Final Year Project Report

Full Unit - Final Report

Playing Games and Solving Puzzles Using Artificial Intelligence

Terique Carnegie

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Michail Fasoulakis

Department of Computer Science
Royal Holloway, University of London

April 15, 2025

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 12,814

Student Name: Terique Carnegie

Date of Submission: 11/04/2025

Signature: Terique Carnegie

Table of Contents

Abstract

The general purpose of artificial intelligence is to give computers the ability to replicate human intelligence through the use of problem-solving algorithms. AI has the potential to outperform human intelligence in solving many different and difficult problems due to its ability to "think" by running problem solving computations at much greater speeds and accuracy than a human and it does this through leveraging the superior processing capabilities of a computer [1].

Due to the nature of problem-solving algorithms, Artificial Intelligence is able to provide assistance and insight into many different fields of expertise making it popular and very much in demand as an asset to humanity. In order to increase the value of AI, it is necessary to deeply research ways to improve problem solving algorithms and to discover which of the algorithms works best for a specific problem, and it is also important to develop methods for humans and AI to safely interact. One way this can be achieved is through games and puzzles. Using games and puzzles as a means to research and further develop AI can be very beneficial as it allows for problem solving algorithms to be tested against fun solvable challenges which can provide insight on how well the algorithms are able to solve problems, using games can also provide a way for humans to interact with artificial intelligence through either testing the speed and efficiency of both the player and the algorithms or through the use of 2 player games like chess and checkers. This type of approach to using artificial intelligence has been seen in the past from the first working checkers program to appear in 1952 [3], and chess playing programs being developed shortly thereafter [2] which had a role in furthering research and development of modern algorithms and uses of Artificial Intelligence.

Chapter 1: **Introduction**

1.1 Artificial Intelligence Solving a Sudoku

Artificial Intelligence has been applied to many different fields of expertise and has had many advancements in their problem solving algorithms. Some of the more modern and complex problem-solving AI algorithms that can be engineered to play and solve challenges in games are backtracking algorithms, constraint satisfaction algorithms, search algorithms, minimax algorithms, Monte Carlo Tree Search, reinforcement learning and neural networks. All of these algorithms allow for AI to be able to play games and potentially interact with human players, however for the sudoku solver, the project will focus on and utilise only a few of these AI algorithms. The selected algorithms for solving this problem will be back tracking and neural network algorithms. The other problem-solving algorithms are capable of providing a solution but my reasonings for my selection will be explained in the following paragraphs.

There are several different types of backtracking algorithms, such as consistency pruning, looking ahead, efficient backtracking and variable and value heuristics, but the basic principle behind backtracking algorithms is to explore all possible solutions through attempting different paths recursively and regressing back to try a different path when the current is no longer deemed viable. A sudoku can be solved by assigning numbers to an empty square but validating the number before to check if it is safe to assign, however this kind of approach has a worst-case time complexity of $O(9^{n \cdot n})$ [4].

Neural networks like convolutional neural networks and recurrent neural networks can be used to solve problems through pattern recognition. It does this by imitating how the human brain works through layers of interconnected nodes that takes in data, performs computations on the data and provides a prediction based on the data analysed. This can be applied to solving a sudoku as if trained on the rules and sequences of numbers of sudoku training data, the algorithm could be able to predict the numbers that will fit sudoku sequence with up to 99% accuracy [5].

1.2 Purpose of my Application

Through this project I plan further the understanding of how Artificial Intelligence can be applied to solve games and puzzles with logical thinking and pattern recognition by developing a platform using the Python programming language for developing my AI algorithms, with the pygame library for the development of the Graphical User interface and the Visual Studio code IDE to effectively manage my work files, to aid in my projects goal of demonstrate the capability of Artificial intelligence and how it can quickly assess and solve puzzles and problems in a gaming environment, by creating multiple problem-solving algorithms which will tackle and solve sudoku puzzles of increasing sizes and complexity, and evaluating which of the following algorithms between both backtracking and neural networks is better for solving a sudoku puzzle efficiently and potentially faster than humans in most cases. I believe that this project will help me to display my skills as a software engineer and can help me to demonstrate a greater understanding in the application of Artificial intelligence and the type of fields and situations that it can be applied to in the technology sector, which will make me very versatile in this industry.

1.3 Professional Issues Faced with the Project

An issue of professionalism that arises when working with artificial intelligence is the concern of AI being integrated into workforces due to its problem solving skill, computational efficiency and overall reduced costs to any company choosing to employ AI for their business. This brings concerns for humanity as many people can be affected by this as they may find less job opportunities due to the rise in applications of artificial intelligence and in some cases, people may lose the jobs they are currently working as they will not be able to keep up with the processing of a machine and are not as cost effective as employing one.

1.3.1 Artificial intelligence replacing human jobs

This problem has been seen arising around the world as companies are opting to reduce costs while increasing their outputs, which is the type of returns artificial intelligence can bring to a business, and example in public domain of this taking place is the implementation of chatbots run autonomously with the power of Artificial Intelligence (reference here). There is currently 23% of customer service companies making use of automated chatbots and around 80% of people having interacted with these AI chatbots which was able to generate over $100 billion in ecommerce transactions [6], as well as saving companies over 2.5 billion working hours [7]. This shows that for tasks that can be improved with the assistance of AI then companies do not have a reason to not employ it as not doing so will only cost them a lot of time and money. In addition to this there is also around 87% of users that reported to be satisfied with their experience when interaction with this implementation of AI [8] as they were pleased with how quick and accurate their responses was, so as long as the models are trained well, then companies will have a means of producing very quick, cheap and effective output that does not involve a large and slow human workforce. With all this information, businesses integrating AI into their workforce in at least one form, like AI chatbots, are projected to increase to around 70% [9].

Regardless of these benefits to the business, AI in some areas can negatively impact the moral of human employees, as in the case of Amazon warehouses which employ managerial AI systems, which makes some of the employees managed by it feel as if they were "disposable cogs" in an "unforgiving" workplace [10], this isn't good for a company as having poor employee satisfaction among your employees can lead to low productivity and even higher staff turnover in the company, which could negatively affect the business's brand reputation. But even with these drawbacks to using AI, the benefits it raises are far superior for the business, and with the rate these algorithms are advancing, AI will be able to perform much better, imitating human work well enough to overcome challenges it may face when aiming to expand into the workforce.

However, despite the rise in AI across the workforce, there are still jobs and career paths that are available and are very resistant to AI changes, these are jobs that require creativity, which AI lacks as it can't replicate the experiences and intuitions of man, empathy, as machines do not have the same behavioral patterns as humans, and leadership, as humanity is not very likely to follow the lead of AI as it lacks initiative that is integral for leading any team of people. Some of these kinds of jobs include fields in social work, politics, education, healthcare and entrepreneurs. As social workers are required to show deep understanding of human emotion and experiences, which although AI could mimic this, it cannot truly relate and deeply understand those that would need help at this would require difficult ethical choices and complex interactions that models are not able to achieve. The lack of these traits also gives rise to a lack of the ability to build rapport and trust in AI, which makes other careers such as politics, education and healthcare extremely troublesome for artificial intelligence to expand into, as there is still many people that would prefer the human presence whether as a customer or an employee in these fields, around 44% of people in a global study would be willing to trust the application of artificial intelligence in healthcare for diagnosis and treatment [11] this shows that in some fields, AI is not yet ready to take over and will be faced with resistance.

1.3.2 The impact of demonstrating AI as a superior problem solver

The AI sudoku solver relates to this professional issue, as it primarily serves as a demonstration of the advanced computational speed and efficiency of AI models working from powerful processors. Although it aims to achieve this through the use of playing games and solving puzzles, the games and puzzles provides a "playground", a safe environment with constraints, the game/puzzle rules, for in which the training model can experiment and learn in hopes that it will be able to quickly adapt and efficiently calculate and predict a correct solution to the problem. This ability is the key trait that makes artificial intelligence attractive to companies that can exploit this to streamline any repetitive tasks that require a degree of problem solving that may take employees an unnecessarily long time to complete. This involves my project with this professional concern as the project itself acts as a proof of concept towards the idea that at some tasks human intelligence is inferior to digital intelligence as its ability to compute multiple instructions in succession, combined with algorithms which give it the ability to evaluate and evolve until it masters a solution to the problem it is presented with, makes it more suitable in the workforce for certain jobs which makes human employees redundant, which is a real fear that some people have in regards to the security of their jobs and careers which they have spent learning, studying and working in.

Chapter 2: **Application Development**

2.1 Project Milestones

There were a couple of milestones achieved in the development of the Artificial Intelligence sudoku solver. The following milestones played a significant role in putting together the application so that it can run and achieve the goals that the project was set to achieve were:

- **The development of the Console Interface:** this milestone had significance in the project as this milestone allowed for displaying the puzzle on the screen both solved and unsolved, this was a needed milestone as if the grid could not be displayed to the screen, it would be difficult to develop any of the AI algorithms as determining if the solving algorithms are outputting the correct results as reading from the array is not very efficient or user friendly due to the lack of a grid layout and sub-grid division.
- **The development of the Backtracking algorithm:** the first solving algorithm is key and very a very important milestone for this project as it displays that AI can be adapted to playing games and solving puzzles, it shows that machines are capable of making an attempt at solving the problem, and if the potential solution is wrong, it can go back an make a different attempt at a viable solution. As well as demonstrating how it is able to do so in a much shorter amount of time when compared to humans.
- **The development of the Graphical User Interface:** having a graphical user interface to display the steps and speed of the algorithm to a user is beneficial to proving that algorithm can learn from its mistakes, mimicking human intelligence, as it shows the algorithm choosing the incorrect solution and when realising its mistake, is displays the algorithm changing its solution to a valid one. This helps to demonstrate that AI can be used to solve problems quickly through the Sudoku grid solving animations that are run in the pygame window. It also provides an interface for the users to be able to interact with the application and switch between different solution algorithms, and sudoku puzzles.
- **The development of the Convolutional Neural Network:** Another integral milestone in the development of this project is the implementation of the secondary artificial intelligence algorithm, which is a CNN (a convolutional neural network), this milestone is important as it shows that a machine is capable to learning distinct patterns and features of a problem and its solution, similarly to a human, and use this knowledge to accurately predict a viable solution to a given problem. With the developed model, my application will be able to demonstrate how artificial intelligence can be trained to attempt to solve a mathematical problem and give a prediction with a suitable degree of accuracy.

2.2 Development of Terminal Interface

The main focus for the start of the sudoku solver needs to be the interface for displaying the sudoku. This is a necessary starting point as in order to demonstrate Artificial Intelligence solving the sudoku, the product needs to be able to display both the unsolved and solved solution in a way that is easily digested by the target audience. The borders, regions, numbers and void values need to be distinct and observable to those trying to understand what the

AI algorithms is trying to achieve, and if the algorithms are doing so correctly. That makes this element of the product integral to the development of my project.

The best approach to initially work on this feature would be to work on a console based interface. This would be effective as the first iteration of the sudoku interface as it only needs to display an easily readable representation of the sudoku puzzle. This was achievable in the console as the borders can be represented through the ’-’ character for row division and ’—’ character for column division, the numbers as just numbers and the void values as an ’*’ character. this gave a simple and easy to read sudoku display on the console.

Division of the grid into sub-grids in the terminal would work through calculating the size of the sub-grids which can be based upon the size of the inputted grid as in most cases the length of the sub-grid is the square root of the length of the grid, for example, a 9x9 grid would have 3x3 sub-grids or a 16x16 would have a 4x4 sub-grids. However, in some cases a sudoku can have sub-grids with unequal length rows and columns, such as a 6x6 grid which has 2x3 sub-grids or a 12x12 which has 3x4 sub-grids. For the solver to be able to adapt so that it can print and solve sudokus of different sizes the application would need to be able to work out the correct row and column length based of the grid size.

A way to approach the interface algorithm is to calculate the square root of the grid length, and storing this value as a row and column divisor, however in the case where the square root is not a whole number, the algorithm rounds the value down for the row divisor and rounds it down for the column divisor but adds 1, which in most cases is enough to find the correct sub-grid lengths, as in the case of a 6x6 grid, the algorithm would calculate the root to 2.449... and round it down to 2 for the row divisor and set the value to 3 for the column divisor, creating the 2x3 sub-grids necessary for displaying the puzzle.

The divisors would be used by the algorithm to print a series of ’-’ characters for every row which is fully divisible by the row divisor, which separates the rows in the sub-grids. And for the columns, the algorithm would print the ’—’ character in the column index which is fully divisible by the column divisor. With this algorithm, the application is capable of taking in sudokus in the form of a 2 dimensional array and displaying a user friendly way of observing both an unsolved and solved sudoku problem.

A look at the final sudoku printing algorithm:

```
def print_grid(self, grid):
if(grid == None):
    print("No found solution!")
    return
divisor = grid.__len__() ** 0.5
if(divisor%1!=0):
    row_divisor = round(divisor)
    column_divisor = round(divisor)+1
else: row_divisor, column_divisor = divisor, divisor
# logic for dividing sudoku grid into regions depending on sudoku length
for i in range(grid.__len__()):
    if i % row_divisor == 0:
        print(" - "*grid.__len__())
        # uses dividing value to place row borders
    row=""
    for j in range(grid[i].__len__()):
        if(j % column_divisor ==0):
            row+= " |"
            # uses dividing value to place column borders
            row+=" "+str(grid[i][j]) if grid[i][j]!=0 else " *"
            # adds number or * to be printed for final sudoku output
        print(row+" |")
    print(" - "*grid.__len__())
    # printing the numbers of grid list or an * where there is a 0
    # and the borders to display the sudoku in the console to the user
```

2.3 Development of Backtracking Algorithm

2.3.1 What is Backtracking?

Backtracking is a main method of creating an AI algorithm to solve a constraint satisfaction problem. Constraint satisfaction problems are problems in which there are a finite set of variables that contain a finite domain of potential values which can be assigned to form a solution. These values will need to satisfy all the constraints of the problem in order for it to be a correct answer. Examples of CSPs are scheduling, puzzle solving such as sudokus and even resource allocation. Backtracking is an example of a complete algorithm, in which it can guarantee the if a solution does or does not exist for the constraint satisfactory problem [12].

Implementing a backtracking algorithm to solve a CSP is done through recursion and may even use a decision tree in order to represent the different potential values for the variable set of the CSP [13] The algorithm would generally work through choosing a value for a variable in the set, validating its choice, and if at any point a valid value for a variable can not be found, it will backtrack to a point were alternative decisions are available and have not all be exhausted and a different path to a potential solution is taken. This is done repeatedly until a solution is found, or there are no longer any unexplored values (all decisions have been exhausted completely) meaning that there is no solution to the constraint satisfaction problem [14].

2.3.2 Implementation of Backtracking Algorithm

A backtracking algorithm works similar to how a human would use backtracking as once a human discovers a mistake, an approach to fixing it would be to go back on their working out and find the cause of fault which gives them the undesirable output. The algorithms would work by making its own attempt to solve a problem. so in the case of a sudoku it would attempt to fill in blank squares with a potential valid value, and if a problem occurs in its solution later on, such as the next blank square not being able to find a single valid value, then the algorithm would need to progress back to the cause of the problem, which would be the incorrect value in the previous square, or the ones before until it finds a different valid value to progress. The backtracking algorithm will then continue its attempt with a different solution for the sudoku, and this is done repeatedly till a viable solution is formed.

When developing the solving algorithm, first came the crucial element of finding a blank square in the grid, this would be represented by a 0 in the sudoku grid's 2 dimensional array. the coordinates on the grid would be returned to the main algorithm when found, then using these, the main loop will attempt to fill the square with numbers ranging from 1 to the max value, which is also the length of the grid (usually 9), until a viable value for the square is found. In order to check the validity of a value, the algorithm would exhaustively search through the row, column and sub-grid (checking every value in that area) for the value that is equal to the value about to be inputted into the blank square if the value is not found in any of the specified areas then it is considered valid, if the same number is found, then the current value is invalid and the algorithm attempts to check the next value for validity. If a viable value cannot be found, the algorithm clears the current square and backtracks to the previous square, and checks the next values for viability, if there are no viable values, the algorithm will clear the current square and backtrack once again to the square before and once again try different values until a different viable value is found and the algorithm searches for the next blank value and the algorithm repeats itself, until there is no more blank squares, or no more values to try on the grid that gives a solution to the problem.

A look at the main Backtracking algorithm:

```
def backtracking(grid, sudoku):
  position = empty_space(grid)
  # locates an empty square on the grid and stores its location
  if position == None:
      return grid
      # if there are no empty squares, the solution has been
      # found and the grid is returned
  for i in range(1, grid.__len__()+1):
      sudoku.update(i, position[0], position[1])
      # updates the GUI to display the current attempt for a viable value
      if safe_value(grid, position, i):
          grid[position[0]][position[1]] = i
          # test a viable number in the open position
          backtracking(grid, sudoku)
          # checks the next space after the previous viable solution
          if(empty_space(grid)==None):
              return grid
              # if the are no empty positions, this solution is correct
      grid[position[0]][position[1]] = 0
      # if the current value is not viable, it is set to 0, but the i remains
      # as the none viable solution to be incremented
      sudoku.update(0, position[0], position[1])
      # updates the GUI to clear the current square
  return None
  # if there is no number that goes into the empty position,
  # an empty grid is returned as there is no solution
```

2.4 Development of Graphical User Interface

For the development of the graphical user interface to display the sudoku and the solving algorithm, pygame was a powerful python library to leverage in the programming of the GUI as it provides an adaptable environment in which the sudoku grid can be drawn on the window and updated with little difficulty while still being able to display a way of visualising the backtracking algorithm. When creating the grid on the window it was best to give it the length 340 which was big enough for the user to see the sudoku and the values in each square clearly. When dividing the grid into sub-grids, the same root and rounded divisor logic used in the console print function, however in the case of the GUI the grid was divided visually using thicker lines to create the division between the whole and sub grids. Then after the grid has been drawn in the window, the algorithm starts adding values to square by placing a new square over the empty square with the intended value.

It is also necessary to be able to clear the squares within the grid, this is so that the sudoku solver will be able to show its iterations of the solution. In the updating squares method, the squares cleared through overwriting the square by filling the surface with a white square then updating the value to a new square to cover the cleared square. this function would need to be called every time the value changed in the algorithm in order to give the visual backtracking effect.

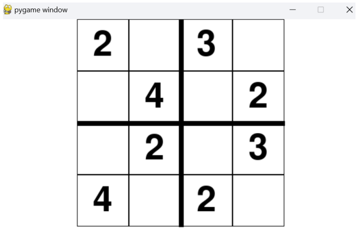
By introducing a GUI to display the solving algorithm, the solver is able to show the step by step solution being formed through mistake adaptation approach my backtracking algorithm takes as well as showing how quickly the computer can calculate this as the GUI updates the values very rapidly, demonstrating how much faster a computer is compared to a human, despite the GUI technically wasting computer processing speed, so it is faster to print the output directly then updating a GUI, but then the algorithm will not be demonstrated in an easily comprehensible manner when being displayed in that manner.

The final iteration of the Graphical User Interface of the application now also incorporates additional sections which are placed around the main grid, which has been adjusted to better center the application as it is the main and primary focus of the project, which displays a guide on the inputs the application uses to allow for the user to interact with the program, giving access to features such as switching between the different solution algorithms and the unseen sudoku problems to watch the application solve with. However, due to the slow nature of the development of the Convolutional Neural Network, it has only been optimised to solve 9x9 sudoku grids, which means the adaptive nature of the grid's rendering algorithm is not fully utilised in the final production application, although this will not compromise the applications ability to provide an insight into how well and efficiently an algorithm can solve the mathematical sudoku problem.

The main loop for the displaying the grid in the GUI

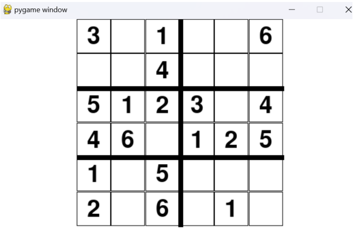
```
for row in range(grid.__len__()):
          self.board.append([])
          # adds a array inside another array, representing rows on the board
          if(row%row_divisor==0 and row != 0):
              pygame.draw.line(self.screen, (0, 0, 0),
              (col_offset, row*square_size),
              (col_offset+(grid.__len__()*square_size), row*square_size), 8)
              # where the board is divided based on divisor,
              # thicker lines are used to show the divisions that
              # make up the sub-grids
          for col in range(grid.__len__()):
              square = pygame.Rect(col_offset+(col*square_size),
              row*square_size, square_size, square_size)
              value = self.font.render("" if grid[row][col] == 0
              else str(grid[row][col]), True, (0, 0, 0))
              val_rect = value.get_rect()
              val_rect.center = square.center
              self.screen.blit(value, val_rect)
              self.board[row].append(square)
              pygame.draw.rect(self.screen, (0, 0, 0), square, 1)
              # the squares are created within the boundaries of the grid,
              # and smaller rectangles with the preset
              # puzzle values are displayed within the squares where their
              # coordinates are found from the input array
```

```
if(col%column_divisor==0 and col != 0):
    pygame.draw.line(self.screen, (0, 0, 0),
    (col_offset+(col*square_size), row*square_size),
    (col_offset+(col*square_size), grid.__len__()*square_size), 8)
# where the board is divided based on divisor, thicker lines are used
# to show the divisions that make up the sub-grids
```



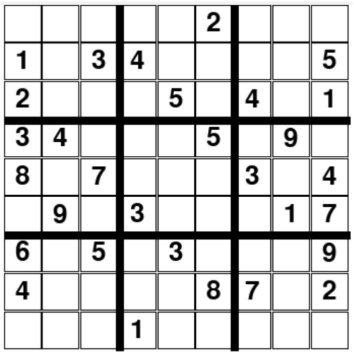
2		3	
	4		2
	2		3
4		2	

Figure 2.1: 2x2 grid



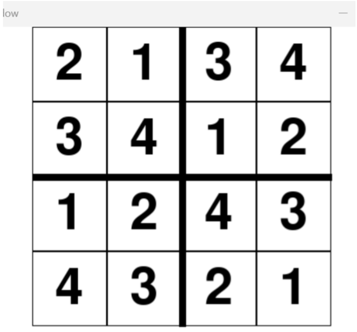
3		1			6
		4			
5	1	2	3		4
4	6		1	2	5
1		5			
2		6		1	

Figure 2.2: 6x6 grid



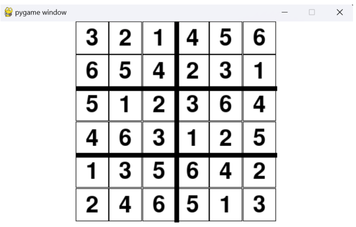
					2			
1		3	4					5
2				5		4		1
3	4				5		9	
8		7				3		4
	9		3				1	7
6		5		3				9
4					8	7		2
			1					

Figure 2.3: 9x9 grid



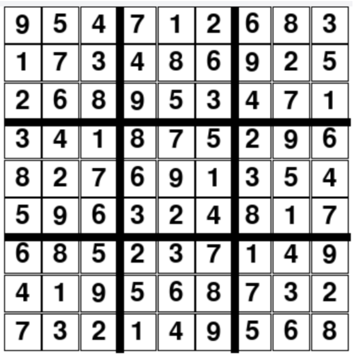
2	1	3	4
3	4	1	2
1	2	4	3
4	3	2	1

Figure 2.4: 2x2 grid



3	2	1	4	5	6
6	5	4	2	3	1
5	1	2	3	6	4
4	6	3	1	2	5
1	3	5	6	4	2
2	4	6	5	1	3

Figure 2.5: 6x6 grid



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

Figure 2.6: 9x9 grid

2.5 Development of the convolutional Neural Network

2.5.1 What is a Neural Network?

A neural network is based off of how the humans are capable of thinking through the use of a network of interconnected neurons. A neural network mimics the human brain's neuron connections through modelling an interconnected network of nodes which can receive information, process the data and output that data either to another layer in order to attempt a greater accuracy in its prediction through additional processing, or to the user as a final prediction output [15]. the model is capable of learning from its mistakes through the use of backpropagation and can also process information in parallel all of which can help the model to do things like pattern recognition, which can be applied to many different fields and problems which could potentially be the best means of finding a solution [16].

This Artificial Intelligence model is implemented by separating the interconnected network into layers. These layers are the input layer, the hidden layer (or layers) and the output layer [17] . The input layer is the primary layer that is used to take in all the inputs to be processed, hence its name. The hidden layers consist of the nodes that begin to process the inputted data, this is done through weights and biases associated with the connection between 2 nodes. These nodes are trained through passing in test data to help set and adjust the weights and biases of the connected nodes, by using a technique called backpropagation the algorithm can analyse predicted outputs and based on the output, make changes to weights and biases to update model and reduce its rate of error, allowing for growth and the ability to learn from its own mistakes [18] . The output layer is responsible for compiling the final prediction to be used as a solution to the problem it was provided to solve.

2.5.2 Convolutional Neural Networks

Convolutional Neural Networks are a specialised type of Neural Network which specialises in image classification, object detection, image segmentation, image generation and more, image classification being the identification of the category of the whole image, object detection being the detection of multiple objects within the image, image classification classifying each pixel in an image to a specific class and image generation works on creating images from scratch based on what it learned from training data. The reason why these types of tasks are better suited to a convolutional neural network is because of the CNN's structure which uses many different types of layers, but most notably and importantly, uses convolutional layers. The other types of layers that can help a convolutional neural network achieve what it does are the pooling layers, normalisation layers, dropout layers, dense layers and flattened layers. These layers will be explained in more depth below:

Conv2D

Conv2d is the key part of a convolutional neural network, as this is the convolutional layer that serves as the basis of the network. The purpose of this layer is to extract features. To do this the layer makes use of filters called kernels to produce feature maps where each neuron on the next layer, calculated as the dot product of the area the filter covers on the inputted data, is the output of the kernel and the combination of this neurons hold values that represent specific features, which aids in pattern recognition in the network. the use of filters allows for parameter sharing, where the same set of weights are applied throughout the input which is less computationally exhaustive for the machine and means less parameters are needed for the network to generalise better as filters cover multiple parts of the input as there is not a need for separate weights for each value in the inputted data due to the filters sharing these parameters in the feature maps.[19]

Batch Normalisation

Batch normalisation is a form of normalisation that takes place on the batches of data being inputted into this function before entering the next layer, it aims to standardise the inputs, by calculating the mean and variance of the data and normalises them while also including learnable scale and shift parameters which is to undo any normalisation if it begins to negatively impact the model performance. By passing normalised inputs to the layer, the process of training becomes more stable and efficient as the standardisation prevents exploding or vanishing gradients which can make gradients too large or too small which will hinder its ability to learn long term dependancies (vanishing gradients) or overflow due to being so large, impacting the learning capabilities of the model (exploding gradient). this allows for a higher rate of learning and quicker training processes; it also helps to prevent overfitting due to adding this bit of ”noise”, random variation, to the input of each layer which will stop the model from learning too specific to the training set, leaving itself unable to accurately classify new data it has not seen before.[20]

MaxPooling2D

MaxPooling2d is a version of Convolutional neural network pooling that works by reducing spatial dimensions of the input data in order to reduce complexity and memory consumption in computations. it does this by keeping the maximum value from the pooling window, which helps to highlight prominent features that has the effect of improving generalisation and reduce overfitting as the reduced spatial dimensions of the convolutional layers feature map decreases the amount of parameters that are being passed in later layers, which in turn makes it unlikely for the model to over learn and only memorise the training set, which will make the model perform poorly and inaccurately when being faced with new data. Other forms of pooling are min pooling and average pooling, which works by keeping the minimum value or average value from the pooling window, instead of the max value but operating all the same elsewise, which provides an effect similar to MaxPooling on the convolutional neural network.[19]

Dropout

The dropout layer hopes to achieve a reduction in overfitting by randomly dropping out neurons (resetting to 0) so that the network is forced to learn redundant representations of the data, also allowing for better generalisation by preventing network from being too specialised to the training data. dropout can also aid in reducing co-adaptation, which is where neurons depend on other specific neurons as the deactivation of randomised neurons will break down the dependence in later iterations of the training phase, this means single neurons can learn more robust features on their own without relying on the input of other connected neurons, which is necessary for the model to be able to produce accurate predictions and classifications when being presented with unseen data.[19]

Flatten Layer

The flattening layer of the convolutional neural network is important for the final stages of the network producing its predicted output as this layer converts the multi dimensional data outputted by the neurons in the previous layer into a singular dimensional output (vector) which is what is needed as input for the dense layers of the CNN so that they can process and learn from the test data, as the extracted features will be merged into one layer, and begin to start any classification tasks, that will distribute probabilities to the different classes, which is required from the model in order to produce a prediction.[19]

Dense Layer

Dense layers is important as they are fully connected layers where each neuron in this layer is connected to every neuron in the previous layer, usually used after all other convolutional, normalising, pooling and flattening layers, so this layer is where all the features are extracted to and therefore every input now and from all previous layers has an impact of every output from this layer. The connection of all these layers to compile all the features is what gives the model the ability to learn the patterns and complex relations of the training data and with that experience with an activation layer (usually SoftMax), predict the final output which may be the classification probabilities which will give the chances of what the input should be classed as.[19]

Types of Activation

In addition to layers in a convolutional neural network, another key factor in the network to help with making accurate classifications and predictions are the activation functions. activation functions are the function in which the neurons in the network uses to learn and make decisions as it determines which neurons should activate and pass data to other layers to form the final prediction. they are useful because they allow for the ability to solve more complex problems as without them, the model would learn like a linear model which is simpler to implement and computationally more efficient, but not very good at solving non linear problems, problems which have multiple variables with complex interactions and potentially many dependencies. The following types of activations that can be used to aid the learning of a convolutional neural network are:

ReLU

The Rectified Linear Unit is an activation function which works by taking in the input x and setting the output to x if positive (output is same as input) and if x is negative ($x <= 0$) the output of the output is 0, this is a simplistic activation which is great for computational efficiency as it reduces unnecessary computations which in turn helps the convolutional neural network concentrate on extracting key features. However, this activation is known for causing the dying ReLU or dying neuron problem were neurons set to 0 are inactive consistently through training iterations and as a result are not learning for some iterations where they are unactive and as a result they do not end up contributing much to the later layers.[19]

Leaky ReLU

Leaky ReLU is an activation variant of the ReLU activation in which it attempts to address the dying ReLU problem. The Leaky ReLU works by allowing for a very small non zero values instead of negative values, which is calculated by multiplying the negative value input with a very small constant (for example 0.01) if the input is $x <= 0$ and just outputting the input x if the input is positive, which is why it is a variant of ReLU as it treats positive values the same way but it lets negative values “leak” into the network through positive constants. this prevents neurons from being repeatedly deactivated in the network as the activation function avoids the output of the neurons being set to 0, which allows for additional contributions from these neurons when trying to learn complex features and patterns in depth.[21]

ELU

Exponential Linear Unit is another type of activation that also aims to prevent the dying ReLU problem. ELU activation does this by running an exponential function calculation on inputs x that is $x <= 0$, this allows for the output to avoid being set to zero, as it decays overtime when the input is 0 (is set closer to 0 but not actually 0), so the dying neuron problem is avoided and the learning phase of the convolutional neural network is able to have the input of more neurons at later layers, which will result in more supporting data to better compile extracted features and produce a more accurate classification prediction. However, due to the inclusion of the exponential function, additional computational costs are needed to calculate the decay of all the neurons gradients and therefore requires a much longer training phase then opting for a different activation function. otherwise, for positive inputs, the output remains the same as the input, making this activation another variation of ReLU as it treats positive values in the same manner as its predecessor.[21]

SoftMax

SoftMax is often a final layer, the output layer, activation function, and is used for classification prediction of multiple classes. This activation function does this by converting the inputted data into probabilities by using a formula to calculate these values from every input into an output, and ensures that all output probability is normalised so that all probabilities add up to 1, so that it is perfect for classification as the SoftMax activation ensures all probabilities is appropriately distributed over classes. Using this the class with the highest probable value is the class that the convolutional neural network deems to classify the input with, as this is what it has been taught through being trained on with the training set.[19]

Sigmoid

Sigmoid is another activation function which makes use of a well known mathematical function used in statistics and machine learning, and in the case of convolutional neural networks it is used by the network to transform the input of a singular neuron into a value between 0 and 1, which acts as the probability of between which 2 class the input should be classified, the highest value being the highest probability of the input being in that class. This makes this activation function very important for binary classification tasks in a convolutional neural network where the input needs to fit between 2 different classification classes. Or this activation can be used in instances where you would want the probabilities of each neuron in a specific layer in as the predictions for some of the classes are not mutually exclusive, meaning the probabilities of some of the other neurons may not be entirely dependent on each other, which would make this useful for multi label classification, where you class and label the different entities from the inputted data.[19]

Model Compilation

The step of compilation in developing a convolutional neural network is integral to how well the model will begin to learn and develop at a faster rate while maintaining a higher degree of accuracy as this is the part of development that manages how the designed model will carry out its training phase in terms of configuring how the model measures accuracy and the errors in its predictions during training, and how much the model adjusts its weights in an attempt to reduce error during training to optimise the model. This also prints the metrics that it evaluated the model with so that its efficiency and effectiveness can be monitored and the model changed if the results are not up a suitable standard, which can help to develop the best model that provides the most accurate predictions for the task it is needed to complete. There are many key components that get passed into the compilation phase that are, the loss function, the optimiser, and the metric type, an explanation of these are found in the paragraphs below.

Network Optimiser

The network optimizer is a vital element of the compilation phase of the convolutional neural network as this parameter is used to reduce the loss (errors) of the network while improving the accuracy of the predictions in its training, which will serve to improve the accuracy of actual predictions of the model outside of its training phase. The optimiser does this through the use of a technique known as backpropagation which is used to update network parameters based on the calculations of the gradients of neurons by computing just how much the weights of the neurons lead to the error in the training prediction and uses this to help adjust the weights of neurons and the biases for the next iteration in training , and this is repeated continuously till training is completed in order to try and attempt to reduce the mistakes made within the neural network while it is making its predictions on the training data, which in turn helps the neural network to efficiently and effectively learn the dataset. There are many different types of optimisers such as, gradient descent, SGD (Stochastic Gradient Descent), RMSprop (Root Mean Squared Propagation), Adam (Adaptive Moment Estimation), each of these optimisers are built of the usual gradient descent, but uses different formulas and calculations in order to better adjust the gradients and weights of the neurons in the convolutional neural network, in the hopes of creating more accurate predictions with the developed model.[25]

Learning Rate

When using an optimiser, the learning rate can affect the accuracy of the model during its learning phase, it is important as this is a parameter in which dictates how much the model's weights adapt when learning. The size in which this parameter value is set can affect the speeds of convergence in the convolutional neural network, where the network training is stable and does not make serious changes in its weights and parameters in changes as the model has finished or begun to plateau with its training phase, and as a result also reduce the training time, as well as how precise the updates in the weights which can improve the accuracy as well as keeping the rate of error (loss) to a minimum. These changes are dictated by how large or how small the parameter is set, as if the learning rate is set too high then the steps in which the optimiser takes can end up making too large of a change to the networks weights and as a result miss the minimum value of loss in the loss function, which can make the training less stable, however with a higher value, the time taken in the training phase is greatly reduced. If the learning rate is too low, then the speed in which the model will reach optimal convergence will be hindered due to the smaller adjustments taking longer to compute, however this can lead to a more stable convolutional neural network which may predict more accurate predictions.[26]

Loss Function

This function is important for the training of the model, as this function is responsible for measuring up and calculating how the accurate of the model predictions are in relation to the target data values, in networks training cycles. The loss function is needed to also work in conjunction with the network optimiser in order to aid in the adjustment of weights and biases by providing the optimiser with these measurements so that the optimiser can use them to try and limit the loss which helps the model to learn more efficiently and predict more accurately. There are several types of loss functions which include, MSE (Mean Squared Error), MAE (Mean Absolute Error) which are both more suited to regression tasks, Binary Cross Entropy, Categorical Cross Entropy which are both more suited to classification tasks. Choosing the right type of loss function is important as this function will play a key role in the learning process when working with the optimiser.[24]

Model Training

How the training data and prepared and the rate of which the data is passed in to be used for training, along with the amount of time the model cycles through the entire dataset and the amount of data kept aside for validation testing, are all things needed to be considered when training a convolutional neural network as this will have an impact on how well the network will learn and help the model to reach convergence with a higher accuracy and a lower rate of error. Setting these parameters to guide how the convolutional network needs to train on the data is an integral part of the convolutional neural network's learning process due to parameters like the number of epochs, the batch sizes and any callbacks the network may use to smooth out the training phase. These parameters and their impact on the training and accuracy of the convolutional neural network are explained below.

Epochs

Each time a model trains through an entire training set, this is referred to as an epoch. the number of epochs refers to the number of times the model will cycle through the same entire training set. having multiple epochs are important as this allows for the model to better understand the data as one train through is not typically enough for the model to learn to effectively and accurately predict. However, there can be consequences for having too many or too few of these epochs in the convolutional neural network then this will impact the how effective and efficient the training of the model is. In the case where there are too many epochs, there is a chance for the model to learn the specific training set and as a result will struggle to predict new examples which means it is overfitting. On the other hand, too few epochs can lead to the model not being able to extract the more complex patterns and features of the training data due to not having enough time with the data to gain better accuracy, therefore the network will not only struggle with predicting with the validation set, but also with the training set, which will result in the model's accuracy being very poor, and the network will not be able to make any good predictions on new data inputs.[22]

Batch Size

The number of samples the network takes from the training set and trains with in one forward and backward pass, making changes to the weights and biases of an neurons that needed adjustments, this is what it is referred to as a batch size. smaller batch sizes can impact the convolutional neural network as it leads to more frequent updates to the model which can help to make better predictions as well as gaining a better, more accurate and stable convergence. A lower batch count could also be better for memory efficiency as less training samples will be simultaneously processed, however due to the repeated updates this will end up consuming more computational resources and will slow down how long it takes to get through all the data in the training set. Larger batches on the other hand, despite the greater computational demand, can train the model much faster, but will have less updates to the network parameters which can make the stable more accurate convergence less likely to take place.[22]

Callbacks

Callbacks can be helpful in the training phase of the model as it can provide a range of functionalities that are used to have better and more manageable control over the training phase of a convolutional neural network. It accomplishes this through running specific tasks between batches or epochs in order to make the model more efficient and effective during its time learning the training data. Callbacks achieve this effect such as improving performance through early stoppage if validation loss stops decreasing (to avoid overfitting) and saving checkpoints for the model while training on the set so that at the end it can restore the point with the best learnt parameters or changing the learning rate based on measured variable or a set interval of batches or epochs. these functionalities are great for adaptational learning for a convolutional neural network as it allows for parameters of training to be adjusted while receiving feedback of how the training is going, which can result in a more robust and precise model capable of predicting reliable outputs, that's training computational cost will scale with the convolutional neural networks rate of learning.[23]

2.5.3 Implementation of the Convolutional Neural Network

The convolutional Neural Network will need to be trained on taking in sudoku solutions as training data, this is so that it will learn what a solution to a sudoku should look like and it should recognise the patterns and rules of a sudoku so that when it is faced against an incomplete sudoku, it will be able to provide an accurate prediction of a solution, that will hopefully be the correct answer to the inputted problem.

The planned structure of the convolutional neural network model would take in the 81 numbers of a sudoku solution in a 9x9 grid format, the model will then need to output a 9x9 grid with the correctly predicted solutions to the missing numbers that it should have learnt to solve from the networks training phase.

The model will need to have a set of convolutional layers in order to attempt to extract the features of a sudoku, such as the rules and constraints the numbers in the solution needs to abide by, so that the convolutional neural network will be able to learn the patterns and apply what it understands from these features to mimic what a solution should look like more accurately when the network is supplied with a new unseen sudoku puzzle. The model will have a typical CNN design where it has repeated convolutional layers in conjunction with MaxPooling layers to highlight the key patterns within a sudoku, as well as batch normalisation and dropout functions in an attempt to maintain a stable training phase that will hopefully not overfit to the data it is being trained with, as the network will be training on a dataset of 1,000,000 sudoku puzzles and their respective solutions.

The output of the convolutional neural network will be a grid of normalised values that will need to be de-normalised into values that will give an actual solution to the sudoku problem. In order to extract the features from the network, necessary for the model to calculate an accurate prediction for the solution, the CNN will need a dense layer which will collect all the features and patterns from the previous layers, and then a final dense layer that will compile all that the model has learnt to hopefully produce an accurate prediction on what the solution for the newly inputted sudoku puzzle should be.

2.5.4 Model 1

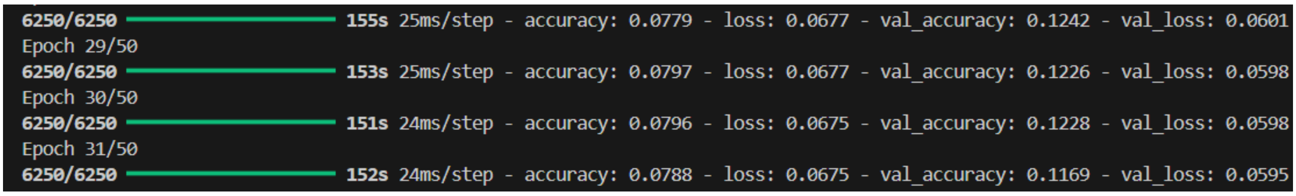
The initial design for the convolutional neural network is as follows

```
    model = Sequential([
        Input(shape=(9, 9, 1)), # takes in a 9x9 grid with only 1 channel
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Dropout(0.4),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Dropout(0.4),
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Dropout(0.4),
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        Dropout(0.4),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(81, activation='linear')  # Output layer with 81 neurons
    ])


model.compile(optimizer=Adam(learning_rate=0.0001),
 loss=MeanSquaredError(), metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
 factor=0.1, patience=3, min_lr=1e-6, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss',
 patience=5, verbose=1, restore_best_weights=True)
# callbacks to try and reduce the probability of the CNN losing accuracy

model.fit(puzzles, solutions, epochs=50, batch_size=128,
validation_split=0.2, callbacks=[reduce_lr, early_stopping])
```

The convolutional neural network makes use of various Conv2D layers, which increases in size from 2 layers that produces 128 filter maps that use a 3x3 filter size and an additional 2 that produces 256 filter maps that use a 3x3 filter size, so that the model can attempt to further draw out any important, key features and patterns that it can potentially find in the training data set. The use of MaxPooling2D that uses a pooling window with 2x2 dimensions, in order to attempt to focus the network onto these features of the sudoku puzzle by reducing unnecessary data and computations, these layers with the addition of the normalisation techniques, such as the dropout of 0.4, which would ignore 40% of neurons to reduce dependencies, would lead to the assumption that the model would work well to extract the rules and constraints of the sudoku puzzle, but as can be seen below, the model is not training very effectively.



```
6250/6250 ━━━━━━━━━━━━━━━━━━━━ 155s 25ms/step - accuracy: 0.0779 - loss: 0.0677 - val_accuracy: 0.1242 - val_loss: 0.0601
Epoch 29/50
6250/6250 ━━━━━━━━━━━━━━━━━━━━ 153s 25ms/step - accuracy: 0.0797 - loss: 0.0677 - val_accuracy: 0.1226 - val_loss: 0.0598
Epoch 30/50
6250/6250 ━━━━━━━━━━━━━━━━━━━━ 151s 24ms/step - accuracy: 0.0796 - loss: 0.0675 - val_accuracy: 0.1228 - val_loss: 0.0598
Epoch 31/50
6250/6250 ━━━━━━━━━━━━━━━━━━━━ 152s 24ms/step - accuracy: 0.0788 - loss: 0.0675 - val_accuracy: 0.1169 - val_loss: 0.0595
```

Figure 2.7: Evaluated Metrics of 1st Model

The measured metrics for this model is not looking very good. It uses the mean squared error loss function to calculate the difference between the predicted value and the actual value the convolutional neural network was supposed to produce. with this it can be seen that the accuracy of this method is not very good as it is disastrously low along with the validation scores, despite the loss and validation loss looking deceivingly low. The model begins to fall off at epoch 30, where the accuracy seems to be falling below the previous, although the validation accuracy still seems to slightly increase, until the next epoch, where we see both the accuracy and the validation accuracy drop below the previous epoch. This seems to show an extreme level of underfitting in my model due to the numerous methods used to try and reduce overfitting, which has hindered the model from learning effectively at all.

2.5.5 Model 2

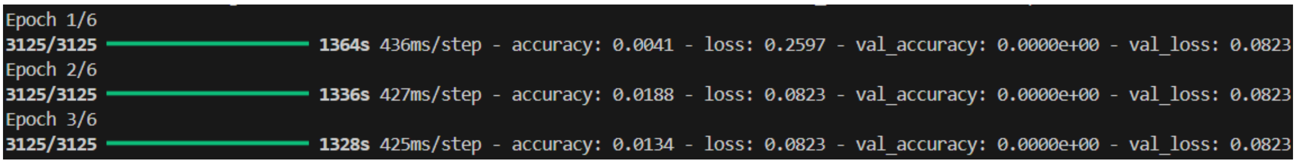
Due to the results of the displayed metric evaluation to the previous model, much needed restructuring done to the model can be seen below.

```
model = Sequential([
    Input(shape=(9, 9, 1)), # takes in a 9x9 grid with only 1 channel
    Conv2D(162, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(162, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(162, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(162, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(162, (3, 3), activation='relu', padding='same'),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.1),
    Dense(81, activation='linear')  # Output layer with 81 neurons
])


model.compile(optimizer=Adam(learning_rate=0.001),
loss=MeanSquaredError(), metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
factor=0.1, patience=3, min_lr=1e-6, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss',
patience=5, verbose=1, restore_best_weights=True)
# callbacks to try and reduce the probability of the CNN losing accuracy

model.fit(puzzles, solutions, epochs=6, batch_size=256,
validation_split=0.2, callbacks=[reduce_lr, early_stopping])
```

In an attempt to reduce the huge underfitting that can be seen in the previous model, it would seem to be effective to remove a significant amount of methods and techniques deployed to reduce overfitting, as the number and sizes of impact of some of these functions seems to be overpowering the convolutional neural networks ability to efficiently learn from the massive dataset provided to it. The model has 5 convolutional layers all producing 162 filter maps, as this should produce a more consistent level of feature extraction. However, due to the nature of sudoku puzzles, it seems very necessary to remove any type of pooling from the network as all of the values in the puzzle contributes to the solution and these values can help to reveal to the model the actual rules and constraints of the problem it is being provided with. This along with the reductions to dropout functions in the network should lead to a better learning environment for the convolutional network and in turn produce better accuracy and validation accuracy scores despite the higher computational costs, that forces a reduction in the number of epochs the model can use to train on the data.



```
Epoch 1/6
3125/3125 ━━━━━━━━━━━━━━━━━━━━ 1364s 436ms/step - accuracy: 0.0041 - loss: 0.2597 - val_accuracy: 0.0000e+00 - val_loss: 0.0823
Epoch 2/6
3125/3125 ━━━━━━━━━━━━━━━━━━━━ 1336s 427ms/step - accuracy: 0.0188 - loss: 0.0823 - val_accuracy: 0.0000e+00 - val_loss: 0.0823
Epoch 3/6
3125/3125 ━━━━━━━━━━━━━━━━━━━━ 1328s 425ms/step - accuracy: 0.0134 - loss: 0.0823 - val_accuracy: 0.0000e+00 - val_loss: 0.0823
```

Figure 2.8: Evaluated Metrics of 2nd Model

Unfortunately, the updated model was unable to overcome the underfitting issues as in the increased amount of time it has to get through the dataset in an epoch, the accuracy seems to increase, but in the next epochs the accuracy and validation accuracy seems to begin to decrease, while the loss and validation loss begins to plateau which is not a good sign for the convolutional neural networks ability to learn and adapt to sudoku problems and provide suitable solutions. To counter the issues that can be observed from the metrics evaluation, additional research will need to be conducted on how to better rework a convolutional neural network model to better learn and attempt to solve such a problem.

2.5.6 Model 3

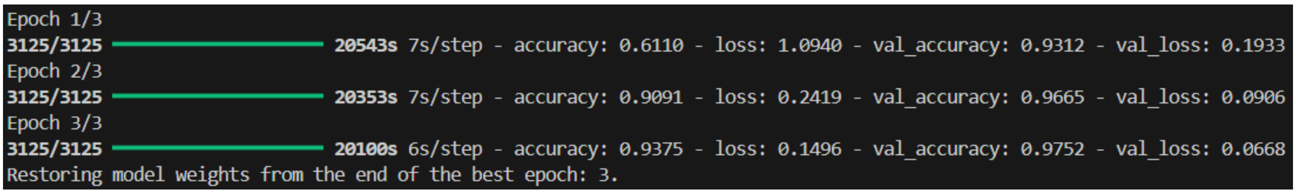
After a considerable amount of research, a study containing a solution of the problem being worked on was found and some of the solution found in the study was applied to the model used in the application to attempt to create a convolutional neural network that will provide a suitable prediction to a provided sudoku problem.

```
model = Sequential([
    Input(shape=(9, 9, 1)),
    Conv2D(512, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(512, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(512, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(512, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(512, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(512, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    Conv2D(512, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        BatchNormalization(),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Flatten(),
        Dropout(0.1),
        Dense((9*9)*9),
        # denses the neurons output into 81 (no of cells in solution)
        # ranges of values between 1-9 (constraint of the solution that
        # cells are between 1 and 9)
        LayerNormalization(axis=-1),
        # normalisation to stabalise and reduce overfitting
        Reshape((9,9,9)),
        # reshapes the condensed layer into a set of 9 columns, 9 rows
        # and an array sized 9 for each cell for the column and rows
        Activation("softmax")
        # this ensures that all output probability is normalised so that
        # all probabilities add up to 1, so that it is perfect for
        # classification
    ])


    model.compile(optimizer=Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy', metrics=['accuracy'])
   reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
    patience=3, min_lr=1e-6, verbose=1)
   early_stopping = EarlyStopping(monitor='val_loss', patience=5,
    verbose=1, restore_best_weights=True)
   # callbacks to try and reduce the probability of the CNN losing accuracy

   model.fit(puzzles, solutions, epochs=3, batch_size=256, validation_split=0.2,
    callbacks=[reduce_lr, early_stopping])
```

Using the style of the 11th convolutional neural network model [27] to create a model for project application, the network was designed with 9 different convolutional layers that output 512 filter maps, all using a 3x3 filter size, but where it differs significantly from previous models in this report, the network outputs a very large matrix where each of the 81 cells in the sudoku problem is represented as their own individual classification matrix which contains values that represent how likely the cells solution will be the number that is the index containing the highest probability. And the network does this by condensing all the neurons into a matrix with the length 729 (matrix sized at 9 for each 81 cells, 9*81 = 729) reshaping the matrix and applying the softmax activation, which distributes probabilities within the matrix so that they all add to 1, therefore only one value in the matrix will be the index most probable to be the cell solution. This multi classification type output from the model can be better measured up against a categorical cross entropy loss function and therefore will be able to provide a much better accuracy and validation accuracy score, while simultaneously keeping the loss and validation loss low. However, due to the size of this network, it would take far too long to run the 5 epochs with the 64 sized batches as seen in the solution from the study, so adaptation to the model fitting has been made to use 3 epochs and a batch sizes of 256 in an attempt to reduce the computational costs of training.



```
Epoch 1/3
3125/3125 ━━━━━━━━━━━━━━━━━━━━ 20543s 7s/step - accuracy: 0.6110 - loss: 1.0940 - val_accuracy: 0.9312 - val_loss: 0.1933
Epoch 2/3
3125/3125 ━━━━━━━━━━━━━━━━━━━━ 20353s 7s/step - accuracy: 0.9091 - loss: 0.2419 - val_accuracy: 0.9665 - val_loss: 0.0906
Epoch 3/3
3125/3125 ━━━━━━━━━━━━━━━━━━━━ 20100s 6s/step - accuracy: 0.9375 - loss: 0.1496 - val_accuracy: 0.9752 - val_loss: 0.0668
Restoring model weights from the end of the best epoch: 3.
```

Figure 2.9: Evaluated Metrics of 3rd Model

Despite the significant improvements to the algorithm and very high accuracy and validation accuracy, along with low loss and validation loss, this model still produces some errors with its output, and to make any adjustments to the model to try and prevent this is far too computationally expensive as it takes 20543 seconds, which equates to around 5-6 hours per epoch which is way too long for most machines to spend on training, this makes any minor adjustments to the model's design infeasible to test and therefore near impossible to improve without spending entire days on training the network, so the next stages of development needs to focus on maintaining this level of accuracy while significantly reducing the computational costs per epoch, so that improvements to the model can be made to try and reduce errors this model produces.

2.5.7 Model 4

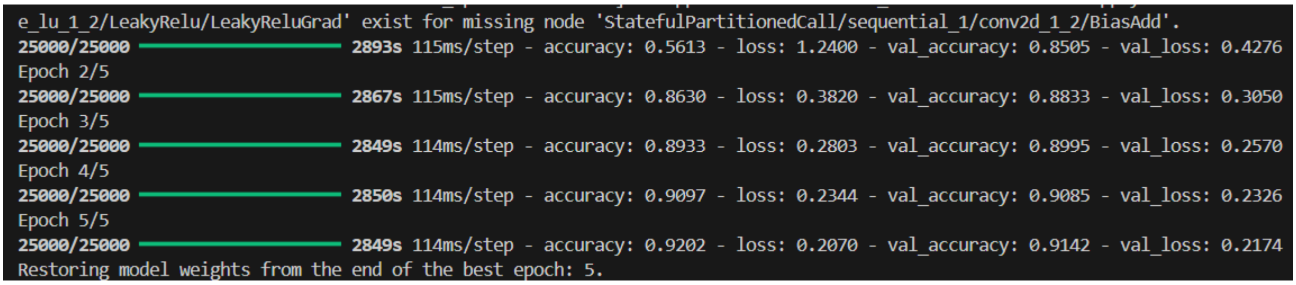
Following the discovery of an accurate yet computationally infeasible convolutional neural network, the following changes has been made to the projects model in order to try and significantly reduce the computational costs on the machine training the neural network.

```
model = Sequential([
    Input(shape=(9, 9, 1)), # takes in a 9x9 grid with only 1 channel
    Conv2D(324, (3, 3), padding='same'),
    LeakyReLU(negative_slope=0.1),
    BatchNormalization(),
    Conv2D(324, (3, 3), padding='same'),
    LeakyReLU(negative_slope=0.1),
    BatchNormalization(),
    Conv2D(324, (3, 3), padding='same'),
    LeakyReLU(negative_slope=0.1),
    Flatten(),
    Dense((9*9)*9),
    # denses the neurons output into 81 (no of cells in solution)
    # ranges of values between 1-9 (constraint of the solution that
    # cells are between 1 and 9)
    LayerNormalization(axis=-1),
    # normalisation to stabalise and reduce overfitting
    Reshape((9,9,9)),
    # reshapes the condensed layer into a set of 9 columns, 9 rows
    # and an array sized 9 for each cell for the column and rows
    Activation('softmax')
    # this ensures that all output probability is normalised so that
    # all probabilities add up to 1, so that it is perfect for
    # classification
])


model.compile(optimizer=Adam(learning_rate=0.0001),
     loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
     patience=3, min_lr=1e-6, verbose=1)
    early_stopping = EarlyStopping(monitor='val_loss', patience=5,
     verbose=1, restore_best_weights=True)
    # callbacks to try and reduce the probability of the CNN losing accuracy

    model.fit(puzzles, solutions, epochs=5, batch_size=32, validation_split=0.2,
     callbacks=[reduce_lr, early_stopping])
```

This iteration of the convolutional neural network attempts to reduce the computational costs of the training phase of the model by reducing the number of convolutional layers the network uses, while also reducing the amount of filter maps each layer produces. This seems like it would work out bad for the convolutional neural network as this would affect how well the network can extract and learn key features and patterns, so to try and offset the potential loss of accuracy while making so many significant reductions to model calculations, changes to the learning rate has been made, reduced from 0.001 to 0.0001 which should help make more precise updates to the network parameters (weights and biases for each neuron), as well as switching the convolutional layers activation from regular ReLU to leaky ReLU so that the network will still benefit from the contribution of neurons even if their inputs are negative, which in the case of ReLU would've set the neuron to 0, thus making it inactive and no contributions would be made from that neuron. with these changes the model should be able to run 5 epochs this time using batch sizes of 32, without it costing too much development time.



```
e_lu_1_2/LeakyRelu/LeakyReluGrad' exist for missing node 'StatefulPartitionedCall/sequential_1/conv2d_1_2/BiasAdd'.
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 2893s 115ms/step - accuracy: 0.5613 - loss: 1.2400 - val_accuracy: 0.8505 - val_loss: 0.4276
Epoch 2/5
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 2867s 115ms/step - accuracy: 0.8630 - loss: 0.3820 - val_accuracy: 0.8833 - val_loss: 0.3050
Epoch 3/5
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 2849s 114ms/step - accuracy: 0.8933 - loss: 0.2803 - val_accuracy: 0.8995 - val_loss: 0.2570
Epoch 4/5
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 2850s 114ms/step - accuracy: 0.9097 - loss: 0.2344 - val_accuracy: 0.9085 - val_loss: 0.2326
Epoch 5/5
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 2849s 114ms/step - accuracy: 0.9202 - loss: 0.2070 - val_accuracy: 0.9142 - val_loss: 0.2174
Restoring model weights from the end of the best epoch: 5.
```

Figure 2.10: Evaluated Metrics of 4th Model

With this model, a very high degree of accuracy and validation accuracy has been achieved. But this model is still less accurate than the previous convolutional neural network, which means it makes more mistakes when attempting to predict a solution to a sudoku problem. Despite the slight decrease in accuracy, this model is still considered a step in the right direction as for this slight decrease in accuracy, a vast improvement of training time has been seen as each epoch only takes around 2893s which is around 50 minutes, less than an hour when compared to the extreme 5-6 hours being dealt with in the previous network!!, which shows a very significant computational cost reduction.

2.5.8 Model 5

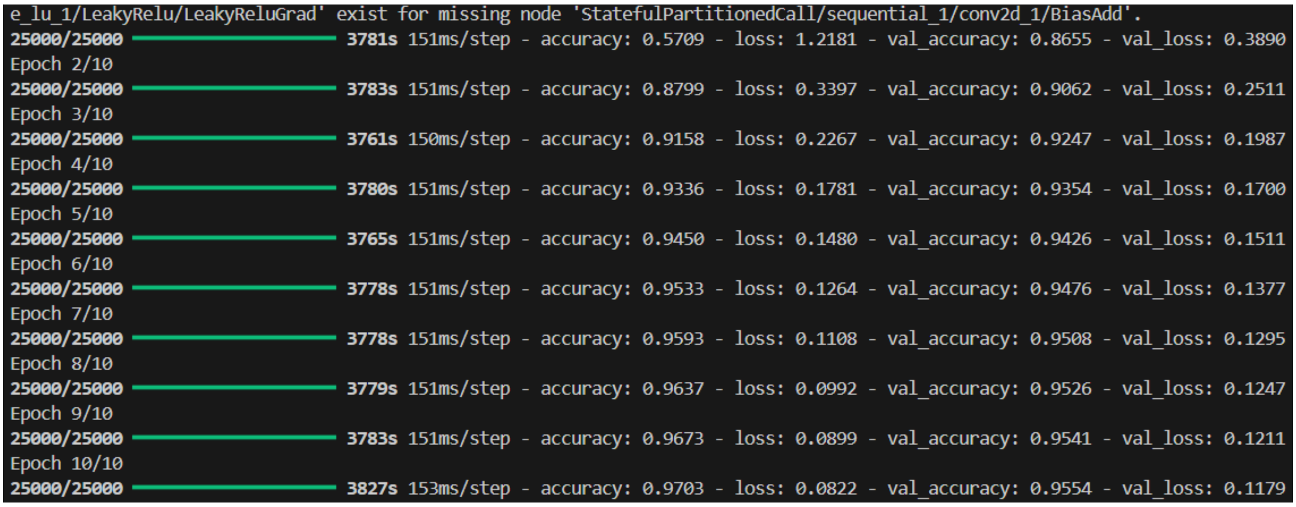
Following the success of its predecessor, this model has been adapted to try and keep its relatively low computational costs while making progress in increasing the accuracy in the convolutional neural network's predictions.

```
    model = Sequential([
    Input(shape=(9, 9, 1)), # takes in a 9x9 grid with only 1 channel
    Conv2D(162, (5, 5), padding='same'),
    LeakyReLU(negative_slope=0.1),
    BatchNormalization(),
    Conv2D(162, (5, 5), padding='same'),
    LeakyReLU(negative_slope=0.1),
    BatchNormalization(),
    Conv2D(162, (5, 5), padding='same'),
    LeakyReLU(negative_slope=0.1),
    BatchNormalization(),
    Conv2D(162, (5, 5), padding='same'),
    LeakyReLU(negative_slope=0.1),
    BatchNormalization(),
    Conv2D(162, (5, 5), padding='same'),
    LeakyReLU(negative_slope=0.1),
    Flatten(),
    Dense((9*9)*9),
    # denses the neurons output into 81 (no of cells in solution)
    # ranges of values between 1-9 (constraint of the solution that
    # cells are between 1 and 9)
    LayerNormalization(axis=-1),
    # normalisation to stabalise and reduce overfitting
    Reshape((9,9,9)),
    # reshapes the condensed layer into a set of 9 columns, 9 rows
    # and an array sized 9 for each cell for the column and rows
    Activation('softmax')
    # this ensures that all output probability is normalised so that
    # all probabilities add up to 1, so that it is perfect for
    # classification
])

model.compile(optimizer=Adam(learning_rate=0.0001),
 loss='sparse_categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1,
 patience=3, min_lr=1e-6, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
 verbose=1, restore_best_weights=True)
# callbacks to try and reduce the probability of the CNN losing accuracy

model.fit(puzzles, solutions, epochs=10, batch_size=32, validation_split=0.2,
 callbacks=[reduce_lr, early_stopping])
```

This final iteration of the convolutional neural network uses an increased 5 convolutional layers with a reduced 162 filter map output, but this time uses a 5x5 dimensional filter which will allow the network to capture more data in a larger space in context to the input, that can potentially result in detecting patterns on a bigger scale, which can have the positive affect of the model requiring less convolutional layers overall. This working in conjunction with the leaky ReLU activations should help to give higher accuracy and validation accuracies while keeping the loss and validation loss low while not increasing the training phase duration too significantly, because of the reduced amount of time the network needs per epoch, the number of epochs has been increased to 10 in an attempt to try and score the highest possible scores on the metric, so that the final convolutional neural network will hopefully make little to no mistakes when trying to make a prediction as to what it thinks the solution to the sudoku problem that is inputted into the model.



```
e_lu_1/LeakyRelu/LeakyReluGrad' exist for missing node 'StatefulPartitionedCall/sequential_1/conv2d_1/BiasAdd'.
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 3781s 151ms/step - accuracy: 0.5709 - loss: 1.2181 - val_accuracy: 0.8655 - val_loss: 0.3890
Epoch 2/10
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 3783s 151ms/step - accuracy: 0.8799 - loss: 0.3397 - val_accuracy: 0.9062 - val_loss: 0.2511
Epoch 3/10
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 3761s 150ms/step - accuracy: 0.9158 - loss: 0.2267 - val_accuracy: 0.9247 - val_loss: 0.1987
Epoch 4/10
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 3780s 151ms/step - accuracy: 0.9336 - loss: 0.1781 - val_accuracy: 0.9354 - val_loss: 0.1700
Epoch 5/10
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 3765s 151ms/step - accuracy: 0.9450 - loss: 0.1480 - val_accuracy: 0.9426 - val_loss: 0.1511
Epoch 6/10
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 3778s 151ms/step - accuracy: 0.9533 - loss: 0.1264 - val_accuracy: 0.9476 - val_loss: 0.1377
Epoch 7/10
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 3778s 151ms/step - accuracy: 0.9593 - loss: 0.1108 - val_accuracy: 0.9508 - val_loss: 0.1295
Epoch 8/10
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 3779s 151ms/step - accuracy: 0.9637 - loss: 0.0992 - val_accuracy: 0.9526 - val_loss: 0.1247
Epoch 9/10
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 3783s 151ms/step - accuracy: 0.9673 - loss: 0.0899 - val_accuracy: 0.9541 - val_loss: 0.1211
Epoch 10/10
25000/25000 ━━━━━━━━━━━━━━━━━━━━ 3827s 153ms/step - accuracy: 0.9703 - loss: 0.0822 - val_accuracy: 0.9554 - val_loss: 0.1179
```

Figure 2.11: Evaluated Metrics of final Model

This model has reached a very high scores in the metrics evaluation, showing a very high degree of accuracy, validation accuracy as well as lower levels of loss and validation loss. With this the convolutional neural network is able to calculate predictions to sudoku puzzles which contain very few errors, which makes it useful for my project as it can help to demonstrate just how effectively and efficiently machines can outperform humans at making fast and accurate calculations, which this convolutional neural network does fairly well despite its small degree of inaccuracy. However, with additional development time, the neural network could be further adapted to yield greater accuracy while potentially reducing computational costs per epoch, which is currently around 3827s which is roughly about an hour, through the use of better callback functions, which would be used to start off the training of model with a higher learning rate with larger batches which would speed up the speed it takes for the network to get through all the test data, but as the models accuracy begins to plateau or fall off, the call back would adjust these values so that in the later stages it will take longer and be more computationally expensive, but will allow for more precise adjustments to the convolutional neural networks parameters later, which could potentially give the same or greater results in much less time, as the earlier epochs are training much quicker.

Chapter 3: **Final Product**

3.1 Application Architecture

The project was divided in a way so that each file would have a specific purpose, a specific feature that would be needed to be used by the main file, in which the entire application is ran from. This organisation of the code base is beneficial as it allows for the program to be well documented, as the comments written in a file is relevant and specific to the feature being programmed. Also this architecture is good for the development of classes and features as they can be built upon and tested independently of one another which is necessary for keeping and maintaining good quality code for the AI sudoku solver, this also has the additional benefit of making feature updates and improvements easier to implement as making changes will only affect the current feature code and potential bugs will not be able to interfere with the progress being made on the other features of the main application.

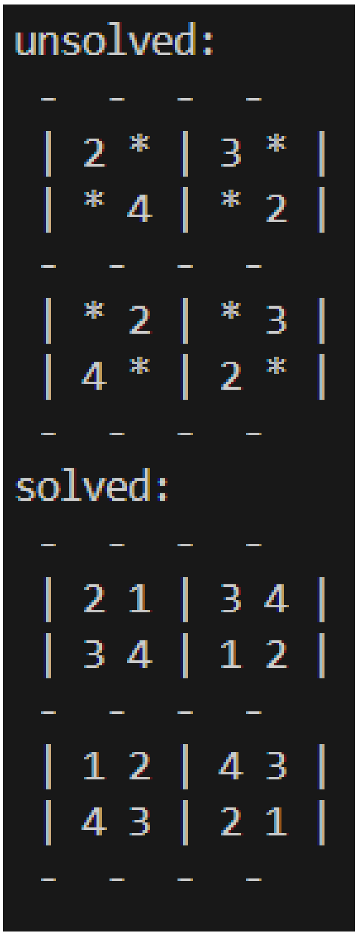
An agile type of methodology style was implemented in the development process of the AI sudoku solver as the primary focus of the term time development cycle was putting together rapidly developed features that is capable of doing just enough to get the application running and connecting them together to get a working prototype which is able to print and solve the puzzle in a way which is easily interpreted by the applications intended users. This style of development will be carried out in the next term of development also as it would be more important to focus on developing the key features of the application so that the prototype is able to meet the set aims and objectives of the project and only once these are achieved, improvements and updates can be made to the code base to improve the application incrementally until the final product has been formed.

3.2 Running the Application

Setting up the environment to run the application is very straightforward, it requires that the following python packages are installed in order to be able to start up the program, which are: the pygame package, which can be installed with the "pip install pygame" command from the system terminal, and the tensorflow python package, whcih is similarly installed with the command "pip install tensorflow" from the system terminal. Once these packages are installed the application will be very simple to run as it is located in the folder called "product", and the application uses the file named "main.py", and running this file will open up and begin running the application. the link to a demonstration of the application being run and a walk through of the features can be found here: Video Demonstration

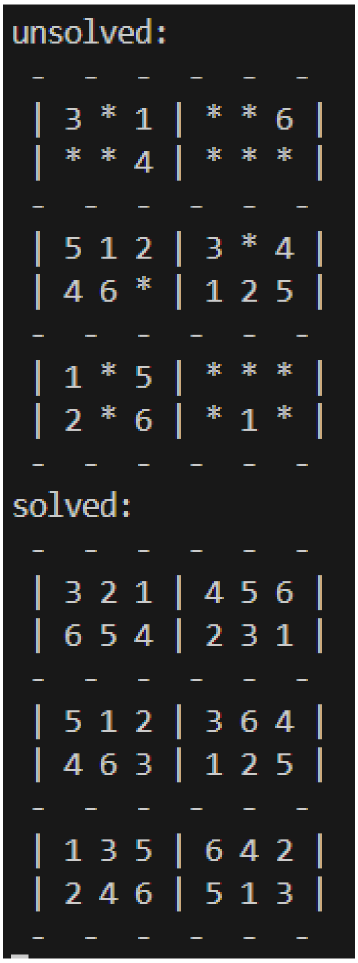
When running the main application there 2 different sets of outputs are provided for the user to examine, the console displayed unsolved and solved sudoku puzzle and the pygame window in which the puzzle is displayed initially unsolved, and then shows the step by step procedures the solving algorithm takes to present the user with an actual solution.

The console based output was beneficial to the first sprint development cycle as it got the job done in terms of displaying what the algorithm needed to solve and the result the AI algorithm produced, this method of output done best at showing just how quickly the AI algorithm could solve the problem as it would almost instantly display both the unsolved and solved solution to the sudoku. However, this method of output was inefficient of displaying the steps and the decisions that the algorithm makes as I would need to print multiple instances of semi-complete solutions, which is not a console friendly design as many iterations and decisions are made, so the mistakes and inaccuracies made before the solution is found is abstracted from the users.



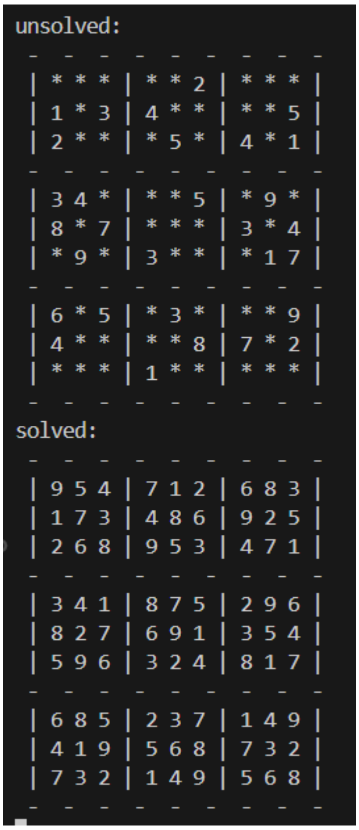
```
unsolved:
 -  -  -  -
 | 2 * | 3 * |
 | * 4 | * 2 |
 -  -  -  -
 | * 2 | * 3 |
 | 4 * | 2 * |
 -  -  -  -
solved:
 -  -  -  -
 | 2 1 | 3 4 |
 | 3 4 | 1 2 |
 -  -  -  -
 | 1 2 | 4 3 |
 | 4 3 | 2 1 |
 -  -  -  -
```

Figure 3.1: 2x2 grid



```
unsolved:
 -  -  -  -  -  -
 | 3 * 1 | * * 6 |
 | * * 4 | * * * |
 -  -  -  -  -  -
 | 5 1 2 | 3 * 4 |
 | 4 6 * | 1 2 5 |
 -  -  -  -  -  -
 | 1 * 5 | * * * |
 | 2 * 6 | * 1 * |
 -  -  -  -  -  -
solved:
 -  -  -  -  -  -
 | 3 2 1 | 4 5 6 |
 | 6 5 4 | 2 3 1 |
 -  -  -  -  -  -
 | 5 1 2 | 3 6 4 |
 | 4 6 3 | 1 2 5 |
 -  -  -  -  -  -
 | 1 3 5 | 6 4 2 |
 | 2 4 6 | 5 1 3 |
 -  -  -  -  -  -
```

Figure 3.2: 6x6 grid

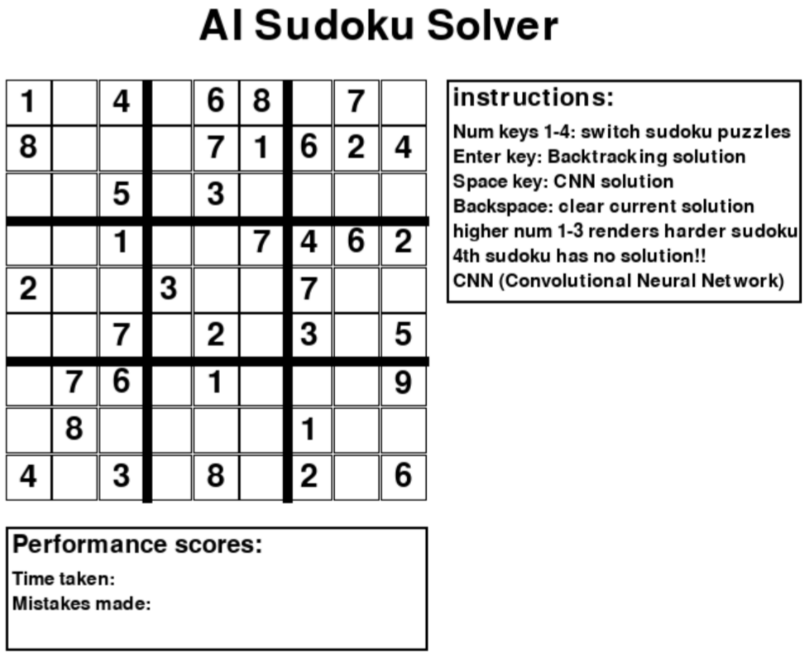


```
unsolved:
 -  -  -  -  -  -  -  -  -
 | * * * | * * 2 | * * * |
 | 1 * 3 | 4 * * | * * 5 |
 | 2 * * | * 5 * | 4 * 1 |
 -  -  -  -  -  -  -  -  -
 | 3 4 * | * * 5 | * 9 * |
 | 8 * 7 | * * * | 3 * 4 |
 | * 9 * | 3 * * | * 1 7 |
 -  -  -  -  -  -  -  -  -
 | 6 * 5 | * 3 * | * * 9 |
 | 4 * * | * * 8 | 7 * 2 |
 | * * * | 1 * * | * * * |
 -  -  -  -  -  -  -  -  -
solved:
 -  -  -  -  -  -  -  -  -
 | 9 5 4 | 7 1 2 | 6 8 3 |
 | 1 7 3 | 4 8 6 | 9 2 5 |
 | 2 6 8 | 9 5 3 | 4 7 1 |
 -  -  -  -  -  -  -  -  -
 | 3 4 1 | 8 7 5 | 2 9 6 |
 | 8 2 7 | 6 9 1 | 3 5 4 |
 | 5 9 6 | 3 2 4 | 8 1 7 |
 -  -  -  -  -  -  -  -  -
 | 6 8 5 | 2 3 7 | 1 4 9 |
 | 4 1 9 | 5 6 8 | 7 3 2 |
 | 7 3 2 | 1 4 9 | 5 6 8 |
 -  -  -  -  -  -  -  -  -
```

Figure 3.3: 9x9 grid

The GUI based output is beneficial for its ability to display the steps and decisions that the algorithm takes to find the solution. It shows the mistakes the algorithm finds and shows the how it works backwards, similarly to how a human would potentially try and find the solution to the sudoku problem. However, this form of output, despite solving the solution fairly quickly, potentially much quicker than most humans could, due to the algorithm requiring extra steps of clearing the squares and updating its value on the window's grid means that this output is generally slower as more computational power is needed from the machine, although this is not that serious of an issue as the speed of the solving algorithm is still capable of proving and meeting the aims and objectives the application was set out to achieve, demonstrating the power and efficiency of artificial intelligence.

After additional changes the layout of the final application was altered to better center the grid on screen as it is the primary focus of the application as this is where the processing power of Artificial Intelligence will be used on the problem and its solution displayed. in addition to this, 2 other sections have been included in the interface to firstly explain to the user how to use the application, which inputs the software takes to allow the user to switch between the backtracking algorithm and the Convolutional Neural Network prediction with the Enter and Space keys, and different preset sudoku problems which the algorithms have not seen before with the Number keys 1-4, where 1 would be the easiest sudoku puzzle and the difficulty increases to 3 the hardest, while 4 is a puzzle without solution to check the algorithms behaviour when presented with this challenge and the Backspace key to clear the current solution and leaving behind the unsolved sudoku. The second section is where the time taken for the solution algorithm to give its final answer, and the number of mistakes the backtracking algorithm had made to reach the final solution or the number of mistakes the final prediction of the Convolutional Neural Network has made.



AI Sudoku Solver

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

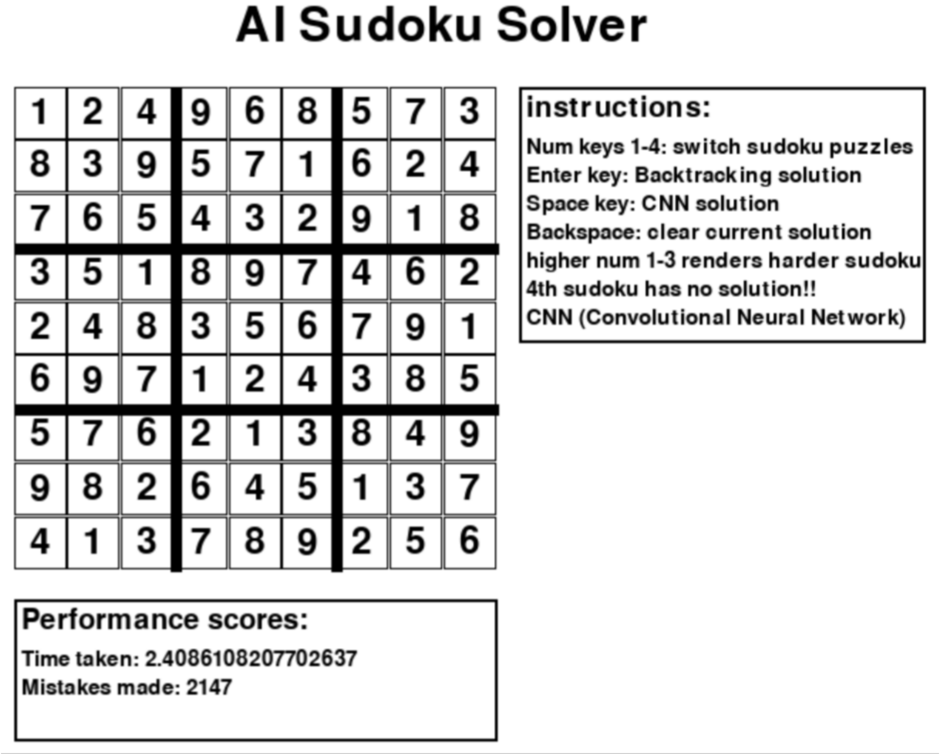
instructions:

Num keys 1-4: switch sudoku puzzles
Enter key: Backtracking solution
Space key: CNN solution
Backspace: clear current solution
higher num 1-3 renders harder sudoku
4th sudoku has no solution!!
CNN (Convolutional Neural Network)

Performance scores:

Time taken:
Mistakes made:

Figure 3.4: Inital Application running

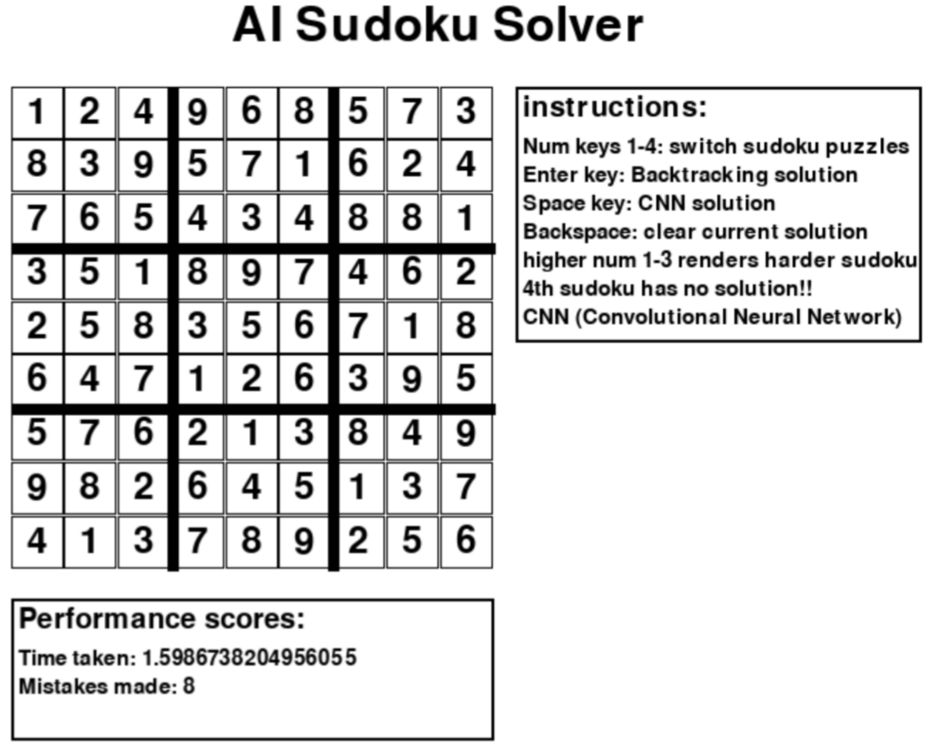

AI Sudoku Solver

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

instructions:
Num keys 1-4: switch sudoku puzzles
Enter key: Backtracking solution
Space key: CNN solution
Backspace: clear current solution
higher num 1-3 renders harder sudoku
4th sudoku has no solution!!
CNN (Convolutional Neural Network)

Performance scores:
Time taken: 2.4086108207702637
Mistakes made: 2147

Figure 3.5: Sudoku solved with backtracking


AI Sudoku Solver

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

instructions:
Num keys 1-4: switch sudoku puzzles
Enter key: Backtracking solution
Space key: CNN solution
Backspace: clear current solution
higher num 1-3 renders harder sudoku
4th sudoku has no solution!!
CNN (Convolutional Neural Network)

Performance scores:
Time taken: 1.5986738204956055
Mistakes made: 8

Figure 3.6: Sudoku solution predicted through CNN

Chapter 4: **Project Evaluation**

4.1 How well the Application Achieves its Aims

The final product provided uses 2 different algorithms to solve the classical mathematical problem, a sudoku. The backtracking algorithm and the Convolutional Neural network both work within the application to demonstrate just how well machines can learn through using 2 different techniques and still provide a suitable solution to a range of sudoku problems of varying difficulty which is more computationally efficient in comparison to a human.

In the case of the Backtracking algorithm it is able to calculate every type of sudoku combinations, learning from failed attempts to discover an answer at speeds most if not all humans would be unable to compete with, and in the case of the Convolutional Neural Network, it is trained on existing problems and solutions to find the patterns and features of the solution so that it can attempt to accurately replicate the steps required to predict a viable solution at speeds even faster than the backtracking approach once the model is fully trained as it does not need to attempt to work through many possible incorrect solution branches to try and calculate what works and what does not, it works through recognising the ruleset of the sudoku and making a prediction that will likely satisfy the same patterns as the previous solutions it was trained with, producing a solution that very accurately follows the constraints of the sudoku problem.

The application shows this through displaying the grid and rendering the solution in real time, along with the time it took the algorithm to calculate its solution and the number of mistakes it has made to get to its conclusion, in the case of the backtracking algorithm, and the mistakes in the final prediction for the CNN. Despite the errors in development that needed to be overcome to develop the project to this state, the product does its job well in meeting its objective of developing AI algorithms that is capable of playing games and solving puzzles, and doing this task more effectively and efficiently than mankind, proving the superior processing capabilities of a computer.

4.2 Potential Improvements for the Application

In the end, the Project done well in meeting the set out expectations, however, there is always room for improvement and there are a range of features that could have made for a better application and would have provided a better user experience, which would have aided the project in superseding its objectives and better demonstrating the power of Artificial intelligence as a means to solve problems. One of the areas that needed improvement was the final convolutional model. Despite the model having a decently high accuracy and validation accuracy, around 97%, these scores were not yet perfect as there was still a degree of uncertainty in the predictions which lead to a few mistakes being generated in some of the 81 cells, which with additional time to develop could have been ironed out to find a truly optimal Convolutional Neural Network.

In addition to the final model, the Backtracking algorithm despite being very fast and guaranteeing a corrected solution to the sudoku problem, given that a solution exists, the current algorithm could benefit from using more efficiency such as opting to use other means to detect already used numbers, other than just calculating an exhaustive search going through each number in rows, columns and sub grids which, although not much, slows down the algorithm and increases its computational time which is undesired when trying to prove the computational superiority of AI algorithms.

The application could have also benefitted from a feature which would have allowed users to input their own sudoku problems, rather than the preset selection to switch between, as this would have shown the effectiveness of the 2 algorithms with a greater variety of problems. Training additional optimal CNN models on variety of sudoku sizes would give the application the ability to show an ability to solve size variants of the classic sudoku problem, which would have allowed for the utilisation of preexisting features such as the adaptive grid rendering and the backtracking algorithms versatility.

Chapter 5: **Conclusion**

In conclusion, the development of the AI sudoku solver has gone well in achieving its goals as significant progress has been made for each milestone such as the development of the console interface, the graphical user interface, the Backtracking solving algorithm and the Convolutional Neural Networks. although the final product produced at the end of this project could be improved to be more efficient, in the case of the backtracking algorithm and more accurate, in the case of the Convolutional Neural Network, the algorithms still performed well with great speeds and very high accuracy despite their rooms for advancements which helps the application to demonstrate how Artificial intelligence is superior to human intelligence due to its computational capacities and speed and how it can be applied and adapted to a gaming environment, which provides a safe, fun and engaging setting to test the limits and capabilities of artificial intelligence, in order to attempt to play games and solve puzzles efficiently through learning from its mistakes and trying to discover an accurate and viable solution for a problem which in the case of this project is a solution to different sudokus of varying difficulty.

Bibliography

[1] AI for Social Good (n.d.). Artificial Intelligence vs. Human Intelligence: Exploring the Debate and Key Points. [online] Available at: https://aiforsocialgood.ca/blog/artificial-intelligence-vs-human-intelligence-exploring-the-debate-and-key-points [Accessed 8 Oct. 2024].

[2] C. Strachey, Logical or non-mathematical programmes, in: Proc. Association for Computing Machinery Meeting, Toronto, ON, 1952, pp. 46–49.

[3] J. Kister, P. Stein, S. Ulam, W. Walden, M. Wells, Experiments in chess, J. ACM 4 (1957) 174–177.

[4] GeeksforGeeks (n.d.). Sudoku — Backtracking-7. [online] Available at: https://www.geeksforgeeks.org/sudoku-backtracking-7/ [Accessed 9 Oct. 2024].

[5] Charles Akin-David, Richard Mantey, n.d. Solving Sudoku with Neural Networks. [pdf] Available at: https://cs230.stanford.edu/files_winter_2018/projects/6939771.pdf [Accessed 10 Oct. 2024].

[6] Rep.ai, *53 Chatbot Statistics For 2024: Usage, Demographics, Trends*, 2024. Available at: https://rep.ai/blog/chatbot-statistics.

[7] *65 Chatbot Statistics for 2025 — New Data Released*, 2025. Available at: https://www.demandsage.com/chatbot-statistics/.

[8] *AI in Customer Satisfaction Statistics: 2025 Data*, 2025. Available at: https://seosandwitch.com/ai-customer-satisfaction-stats/.

[9] Nexford University, *How Will Artificial Intelligence Affect Jobs 2024-2030*, 2024. Available at: https://nexford.org.

[10] APA PsycNet, *APA PsycNet FullTextHTML page*, 2025. Available at: https://psycnet.apa.org/fulltext/2024-87092-001.html

[11] Statista, *Trust and acceptance of healthcare AI worldwide 2022*, 2022. Available at: https://www.statista.com.

[12] S. C. Brailsford, C. N. Potts, and B. M. Smith, *Constraint satisfaction problems: Algorithms and applications*, European Journal of Operational Research, vol. 119, no. 3, pp. 557-581, 1999. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0377221798003646

[13] P. van Beek, *Backtracking Search Algorithms*, Foundations of Artificial Intelligence, vol. 2, pp. 85-134, 2006. Available at: https://www.sciencedirect.com/science/article/abs/pii/S1574652606800088

[14] Melanie, *Backtracking: What is it? How do I use it?*, DataScientest, 29 Mar 2024. Available at: https://datascientest.com/en/backtracking-what-is-it-how-do-i-use-it

[15] A. D. Dongare, R. R. Kharde, and A. D. Kachare, *Introduction to Artificial Neural Network*, International Journal of Engineering and Innovative Technology (IJEIT), vol. 2, no. 1, pp. 189-194, 2012. Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=04d0b6952a4f0c7203577afc9476c2fcab2cba06

[16] C. Woodford, *How neural networks work - A simple introduction*, Explain that Stuff, 12 May 2023. Available at: https://www.explainthatstuff.com/introduction-to-neural-networks.html

[17] IBM, *What is a Neural Network?*, IBM, 2024. Available at: https://www.ibm.com/topics/neural-networks

[18] J. Schmidhuber, *Deep Learning in Neural Networks: An Overview*, Neural Networks, vol. 61, pp. 85-117, 2015. Available at: https://arxiv.org/abs/1404.7828

[19] UpGrad, *Basic CNN Architecture: 5 Layers Explained Simply*, Blog Article, 2025. Available at: https://www.upgrad.com/blog/basic-cnn-architecture/

[20] Machine Learning Mastery, *Using Normalization Layers to Improve Deep Learning Models*, Blog Article, 2025. Available at: https://machinelearningmastery.com/using-normalization-layers-to-improve-deep-learning-models/

[21] Data Science Stack Exchange, *ReLU vs Leaky ReLU vs ELU with Pros and Cons*, Forum Discussion, 2025. Available at: https://datascience.stackexchange.com/questions/102483/relu-vs-leaky-relu-vs-elu-with-pros-and-cons

[22] Machine Learning Mastery, *Difference Between a Batch and an Epoch in a Neural Network*, Blog Article, 2025. Available at: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

[23] Keras, *Callbacks API*, Documentation, 2025. Available at: https://keras.io/api/callbacks/

[24] *Loss Functions in Deep Learning: A Comprehensive Review*, Preprint, arXiv, 2025. Available at: https://arxiv.org/pdf/2504.04242

[25] Machine Mindscape, *Understanding Optimization Algorithms in Deep Learning*, Blog Article, 2025. Available at: https://machinemindscape.com/understanding-optimization-algorithms-in-deep-learning/

[26] Machine Learning Mastery, *Understand the Dynamics of Learning Rate on Deep Learning Neural Networks*, Blog Article, 2025. Available at: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[27] J. Yu, *Solving Sudoku with Convolutional Neural Networks*, GitHub repository, 2025. Available at: https://github.com/JacksonYu92/Solving-Sudoku-with-Convolutional-Neural-Networks/blob/main/Sudoku_cnn.ipynb

Appendix

Sudoku Interface

Importance of Interface Implementation

The main focus for the start of my project needs to be the interface for displaying the sudoku. this is a necessary starting point as in order to demonstrate Artificial Intelligence solving the sudoku, my product needs to be able to display both the unsolved and solved solution in a way that is easily digested by the target audience. the borders, regions, numbers and void values need to be distinct and observable to those trying to understand what the AI algorithms is trying to achieve, and if the algorithms are doing so correctly. that makes this element of the product integral to the development of my project.

Implementation of Sudoku Interface

In order to implement this feature, I have decided to initially work on a console based interface. I felt this was necessary as my first iteration of the sudoku interface only needs to display an easily readable representation of the sudoku puzzle. this was achievable in the console as the borders can be represented through the '-' character for row division and '—' character for column division, the numbers as just numbers and the void values as an '*' character. this gave a simple and easy to read sudoku display on the console.

Planned Improvements to the Interface

Once the other features of my project have become slightly more developed, I plan to return to improve the sudoku interface through various means, such as randomised sudoku generation of varying difficultly, as well as a better user interface to display the sudoku rather than primarily using the console, which works well and is readable, but is not the best means of displaying the puzzle in a visually appealing and user friendly fashion.

Solving Algorithm

How a Backtracking Algorithm could Solve a Sudoku

A backtracking works similar to how a human would back track. The algorithms makes an attempt to solve a problem. so in the case of a sudoku it would attempt to fill in blank squares with a potential solution, and when a problem occurs in its solution, it goes back to the cause of the problem, which would be the invalid potential solution in that square or a previous one, and continues its attempt with a different solution for the sudoku, and this is done repeatedly till a viable solution is formed.

Implementation of Backtracking Algorithm

When developing the solving algorithm, first came the crucial element of finding a blank square in the grid, this would be represented by a 0 in the sudoku grid array. the coordinates on the grid would be returned to the main algorithm when found, then using these, the main loop will attempt to fill the square with numbers ranging from 1 to the max value, which is also the length of the grid (usually 9), until a viable value for the square is found (a value that is not repeated in the row, column or sub-grid). if a viable value cannot be found, the algorithm clears the current square and backtracks to the previous square, and checks the next values for viability, if there are no viable values, the algorithm will clear the current square and backtrack once again to the square before and once again try different values until a different viable value is found and the algorithm searches for the next blank value and the algorithm repeats itself, until there is no more blank squares, or no more values to try on the grid that gives a solution to the problem.

Planned Improvements for the Backtracking Algorithm

In order to improve the speeds of my backtracking algorithm, I would need to look into my uses of exhaustive searches which is used a lot in my algorithm for different functionalities such as, finding an empty square by looping through every value in the grid array, checking validity of a value by searching through every value in the row, column and sub-grid for a repeated value. this is very inefficient, and I believe this can be rectified by maybe implementing faster search algorithms or making use of other methods and techniques to make the searches slightly quicker and efficient, which is necessary to prove AI's superior problem solving capabilities.

Improvement of the User Interface

Creating a Graphical User Interface

When creating my graphical user interface to display the sudoku and the solving algorithm, I decided to pygame as it provides an adaptable environment in which the sudoku grid can be drawn on the window and updated with little difficulty while still being able to display a way of visualising the backtracking algorithm. when creating the grid, I decided to give it the length 340 which was big enough for the user to see the sudoku and the values in each square clearly. When dividing the grid into sub-grids I used the same divisor logic used in the console print function, however I divided the grid visually using thicker lines to create the division between the whole and sub grids. I then started adding values to square by placing a new square over the empty square with the intended value, this could be cleared through overwriting the square by filling the surface with a white square then updating the value to a new square to cover the cleared square. this function would need to be called every time the value changed in the algorithm in order to give the visual backtracking effect.

The Benefits of Having a GUI to Display the Problem Solving Algorithm

By introducing a GUI to display the solving algorithm, I am able to show the step by step solution being formed through mistake adaptation approach my backtracking algorithm takes as well as showing how quickly the computer can calculate this as the GUI updates the values very rapidly, demonstrating how much faster a computer is compared to a human, despite the GUI technically wasting computer processing speed, so it is faster to print the output directly then updating a GUI, but then the algorithm will not be demonstrated in an easily comprehensible manner.

Planned Improvements to the GUI

In order to improve the Graphical User Interface I plan to make use of radio buttons or checkboxes to select which algorithm to use when solving the puzzle, I also feel a reset button to clear the grid and start button to start the algorithm would also be a nice touch, and will make the application experience more convenient. Additional text would also be needed, and useful for displaying the time it takes for solving the problem as well as potentially a mistakes counter for how efficiently/accurately it initially solves the problem.

Next Solving Algorithm

How could a Neural Network Solve a Sudoku

My focus moving forwards will be on developing a Neural Network that will be trained on taking in sudoku solutions, this is so that it will learn what a solution to a sudoku should look like and it should recognise the patterns/rules of a sudoku so that when it is faced against an incomplete sudoku, it will be able to provide an accurate prediction of a solution, that will hopefully be the correct answer to the inputted problem.

Layers of the Network

Conv2D is used to extract features, makes use of filters called kernels to produce feature maps, which aids in pattern recognition in the network. the use of filters allows for parameter sharing which means less parameters are needed for the network to generalise better as filters cover multiple parts of the input. currently uses varied filter sizes and kernel sizes 3x3.

BatchNormalisation aims to standardise the inputs so that the process of training becomes more stable and efficient. this allows for a higher rate of learning and quicker training processes, it also helps to prevent overfitting by adding on a bit on ”noise” to the input of each layer. does this through working out the mean and variations of the inputs, then normalises based on the results.

MaxPooling2D works by reducing spacial dimensions of the input data in order to reduce complexity and memory consumption in computations. it does this by keeping the maximum value from the pooling window, which helps to highlight prominent features, which improves generalisation and reduce overfitting.

Dropout hopes to achieve a reduction in overfitting by randomly dropping out neurons (resetting to 0) so that the network is forced to learn redundant representations of the data, also allowing for better generalisation by preventing network from being too specialised to the training data. dropout can aid in reducting co-adaptation which is where neurons depend on other specific neurons, this means single neurons can learn more robust features independently.

Flatten layer converts the multidimestional data into a singular dimension which is what is needed as input for the dense layers of the CNN so that they can process and learn from the test data.

Dense layers are fully connected layers as each neuron in this layer is connected to every neuron in the previous layer, so every input now has an impact of every output, which gives the model the ability to learn the patterns and complex relations of the training data.

Types of Activations

Activation Functions this is the function in which the neurons in the network uses to learn and make decisions as it determines which neurons should activate and pass data to other layers to form the final prediction.

ReLU this is an activation function which works by taking in the input x and setting the output to x if positive (output is same as input) and if x is negative (x¡=0) the output of the output is 0, this is a simplistic activation which is great for computational efficiency however, can lead to dying ReLU problem were neurons set to 0 are inactive consistently and therefore not learning.

Leaky ReLU this is a variant of ReLU which attempts to address the dying ReLU problem by allowing for a very small non zero values which is calculated by multiplying the input with a very small constant (for example 0.01) if the input is x¡=0 and just outputting x if the input is positive. this prevents neurons from being repeatedly deactivated in the network as they avoid the output being set to 0.

ELU this activation also aims to prevent the dying ReLU problem by using and exponential function on inputs ¡=0, this allows for the output to avoid being set to zero, as it is set closer to 0 but not actually 0, so the dying neuron problem is avoided. otherwise, for positive inputs, the output remains the same as the input.

Softmax softmax is a final layer activation which this ensures that all output probability is normalised so that all probabilities add up to 1, so that it is perfect for classification as the probabilities is appropriately distributed over classes (perfect for sudokus as each cell has classes 1-9 as a potential solution).

Model Compilation

Network Optimizer a network optimizer is used to help adjust the weights of neurons and the biases in the neural network while in its training phase. it works to reduce the loss (errors) of the network while improving the accuracy of the predictions. it uses backpropagation to update network parameters based on gradient calculations, which in turn helps the neural network learn the dataset.

Learning Rate when using the Adam optimiser, the learning rate can effect the accuracy of the model during its learning phase, it is important as this dictates how much the models weights adapt when learning. the size in which this value is set can effect the speeds of convergence and as a result the training time, as well as how precise the updates in the weights which can improve the accuracy as well as keeping the rate of error (loss) to a minimum.

Loss Function this function is important for the training of the model, as this function is how the accuracy of the model prediction is calculated. this is needed to provide a measure on how well the model is performing on the training dataset. this function also works with the loss function to aid in the adjustment of weights and biases, to try and limit the loss which helps the model to learn more efficiently and predict more accurately. with classification models it is common to use a Cross-Entrophy loss function, as this helps with the models learning process.

Model Training

Epochs each time a model trains through an entire training set, this is called an epoch. the number of epochs refers to the number of times the model will cycle through the same entire training set. having multiple epochs are important as this allows for the model to better understand the data as one train through is not typically enough for the model to learn to effectively and accuratly predict. However, if there are too many epochs, there is a chance for the model to learn the specific training set and as a result will struggle to predict new examples which means it is overfitting.

Batch size this is the amount of samples the network takes from the training set and trains with in one forward and backward pass. smaller batch sizes can lead to more frequent updates to the model which can help to make better predictions as well as consuming less computational resources, but larger batches despite the greater computational demand, can train the model much faster as well as having more stable updates which give smoother convergence.

Callbacks callbacks can be helpful in the training phase of the model as it can provide a range of functionalities such as improving performance through early stoppage if validation loss stops decreasing (to avoid overfitting) and saving checkpoints for the model while training on the set so that at the end it can restore the point with the best learnt parameters.

Problems Encountered

Severe Underfitting A serious issue that came with my first few drafts at developing my model is extreme underfitting. this is where the model was failing to properly learn from the training set, this was a serious issue as if it wasn't accurately learning and predicting with the training set, when I test it on new data, the prediction I get from the model is very much inaccurate and could not even pass as an almost solved sudoku. I believe the reasons for this the low numbers of conv2D layers, as well as very small filters and kernel sizes, in addition to extremely high batch sizes. these will need to be more finely adjusted to try and discover the appropriate model for solving a sudoku puzzle.

Extreme Training Durations after adjusting the sizes of filters and conv2D layers, as well as the types of activation those layers required, I found when having too many layers, around 5 or more, and filters above 162 and greater, with kernels (5,5) and higher each and in combination gave extremely high training times per epoch, some parameters coming up to 5 hours on each epoch, which is very computationally exhaustive and in some cases not even beneficial as the models accuracy would be barely increasing in these hours, meaning the model is training past its peak which could be and indication of overfitting.

Slight Overfitting with some of my larger model attempts, I would yeld better predictions to the sudokus I would pass in, however due to the long times spent training, the model would provide very high accuracy but slightly lower validation accuracy and when testing on new data would provide sudoku solutions with quite a few mistakes. I think moving forwards the model will need some parameters reduced and some of the data and layers more normalised or even dropped out in order to try and avoid neuron dependancy and overfitting in the model, which should smooth out and reduce mistakes made when solving the problem.

Potential Solutions to the Problems Encountered

Number of Conv2D Layers increasing the number of convolutional layers can increase the models ability to learn by better extracting features from the training set being passed in so that it can use those features to classify and predict far more accurate predictions. however too many layers brings overfitting and can increase computational costs significantly.

Increased Number of Filters by using more filters the convolutional layers are able to extract a deeper representation of the data, detecting patterns and features more precisely and as a result returning predictions with far less errors. although with greater increases in filters, the greater the demand on training durations, which may not give much in return.

Increased Kernel Sizes larger kernels allows for the network to capture more extensive spatial features of the training data, this allows for smoother feature extractions as well as reduction the amount of layers the network would need to capture these features. similarly to the layers and filters, too many can still increase the computational costs in training the model and this could also lead to overfitting.

Final Improvements to the User Interface

Improved Layout adjustments to the User Interface has been made to better center the sudoku on screen and also providing an instructions section to the application describing how the user can use the app, as well as providing a section in which the time the algorithm takes to provide the solution along with the number of mistakes it has made to come up with its final solution.

Use of Additional Inputs allows user to use keys on the keyboard to navigate through different features of the app, such as number keys to switch between the sudoku problems, the enter key to solve with backtracking, the space key for the Neural Network solution and the backspace to clear the current solution leaving the user back with the unsolved sudoku puzzle.