degree V.
2. Find the set of non-adjacent vertices to V.
3. From this set select the vertex Y of maximum common vertices with V.
4. Contract Y into V to be coloured with the same colour.
5. Remove Y from the set and repeat steps 3-5 until the list is empty.
6. Remove vertex V from the graph.
7. Repeat steps 1-6 until the resulting graph has all contracted nodes adjacent to each other.

[15]

2.4.4 Sudoku as a Constraint Satisfaction Problem

Sudoku puzzles are a part of a set of mathematical problems called constraint satisfaction problems in which the aim is to find an assignment to variables that satisfies a set of constraints over them. In this section, we discuss approaches for constraint satisfaction problems. [16]

2.4.4.1 Backtracking Heuristics

One such approach is using brute force and trying all possible combinations via backtracking (depth-first search). This solution, however, is rather slow, but it can be optimised with some heuristics that specify which variable should be assigned a value next. Heuristics are applied in the order described below. Heuristics that are further down in the list are used as tiebreakers.

- Most constrained variable a.k.a. minimum-remaining-values (MRV) heuristic: choose the variable with the fewest legal values.
- Most constraining variable a.k.a. degree heuristic: choose the variable with the most constraints on remaining variables thus reducing branching.
- Least constraining value: given a variable, choose the least constraining value - the one that rules out the fewest values in the remaining variables.

2.4.4.2 Arc Consistency AC-3

The AC-3 algorithm (Arc Consistency Algorithm #3) is one of a series of algorithms for constraint satisfaction problems. It was developed by Alan Mackworth in 1977.

AC-3 works on a graph where nodes are variables with their corresponding domains and edges (arcs) represent constraints. It propagates constraints so that each arc ends up being consistent. An arc $X \rightarrow Y$ is consistent if and only if for every value x of in the domain of X there is some allowed y in the domain of Y. If X loses a value, all neighbours of X need to be rechecked. The algorithm can be run as a preprocessor or after each assignment of values.

```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
while queue is not empty do
    X_i, X_j ← REMOVE-FIRST(queue)
    if REVISE(csp, X_i, X_j) then
        if size of D_i = 0 then
            return false
        end
        foreach X_k in NEIGHBOURS(X_i) - X_j do
            add (X_k, X_i) to queue
        end
    end
end
return true
end function
```

```
function REVISE(csp, X_i, X_j) returns true iff the domain of X_i needs to be revised
revised ← false
foreach x in D_i do
        if no value y in D_j allows (x, y) satisfying the constraint (X_i, X_j) then
            delete x from D_i
            revised ← true
    end
end
Return revised
```

2.4.5 Knuth's Algorithm X and Dancing Links

We can represent every sudoku in the form of a matrix as follows:

- Rows 1-729 indicate whether the chosen assignment of values includes each number in each position.

Columns represent the rules of the puzzle as constraints:

- Constraints 1-81: We must place some number in each cell.
- Constraints 82-162: We must place each number somewhere in each row.
- Constraints 163-243: We must place each number somewhere in each column.
- Constraints 244-324: We must place each number somewhere in each block.

Thus, a cell in this matrix contains 1 if the $(value,\ position)$ pair from the corresponding row satisfies the constraint in the corresponding column. The constraint matrix contains 236196 cells in total, but any candidate solution only fills 2916 of them with 1's.

Given a matrix of 0's and 1's (such as the one described above) the exact cover problem concerns with finding an assignment of values such that there is exactly one 1 in each column. In order for a solution of a sudoku puzzle to be correct, this is precisely what the case must be.

Assignment	Constraint																														
	There is a number in row									The number									The number									The number								
	1			...			9			1			...			9			1			...			9			1			...			9		
	and column									must appear in row									must appear in column									must appear in block								
	1	...	9	...	...	...	1	...	9	1	...	9	...	...	...	1	...	9	1	...	9	...	...	...	1	...	9	1	...	9	...	...	...	1	...	9
1 in (1,1)																																				
2 in (1,1)																																				
...																																				
9 in (1,1)																																				
...																																				
1 in (9,9)																																				
...																																				
9 in (9,9)																																				

Figure 2.14: Constraint matrix [17]

2.4.5.1 Algorithm X

Donald Knuth has created an algorithm that finds a solution to the exact cover problem defined by such matrix. He decides to call it algorithm X "for lack of a better name" [18] as he states in his paper. Algorithm X works in the following way:

1. If there are no unsatisfied constraints remaining, the solution set is complete. Otherwise, pick an unsatisfied constraint (a column that does not contain a 1 in

any of the rows in the solution set).

2. Pick a row that satisfies that constraint (one that has 1 at its intersection with the chosen column). If no such row exists, the solution cannot be continued. Backtrack to the previous time a row was chosen and select the next one instead.

3. Add that row to the solution set.

4. Delete all rows that satisfy any of the constraints satisfied by the chosen row: that is, all rows that have 1 in the same column as any of the cells containing 1 in the chosen row.

5. Return to step 1.

[19]

2.4.5.2 Dancing Links

Dancing links is the technique suggested by Donald Knuth to efficiently implement his Algorithm X. The idea for this technique is based on the way doubly linked lists work. A doubly linked list is a data structure consisting of a sequence of nodes, where each node has a reference (link) to the one preceding it and the one following it. Doubly linked lists allow for faster insertion and deletion of elements.

- Element deletion:

$$L[R[x]] \leftarrow L[x]$$

$$R[L[x]] \leftarrow R[x]$$

- Element insertion:

$$L[R[x]] \leftarrow x$$

$$R[L[x]] \leftarrow x$$

The constraint matrix can be represented by the cells that contain 1's. Each of them is a data object x with five fields $L[x]$, $R[x]$, $U[x]$, $D[x]$, $C[x]$ - for the values to the left, right, up and down and the column header. Rows and columns in the matrix are doubly linked as circular lists. Knuth refers to removing and inserting columns as "covering" and "uncovering". Using dancing links, these operations require a significantly smaller amount of time. [18]

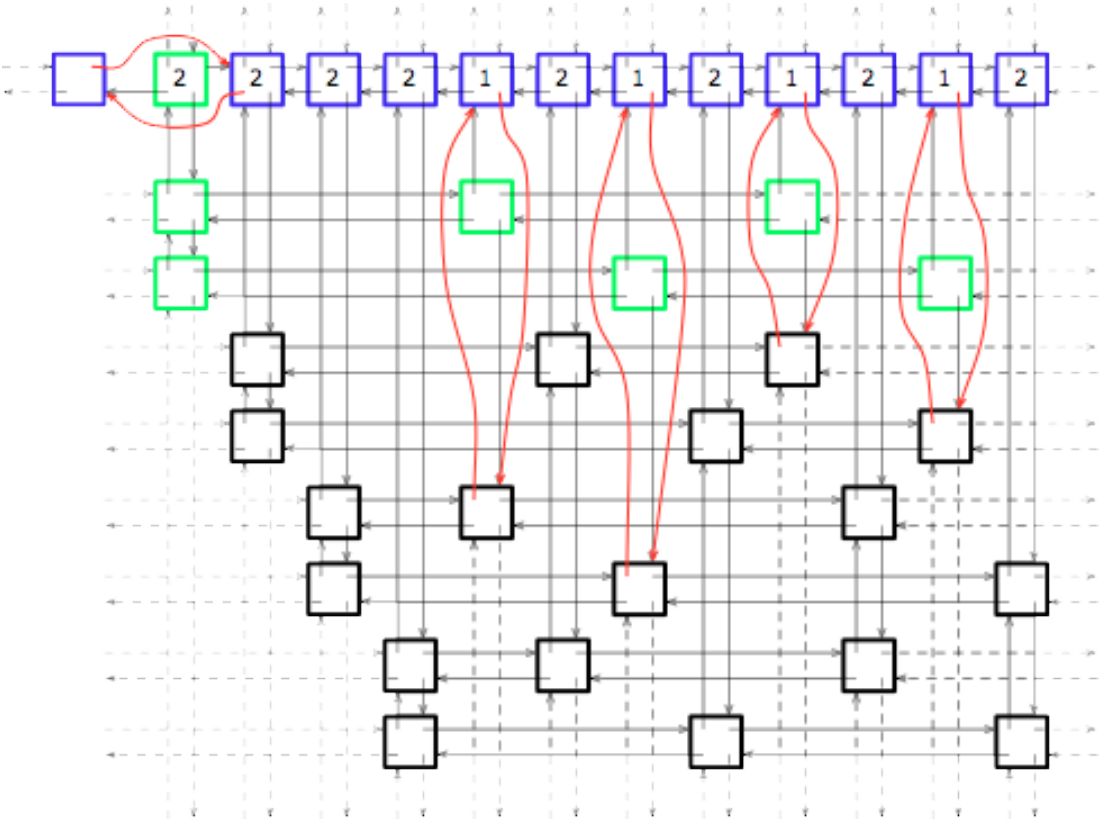


Figure 2.15: Covering a column. [19]

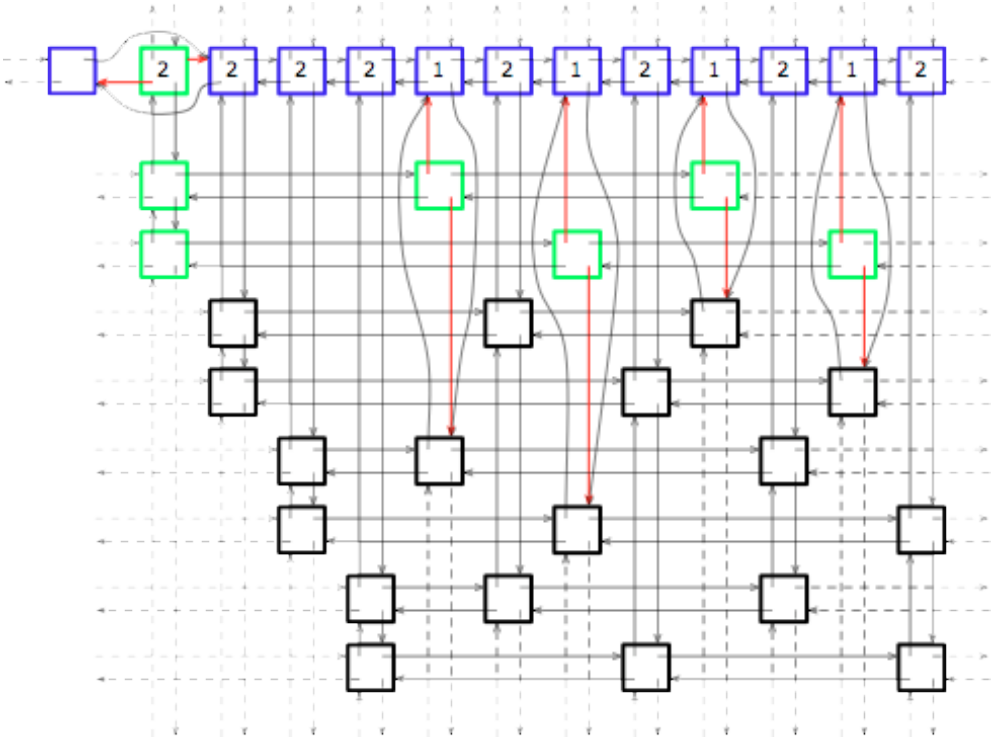


Figure 2.16: Uncovering a column. [19]

Chapter 3

Let the iPhone See

The very first step when building the app was to figure out how to control the iPhone's camera to get the visual input and output it back to the screen for the user to see. The framework that Apple provides for those purposes is called AVFoundation.

3.1 AVCaptureSession Inputs and Outputs

AVCaptureSession is an object that manages capture activity and coordinates the flow of data from input devices to capture outputs. We need to initialise one on app load, configure the capture device and add inputs and outputs. Next, we have to present the user with the feed from the iPhone's camera by displaying it on the screen. We do this simply by connecting an AVCaptureVideoPreviewLayer to our capture session. We then add this layer to the UIView that takes up the whole screen and finally start the capture session. [20]

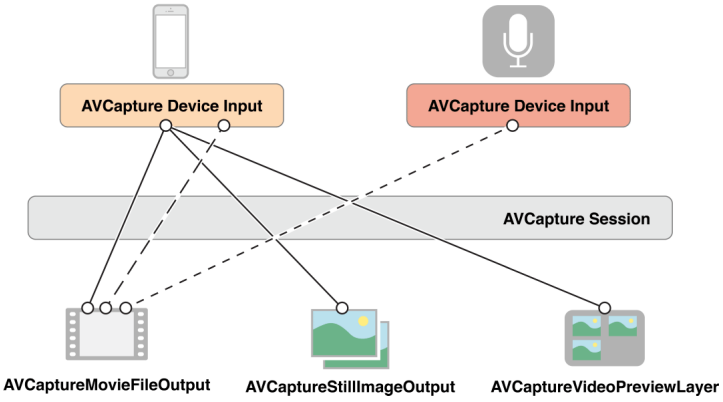
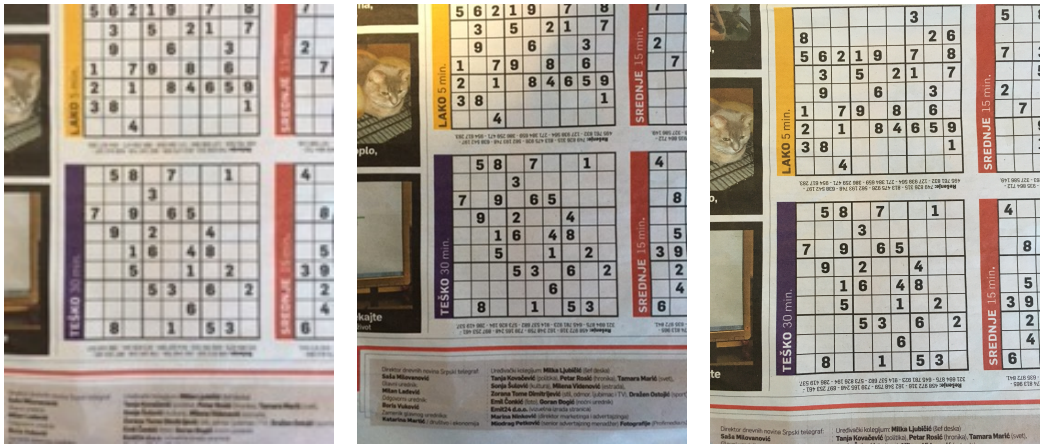


Figure 3.1: AVCaptureSession [21]

3.2 Image Quality

We use the property sessionPreset to select an appropriate quality level of the bitrate of the output. We chose the medium preset which specifies settings that produce a 480x360 pixel video output. This allows for maintaining a relatively good quality of the images that we use to process, while not letting them become too large. A big size of the image is problematic because it means more pixels, hence more processing time required. [21]



(a) Low (b) Medium (c) High

Figure 3.2: Comparison of presets

Chapter 4

Image Processing

Our vision task is to recognise the sudoku puzzle in an image. We need to manipulate our images to convert them to the correct format for recognition.

4.1 Colour Normalization

Colour normalization is widely used in computer vision for colour based object recognition. Different levels of illumination cause for the same colour to have different colour values across the image. Colour normalization copes with varying lighting and reduces shadow effects. To normalise the colours of an image, we take the three values (r,g,b) - of the red, green and blue channel and transform them to new ones (r',g',b') in the following way:

$$r' = \frac{r}{r+g+b}$$

$$g' = \frac{g}{r+g+b}$$

$$b' = \frac{b}{r+g+b}$$

Colour normalization also helps reduce space, as we now only need to store two of the colour values for each pixel. The third can be calculated by:

$$b' = 1 - r' - g'$$

However, this method proved to be completely inefficient for our task. Black and white are essentially different shades of the same colour. This means that white, black and all shades of grey between them always have the same hue and saturation - 0. The only thing that differentiates them in the HSV spectrum is their value (lightness). In terms of the RGB spectrum, the values of the three channels (r,g,b) for a white, black or grey colour are equal and normalizing them always results in $r' = \frac{1}{3}$, $g' = \frac{1}{3}$, $b' = \frac{1}{3}$. Hence, white and black become indistinguishable.

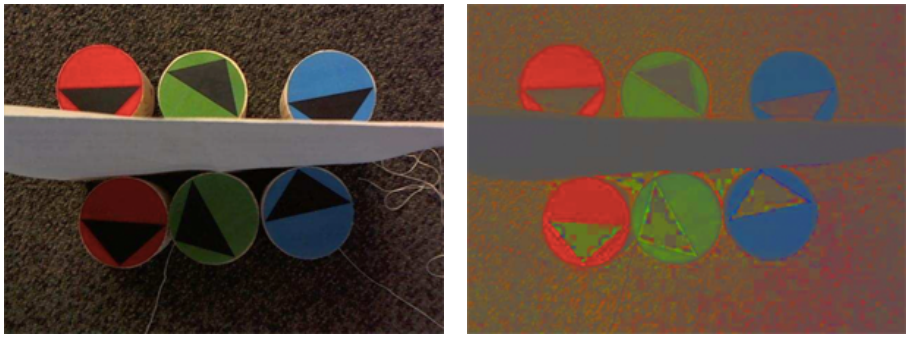


Figure 4.1: Colour normalization [22]

4.2 Greyscaling

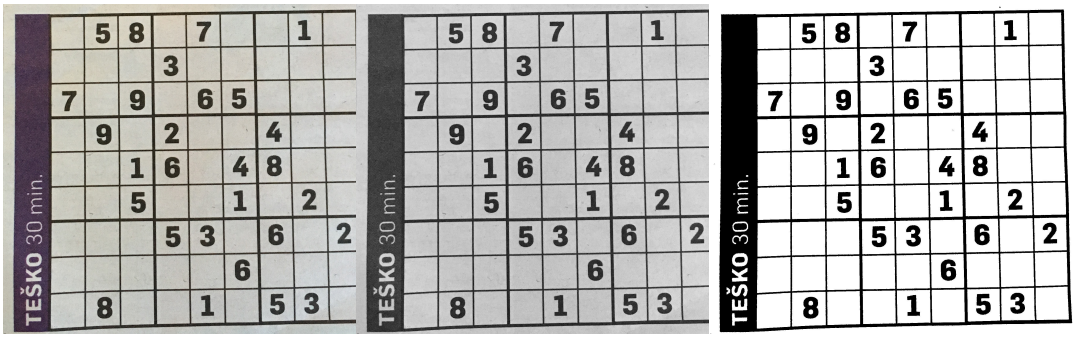
Since colour does not give us any useful information for our recognition task, we can easily discard it from our images and focus on the shapes in them by greyscaling. A greyscaled image only contains one channel representing the light intensity of the pixel, as opposed to the usual three channels. There are a number of algorithms for such conversion so and we had to find one that fits for our task well. The choice of an algorithm has a significant impact on the recognition performance. Most of the methods for greyscaling use the tree channels in RGB colour space. The simplest method just averages out the values of the three channels.

$$grey = \frac{r+g+b}{3}$$

The reason why it looks like it does not produce very good results is that it does not represent luminosity the way the human eye perceives it. Humans have three types of cones in their with different sensitivity to wavelengths. The perceived brightness is often dominated by the green component. *Luminance* uses a weighted average of the three channels to produce a representation which is close to human perception.

$$grey = 0.2989 \times r + 0.5870 \times g + 0.1140 \times b$$

This approach is used in many programs, including Matlab's *rgb2gray* function, and is popular in computer vision. We decided to use it for our project as it gives good results. There are other, more complicated approaches. In general methods that incorporate a form of gamma correction usually perform better than purely linear methods.



(a) Original (b) Greyscaled (c) Bitmap

Figure 4.2: Image processing

4.3 Bitmap

From the greyscaled image we can produce a black-and-white one. Terminology disputes exists, but we shall refer to a representation of the image, where only two colours - black and white (and no additional shades) are used, as a bitmap (a binary image). What we create is essentially a matrix where each element corresponds to a pixel in the image. The values that these elements can take are $\{0, 1\}$ (white/black). These values are determined by whether the grey value of the corresponding pixel is above or below a threshold.

Figure 4.3: Bitmap of the number 9

4.4 Thresholding

Different methods for finding a threshold exist. We briefly discuss a couple of them in the following subsections.

4.4.1 Using a Histogram

This method assumes that we are looking for a dark object on a light background. The first thing we need to do for it is to produce a histogram of the greyscaled image. The key idea is that that histogram will have all the pixel values in two main clusters - one for the dark pixels and one for the light ones. The issue is that it is not necessary that the peaks in the histogram will be only two. We can fix this by smoothing the histogram with convolution. Thus we are left with the big peaks only. The largest peak, then, must be the background. The second biggest peak in the darker direction is our object of interest. The threshold lies in the deepest valley between those peaks. [23]

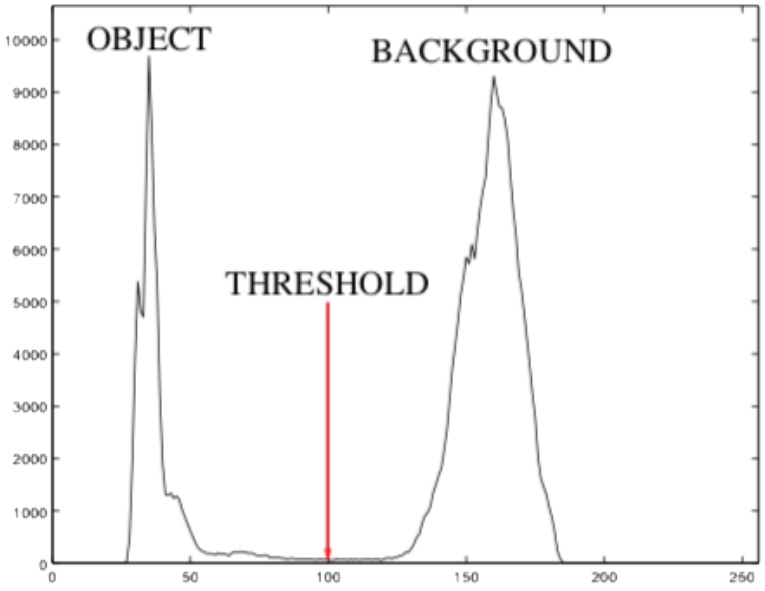


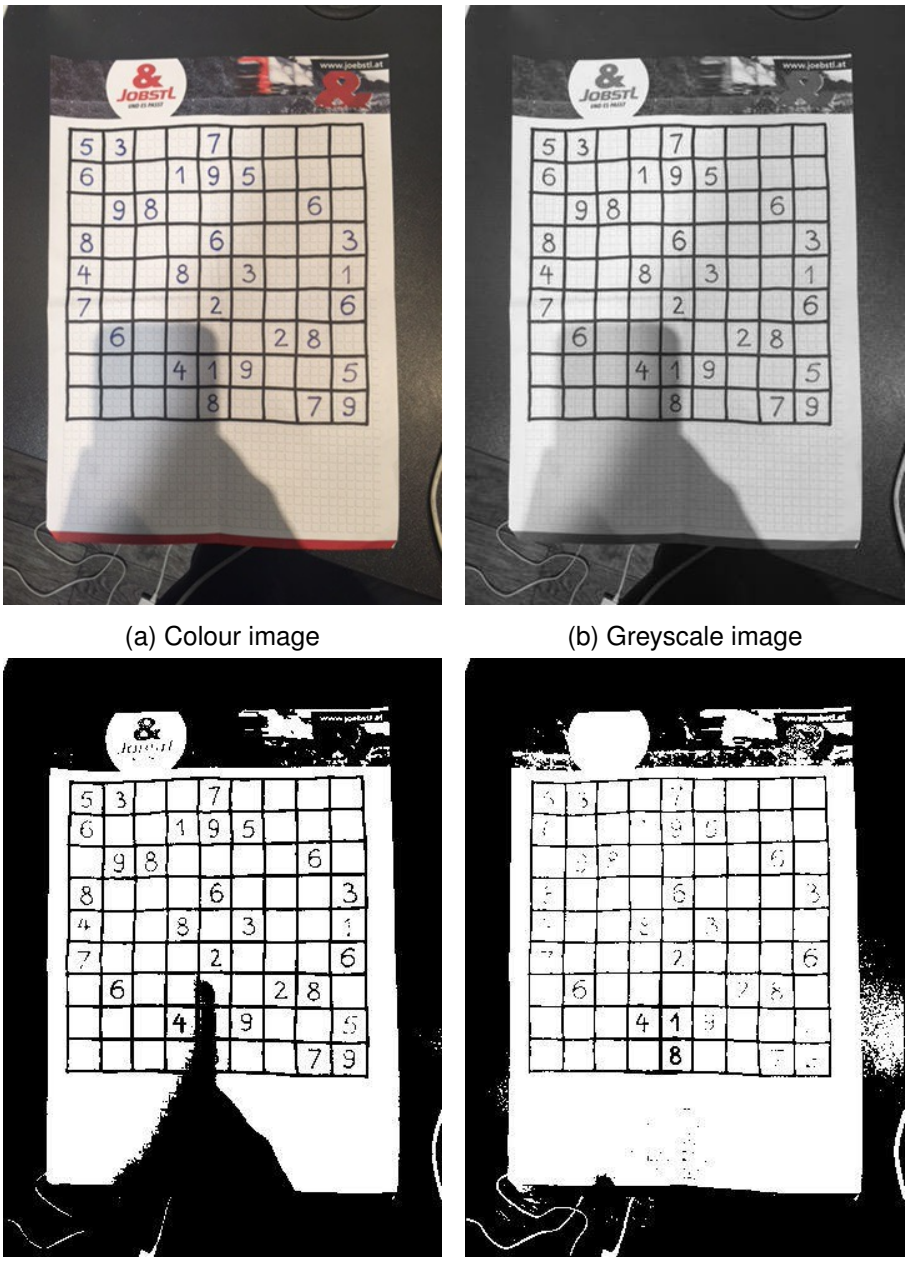
Figure 4.4: Thresholding using a histogram [23]

4.4.2 Adaptive Thresholding

The idea behind adaptive thresholding is to define a separate threshold for each pixel by looking at its neighbouring pixels. There are different ways to derive the formula for the threshold. [24], [25], [26], [27], [28], [29], [30] all propose adaptive threshholding techniques. These methods are suitable for removing shadows due to non-uniform illumination.

4.4.3 Fixed Threshold Method

Global binarization methods fix one threshold only and use it to binarise the image. The reason why we chose such method is its speed. Moreover, most of the local thresholding techniques require a constant, which is difficult to set one that will work well on all inputs, without causing the grid or digits to disappear. In our project, we calculate the threshold by averaging out the values for the brightest and the darkest pixels in the image.



(a) Colour image (b) Greyscale image

(c) Local binarization (d) Global binarization

Figure 4.5: Comparison of binarization techniques

Chapter 5

Detecting the Grid

5.1 Initial Approach

The initial approach we took towards detecting the grid was under the assumption that since the sudoku puzzle is the main object in the scene it will be the one with the largest size. We did that by using the bitmap of the image. This bitmap contains a number of connected components corresponding to different objects in the image. Two pixels are part of the same component if they are neighbouring and they are both black. Under good lighting conditions, the outline of the grid will be continuous and is thus likely that the component it comprises will have the biggest size. This approach did have several problems with it. Any gaps in the lines would lead to a component broken into two or more pieces, the grid - not recognised correctly and thus locations of the digits misplaced. The speed of the execution of this algorithm is dependent on the number of components in the whole image. In most cases, it takes several seconds to run, which is too slow for the smooth execution of the program. The process needs to be repeated at least once every second to account for very small changes in the position of the camera or the object of interest.

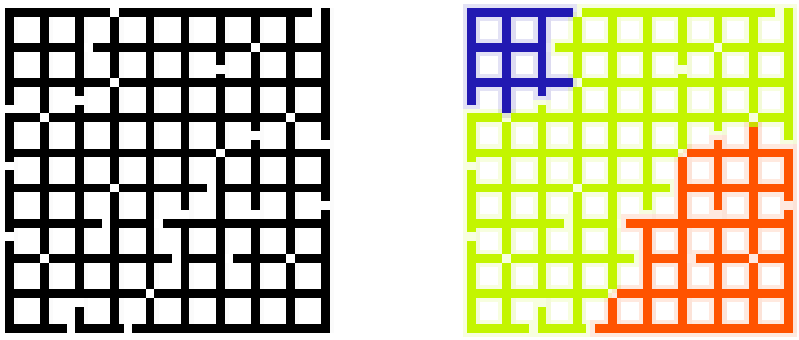


Figure 5.1: Gaps in the grid causing it to appear as three separate components

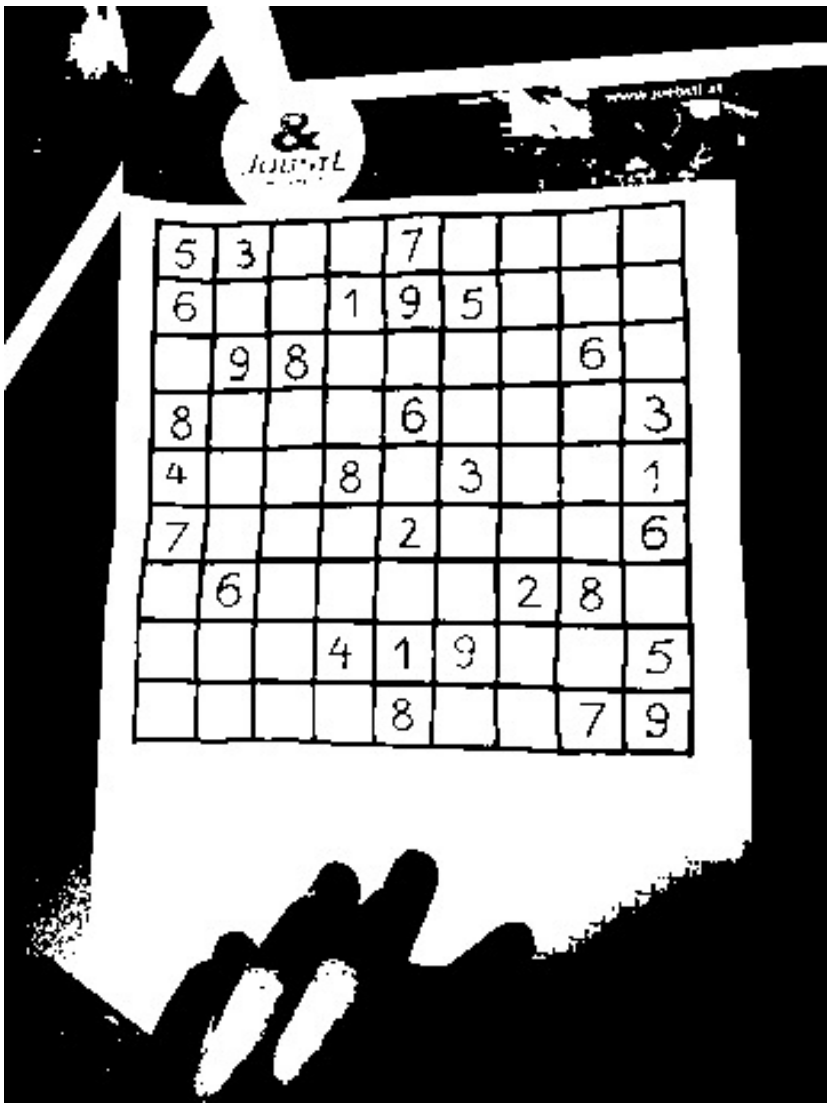
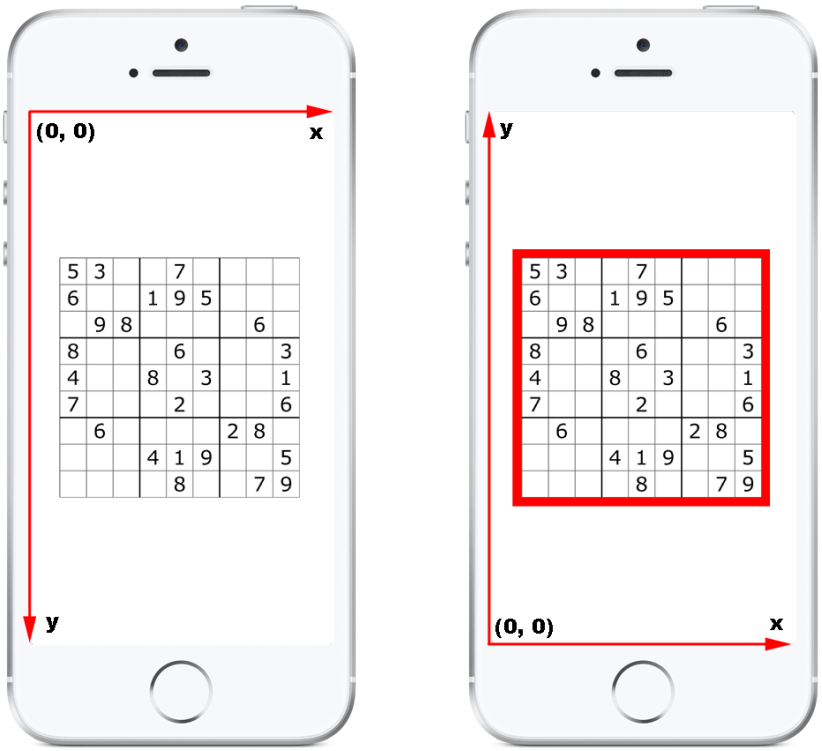


Figure 5.2: The bitmap of an image with shadows and a dark background

The assumption we made was very strong in the first place. When converted into a bitmap, all dark objects in the background, as well as shadows, will appear black. Thus, they will quite often comprise big components. Since we measure the size of a component by the number of pixels in it, no matter how large the grid is, it will never contain many pixels. This is due to the fact that its lines are relatively thin and most of the pixels within the grid are white. This is a big issue because in scenes like the one displayed in the figure below. The program would not even be able to detect the grid correctly and continue with further processing.

5.2 Apple's Vision Framework

With the iOS 11 release in September 2017, Apple also released a new powerful Vision framework. It uses traditional computer vision and deep learning algorithms to detect



(a) UIKit (b) Core Graphics

Figure 5.3: Coordinate spaces

certain landmarks. Conveniently, these include rectangle shapes. The detection runtime is much less than a second.

Sending a VNDetectRectanglesRequest with our camera image returns an array of VNRectangleObservations. For our program, in most cases, there will only be one observation - the outline of the sudoku. Each observation has a boundingBox property with the size of the detected rectangle as well as its coordinates. The boundingBox is a rectangle of type CGRect containing the detected object.

The coordinate system that Core Graphics uses is different from the one in the original image. This means that the coordinates cannot be used directly and therefore we need to apply a transformation first to convert them to the appropriate form. UIKit coordinate space has its origin in the top left corner, whereas Core Graphics coordinate space's origin in the bottom left corner.

Chapter 6

What is in the Box?

6.1 Separating the Squares

After detecting the grid, finding the coordinates of the separate squares becomes trivial. Since we know that they are of equal size, we can just cut the grid into 9x9 equal parts. Going through all pixels in the image in the bounds of the grid, we check which of those parts this pixel belongs to according to its coordinates. We generate 81 new bitmaps - of each of the cropped squares.

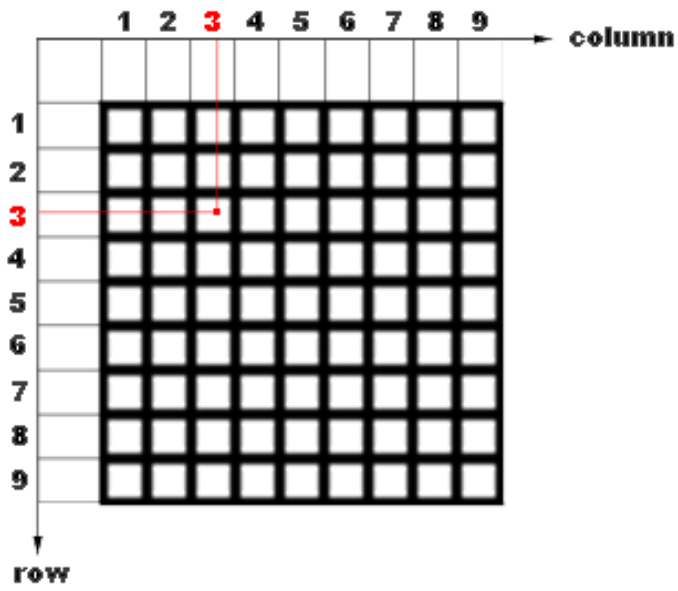


Figure 6.1: Separating the squares

If the coordinates of the top left corner of the grid are $(left, top)$ and the lengths of its sides are respectively *width* and *height*, then we have the following:
Each little square has sides of length w and h, where

$$w = width/9$$

$$h = height/9$$

Then a pixel belongs to the square on row *row* and column *col* if its image coordinates (x, y) satisfy the following inequalities:

$$left + (col - 1) \times w \leq x \leq left + col \times w$$

$$top + (row - 1) \times h \leq y \leq top + row \times h$$

6.2 Cropping the Digits

When we obtain the images of all the squares, as well as a digit, most of them also have pieces of the grid which we need to remove.

Figure 6.2: Separating the digits from the grid

We can treat the bitmap image as a graph in which the pixels are the vertices. Two vertices are then connected by an edge if they share a side i.e. if they are neighbouring. Thus the bitmap can essentially be looked at as an adjacency matrix. We can take the pixel in the centre as the root and start traversing the graph from it breadth-first using the flood fill method. We stop when we encounter a black (non-empty) pixel or when we have gone through all of the pixels. If the image only contains white pixels, then the corresponding square does not contain a digit. Otherwise, we initialise a second traversal, starting from the pixel that we found in the previous one. This time, we only visit black adjacent pixels. What we are left with is a mask of the digit that we can apply to the initial image.

The traversal works as follows:
We start at a given vertex and add it to the to the queue Q of vertices to be visited.

```
while Q is not empty do
    take the first element of Q;
    mark E as visited;
    remove E from Q;
    foreach neighbour N of the E do
        if N is not visited then
            add N to Q;
        end
    end
end
```

Algorithm 1: Flood fill algorithm

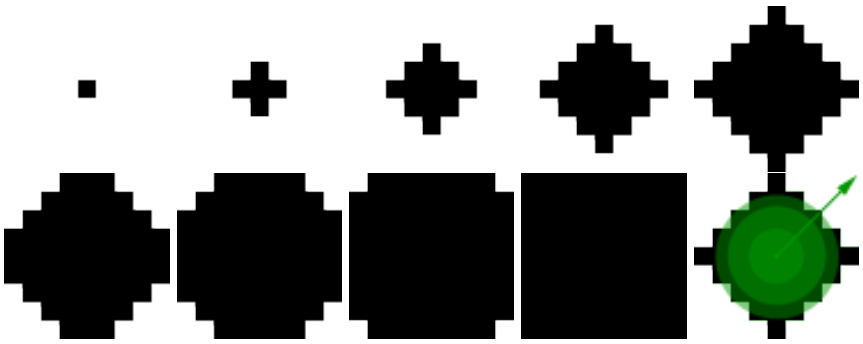


Figure 6.3: Flood fill illustration

Some squares will not contain a digit at all. However, this does not imply that the search will necessarily find no black pixels.
Random noise from the camera, ink marks and impurities on the paper can cause small clusters of black pixels to appear in the bitmap. To account for that, we can compute the size of every found component and disregard that component if the size is less than a threshold. After some experimentation, we found that the digit taking up least space was 1 with an average of 10% of the square's area. Therefore, we set the threshold to be 0.1 times the size of the given square.
Furthermore, pieces of the grid can still be found on squares with no digits. If this is the case, the search will find these so we need a way to determine if a certain component is a digit or a part of the grid. By calculating the difference in coordinates in the leftmost and rightmost pixels in the component, as well as the one in the coordinates of the pixels at the very top and very bottom, we can find the object's width and height. If either of those components is equal to more than 95% of, respectively, the width or the height of the square, this means the object is a part of the grid and regard this square as empty. The reason for the 5% interval we have left is that if we are dealing with one of the corners, it is not necessary for the lines to pass through the whole square. The assumption we make here is that no digit is going to take up it's square end to end, in other words, the font will be small enough to leave sufficient gaps between the digits and the grid.

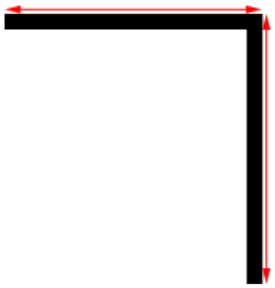


Figure 6.4: Empty squares

Chapter 7

Digit Recognition

7.1 Core ML

The release of iOS 11 also provides developers with a new Machine Learning framework. Core ML allows the developer to integrate a trained machine learning model into their app. Models are created and trained using a third-party machine learning framework. Thus, training is done offline, saving both run-time and space on the app. Core ML Tools is a Python package that converts a variety of model types into the Core ML model format. There are also third-party conversion tools that convert a model to the Core ML model format. [31][32]

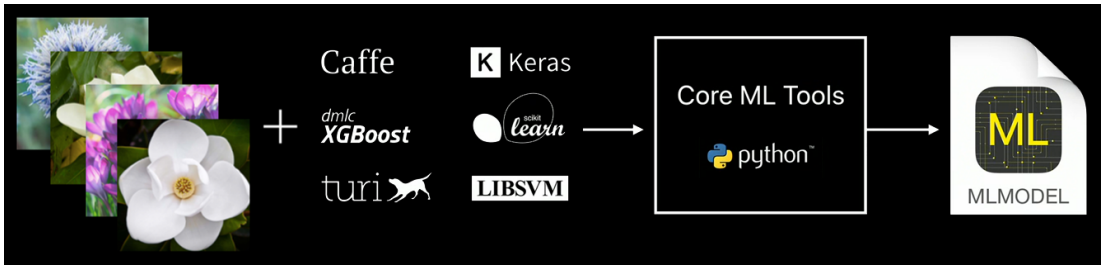


Figure 7.1: Core ML conversion [31]

Model type	Supported models	Supported frameworks
Neural networks	Feedforward, convolutional, recurrent	Caffe v1 Keras 1.2.2+
Tree ensembles	Random forests, boosted trees, decision trees	scikit-learn 0.18 XGBoost 0.6
Support vector machines	Scalar regression, multiclass classification	scikit-learn 0.18 LIBSVM 3.22
Generalized linear models	Linear regression, logistic regression	scikit-learn 0.18
Feature engineering	Sparse vectorization, dense vectorization, categorical processing	scikit-learn 0.18
Pipeline models	Sequentially chained models	scikit-learn 0.18

Table 7.1: Models and third-party frameworks supported by Core ML Tools [32]

7.2 MNIST Dataset

Our aim is to build an app that recognises as many sudoku puzzles as possible, regardless of the font or ink colour. This is why we want it to work accurately with both typeset and handwritten digits, hence our choice for a training set - the MNIST dataset. The MNIST (Modified National Institute of Standards and Technology) database is a large collection of annotated images of handwritten digits. It is widely used for training digit recognition systems.

Figure 7.2: Sample images from MNIST test dataset. (Image credit: Josef Steppan)

7.3 The Model

There are multiple already compiled models online for various purposes. Sri Raghu Malireddi's model for prediction of hand written digits uses precisely the MNIST dataset for training. It is build using a convolutional neural network. We integrated this model into our app. The input it takes are 28x28 pixel greyscale images. Therefore, after we crop the images of the digits, we have to scale them to the appropriate size before using them as an input to the classifier.

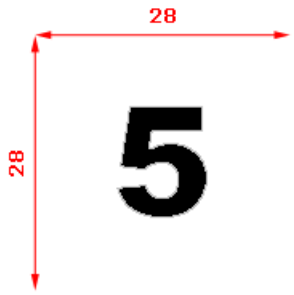


Figure 7.3: Scaling a digit

7.4 Classification Accuracy

To calculate the accuracy of the digit recognition in our app, we compiled a set of 100 different images for each of the 10 classes (each of the digits 1-9 and the absence of a digit). The pictures were taken under different lighting conditions and the digits were written or printed with different ink on different kinds of paper. We hand-labelled each image with its true class and made a prediction using the model. We obtained an accuracy of 90.1%. The resulting confusion matrix is displayed below. We use 0 to represent an empty square.

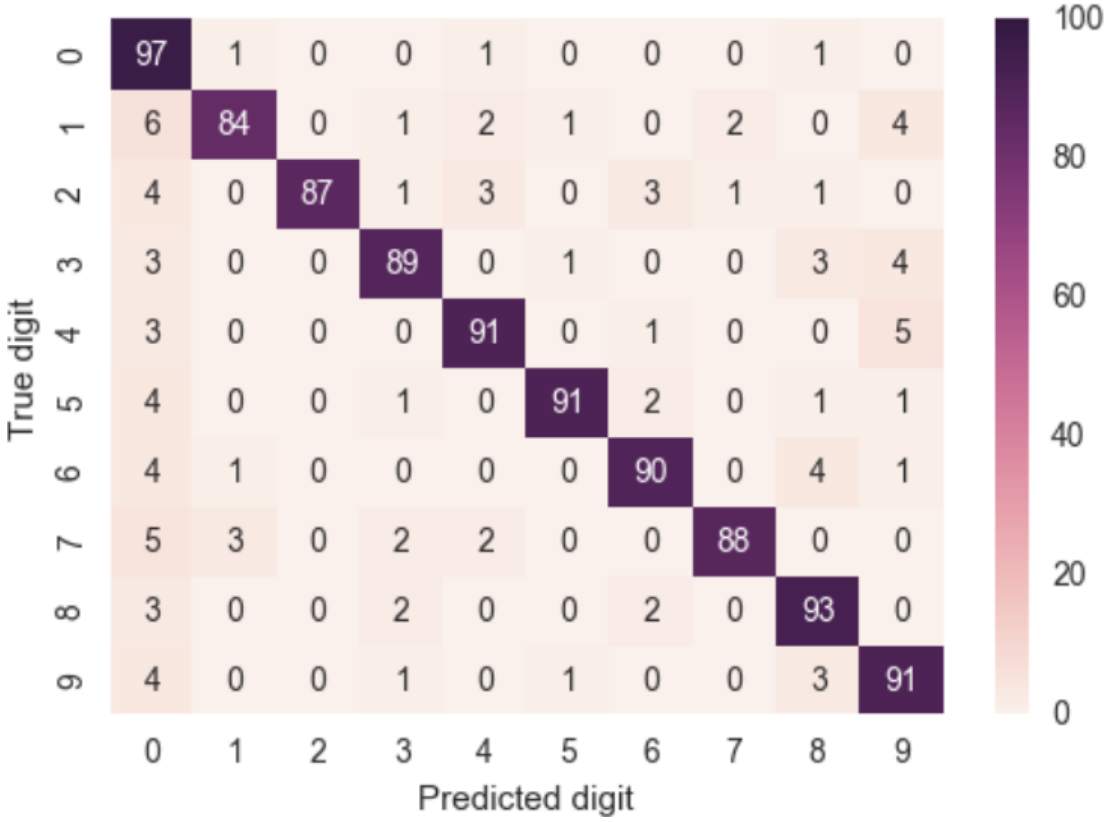


Figure 7.4: Confusion matrix

Chapter 8

Solving the Puzzle

8.1 Backtracking

Sudoku puzzles are NP-complete problems and as such, their solution can be found by performing an exhaustive search. Backtracking is a depth-first search brute force algorithm that guarantees to find a solution as long as one exists. The algorithm assigns numbers to empty cells so that the puzzle remains valid and recursively checks whether this assignment can lead to a solution or not. We shall call a puzzle valid when there are no two repeating digits in any row, column or block. The algorithm returns *true* and fills the grid with the correct solution if such exists, and returns $false$ otherwise.

```
Find an empty cell with coordinates (row, col).
if no cell is found then
    return true;
end
for digits x from 1 to 9 do
    if we can place x at position (row, col) and the puzzle remains valid then
        recursively continue to fill the rest of the puzzle;
        if recursion succeeds i.e. returns true then
            return true
        end
        else
            free cell and try with the next digit;
        end
    end
end
if none of the digits lead to a solution then
    return false
end
```

Algorithm 2: Backtracking

The time needed for the algorithm to run is not directly related to the difficulty of the puzzle. It is dependent on the number of times the recursion is repeated (the function calls itself). There have been puzzles constructed such that the performance of the

backtracking algorithm is very slow. One such puzzle is the following:

9	8	7	6	5	4	3	2	1
2	4	6	1	7	3	9	8	5
3	5	1	9	2	8	7	4	6
1	2	8	5	3	7	6	9	4
6	3	4	8	9	2	1	5	7
7	9	5	4	6	1	8	3	2
5	1	9	2	8	6	4	7	3
4	7	2	3	1	9	5	6	8
8	6	3	7	4	5	2	1	9

Figure 8.1: Sudoku designed so the backtracking algorithm works slow on it.

Looking at the first row of the solution, it becomes obvious why the program takes so long to finish - it needs to backtrack all the way to the first cells numerous times.

8.2 Optimization

The number of steps needed for completion of the algorithm is dependent on the order in which the cells are filled. Kevin Coulombe has described some heuristics that can optimise the program by choosing the cells in the most efficient order. We have slightly modified his rules so they became as follows:

1. If no other value is allowed in a cell, fill the cell with that value.
2. If a certain value is allowed in no other cell in the same row/column/block, fill the cell with that value.
3. If a certain value is allowed only on one column or row inside a block, we can eliminate this value from that tow or column in the other blocks.
4. If none of the above is applicable, make a guess or backtrack as needed.

In order to use these, we need to keep a list of the allowed values for each cell. Instead of keeping those in an array, we can represent them as 9-bit binary numbers for each cell. Thus, if for a certain cell, the bit in position x is 1, the value x is allowed for the cell. Otherwise, if the bit in position x is 0, the value x is not allowed. Having this mapping, we can use bitwise operations to update the allowed values. Bitwise operations are much more time efficient than looping through and accessing arrays.

Working with bitwise operations is done the following way:

- To obtain a mask for a number which has 1's in positions x_i we simply sum over all 2^{x_i-1}:

$$\sum_{x_i} 2^{x_i-1}$$

- To set the i^{th} bit of a number to 1 we simply need to *OR* that number with a number whose i^{th} bit is 1.
- To set the i^{th} bit of a number to 0 we simply need to *AND* that number with a number whose i^{th} bit is 0.

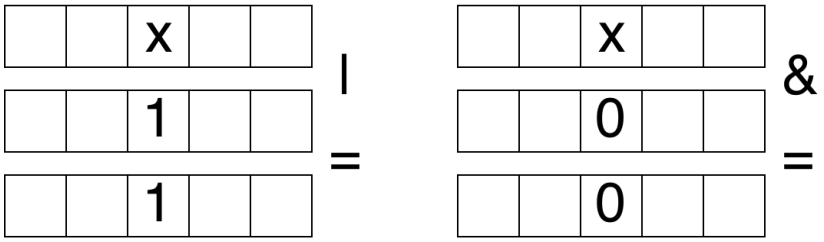


Figure 8.2: Bitwise operations

However, even optimized, backtracking's run-time might still go up to a few seconds depending on the puzzle. We have left implementing Algorithm X [18] for future improvements due to a shortage of time.

Chapter 9

Solution Overlay

After we are done with the most crucial part (solving the puzzle), there is just one thing left to do: display the solution on the screen. We began by overlaying the camera preview with a transparent image of an empty grid. We set the opacity of this image to 50% so the actual puzzle and the digits underneath can still be clearly seen.

The grid image has two purposes. The first one is mainly for aesthetics. We present the solution to the user by turning it into an array of UILabels. We modify the text and the position of those labels accordingly and display them on top of the grid. The grid serves as a frame and makes the numbers appear more organised. Having this frame, we also do not need to worry about labels' size as it stays fixed.

The second purpose of the grid is the more important one. It helps the user align the camera and the puzzle so that the app has a better chance of recognising it.

In terms of recalculating the solution and updating the UILabels, we experimented with three different ways of dealing with the problem.

The first one was to keep repeating the whole process after each iteration of the algorithm. This approach had a problem when the user's hand is not stable enough or the object of interest in the scene is not static. We have no way to stabilise the image. The user has no way to determine the exact moment when the app is going to extract the next frame. Therefore, it is very likely that the sudoku will not be in focus and the program will not produce satisfactory results.

The second one was to repeat the process at a certain frame-rate. Apart from the issue with the first approach, this one also had another drawback. Different sudoku puzzles take different amount of time to be solved. Hence, we have no way of knowing how to set the frame-rate in such way to be sure that the whole process is finished before starting it again. If the frame-rate is too high, the app will never manage to get to the end and will never display the solution.

The approach we finally decided on was to add a ”Solve” button and make the app extract the image frame from the moment that this button is pressed. This approach lacks the problems of the previous two and further sets the stage for adding some future app functionalities as explained in the next chapter. We also added a ”Clear” button as an extra feature in case the user wants to hide the solution on the screen.

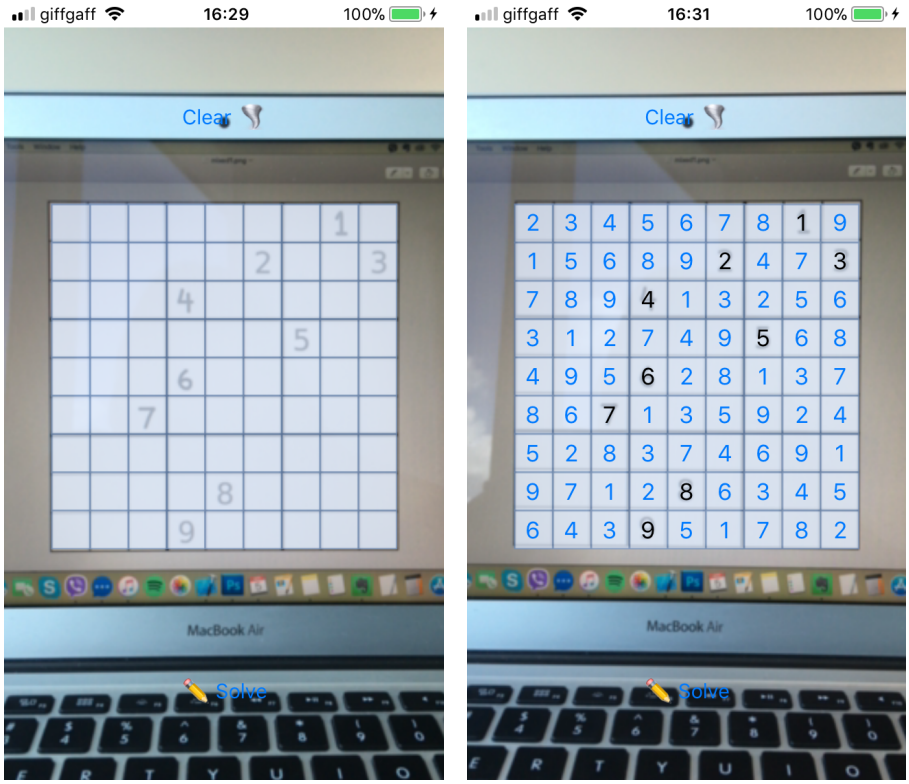


Figure 9.1: Overlaying the Solution on Screen

Chapter 10

Conclusion

10.1 Limitations and Future Improvements

10.1.1 Perspective Distortion

The biggest issue of our app is its lack of an ability to cope with distortion of shapes in the image due to the perspective (angle between the shooting plane and the image plane). We can fix this by applying homography. Knowing the coordinates of the corners of the grid, we can fix one of them in place and set the new positions of the other three corners so that they form a square. These can be used as our 4 points of correspondence.

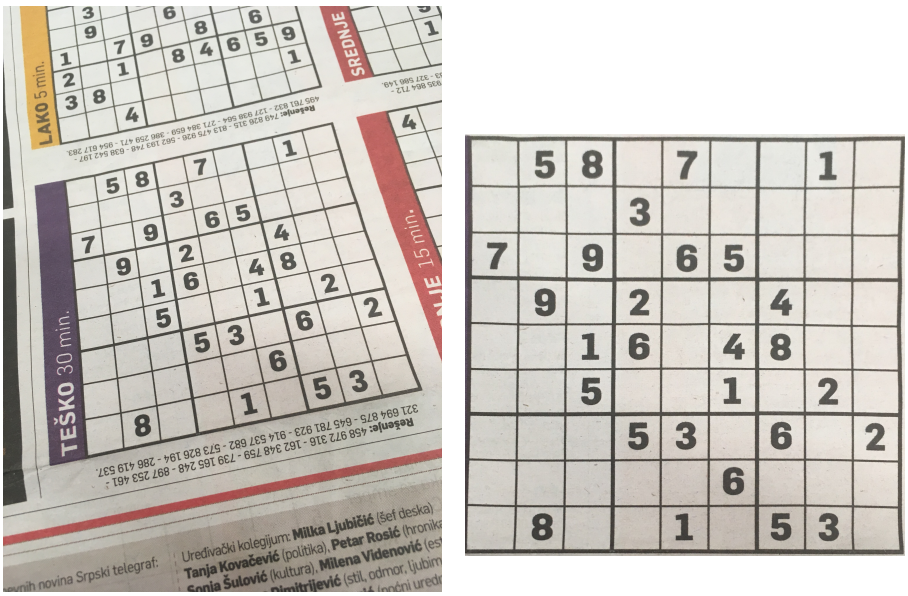


Figure 10.1: Perspective crop

10.1.2 Augmented Reality

Instead of displaying the numbers in a grid as we now do, we could make use of Apple's ARKit framework and use images of the numbers instead of labels. Those pictures can then be resized and skewed so that they can be mapped back into the scene for a more natural presentation of the solution. [33]

There is a product on the market that already supports this functionality. The app is called Magic Sudoku and has been developed by Hatchlings, Inc. Its capabilities also include solving multiple puzzles simultaneously. [34]

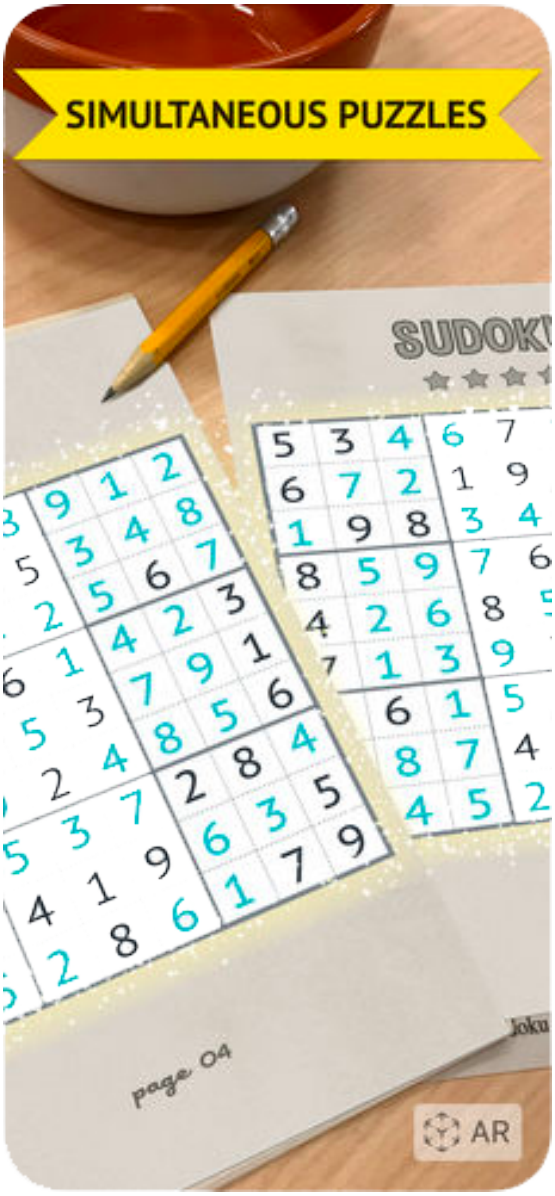


Figure 10.2: Magic Sudoku [34]

10.1.3 Object Tracking

The Vision framework has an object tracking feature. If we can track the sudoku grid's movement, we can mimic and use it to move the positions on the numbers as well, instead of having to recognise the puzzle all over again each time it moves. [35]

10.1.4 Additional Features

Features that can be added to the app in the future to improve user experience include:

- Showing the recognised numbers that were in the sudoku originally before the rest of the numbers in the solution and making them editable by the user in case the recognition wasn't correct.
- Offering the option of showing only a hint instead of the complete solution. The hint may be either in the form of showing the value in the next square that an algorithm using human solving techniques would fill or revealing the value of a square chosen by the user.
- We can add the option of saving an overlaid image to the user's picture gallery.

10.2 Evaluation

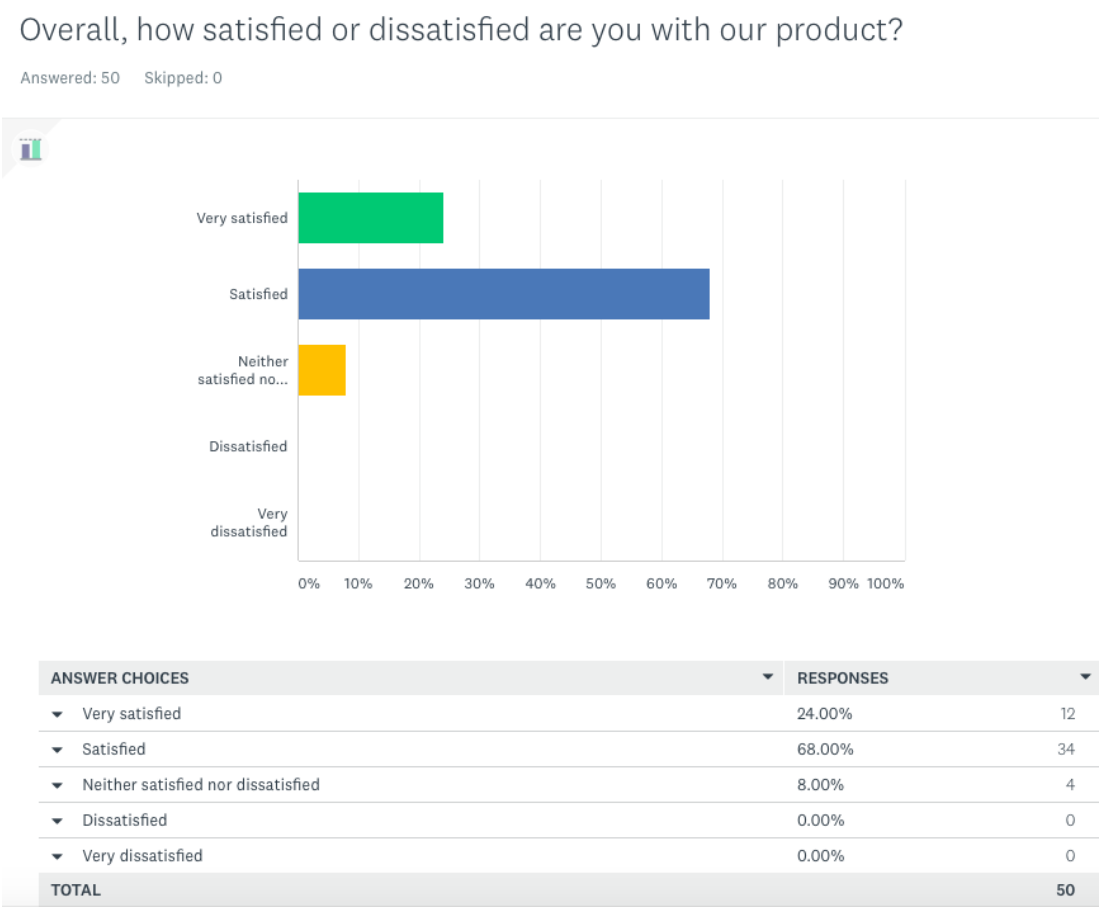
10.2.1 Choosing a Method

We needed a way to quantitatively evaluate our product in a relatively short period of time. Considering that out app only has one screen and a couple of buttons, applying Neilson's 10 Heuristics or having an expert perform a cognitive walk-through would not be of much benefit to us. We decided to design a survey to get our audience's opinion on the app. This was a quick way to reach out to and collect information from a big number of people. The closed questions let us easily analyze the data. Some of the questions we left open so that we could get a deeper understanding of the answers. Prior to this study, users were presented with a consent form, followed by a demonstration of the app.

10.2.2 Survey Results

In this section, we present the results we obtained from surveying 50 people (25 male and 25 female) with a wide range of age and degree of study or occupation.

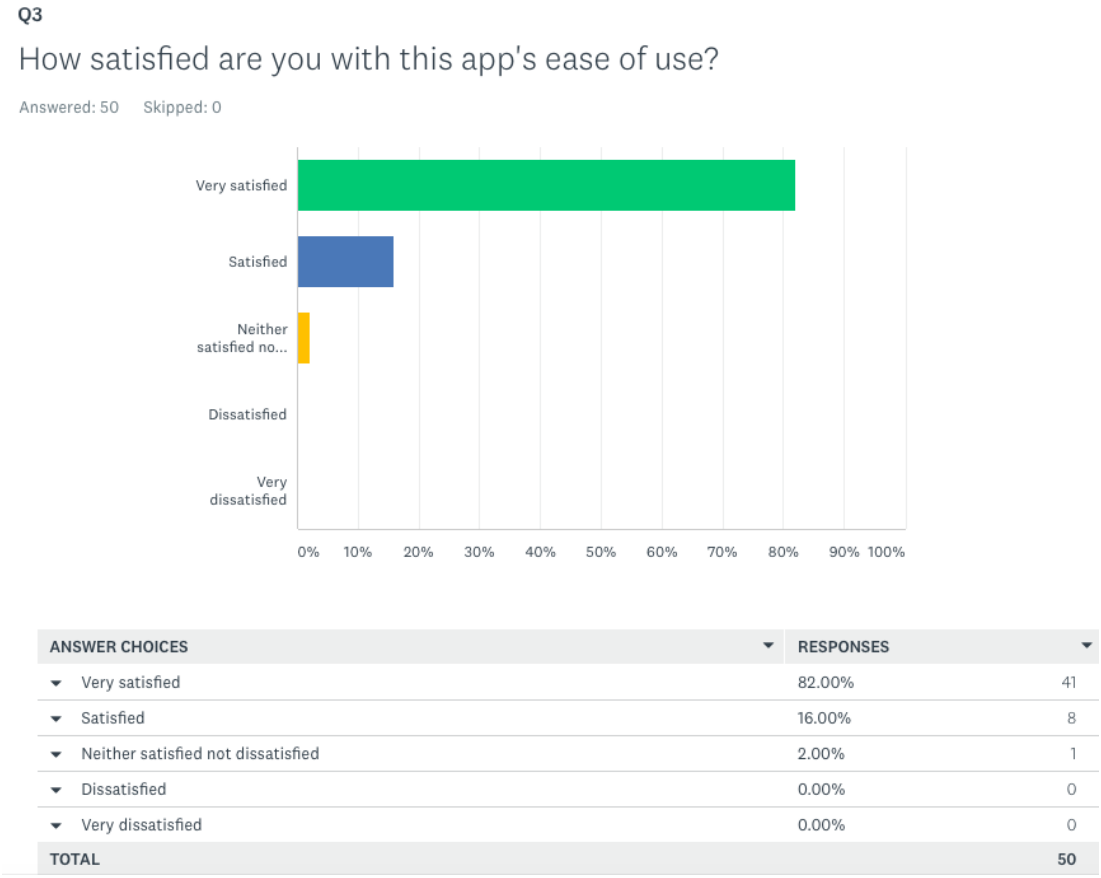
10.2.2.1 Overall User Satisfaction



ANSWER CHOICES	RESPONSES	
Very satisfied	24.00%	12
Satisfied	68.00%	34
Neither satisfied nor dissatisfied	8.00%	4
Dissatisfied	0.00%	0
Very dissatisfied	0.00%	0
TOTAL		50

Figure 10.3: Overall satisfaction

10.2.2.2 App's Ease of Use



ANSWER CHOICES	RESPONSES	
Very satisfied	82.00%	41
Satisfied	16.00%	8
Neither satisfied not dissatisfied	2.00%	1
Dissatisfied	0.00%	0
Very dissatisfied	0.00%	0
TOTAL		50

Figure 10.4: Ease of use

10.2.2.3 Users' Likelihood of Recommendation

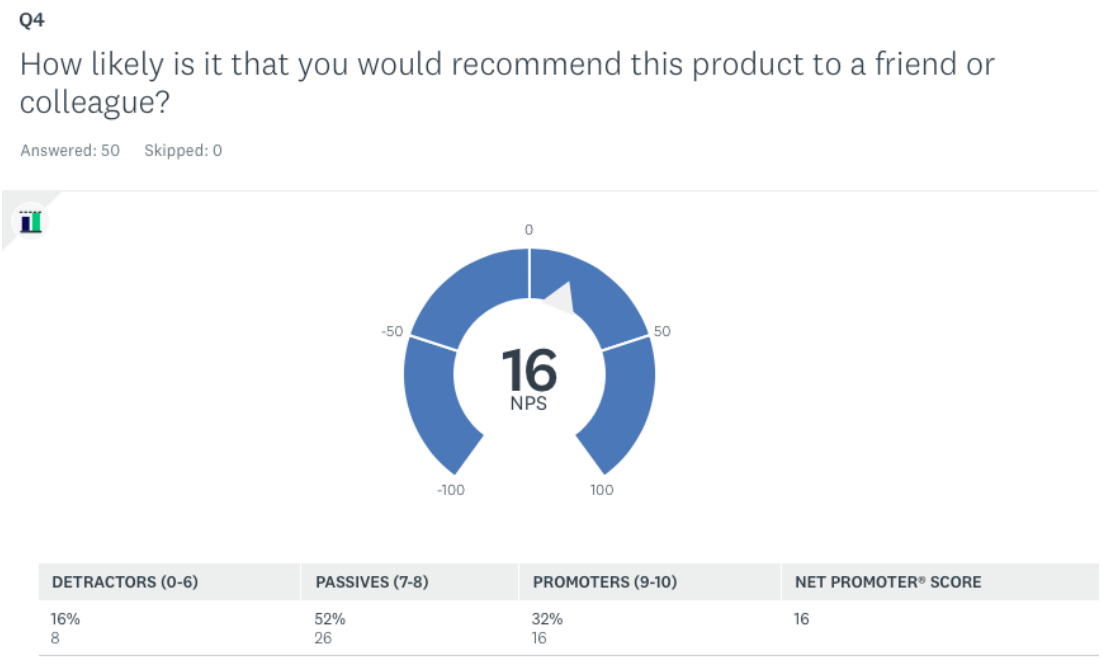


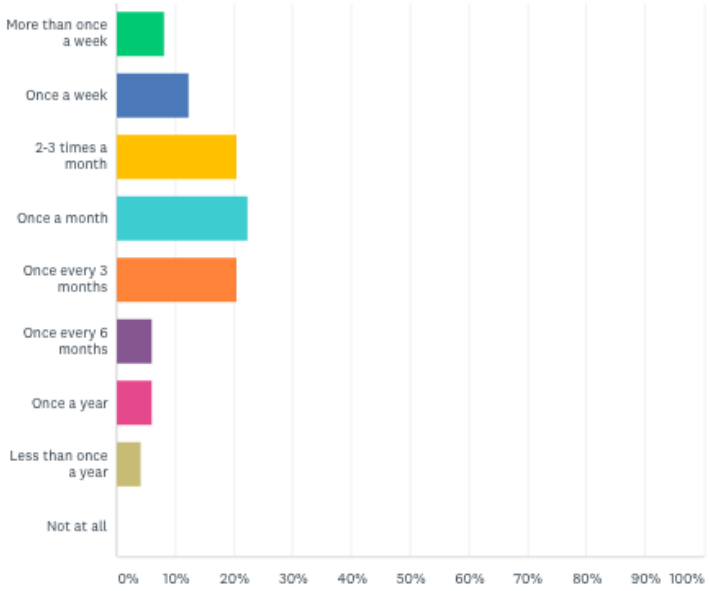
Figure 10.5: Likelihood of Recommendation

10.2.2.4 Users' Frequency of Solving Sudoku Puzzles

Q5

About how often do you solve Sudoku puzzles?

Answered: 49 Skipped: 1



ANSWER CHOICES	RESPONSES	
More than once a week	8.16%	4
Once a week	12.24%	6
2-3 times a month	20.41%	10
Once a month	22.45%	11
Once every 3 months	20.41%	10
Once every 6 months	6.12%	3
Once a year	6.12%	3
Less than once a year	4.08%	2
Not at all	0.00%	0
TOTAL		49

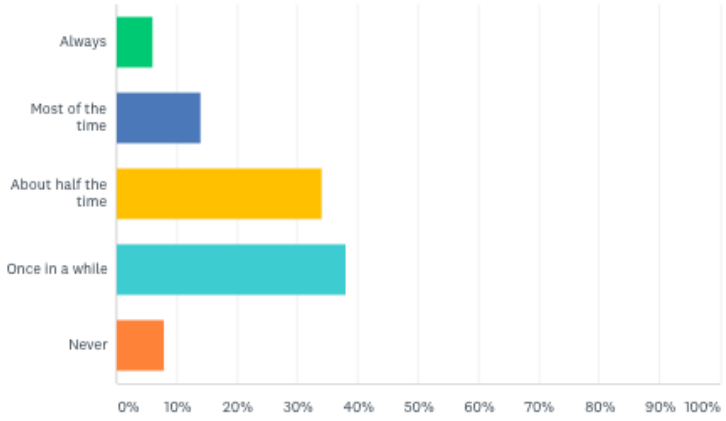
Figure 10.6: Frequency of sudoku solving

10.2.2.5 Users' Need of Help when Solving Sudoku

Q6

How often do you need help solving a Sudoku?

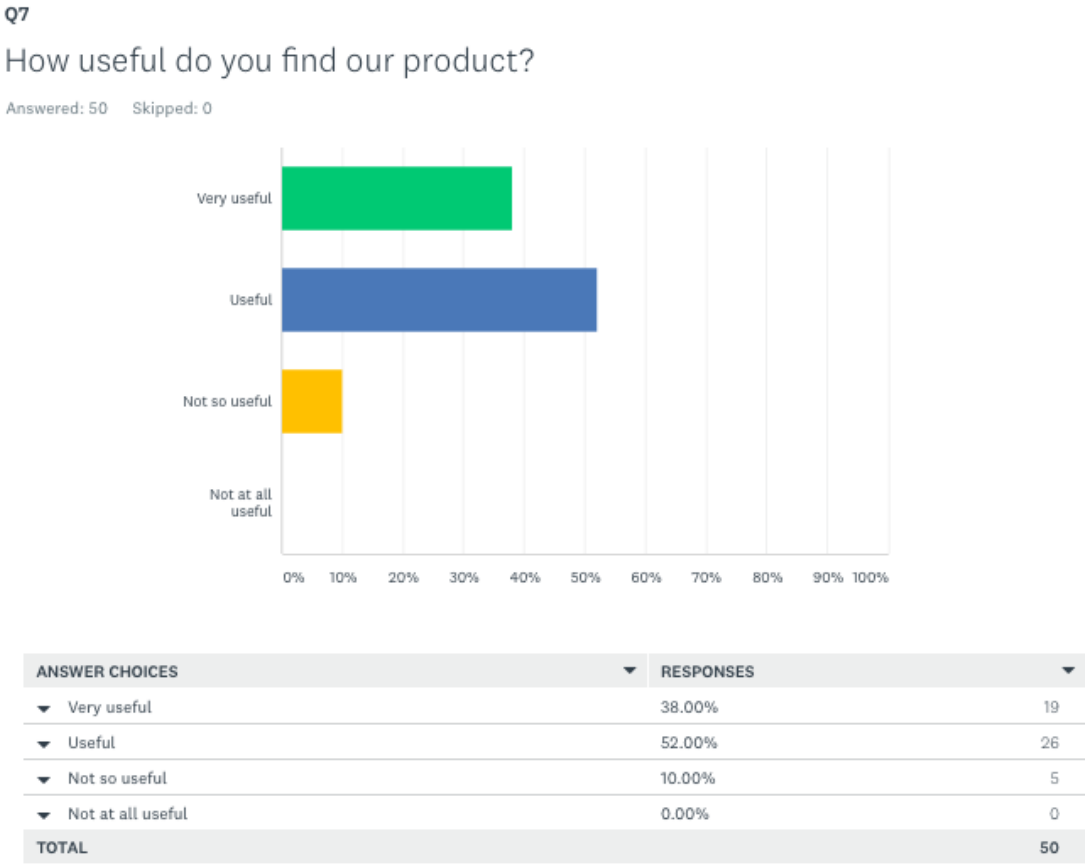
Answered: 50 Skipped: 0



ANSWER CHOICES	RESPONSES	
Always	6.00%	3
Most of the time	14.00%	7
About half the time	34.00%	17
Once in a while	38.00%	19
Never	8.00%	4
TOTAL		50

Figure 10.7: Need of help for solution

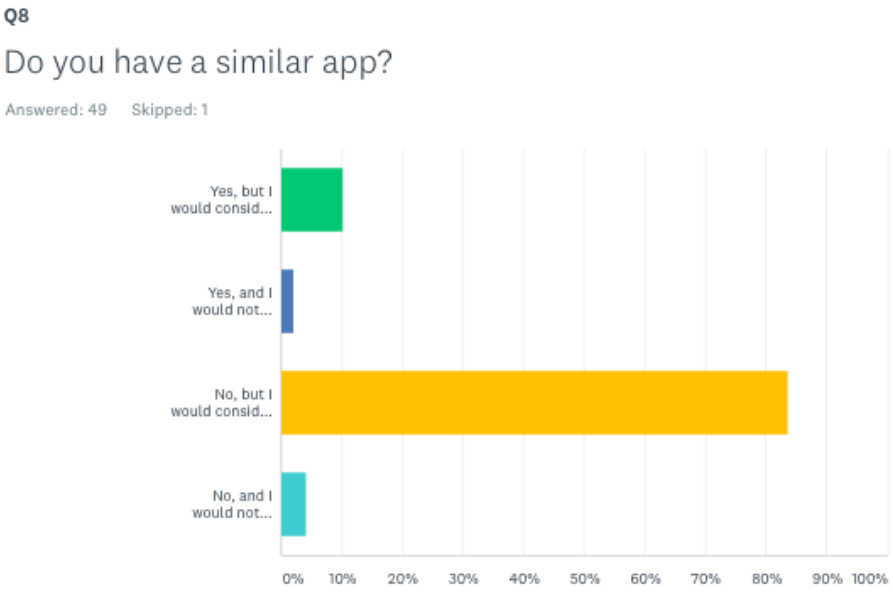
10.2.2.6 Usefulness of the App



ANSWER CHOICES	RESPONSES	
Very useful	38.00%	19
Useful	52.00%	26
Not so useful	10.00%	5
Not at all useful	0.00%	0
TOTAL		50

Figure 10.8: Usefulness of the app

10.2.2.7 Do Users Have a Similar App



ANSWER CHOICES	RESPONSES	
Yes, but I would consider changing it for a better one.	10.20%	5
Yes, and I would not install a new one.	2.04%	1
No, but I would consider installing such app.	83.67%	41
No, and I would not install one.	4.08%	2
TOTAL		49

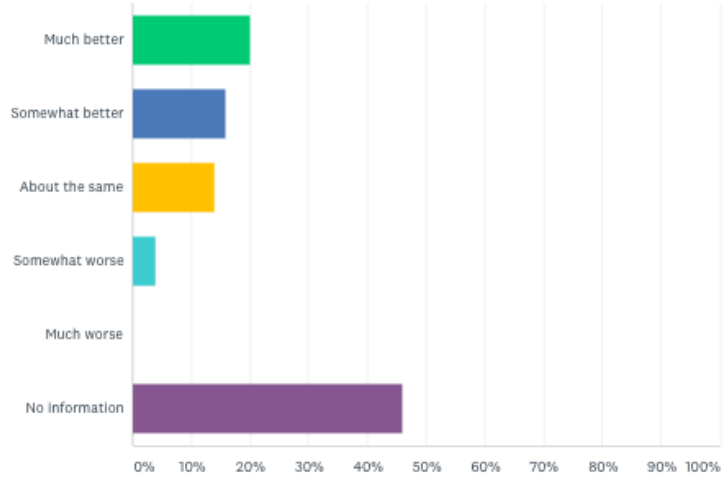
Figure 10.9: Do users have a similar app

10.2.2.8 App Quality Comparison

Q9

Compared to our competitors, is our product quality better, worse, or about the same?

Answered: 50 Skipped: 0



ANSWER CHOICES	RESPONSES	
Much better	20.00%	10
Somewhat better	16.00%	8
About the same	14.00%	7
Somewhat worse	4.00%	2
Much worse	0.00%	0
No information	46.00%	23
TOTAL		50

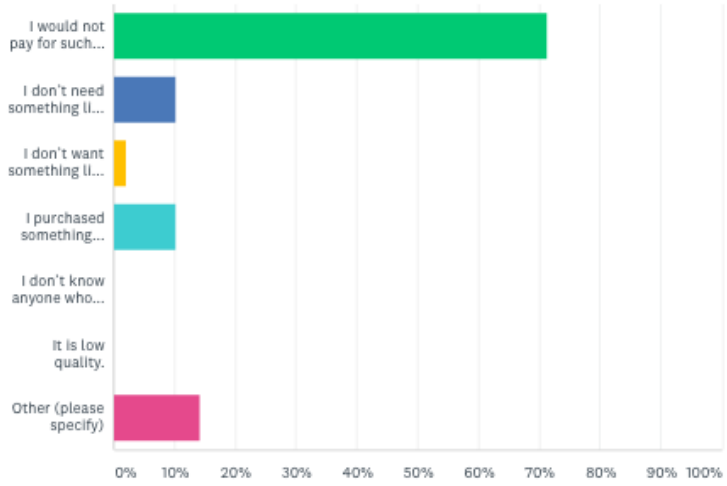
Figure 10.10: App quality comparison

10.2.2.9 Reasons for Users not to Purchase the App

Q10

Which of the following are reasons that you might not purchase this product? Please select all that apply.

Answered: 49 Skipped: 1



ANSWER CHOICES		RESPONSES	
I would not pay for such product, but I would install it for free.		71.43%	35
I don't need something like this.		10.20%	5
I don't want something like this.		2.04%	1
I purchased something similar and I am satisfied.		10.20%	5
I don't know anyone who would want this.		0.00%	0
It is low quality.		0.00%	0
Other (please specify)	Responses	14.29%	7
Total Respondents: 49			

Figure 10.11: Reasons for users not to purchase the app

When we asked participants for reasons that would stop them from purchasing our app, most of the ones that voted for *Other* stated that they would like to see more features. One of the participants explained that seeing the whole solution ruins their experience of solving the puzzle and that they would only like to see a hint. Another one said that they are more likely to download the app with limited functionality and make an in-app purchase to unlock more features.

10.3 Discussion

Throughout this report, we have discussed different Vision and Machine Learning algorithms as well as algorithms for solving sudoku puzzles. We have selected and implemented some of them to successfully achieve the goal of the project. We have built an app for identifying and solving sudoku puzzles using a mobile device's camera. Although there is room for improvement, the app is highly useful and owing to its easy to understand interface - accessible to a wide target audience.

Bibliography

[1] R. B. Fisher and V. Ferrari, “Homography and transfer.” [Online]. Available: https://www.learn.ed.ac.uk/bbcswebdav/pid-2044540-dt-content-rid-3799043_1/courses/INFR090192016-7SV1SEM1/ivr_homographyf.pdf

[2] M. Brubaker, S. Mathe, D. J. Fleet, and A. Jepson, “2503 tutorial: 2d homographies.” [Online]. Available: http://www.cs.toronto.edu/~jepson/csc2503/tutorials/homography.pdf

[3] E. Vincent and R. Laganiere, “Detecting planar homographies in an image pair,” in *ISPA 2001. Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis. In conjunction with 23rd International Conference on Information Technology Interfaces (IEEE Cat.)*, 2001, pp. 182–187.

[4] R. B. Fisher and V. Ferrari, “Description of segments.” [Online]. Available: https://www.learn.ed.ac.uk/webapps/blackboard/content/listContent.jsp?course_id=_51138_1&content_id=_2044442_1

[5] B. Marr, “A short history of machine learning – every manager should read,” *Forbes*. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2016/02/19/a-short-history-of-machine-learning-every-manager-should-read/#4506982d15e7

[6] D. C. Cirean, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Deep, big, simple neural nets for handwritten digit recognition,” *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010, pMID: 20858131. [Online]. Available: https://doi.org/10.1162/NECO_a_00052

[7] V. Lavrenko and N. Goddard, “Neural networks.” [Online]. Available: https://www.learn.ed.ac.uk/bbcswebdav/pid-2768533-dt-content-rid-5352273_1/courses/INFR100692017-8SV1SEM1/ann.pdf

[8] D. Green, “Conceptis sudoku difficulty levels explained.” [Online]. Available: http://www.conceptispuzzles.com/index.aspx?uri=info/article/2

[9] M. Ercsey-Ravasz and Z. Toroczkai, “The chaos within sudoku,” August 2012. [Online]. Available: ”https://arxiv.org/abs/1208.0370”

[10] “Mathematics of sudoku leads to ”richter scale” of puzzle hardness,” August 2012. [Online]. Available: ”https://www.technologyreview.com/s/428729/mathematics-of-sudoku-leads-to-richter-scale-of-puzzle-hardness/”

[11] “Learn sudoku!” [Online]. Available: ”https://arxiv.org/abs/1208.0370”

[12] “Conceptis puzzel sudoku techniques.” [Online]. Available: ”http://www.conceptispuzzles.com/index.aspx?uri=puzzle/sudoku/techniques”

[13] A. Brostrm and S. Johansson, “Sudoku solvers. man versus machine.” [Online]. Available: ”http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand13/Group5Roberto/final/Andreas.Brostrom.Simon.Johansson.report.pdf”

[14] V. Chytrý, *Sudoku game solution based on graph theory and suitable for school-mathematics.* Nitra: Constantine The Philosopher University in Nitra, Faculty of Natural Sciences, 2014, pp. 55–61.

[15] M. Eissa, “A sudoku solver using graph coloring,” July 2014. [Online]. Available: ”https://www.codeproject.com/Articles/801268/A-Sudoku-Solver-using-Graph-Coloring”

[16] S. Albrecht, J. Fleuriot, M. Rovatsos, and M. Herrmann, “Smart searching using constraints.” [Online]. Available: ”https://www.inf.ed.ac.uk/teaching/courses/inf2d/slides/08_Smart_Searching_Using_Constraints.pdf”

[17] “Exact cover matrix.” [Online]. Available: ”http://www.stolaf.edu/people/hansonr/sudoku/exactcovermatrix.htm”

[18] D. E. Knuth, “Dancing links,” November 2000. [Online]. Available: ”https://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/0011047.pdf”

[19] G. Rees, “Zendoku puzzle generation.” [Online]. Available: ”http://garethrees.org/2007/06/10/zendoku-generation/”

[20] Apple, “Avcapturesession.” [Online]. Available: https://developer.apple.com/documentation/avfoundation/avcapturesession

[21] ——, “Avfoundation programming guide.” [Online]. Available: https://developer.apple.com/library/content/documentation/AudioVideo/Conceptual/AVFoundationPG/Articles/04_MediaCapture.html

[22] R. B. Fisher, “Finding objects by background removal.” [Online]. Available: https://www.learn.ed.ac.uk/bbcswebdav/pid-2044474-dt-content-rid-3799094_1/courses/INFR090192016-7SV1SEM1/ivr_backgroundf.pdf

[23] ——, “Thresholding based segmentation.” [Online]. Available: https://www.learn.ed.ac.uk/bbcswebdav/pid-2044470-dt-content-rid-3799056_1/courses/INFR090192016-7SV1SEM1/ivr_thresholdf.pdf

[24] T. R. Singh, S. Roy, O. I. Singh, T. Sinam, and K. M. Singh, “A new local adaptive thresholding technique in binarization,” *CoRR*, vol. abs/1201.5227, 2012. [Online]. Available: http://arxiv.org/abs/1201.5227

[25] C. Chow and T. Kaneko, “Automatic boundary detection of the left ventricle from cineangiograms,” *Computers and Biomedical Research*, vol. 5, no. 4, pp. 388 – 410, 1972. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0010480972900705

[26] K. V. Mardia and T. J. Hainsworth, “A spatial thresholding method for image segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 10, no. 6, pp. 919–927, Nov. 1988. [Online]. Available: http://dx.doi.org/10.1109/34.9113

[27] W. Niblack, “An introduction to digital image processing,” 01 1986.

[28] T. Taxt, P. J. Flynn, and A. K. Jain, “Segmentation of document images,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 12, pp. 1322–1329, Dec. 1989. [Online]. Available: http://dx.doi.org/10.1109/34.41371

[29] S. D. Yanowitz and A. M. Bruckstein, “A new method for image segmentation,” in *[1988 Proceedings] 9th International Conference on Pattern Recognition*, Nov 1988, pp. 270–275 vol.1.

[30] J. J. Sauvola, T. Seppänen, S. Haapakoski, and M. Pietikäinen, “Adaptive document binarization,” in *Proceedings of the 4th International Conference on Document Analysis and Recognition*, ser. ICDAR ’97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 147–152. [Online]. Available: http://dl.acm.org/citation.cfm?id=646270.685453

[31] Apple, “Introducing core ml, wwdc 2017.” [Online]. Available: https://developer.apple.com/videos/play/wwdc2017/703/

[32] ——, “Converting trained models to core ml.” [Online]. Available: https://developer.apple.com/documentation/coreml/converting_trained_models_to_core_ml

[33] ——, “Arkit,” 2018. [Online]. Available: ”https://developer.apple.com/arkit/”

[34] “Magic sudoku.” [Online]. Available: ”https://itunes.apple.com/us/app/magic-sudoku/id1286979959?mt=8”

[35] Apple, “Vntrackingrequest,” 2018. [Online]. Available: ”https://developer.apple.com/documentation/vision/vntrackingrequest”

Appendix A

Consent Form

My name is Martina Kotseva. I am a student at the University of Edinburgh. For my 4th Year Honours Project, supervised by Professor Nigel Topham, I am conducting a study aimed at evaluating the usability of a Sudoku Solver application and gathering ideas for future improvements. I have designed a survey for this purpose. The survey contains 10 questions and takes about 3 minutes to complete.

Your participation in this study is voluntary. You may choose not to participate. If you decide to participate in this survey, you may withdraw at any time. You may skip any questions you do not feel comfortable answering.

All data that you provide will be stored on SurveyMonkey and access will be restricted to me. It will be destroyed promptly after completion of the project. Your anonymity will be preserved throughout the entire study.