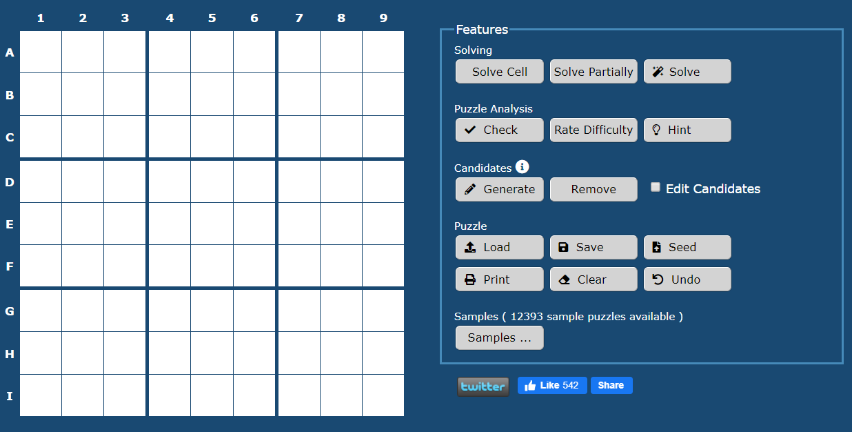
**Introduction and current problems:**

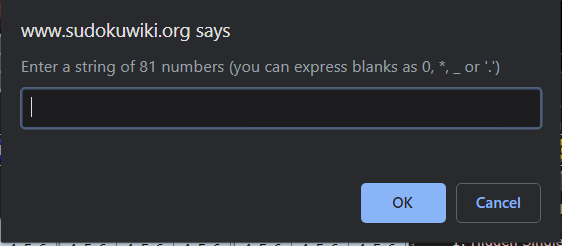
When solving numerical puzzles, especially a difficult one, sometimes people look for the answers to help them solve the problem. In places such as the newspaper or some puzzle books, the answers are not printed. The solution to this would be to get a puzzle-solving algorithm to solve it for you quickly. However, a significant problem currently is that online solving algorithms are inferior in their user interface. Although they compensate for this with fast solvers, it is very time-consuming to input numbers into a grid.

Tediously typing out numbers into a template is the main problem area that this project will be focusing on.

For this software to work, first, it needs to detect the borders of a puzzle and classify the numbers in the puzzle. This application will be used by programmers who are looking to improve their current user interface and make it so that the user's experience is as good as possible. Their likely use of my program will allow the user to import a picture of the Sudoku and let my application process the image. After the user's solver finds a solution, the algorithm will also allow the final computed solution to be masked over the original. The time difference between opening the application and getting an answer in traditional systems is significant and might not be appealing to the user. However, by introducing the aspect of pictures, what would have taken you 5 minutes of clicking and typing will now take a matter of seconds.



In online solvers such as this one, the time taken to input all the numbers can take anywhere from a minute to 2 minutes. Accompanied by clumsy buttons on the side, most of which an average user will likely not use or will have to spend time understanding, proves a lousy user experience

Another example of a poorly designed site that doesn't care about poor user experience requires you to type in an 81-long string of numbers. As you can see the likelihood of an error is very high and getting one number wrong would waste an additional 5 minutes trying to identify the mistake

**Purpose of project:**

The purpose of this project is to create a **tool** that **programmers** can incorporate into their own sudoku solver algorithms in order to bypass the issue of having to type out numbers into a grid themselves. It will be used for image recognition purposes; it'll compress and abstract the image of any irrelevant details and noise. Afterwards, it'll identify the borders of the sudoku puzzle (which could be rotated), rotates and crops the puzzle, extracts number data along with their position, tries to classify numbers and output into a grid that will be used by others as a base to solve their puzzles.

**Clients:**

There will be two clients that will be using this system the most: my friend, a novice programmer James Anderson who adores problem-solving and has developed a sudoku solver, and my grandmother who likes to keep her mind in check by solving sudoku puzzles.

James has developed a sudoku solver that is capable of solving any sudoku, given that it has a solution. The next step in his project would be to implement a user interface where users input numbers into a grid. However, his program can further be improved by introducing image recognition. This will get rid of the step of having the user input numbers into a grid.

My grandmother likes to solve sudoku puzzles that are on the newspaper that day, but finds it frustrating when she makes a mistake and has no reference to check it against. By introducing the image reading, she won't have to use the silly websites to check her puzzles any more, but use a more streamline user-experience-optimized program that James and I will have created. Just to reiterate, my gran won't be doing any of the coding, she'll be using the product of mine and James' programming.

**Interviews:**

1. James is looking for an easy to use library:

What are some of the functions that you are looking for?

I want to be able to print my functions

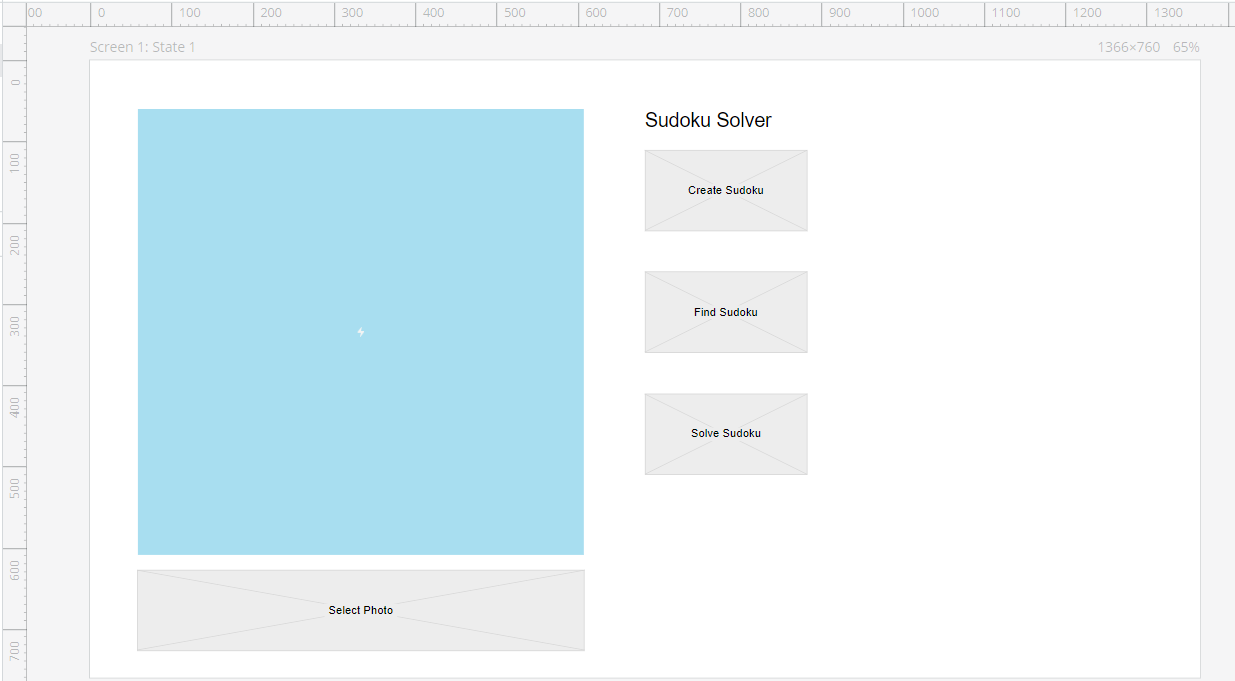
What should the structure of the library be like?

I'm looking for an easy to use tool, so your different tools should be split into object and contain enough functions for me to incorporate them the way I want to.

But I don't want to read into the code too deeply, I want very few lines of code which processes the image so that it takes up as little space as possible.

What do you expect to come with the code?

Documentation where I can quickly look at requirements and outputs of functions and classes.

James also provided me a screenshot of what his UI might look like, which helped understand his requirements more.

1. One of the users that will be using elements of my final outcome is my Grandma, who emphasizes in the interview the importance of a UI and not having to input numbers yourself. She has no IT skills and prefers solving numerical puzzles such as Sudoku on paper, with a pen. She has a laptop with a webcam, which could be used to get a snapshot of the problem for analyzing.

How do you prefer to do your puzzles?  
I can only do my puzzles using pencil and paper. I can write in notes in the boxes and erase numbers with ease, and then lock in my answer with a pen. Besides, I love the feeling of writing notes with a pencil. Doing puzzles on the screen is much less enjoyable and hurts my eyes after a while.

What are you looking for in this application?

I am looking for an easy way to get a solution to my puzzle. I can't stand writing numbers into a grid with my mouse and keyboard; it takes so long, and sometimes I press the wrong button or click the wrong box, so I have to spend even longer trying to find my mistake! I think it's just incredibly clumsy, time-consuming and a waste of time. By the time I finished putting in numbers , I could have probably figured it out myself.

They have a website on their front cover where they give you the answers, but I don't care for the detail of typing it out. I'd then have to shift through all the problems until I find my puzzle.

**Interview analysis:**  
Image Analysis:

The first step is the abstracting the image. This includes turning the image into greyscale because I don't need colour to be able to solve most puzzles. Noise reduction to get rid of any indents from previous writing or accidental lines drawn or any other detail that would pollute my image.

Next, I will need to identify the vertices of the puzzle so that I can construct a virtual grid. This will also allow me to get rid of the background and other unnecessary details by cropping the image.

After that, I will detect if there are any numbers in the cells and if there are take a snippet of that cell and try and identify the number to then input into a 2x2 matrix to solve. By taking a snippet of the cell, I don't have to worry about there being more than one digit in the cell, which makes my life easier because I only expect one number.

Program Structure:  
My friend wants an easy to use tool that will also allow him to use it with enough flexibility and with minimal hassle. This means that most of the work has to be done for him in functions but also allow him to use them as he sees fit. This means that most of the functions can be private, and the only public functions will be to carry out one step in the algorithm or to fetch class parameters. So as to not corrupt the program, I think that all of the parameters will be private, just so that an accidental modification won't cost him the program.

From UI:

From the UI, it's evident that he'll be showing the image on the large square. It's therefore important for this project to have a resize option as well as make the image data available so that the UI software can scale it and do with it what it needs.

**Purpose of the project after the interview:**

From the interviews, it is evident what my project should contain:

The purpose of my project is to create an easy to use tool that programmers can incorporate into their own sudoku solver algorithms in order to bypass the issue of having to add numbers into a grid themselves. My tool will be used for image recognition purposes; it'll compress and abstract the image of any irrelevant details and noise, identify the borders of the sudoku puzzle (which could be rotated), rotates and crops the puzzle, extracts number data along with their position, tries to classify numbers and output into a grid that will be used by others as a base to solve their puzzles.

My tool will be used to serve an end goal of it being used to incorporate image processing into their sudoku solvers. It is necessary to understand that this project will not contain a UI; this is because it is made for programmers to incorporate into their projects which will have no aspects of image processing.

**Different types of solution that might be available**

Detecting numbers:

1. **Neural Networks:**

Researching into different Neural networks that I can use:

**CNN:**

[*https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53*](https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53)

A CNN consist of an input and an output layer, as well as multiple hidden layers, which consist of a series of convolutional layers that convolve (combine). This network takes an input image and assigns importance to various aspects, like a particular pen stroke in my case.

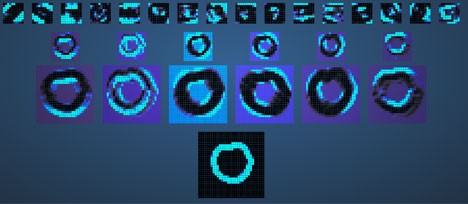
Typically, an input image (a linear list of numbers) will also be assessed by different colour channels, but because I will be analyzing handwritten text, I can downscale the colour to a simple greyscale; black and white. In the case below, this image would be split into three layers (as shown), red, green and blue. In my case, because of the simplicity of the input image, I will not have to worry about any sort of multiple simultaneous problems, which is an advantage of this project; not as computationally intensive as it gets.

A kernel convolves the image, which is demonstrated by the dashed squares in the below image. This filter applies different filters to the image. For example, the kernel matrix below is used to sharpen an image, as demonstrated:



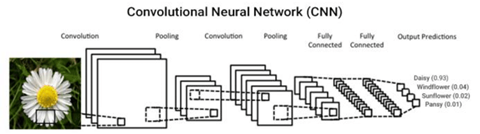
[*https://www.saama.com/blog/different-kinds-convolutional-filters/*](https://www.saama.com/blog/different-kinds-convolutional-filters/)

Therefore, kernels are used to perform a transformation to the image, for example, inversions, sharpening an image, applying a blur, filter, or much more. In my case, the kernel that I will want would be that which will extract only the most crucial information from my image. However, that is improbable, so the kernel is left for the CNN to learn.



Here is a demonstration of how a kernel is applied to the letter 'O'. It is this ability of CNNs to be able to detect abstract and complex features such as boundaries that makes them so attractive in image recognition problems. The resulting multiplication of the matrix with the image in the input will be a very big number, which will be interpreted as the essential information.

One big concern with using this approach, above any, is the time cost when using a kernel to move along the image. They are very computationally intensive, and using a large filter will be very problematic. An easy solution is to decrease the size of the filters and increase the size of the strides taken.



Next comes the Pooling layer, which reduces the spatial size by extracting dominant features (rotational and positional invariant) for decreasing computational power. The Max Pool returns max value from a portion of the image covered by kernel, where the average pooling returns average of all values. For my purpose, it would make sense to use the max-pooling because it not only performs a lot better but also it acts as a Noise Suppressant.

The fully connected hidden layer is a way of learning how to identify different objects from others. It displays the resulting image from convolution and pooling as a column vector. It is hidden because we know not of how the data is processed, but we only know that something rather happens inside which then points to a valid object.

In an effort to display the weights of the pointers, a SoftMax layer converts numerical values into percentages (0 to 1). This is why it is typically the final layer used in CNN. This is great at determining multi-class probabilities, but the SoftMax can be costly as number of classes grow. In that case, the alternative is to use multiple logistic regressions instead.

[*https://pdfs.semanticscholar.org/d50d/ce749321301f0104689f2dc582303a83be65.pdf*](https://pdfs.semanticscholar.org/d50d/ce749321301f0104689f2dc582303a83be65.pdf)

**Backpropagation:**

Each input pixel value contributes to a weighted sum for each output unit. The output unit with the highest sum (including the contribution of a bias constant) indicates which digit the number is. It is also attributed to an Aritificial Neural Network (ANN), which contains one or more layers which are linked to the next layer. There are a series of hidden layers in-between the input and output layers. In the code provided in9, the network is trained until the error rate is less than a specified value and until it reaches a maximum number of iterations. During this, it takes each image from the training set and forward propogates (finds inputs and outputs for the next layers and calculates the output) and then backpropogates, to try to reduce the error rate by changing the weights. It then iterates until the gradient descent tends towards a global minimum.10

Advantages of this neural network include:

1. Adaptive learning
2. Self-Organization
3. Can create its own or representation of the information it receives
4. Real-Time Operation
5. Fault Tolerance via Redundant Information Coding
6. **Templates using cv2**

A simpler solution to implement would be to use a template matching algorithm.

Cv2 library has a function which matches a template onto an image. Therefore, by simply making a template file anybody could match any number to a template to quickly identify its nature.

This would only work with typed numbers as the randomness and unpredictable nature of humans' handwritten digits can only be analyzed with neural networks, which are very complex.

This would mean using **python** as my language because extensive documentation exists in this language primarily.

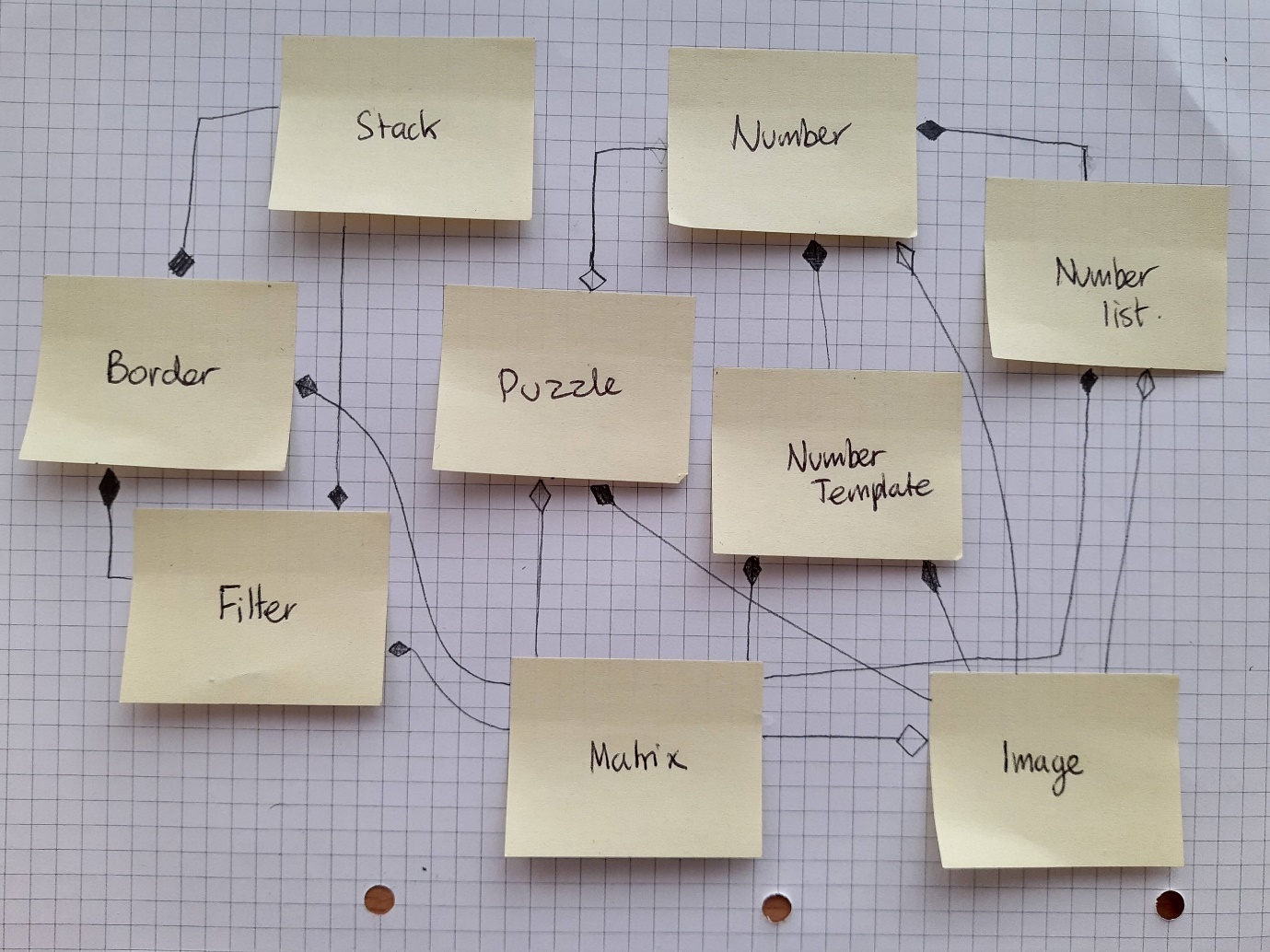


My final solution would be to use the templates of cv2 because it is much easier to implement, will execute faster than a neural network, won't require any training, is a deterministic solution. This will also mean that my final solution will be using **python**.

**Modelling of the project area**

This project wont need any database approaches, because all the system does is analyze images, not save them in any way. Further, it won't save anything unless specified, where the only thing that will be saved is the image.

I will approach this through an Object-Oriented Approach because of nature of how different parts of my program will interact.

****

**Object descriptions:**

Image -> work with image data, compress and adding elements to it

Border -> identifying the border of the large quadrilateral surrounding the puzzle

Filter -> traverse the large quadrilateral and find the corners

Puzzle -> handling forming the puzzle out of the image data, operates under the assumption that there is a 9x9 grid with at most a one-digit number in each square

Number -> handles numbers, contains the actual classified number, position in the grid and the number image

Number Template -> handles templates of numbers

Number List -> handles a lot of numbers to perform operations based on the number, position etc

Matrix -> handles image matrix get functions such as get\_pixel(x,y) or get length

Stack -> handles backtracking and stack data-structures

**List of numbered SMART objectives (what the new system MUST do)**

My solution to the problem at hand will be to program my code in python. Alongside using python for its simple syntax and for using cv2, I further chose to use python because of its broad diversity of user-made libraries, which means that my job can be easier when fitting in these functions which have been tried and tested by people on GitHub and other community congregations.

I further intend to use an Object-Oriented approach which is the most sensible approach as I will be working with a lot of things that will be repetitive. By introducing classes, I will compress my code down into objects. It will also offer the following benefits:

* Easy troubleshooting:
  + When working with object-oriented programming, I'll know exactly where to look, so if my corner identification stops working, I know that the logic is wrong in the corner Class.
* Reusing classes with inheritance
  + It is an inevitable outcome of my project that some of my classes will go better together if they're integrated and/or inherited.
* Easy implementation and modularity
  + It is important not to forget that the aim of my project will be to create a library for my user to implement into his code. It'll be easier for the programmer and for the user if I organize my project into objects.

|  |  |  |
| --- | --- | --- |
| **No** | **Objective** | **How and Why** |
| **1** | Read Image Files from where | Either by looking up the standard for different image encryption algorithms or by using a pre-written library to extract pixel information |
| **2** | The image should be compressed by compressing it into a small file size and turned to black and white | Grey pixels introduce a certain uncertainty into whether the pixel is worth analyzing or not. By removing redundant grey pixels and setting the darker greys to black, it is possible to |
| **3** | Trace the outside of the largest quadrilateral | To identify the quadrilateral, you need to be able to first identify whether the black outline of the square is the main puzzle |
| **4** | Identify the corners of the quadrilateral | So that further number analysis and puzzle structure can be identified.  And so that the puzzle can be analyzed independently of the background, it is irrelevant what surrounds the quadrilateral |
| **5** | Rotate and cropping the puzzle out of the image | So that the background is cut out. All that is left in the image is the puzzle that will be analyzed; I'm only left with the parts that I care about; the puzzle, not random noise surrounding it |
| **6** | Extract numbers from each small square of the sudoku puzzle | So that I can classify the number (I should first extract the number before trying to identify it. |
| **7** | Creating a number handling object that will store the number that has been cropped out of the puzzle | To do everything to do with number operations, such as optimizing the number and getting rid of the white space around it, identifying the number and scaling it up to a certain size |
| **8** | Create a template structure to hold template number information | So that I can compare the numbers to the template when I'm trying to classify it |
| **9** | Classify numbers | So that the numbers from the image can be transferred to an array where it can be printed or further processed. For example is it a 1 or 2? |
| **10** | Insert numbers into the image | So that the solution (if there is one) or numbers can be pasted into the squares of the image. It should see whether there is any data there already before pasting. |
| **11** | Must be able to update the number grid after inserting new numbers | So that this number of data can be displayed on a more neat number grid that is drawn in execution for example |

**Optional Targets:**

|  |  |  |
| --- | --- | --- |
| **1** | Implement a machine learning approach for classifying the numbers | For 99% accuracy on number classification |
| **2** | Solve the sudoku puzzle | From the numbers obtained from the image processing get missing square values |

**Problems that might arise from the system that I will design:**

Programming a number recognition project without the use of neural networks will be a tough thing to do well; there will be anomalies in the digits in the sense that human handwriting, an unfortunate angle or any other factor will skew the digit from being recognized by a template. In my project, I will be looking for an 80% success rate in identifying numbers of a puzzle.

I cannot program a solver for this project, because the poor image digit recognition will affect the final outcome, so it cannot be used as its result won't reflect the actual solution. Besides, this project focuses on computer vision and not solutions to sudokus.

This code will also be running under the following assumptions:

* The sudoku puzzle will be black on a white background with minor noise, any noise that might be considered a closed body might corrupt results
* The photo will be an image that is straight on, with no tilt.
* The photo will be in the correct orientation, not upside down or sideways, although it may be tilted up to 45 degrees left or right
* The numbers in the puzzle are printed and so will not have any anomalies such as flicks of the hand etc.
* The puzzle won't be in a fancy font; it'll be regular numbers from the most common font