Automated Sudoku Solver with Deep Learning for Digit Recognition

Feisal Kiiru
fkiiru26@amherst.edu

David Kairu
dkairu25@amherst.edu

Christian Manzi
cmanzi25@amherst.edu

***Abstract*—This project investigates the development of a Sudoku puzzle that utilizes a custom trained digit classifier to extract digits from various image inputs containing Sudoku puzzles. The project leverages the Open Computer Vision Library (OpenCV) for image preprocessing, grid detection, and cell segmentation. The custom digit recognition model is built with a Convolutional Neural Network (CNN) and trained on the Modified National Institute of Standards and Technology dataset (MNIST) dataset developed by Yann LeCun, Corinna Cortes and Christopher Burges. The project aims to integrate the current Sudoku solver with the OCR capability and examine ways to refine the model's accuracy to handle seemingly ambiguous puzzles. On the refinement of the model's accuracy, various methods are explored that range from the choice of a Stochastic Gradient Descent (SGD), the appropriate combination of hyperparameters and neural network layer configurations (including layer-specific parameters).**

***Index Terms*—Sudoku Solver, digit recognition, image processing, neural network, reinforcement learning, Q-learning, MNIST.**

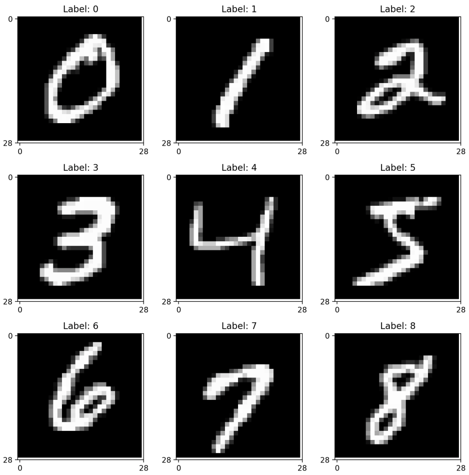


Fig. 1. Sample digits from MNIST Dataset

I. Introduction

Sudoku solving is a well-known computational challenge with applications in both recreational and educational contexts. While current solutions primarily rely on input grids, incorporating Optical Character Recognition (OCR) allows to easily digitize many puzzles either for studies that require an enormous number of puzzles such as in training neural networks or in determining an existence of their solution. Converting a 9x9 puzzle from paper to image does not seem hard of a task but as the dimension enlarge, one can easily imagine how quick and hectic the task of translating the puzzle to the correspoing machine readable format becomes. This project focuses on developing a custom digit recognition model that is extensive and easily replicable on other digit recognition tasks.Studies suggest that deep neural networks are more advantageous for such tasks that rely heavily on the capabilites of supervised learning. One particular variant of the deep learning field, CNNs, which has wide applications in image classification, object recognition, computer vision, face recognition proved to be efficient for the task that our project addresses. CNNs require minimal efforts for pre-processing, feature extraction, and classification steps. CNNs also provide considerably higher accuracy even amid the availability of insufficient training data [1]. These two factors, that will be elaborated on later in the report, served as a strong basis for the choice of CNN as the architecture to train the digit classifier underlying the project model. The project utilizes the MNIST dataset which is publicly available and popular among handwritten digit recognition studies. The project is evaluated with a focus on recognition accuracy, grid detection success rate, and the solver's overall performance when working with real-world image inputs. The rest of the report is organized as follows: Section II describes the motivation and background; Section III and Section IV describe the CNN architecture, project setup, and evaluation methodology; Section V discusses the results; and Section VI presents the conclusions.

II. Motivation and Background

Existing Sudoku solvers require manual input of the puzzle grid. This hinders accessibility for users with images containing the puzzle and, even further, for studies that rely on a dataset of puzzles in form of images. An example for the latter was our experience when building the Z3-based solver where we had to type each digit in the unsolved puzzle into a 2D array to be solved. With this project, the previous shortcomings can be alleviated. To get started, we needed a library that excels in image pre-processing tasks like resizing, grayscaling, blurring, as well as perspective transformation for skewed grids. OpenCV has been consistently used in recognition tasks due to its performance over alternatives like Python Imaging Library (PIL). To complement OpenCV's pre-processing capabilities, TensorFlow was the ideal choice for implementing our custom classifier. While traditional OCR tools like Tesseract are general-purpose, TensorFlow allows us to train a specialized CNN using MNIST data, tailored to recognizing Sudoku digits. This approach ensures high accuracy, even for uniquely styled or ambiguous digits. In the project proposal, we initially emphasized leveraging Q-learning as the training mechanism for our digit classifier. Through further exploration, we noticed the excessive complexity it introduces, particularly in regards to the high-dimensional state space associated with image data. This was complemented by studies done in [3] revealing how employing Q-Learning in traditional frameworks can lead to large state-spaces, complicating both the optimization process and the tuning of hyperparameters. In the same paper [3], the author also argues that this complexity often results in significant computational resource demands, which can hinder efficiency, especially in tasks requiring rapid digit recognition such as our framework. This led us to explore other plausible options such as CNNs and ANNs.

CNNs are increasingly favored over traditional ANNs for digit recognition tasks due to their ability to efficiently capture and learn complex spatial features and patters in image data [4]. Authors of the paper [4] continue to argue that CNNs excel in automatic feature extraction through their layered architecture, which applies convolution operations that allow the model to recognize patterns and details at various levels of abstraction. This ability to learn features, from simple edges to intricate digit shapes, positions CNNs as better candidates for our recognition task compared to ANNs, which rely on flattened inputs and do not take advantage of the spatial relationships inherent in images.

While the study [2] shows that ANNs can also achieve high recognition accuracy rates, they often require extensive pre-processing and manual feature engineering to optimize performance for image data. CNNs, by their design, eliminate the need for such overhead steps as they directly utilize the raw pixel values to generate feature maps through convolution, which simplifies the workflow [4]. Furthemore, according to the authors of [1], CNNs achieve state-of-the-art results in image classification tasks, as supported by the reported recognition accuracy of 99.87% for similar digit recognition tasks.

CNNs' architecture, including convolutional and pooling layers, not only reduce the dimensionality of data but also enhance robustness against variations in input, such as shifts and distortions. Slight distortions can greatly impact the model's recognition of an image, therefore CNNs abilities to perform well with such data points was an essential considerations for our project. The pooling mechanism ensures translation invariance—an essential property for digit recognition, as the precise positioning of digits can vary significantly in handwritten text. In comparison, ANNs may struggle with maintaining performance under such conditions, as highlighted in the insights from the authors of [2].

Overall, while ANNs can achieve commendable accuracy in specific applications, the inherent strengths of CNNs in handling image data efficiently make them the preferred choice for digit recognition tasks and hence our preferred choice for this project.

III. Methodology

The framework underlying the project is divided into five key stages: Image Processing, Grid Detection, Cell Segmentation, Digit Recognition (OCR), and Sudoku Solving.

1. Image Pre-processing: At this stage, we load the MNIST dataset to be pre-processed using OpenCV. This is done to ensure consistent input data for training the model and the following three techniques are used:

- **Normalization/Scaling:** MNIST images have pixel values ranging from 0 to 255. These values are normalized to the range of 0 to 1 by dividing each pixel value by 255. This scaling is necessary to improve the model performance and convergence during training since we ensure a consistent input scale.
- **Reshaping:** CNNs require input data in a 4D array format: (num_samples, height, width, channels). MNIST images are 28x28 grayscale images. Each image is reshaped to (num_samples, 28, 28, 1), where 1 represents the single grayscale channel.
- **One-hot Encoding:** MNIST labels are integers from 0 to 9. These values are converted to a one-hot encoded format. Each label is represented as a binary vector (e.g., '3' as [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]) to ensure steady numerical input and ouput for the model .

2. Grid Detection: The second stage involves locating and isolating the Sudoku grid within the input image using OpenCV. The objective of this step is to accurately identify the grid's position and orientation which can vastly distort the output label. This detection was done using OpenCV functions through:

- **Contour Detection:** This involves identifying the outlines of shapes within the image. By applying contour detection to a thresholded (binary) version of the image, the outlines of potential shapes, including the Sudoku grid, get extracted. The largest detected contour, assumed to correspond to the Sudoku grid, is then selected as the puzzle.
- **Perspective Transformation:** Given different angles through which images are taken, this step handles perspective distortion in the image. The coordinates of the four corners of the detected grid are used to perform a perspective transformation, warping the grid into a perfect square.

3. Cell Segmentation: This stage divides the extracted and straightened Sudoku grid into individual cells using OpenCV. The objective is to isolate each digit within its own cell. To achieve this, the grid is divided into a 9x9 matrix of cells by calculating the width and height of each cell and extracting each as a separate image to be classified by the model. The cells are resized to ensure they match the initial train and test data format (MNIST data point format). At this stage, a potential digit-containing cell is isolated into its own image for the digit recognition stage.

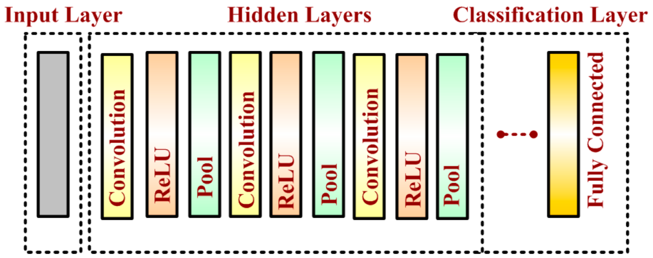
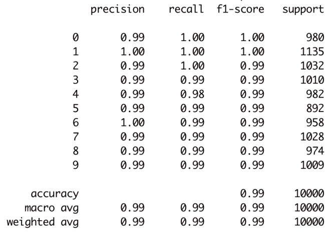


Fig. 2. Convolutional Neural Network Architecture. Source: [1]

4. Digit Recognition (OCR): This stage employs a custom-built image classifier based on CNN architecture. Instead of using reinforcement learning (such as Q-Learning) as previously stated in the proposal, this implementation utilizes a pure CNN architecture, as research has demonstrated its effectiveness when parameters are carefully optimized [1]. CNN architeture allows for the extraction of vital features from the input images through the process to be described. The model begins with multiple convolutional layers; the first applies 32 filters with a kernel size of (5x5), while the second layer utilizes 64 filters of (3x3). Both layers employ Rectified Linear Unit (ReLU) activation functions to introduce non-linearity, which enhances the network's capability to learn intricate patterns. Following these convolutional layers, pooling layers with a (2x2) pool size)reduce the dimensionality of the feature maps. The pooling operation helps ensure that extracted features maintain minor positional shifts, which is particularly beneficial in digit recognition tasks.

The model transitions to classification through fully connected layers, including one layer with 128 neurons and another with 64 neurons, both activated by ReLU. This step aggregates features extracted by previous layers for final classification. Incorporating dropout layers with a 25% rate further mitigates overfitting by randomly "dropping out" neurons during training, encouraging the learning of more robust features—an approach supported by the study which found that finely tuned pure CNN models achieve greater recognition accuracy compared to more computationally expensive ensemble architectures [1]. For the training process, the model uses a learning rate of 0.001, an epoch size of 20, and a batch size of 128, which are supported by the study in [1]. The Adam optimization algorithm is employed due to its adaptive learning rate capabilities, offering improvements in convergence speed and overall training performance.

5. Sudoku Solving: This stage utilizes the pre-built Python Sudoku solver (from a class project) to solve the extracted puzzle. The objective is to generate and display the solution to the Sudoku puzzle, if the extracted puzzle is valid (i.e the digits have been accurately classified). The extracted digits, represented as a 2D array (9x9 grid), are fed into the solver, which returns a 2D array solution. Both arrays are display once the code is run in the terminal.

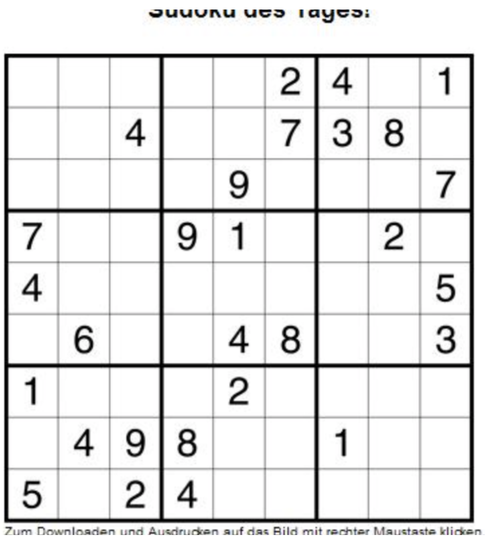


```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       980
           1       1.00      1.00      1.00      1135
           2       0.99      1.00      0.99      1032
           3       0.99      0.99      0.99      1010
           4       0.99      0.98      0.99       982
           5       0.99      0.99      0.99       892
           6       1.00      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

Fig. 3. Performance Metrics Per Label



Sudoku des Tages:

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

Zum Downloaden und Ausdrucken auf das Bild mit rechter Maustaste klicken

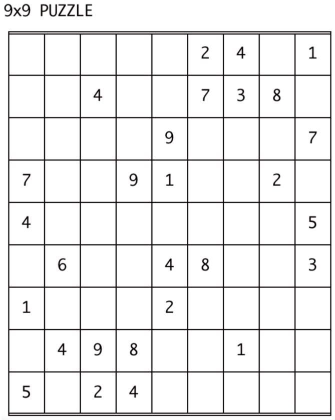
Fig. 4. Sample image of a Sudoku Puzzle
source: Link on Github Repo

IV. Evaluation Methodology

We evaluate the success of our project using two primary metrics: the **Accuracy of the Classifier** and the **Cell Extraction Success Rate**. These metrics provide an assessment of the system's performance and highlight its effectiveness in solving the overall problem.

First, we measure the **Accuracy of the Classifier**, focusing on its ability to correctly identify digits within Sudoku cells. We test the classifier on a diverse set of images (a sample of which are shown in the Table 1 with the model's performance on each) to ensure generalization. We visualize the classifier's performance using a confusion matrix, displayed as a heat map in Fig. 7. This heat map highlights the areas where the model performs well and where misclassifications occur, helping us identify specific digit pairs that may require further refinement.

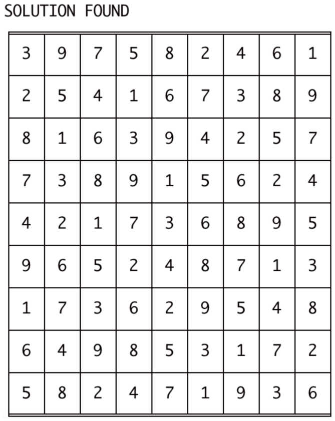
Next, we assess the **Cell Extraction Success Rate (CESR)**, which encompasses both grid detection and cell segmentation. This metric evaluates how reliably the system detects the Sudoku grid and segments it into individual cells. Since successful cell extraction relies on accurate grid detection, this combined measure provides a streamlined way to evaluate the framework performance. We calculate the CESR as the percentage of correctly and clearly segmented cells for each puzzle. To ensure an extensive evaluation of our framework, the images used are selected from a sample of puzzles with



9x9 PUZZLE

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

Fig. 5. Digit Classification Output of Sudoku Puzzle in Fig. 4



SOLUTION FOUND

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

Fig. 6. Solution to Sudoku Puzzle in Fig. 4

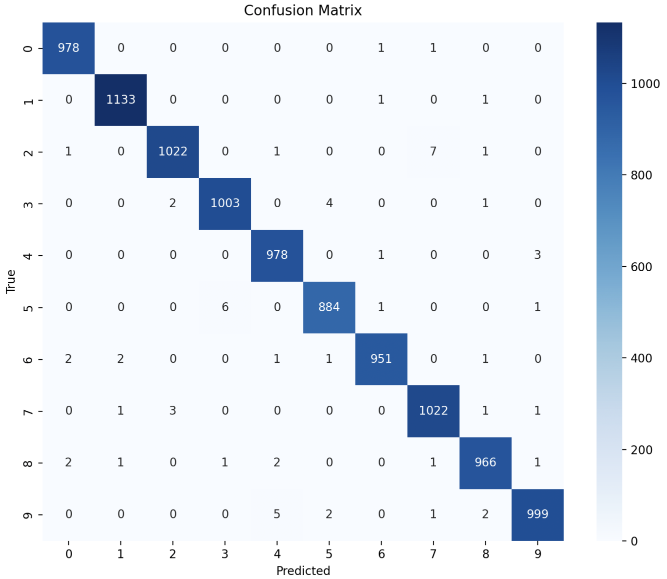
varying conditions: challenging orientations, lighting conditions, and image qualities.

$$\text{CESR }(\%) = \left(\frac{\text{Num\_Correctly\_Classified\_Digits}}{\text{Num\_Initial\_Digits}}\right) \times 100$$

V. Results

We present the results of our evaluation with a sample example of Sudoku puzzle in Fig. 4, the OCR'd version in Fig. 5, and the solution in Fig. 6. The confusion matrix on the test set shown in Fig. 7 offers a clear and intuitive visualization of the model's classification accuracy and performance when classifying different digits. This graphical representation is useful for identifying patterns in misclassifications, such as confusion between digits like "3" and "5" or "4" and "9."

For the accuracy in cell segmentation, our framework fails to correcly classify some digits due to misclassifications or unclear images (as illustrated in Table 1). The model achieves an average CESR of 94.77% across different images shown in Table 1. The images include different variations such as clear and well lit (p1_clear), tilted (p2_tilt), small (p3_small), blurry (p4_blur and p6_blur), and misaligned (p5_mis). The project code and installation/execution guide are hosted on Github and can be accessed via https://github.com/cmanzi00/cosc-241-final-project



Confusion Matrix

True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	978	0	0	0	0	0	1	1	0	0
1	0	1133	0	0	0	0	1	0	1	0
2	1	0	1022	0	1	0	0	7	1	0
3	0	0	2	1003	0	4	0	0	1	0
4	0	0	0	0	978	0	1	0	0	3
5	0	0	0	6	0	884	1	0	0	1
6	2	2	0	0	1	1	951	0	1	0
7	0	1	3	0	0	0	0	1022	1	1
8	2	1	0	1	2	0	0	1	966	1
9	0	0	0	0	5	2	0	1	2	999

Fig. 7. Test Confusion Map

Puzzle	num_initials_digits	num_class_digits	num_unclass	puzzle_solved
p1_clear	21	20	1	Yes
p2_tilt	33	33	0	Yes
p3_ small	27	27	0	Yes
p4_blur	26	23	1	No
p5_mis	32	30	8	No
p6_blur	33	30	0	No

TABLE I
Cell Extraction Data for different puzzles

VI. Conclusions

Our project successfully demonstrates the ability to process Sudoku puzzles from images, achieving an average CESR of 94.77% cell segmentation rate across a sample of six different puzzles. The classifier accurately extracts the puzzle grid and analyzes individual cells in most test images. Similarly, our custom classifier performs well in recognizing digits, as evidenced by the consistency and detailed insights from the confusion matrix heat map (Fig. 7).

However, our results highlight certain limitations in the OpenCV-based approach. While effective for clean, well-lit, and properly aligned images, our classifier struggles with subtle variations in lighting, angle, or distortion. These challenges occasionally lead to failures in grid detection and inaccurate cell segmentation, impacting the overall system's reliability. This inconsistency underscores the need for methods that can adapt to diverse image conditions.

Therefore, the next steps would be to explore a composition of our digit classifier with reinforcement learning, which we believe can handle a significant portion of the problem effectively. The reasoning behind this proposition is the adaptability of reinforcement learning which can enable the system to adjust dynamically to variations in image quality, improving grid detection and cell extraction in challenging cases.

In conclusion, while our current work demonstrates substantial progress in solving Sudoku puzzles from images, further refinements, particularly through the integration of RL, hold great potential for improving the robustness and versatility of our digit classifier.

References

[1] Ahlawat, Savita, et al. ”Improved handwritten digit recognition using convolutional neural networks (CNN).” *Sensors* 20.12 (2020): 3344.

[2] Islam, Kh Tohidul, et al. ”Handwritten digits recognition with artificial neural network.” *2017 International Conference on Engineering Technology and Technopreneurship (ICE2T)*. IEEE, 2017.

[3] Hafiz, Abdul Mueed. ”Image Classification by Reinforcement Learning With Two-State Q-Learning.” *Handbook of Intelligent Computing and Optimization for Sustainable Development* (2022): 171-181

[4] Krichen M. Convolutional Neural Networks: A Survey. *Computers*. 2023; 12(8):151. https://doi.org/10.3390/computers12080151

[5] OpenCV Documentation, Available from https://docs.opencv.org.

[6] Tensorflow Documentation, Available from https://www.tensorflow.org