Augmented Reality Sudoku Solver

Tarun Raheja, Rohan Saraogi

December 14, 2023

Abstract

This study presents an Augmented Reality (AR) Sudoku Solver that integrates image processing and machine learning for the detection and solution of Sudoku puzzles in both static images and video feeds. The system employs classic image processing techniques and a convolutional neural network (CNN) for grid and digit extraction. For puzzle solving, an efficient implementation of Donald Knuth's Algorithm X, using Dancing Links, is utilized. An innovation is the integration of a distance-based grid tracking algorithm for dynamic video analysis, enhancing real-time processing capabilities. The system supports detection of multiple grids in the same image, detection of handwritten and printed digits, and demonstrates robustness in varied lightings, orientations, and Sudoku media (printouts, newspapers, computer screens). All of this is done without significant resources using a smartphone camera wirelessly connected to a MacBook Air running on CPU.

1 Introduction

The classic Sudoku is a puzzle game consisting of a 9x9 grid partially filled with numbers from 1 to 9. The objective is to fill the grid in a way such that every row, column, and 3x3 subgrid contains every number from 1 to 9 exactly once. The puzzle has a very large audience and is commonly found in print media including newspapers and books. In this form, finding immediate solutions can be challenging and time-consuming for enthusiasts. In an era where cameras have become an integral part of daily life, the opportunity to integrate them into Sudoku solving is an interesting avenue to explore. Augmented Reality (AR), an emerging technology that superimposes digital information onto the real world, provides a perfect platform to enhance this experience. And as avid Sudoku solvers ourselves, we find the prospect of enhancing the Sudoku solving experience using AR to be very compelling.

To this end, our project aims to build an end-to-end pipeline that employs image processing and machine learning-based computer vision techniques. The objective is to automatically detect Sudoku grids from images or video feeds, extract the grid along with the digits, solve the puzzle, and subsequently overlay the solution back onto the original image or video. By merging the realms of traditional puzzle-solving with AR, we seek to enrich the user experience of tackling Sudoku puzzles.

We differentiate ourselves from previous work by building an integrated system for both images and video. We provide support to extract multiple grids of different sizes from the same image. We also use our own implementation of a very efficient Sudoku solver and a grid tracker to work smoothly with hard Sudokus in real-time with video.

2 Related Works

Various approaches have been employed for Sudoku puzzle detection and solving. In 2012, the authors in [**simha2012**] used efficient area calculation techniques to identify the Sudoku grid, a template matching algorithm to identify digits within each cell, and solved the Sudoku using backtracking. The authors in [**kamal2015**] used similar ideas with a focus on comparing three Sudoku solving algorithms: backtracking, simulated annealing, and genetic algorithms. The authors in [**wicht2015**] utilized Hough transforms and contour detection techniques for puzzle detection, and a convolutional deep belief network for digit classification.

We took inspiration from these works to build our grid extractor using classic image processing techniques. For digit classification, we adapted a CNN based model architecture of one of the best performing models on Kaggle's **Digit Recognizer** competition. For solving the extracted grid, we used our own implementation of a very efficient Sudoku solver built using the **Dancing Links** implementation of Donald Knuth's Algorithm X. And finally, for grid tracking, we used a **distance based algorithm**. We present details for each of these components in the Methodology section.

3 Methodology

Our system design is shown in Figure 1. The design works for both images and video, with the caveat that for video the digit extractor and Sudoku solver components work in tandem with grid tracking. We discuss each component below.

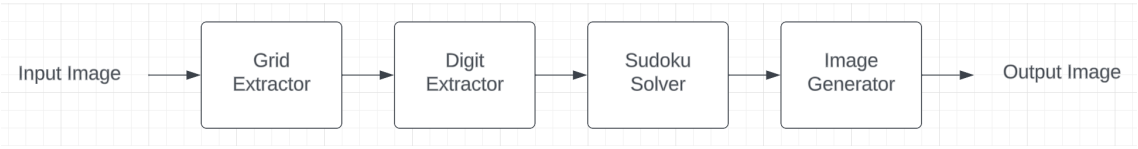

Figure 1: System Design

3.1 Grid Extractor

We built our grid extractor using classic image processing techniques.

1. As initial setup, we used gray-scaling, normalization, Gaussian blurring, adaptive thresholding, and morphological transformations. We started with off-the-shelf parameters from OpenCV documentation and online tutorials and performed some parameter tuning using trial-and-error.

2. We then used contour based techniques to extract grid corners using OpenCV's `findContours` function. One of our innovations was to leverage contour area to extract multiple grids. In particular, we sorted contours in descending order of their area and considered all contours with area greater than a threshold. By default, the threshold was set to 1/8th of the maximum contour area in the image. This allowed grids of multiple sizes to be detected and also served as a latency parameter: setting the threshold to a value that was too low allowed smaller grids to be detected but also increased system latency.

3. For each set of extracted grid corners, we used a perspective transform on the original (without pre-processing) image to extract the grid as a 450x450 image and computed the inverse transform matrix. These extracted grid corners, 450x450 images, and inverse transform matrices were passed on to components later in the pipeline.

3.2 Digit Extractor

1. Similar to the grid extractor, we used gray-scaling, normalization, Gaussian blurring and adaptive thresholding as initial setup for each grid extracted by the grid extractor. The parameters were the same as for the grid extractor.

2. We then used contour based techniques using OpenCV's `findContours` function to extract 28x28 digit images.

3. We then used a CNN based model for digit classification. We passed each digit image through the model and classified the image as per the model prediction if the associated prediction probability was at least 98% (ignoring the prediction otherwise). The model architecture is shown in Figure 2. For training, we experimented with both the MNIST and Char74k datasets, used a train/validation split of 90%/10%, RMSprop as the optimizer, and categorical cross-entropy as the loss function. We used data augmentation (shown in

Figure 3) and trained the model for 30 epochs with Keras' `ReduceLROnPlateau` callback with a patience value of 3. The configuration parameters were adapted from one of the best performing models on Kaggle's **Digit Recognizer** competition.

4. We then checked if there were at least 17 classified digits (which is needed to ensure that the Sudoku has a unique solution), and ensured that the classified digits didn't violate the Sudoku constraints. If these checks were successful, we passed the grid to the next component.

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5, 5), padding="Same", activation="relu", input_shape=(28, 28, 1)))
model.add(Conv2D(filters=32, kernel_size=(5, 5), padding="Same", activation="relu"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), padding="Same", activation="relu"))
model.add(Conv2D(filters=64, kernel_size=(3, 3), padding="Same", activation="relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
optimizer = RMSprop(learning_rate=0.001, rho=0.9, epsilon=1e-08)
model.compile(optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"])
```

Figure 2: Digit Classification Model & Optimizer

```
image_data_generator = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=False,
    vertical_flip=False
)
image_data_generator.fit(X_train)
```

Figure 3: Data Augmentation

3.3 Sudoku Solver

One of our innovations was to use our own implementation of a very efficient Sudoku solver built using the **Dancing Links** implementation of Donald Knuth's Algorithm X. The efficiency of the solver was crucial for our system to work for hard Sudokus in real-time with video. Our solver also gracefully handled issues with digit classification, returning one solution if the extracted grid had a non-unique solution, and no solution if the grid was unsolvable. We passed each of the extracted grids to the solver, and the solved grid (if the grid was solvable) to the next component.

3.4 Image Generator

Using each of the solved grids and associated inverse perspective transform matrices, we added the solved digits to the original image and returned the generated image for display.

3.5 Grid Tracker

One of our innovations was to use a distance based algorithm to track grids to avoid running the entire pipeline for every video frame. This helped avoid running the digit extractor and Sudoku solver every time a grid was extracted. For a detailed look at the algorithm, please check **here** and the project code. We summarize the broad ideas of the algorithm below.

1. The algorithm maintains a cache of solved grids. For each grid, it stores the extracted corners, the 450x450 transformed image, the inverse perspective transform matrix, the unsolved grid, the solved grid, and the no. of frames since it was last detected.

2. The grid extractor is run for every frame and each extracted grid is uniquely matched with the nearest cached grid based on the Euclidean distance between their extracted corners. To avoid grids that are far away from getting matched with each other, matches are only valid if the distance is bounded by `max_distance` (set to 100 by default). The cached values for a matched cached grid are updated with the values for the matched extracted grid.

3. Unmatched extracted grids are passed through the digit extractor and Sudoku solver and added to the cache if they have solvable grids. Likewise, unmatched cached grids are removed from the cache if they haven't been matched for more than `max_disappeared` consecutive frames.

4 Experiments & Results

Before discussing our experiments and results, we show sample results for our system in Figure 4. Our system was robust and versatile, being able to extract Sudokus in printouts, newspapers, computers, and in different orientations. As can be seen in the top left image, it was also able to extract handwritten digits.

Figure 4: Sample results for our system. Our system was robust and versatile, being able to extract Sudokus in printouts, newspapers, computers, and in different orientations. As can be seen in the top left image, it was also able to extract handwritten digits.

We now discuss key design choices, experiments, and results.

4.1 Adaptive Thresholding

A key design choice in image processing prior to grid and digit extraction was our use of adaptive thresholding. As shown in Figure 5, unlike simple thresholding, adaptive thresholding allowed us to extract grids for different lighting conditions and also when some of the cells in a Sudoku grid had different background color compared to other cells.

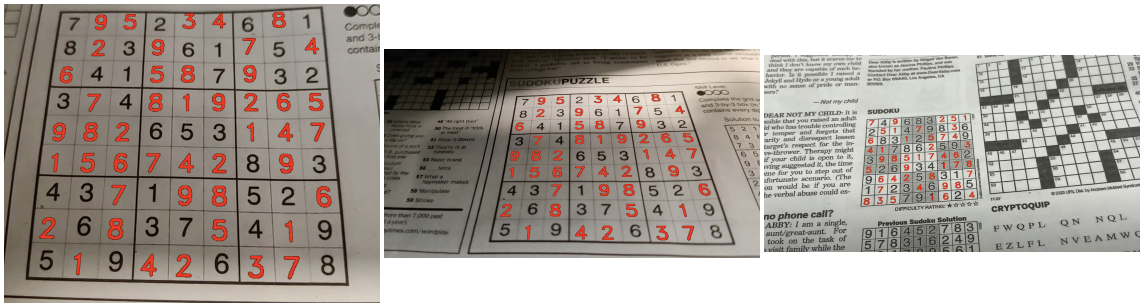

Figure 5: Adaptive thresholding allowed us to extract grids for different lighting conditions and also when some of the cells in a Sudoku grid had different background color compared to other cells.

4.2 Contour Area Threshold

A key design choice while implementing the grid extractor was the minimum threshold for the contour area for extracted grid corners. Setting the threshold to a value that was too low would potentially increase the no. of extracted grids and increase system latency. On the other hand, setting it to a value that was too high would prevent valid Sudokus of different sizes from being detected in the same image. Based on our experiments, setting the threshold to 1/8th of the maximum contour area in the image worked well for us. As shown in Figure 6, this allowed us to extract grids that were slightly far from the camera and multiple grids of different sizes in the same image. This would not be possible if we set the contour area threshold to a higher value eg. 1/2 of the maximum contour area in the image.

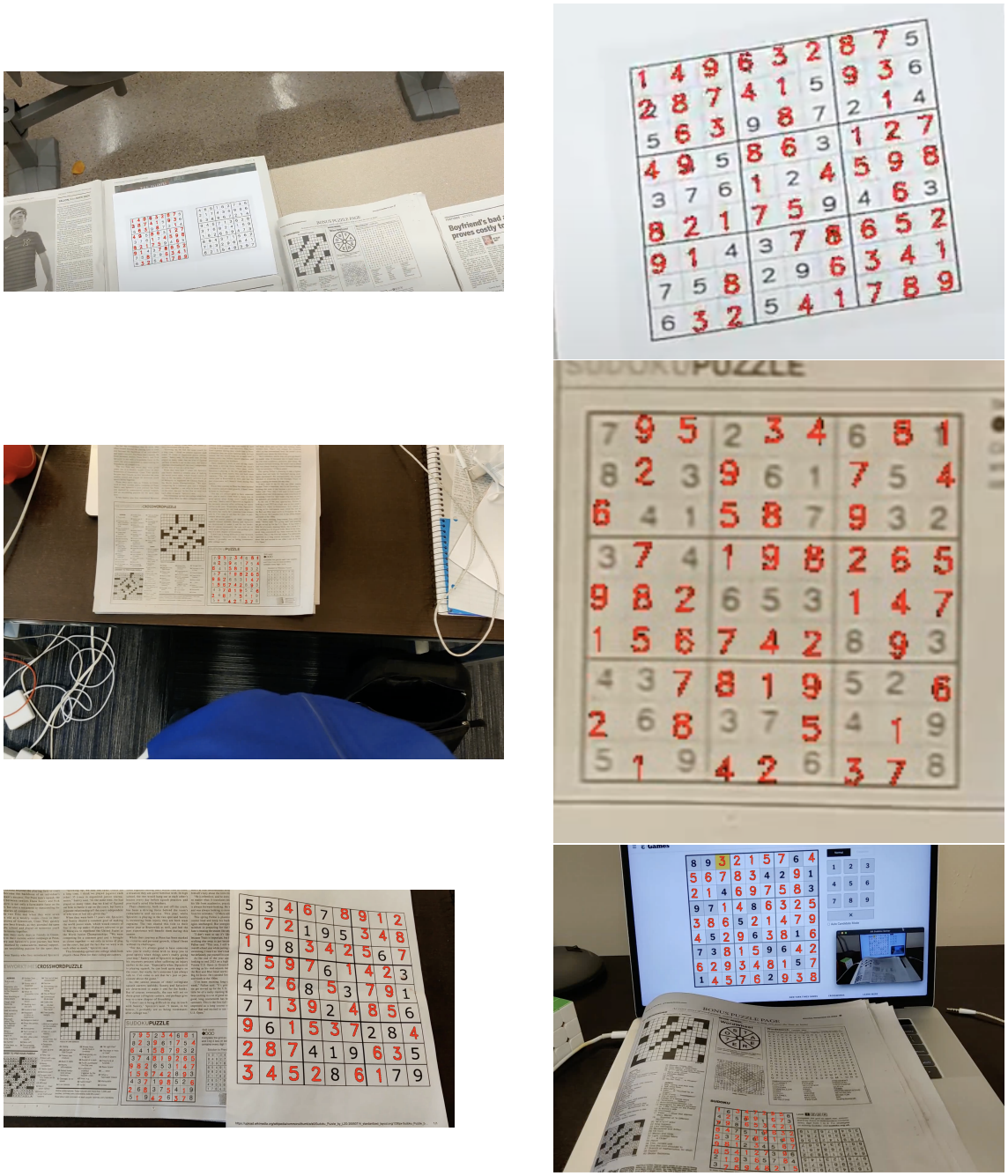

Figure 6: For the first two rows, the images on the left contain grids at some distance from the camera and the associated images on the right are zoomed in on the grid to show correctness of the extraction. For the last row, the images contain multiple extracted grids of different sizes.

4.3 Digit Classification

We experimented with training our digit classification CNN model on both the MNIST and Char74k datasets. In each case the model achieved near perfect performance. Figures 7-10 show the train and validation loss and accuracy curves and the validation confusion matrices. We noticed no substantial differences in the performance of both models in our experiments and chose to use the Char74k model as our final model.

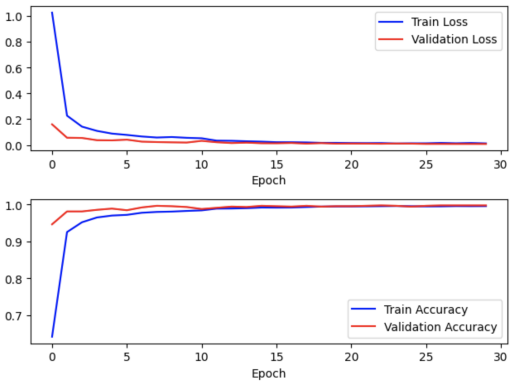

Figure 7: Char74k Train/Val. Loss/Acc. Curves

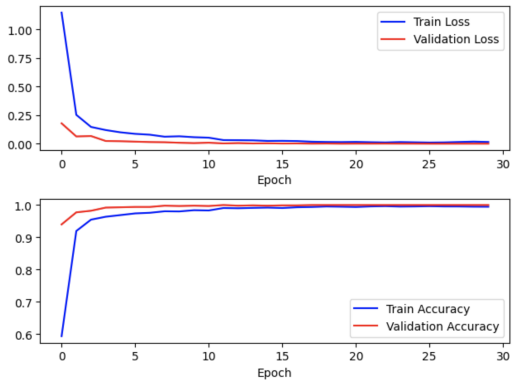

Figure 8: MNIST Train/Val. Loss/Acc. Curves

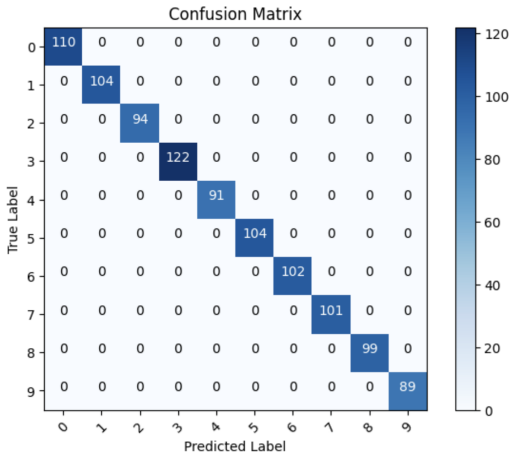

Figure 9: Char74k Val. Confusion Matrix

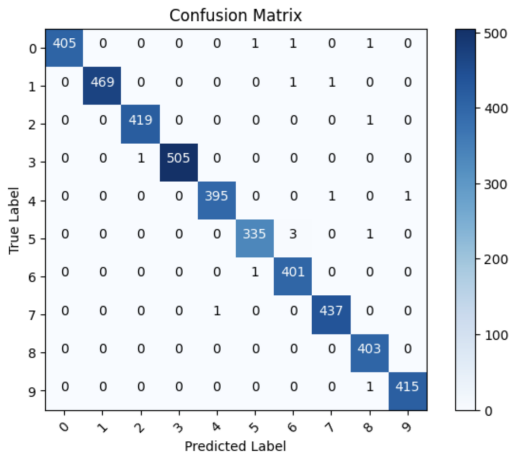

Figure 10: MNIST Val. Confusion Matrix

4.4 Grid Tracking

As mentioned previously, we introduced grid tracking to avoid running the digit extractor and Sudoku solver for every extracted grid. We have included video snippets with the project deliverables to compare the performance of our system with and without grid tracking. Looking at the video snippets, we notice that without tracking the solved grid flickers in and out (and the solved digits potentially change) depending on whether the grid/digits were correctly extracted for that image. This problem is subtantially reduced with grid tracking, leading to a smoother experience.

A key design choice while implementing the tracker was the value for `max_distance`, the maximum allowed distance between the extracted corners for extracted and cached grids to be matched with each other. Setting `max_distance` to a value that was too low would decrease matches and defeat the purpose of tracking. On the other hand, setting it to a value that was too high could result in incorrect matching. An example of incorrect matching is shown in Figure 11. Assuming that the camera is moving gently, setting `max_distance` to 100 worked well for us. We think this assumption is reasonable as allowing for rapid camera movements increases the likelihood of grids being incorrectly matched and the system (as it is currently designed) cannot correct for this as it is not performing digit extraction and solving the Sudoku for grids once they have been cached.

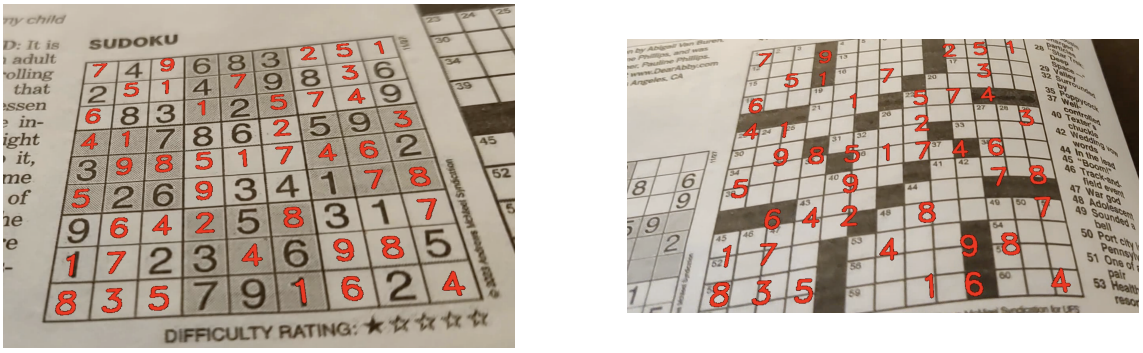

Figure 11: If `max_distance` is too high (2000 here), the tracker can incorrectly match an extracted grid to a cached grid if the cached grid goes out of focus. The originally extracted and cached grid is shown on the left. When the camera is moved and the grid goes out of focus, the tracker incorrectly matches the grid with a crossword grid that is nearby.

4.5 Sudoku Dataset

We used a Sudoku dataset from **here** as one method to validate our pipeline. The dataset consists of 200 Sudoku images and associated extracted grids. Our system was able to extract 67% of the grids in the dataset and correctly classify 86.7% of the digits extracted (for 20.9% of the extracted grids it was able to correctly classify every single digit). The confusion matrix for digit classification for the dataset is shown in Figure 12.

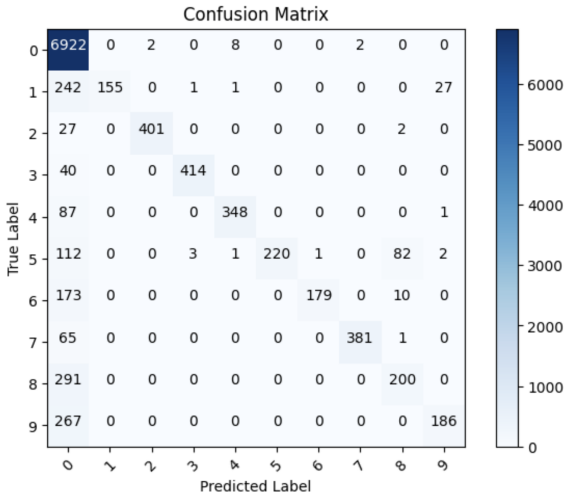

Figure 12: Sudoku Dataset Digit Classification Confusion Matrix

We were quite happy with these results as many of the images in the dataset are of very poor quality. As can be seen in the confusion matrix, the primary issue was that the digit classification model wasn't sure about many digits and therefore classified them to 0 by default (it needed at least 98% predicted probability to classify a digit image to a non-zero value). And the lack of off-diagonal values otherwise suggests that when the model was confident about a digit image, it was generally correct.

In Figure 13 we show sample results from the dataset.

Figure 13: Sample results from the Sudoku dataset. The top row shows some images for which the system worked perfectly. The bottom row shows some images that the system got partially correct. The empty cells in this row suggests that the system incorrectly extracted a digit when it wasn't there. The filled digits superimposed on already provided digits suggests that the system failed to extract the corresponding digits. The two images at the bottom right show how poor the image quality for some of the images in the dataset can be.

5 Qualitative Analysis

Most of our design decisions were guided by the use-case and target audience we had in mind - to provide an augmented reality experience to Sudoku lovers. We highlight and analyse our decisions in the context of this use-case below. We also mention some of the implementation challenges we faced.

5.1 Low Resource Computation

We built our system with the intent that it should not consume significant resources. To this end, we used efficient techniques including classic image processing, a small digit classification model, an efficient Sudoku solver, and grid tracking to avoid repeated computations. Almost all of our experiments were conducted using a smartphone camera wirelessly connected to a Macbook Air M1 laptop running on CPU. This is in line with our expectation that the target audience would primarily use their smartphone to interact with Sudokus. While we have not had the time to do so, an obvious next step in this project would be to integrate the code and deploy it directly on the edge.

5.2 Real-time Video Experience

While it would be possible to get the AR experience by clicking images of Sudokus and passing it through our system, we felt the experience would be much better with real-time video. Hence a key objective of ours was providing support for both static images and real-time video. The aforementioned efficiencies helped achieve this, with the grid tracker playing a crucial role. We would not expect our user to be holding their camera in a perfectly stable manner, and the grid tracker helped ensure a smooth video experience in the presence of minor instabilities. We also invested noticeable time in error/edge case handling and code robustness to minimize the chances of system crashes.

5.3 Multiple Grids Support

We do not expect our users to want to extract multiple Sudoku grids of different sizes at the same time. While our system does support this, one of our motivations for adding this functionality was driven by the fact that we would expect there to be multiple (potentially bigger) grids in the image other than the Sudoku eg. if the user was pointing their camera at something grid-like such as a laptop screen or a newspaper containing a Sudoku, or the Sudoku wasn't extremely close up to the camera. We designed our grid extractor and set the contour area threshold to support this.

5.4 Grid & Digit Extractor Versatility

We attempted to make our extractor versatile, being able to extract grids from different orientations, lighting conditions, and media (print-outs, newspapers, laptops etc.), and also have some support for handwritten digits. The simplicity of our approach came with its own limitations. For example, our system fails to work with deformed grids eg. if the grid is folded or crumpled or extreme orientations eg. if the grid is flipped. There are some other quirks eg. our system struggles if the grid borders and digits are white and the background is black. We argue that these are reasonable limitations. We would expect our users to have their grid on a relatively flat surface, at a reasonable orientation, and with the digits and borders in a darker color with a lighter background.

5.5 Lack of Sudoku Image Data

Other than the 200 images Sudoku dataset we have mentioned in the Experiments & Results section, we couldn't find any Sudoku images dataset. This limited our ability to experiment with deep learning techniques for grid extraction and to validate our model in a more exhaustive manner.

5.6 Privacy

Given that one use-case of our system is with real-time video, it has a potentially intrusive aspect as it is tracking what the user is pointing their camera to. Assuming that we make the necessary changes to deploy our system on the edge, it doesn't require any user-specific data for functioning and is self-contained without the need to communicate with any other resource. Hence it also protects user privacy.

6 Conclusion

In this study, we developed an innovative Augmented Reality (AR) Sudoku Solver, effectively combining image processing and machine learning to automate the detection and solution of Sudoku puzzles in both static and dynamic video formats. The system's core, an image processing and CNN based grid and digit extractor, is complemented by an efficient implementation of Donald Knuth's Algorithm X with Dancing Links, which provides rapid and reliable puzzle solving capabilities. An important component of our work is the development of a grid tracking algorithm tailored for video feeds. This algorithm noticeably improves the stability and processing efficiency of the system in real-time scenarios, addressing one of the primary challenges in AR applications. The system's capability to handle multiple grid sizes and adapt to diverse lighting conditions and orientations demonstrates its robustness and adaptability, making it a versatile tool for Sudoku enthusiasts.

Working on this project gave us the opportunity to gain some experience with building an end-to-end system, focusing on efficiency and robustness. This entailed grappling with different design choices and associated trade-offs such as the contour area threshold and the grid tracking parameters. We also got the opportunity to work with image processing and computer vision based techniques on a real-world problem. While any given component of our system was not particularly complex, we were quite happy with how they were stitched together and the performance of the overall system. Naturally, simplicity came with its own trade-offs such as for example the difficulty in working with deformed grids and adverse grid orientations. There is scope to work on each component and to also integrate the system and deploy it on the edge. These are future directions for our work.

References

[1] Simha PJ, Suraj KV, Ahobala T. Recognition of numbers and position using image processing techniques for solving Sudoku Puzzles. In: IEEE-international conference on advances in engineering, science and management (ICAESM-2012).

[2] Kamal S, Chawla SS, Goel N. Detection of sudoku puzzle using image processing and solving by back-tracking, simulated annealing and genetic algorithms: a comparative analysis. In: 2015 third international conference on image information processing (ICIIP).

[3] Wicht B, Hennebert J. Mixed handwritten and printed digit recognition in sudoku with convolutional deep belief network. In: 2015 13th international conference on document analysis and recognition (ICDAR).

[4] Knuth, Donald E. (2000). Dancing links. In: Millennial Perspectives in Computer Science.

[5] Simple object tracking with OpenCV (https://pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/).

[6] Kaggle's Digit Recognizer competition (https://www.kaggle.com/competitions/digit-recognizer).