ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

IT4342E - Computer Vision

Hanoi University of Science and Technology

School of Information and Communications Technology

Digit recognition and its application to sudoku solving

Group
Johannes Herz 2023T036
Billy Tom Herrmann 2023T033
Toni Steven Dix 2023T038
Daniel Stierle 2023T030
Yannik Elias Hanel 2023T039

Instructor:
Prof. Nguyen Duc Dung

Contents

Abstract

1 Introduction

The integration of computer vision, deep learning, and image processing has paved the way for innovative problem-solving solutions across various fields. Our project tackles the challenge of sudoku solving, employing state-of-the-art technologies to create a comprehensive system that combines image processing, deep learning, and logical puzzle-solving methods.
Our approach begins with extracting the sudoku puzzle using image processing techniques like thresholding, contour finding, and image transformation. This initial step outlines the puzzle's structure, laying the groundwork for subsequent analyses. Following this, we extract individual cells from the puzzle and employ deep learning models to classify the digits within each cell. These models play a crucial role in accurately identifying and categorizing the numbers in the sudoku grid.
To achieve a complete sudoku-solving solution, our project incorporates a recursive sudoku solver. This component synthesizes the outputs of the deep learning models, providing a systematic resolution to the puzzle. Going beyond mere solving, our system introduces sudoku image remapping. This process seamlessly integrates the solved sudoku grid back onto the original image, offering a visually intuitive representation of the solution within the context of the puzzle.
To be able to train the used deep neural networks our project involves generating large amounts of training data. This step is essential for equipping our models with the ability to accurately recognize and classify digits within sudoku grid cells. The generated data serves as the foundation for training, enhancing the adaptability and robustness of our deep learning models to the complexities of sudoku puzzles.

2 Solver

2.1 Sudoku Detection and Extraction

The identification and extraction of Sudoku grids from digital images play a crucial role in the development of automated Sudoku-solving algorithms. This section presents a systematic methodology aimed at improving grid visibility, recognizing potential grid contours, and fine-tuning the selection process to isolate the Sudoku grid for subsequent analysis. To provide a more in-depth comprehension of the procedural steps involved, we will clarify the process using a sample input image of arbitrary size and RGB color

space, as illustrated in Figure 1.

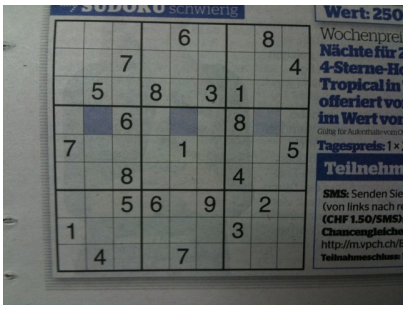

Figure 1: Sample input image of arbitrary size and RGB color space

In the continuous process of identifying a Sudoku grid contour from an input image, the initial step involves converting the image to grayscale and thereby simplifying the RGB image to a single intensity per pixel. Subsequently, a Gaussian blur filter is applied to reduce noise and enhance overall grid visibility. Following this, adaptive and binary thresholding is employed to segment the grid from the background, taking into account variations in lighting conditions. The resulting binary image is inverted to emphasize grid lines, facilitating the subsequent identification of contours.

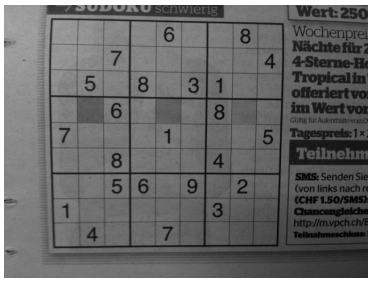

Figure 2: Gray Scale input image

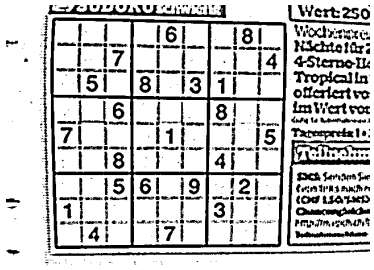

Figure 3: Thresholded input image

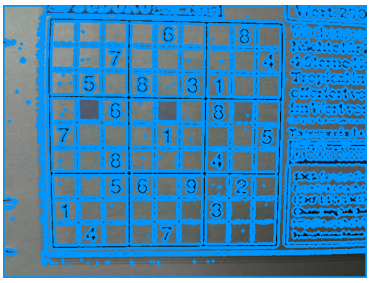

Figure 4: Contours found in input image

In the analysis of the sample image presented in Figure 4, the identified contours encompass a variety of shapes, extending beyond the specific Sudoku grid contour our program is designed to process further. To narrow down our focus, we employ threshold-based selections to reduce the total number of potential Sudoku contours. This selection process comprises two distinct preselection methods.
The initial method eliminates contours that are either excessively large or too small to be considered a Sudoku grid contour. Exclusion of contours that closely match the size of the image is crucial because the contour-detection method not only identifies shapes within the image but also includes the outline of the image itself as a separate contour. Consequently, contours with a size similar to that of the image are excluded as potential

Sudoku grids. Additionally, contours that are considered too small are not processed further. This decision is based on the assumption that such small contours either do not represent the Sudoku grid, or even if they do, we opt not to process them to optimize the performance of subsequent Sudoku-solving steps.
The second selection method addresses the presence of contours that pass the initial size check but fail to represent a square, and consequently, a Sudoku grid. To address this, the ratio between the width and height of the contours is examined. This step aims to eliminate contours that do not exhibit square-like proportions. It is important to note that if in this final selection multiple squares are detected, the square occupying the largest area is designated as our Sudoku grid for further processing.

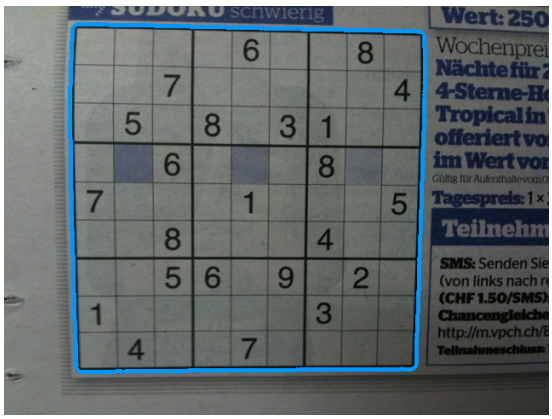

Figure 5: Sized checked contours

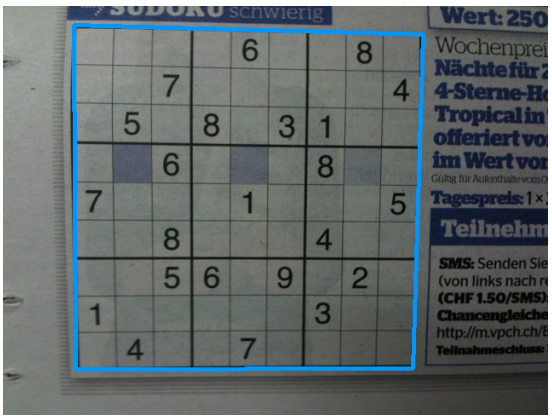

Figure 6: Shape checked contours

In the concluding phase of this step, the identified contour is used to extract the Sudoku grid from the original image and transpose it onto a new canvas with a predetermined size of 900x900 pixels. This step holds significance as it standardizes the initially arbitrary-sized input image, concentrating on the specific region of interest.

Figure 7: Gray scale Sudoku extracted from input image

2.2 Cell Extraction and Classification

To advance image processing and construct an internal representation of the Sudoku's digits and their positions within the grid, it is essential to isolate individual cells from the obtained Sudoku image. Utilizing this cell extraction, a deep learning model can then classify the cells based on their numerical values, thereby creating the internal representation of the identified Sudoku board.
The extraction of individual cells from the obtained Sudoku grid involves a straightforward sub-imaging approach. This entails treating every 100x100 pixel segment of the image as an independent cell of the Sudoku. This method is feasible due to the uniformity established in the previous step and the general information that cells in every Sudoku share the same size.

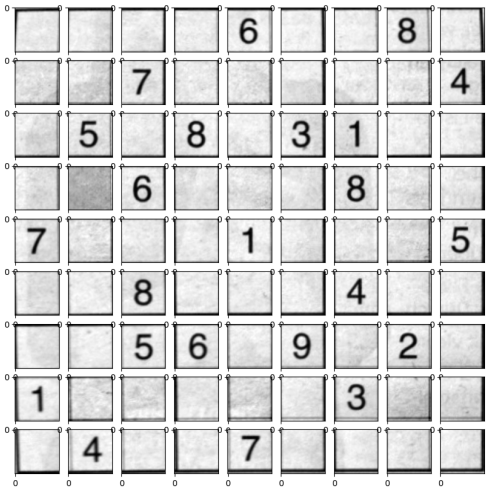

Figure 8: Extracted Sudoku image as individual cells

To prepare individual cells for the classification task, additional processing is required. This step also involves determining whether the cell contains a digit or not, simplifying the later classification task to 9 instead of 10 digits.
The initial action is to eliminate any remaining borders within the cell to retain information solely about the digit represented in the cell image. To achieve this, a binary threshold is applied, with the values of white and black inverted. This inversion is necessary because the function used to remove remaining borders transforms every pixel with a value of 1 to 0 if it's connected to the outline of the image.
Subsequently, on this border-cleared image, contours are detected to ascertain if the cell contains any digit and if so to eliminate any noise present in the image. To determine

if the cell contains a digit, the assumption is made that the largest contour within the image must be the digit if the cell indeed contains one. Since even in empty cells a contour might be detected due to existing noise, the contour undergoes two threshold-based area checks. These checks involve verifying if the largest contour found is sufficiently large. If it fails those checks, it is considered noise, and the cell is assumed to be empty. Conversely, if the contour is large enough to be considered a digit, a digit mask is generated to selectively choose pixels within this contour for display in the processed cell image, with the remaining area set as a white background.
The processed cells from the Sudoku are depicted in Figure 9. It is important to highlight that images comprising solely white pixels have been categorized as empty cells prior to leveraging deep learning models. For the remaining cells a deep learning model can be used to classify them between 1 and 9 to find the numerical values behind each. This predetermination and classification results in an internal representation of the Sudoku grid used in following steps.

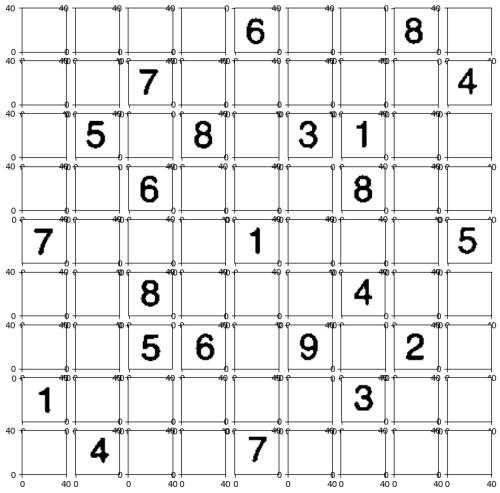

Figure 9: Processed cells

2.3 Sudoku Solving and Remapping

The last phase of the Sudoku solver involves the actual solving of the found and reconstructed Sudoku as well as remapping a newly created image containing the solved Sudoku onto the original image at the exact place the original grid was found.
To solve the now internal representation of the Sudoku grid no deep learning approach is needed. A basic recursive function can be used that tests every empty cell with every

possible digit for this cell and returns True and the filled Sudoku if possible and False if the Sudoku inputted can not be solved due to digit conflicts.
Assuming the deep learning models had a high enough accuracy and are capable of reconstructing the Sudoku's digits flawlessly the solution can always be found and can be used to generated a new, filled Sudoku image as shown in Figure 10. To only map the needed digits onto the original image the digits that have already been in the grid are excluded in this generation step.

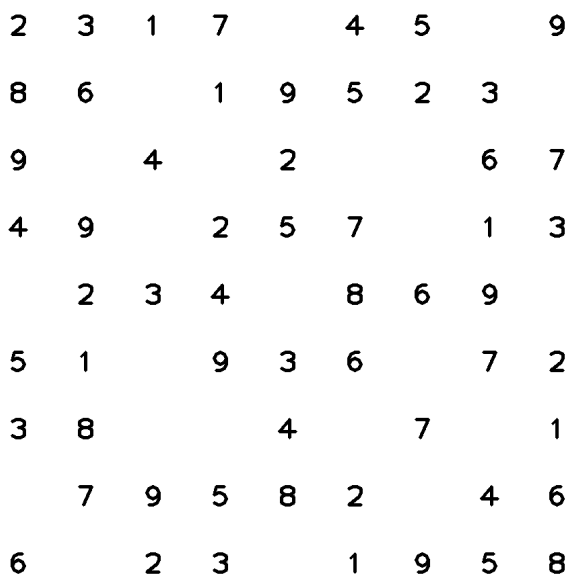

Figure 10: Newly created image containing the solved Sudoku in gray scale

Utilizing the contours of the Sudoku found in the original image this newly created image can now be remapped onto the original input image creating a masked Sudoku image containing the new Sudoku on a canvas the same size as the input image, at the same position and rotation of the original found grid. Note that if the values of the created Sudoku image are inverted before this process the output image of this steps provides a mask that contains white pixels everywhere where the digits should be drawn onto and black else wise (see Figure 11).
This mask can now be used to put the missing digits onto the original image. TO achieve this for every black pixel in the mask the corresponding pixel from the initial image is taken. Otherwise for every white pixel in the mask the corresponding pixel in the output image is colored a specific color - red in this case. This finalizes our Sudoku solver providing the initial image with colored digits inside the empty grid cells.

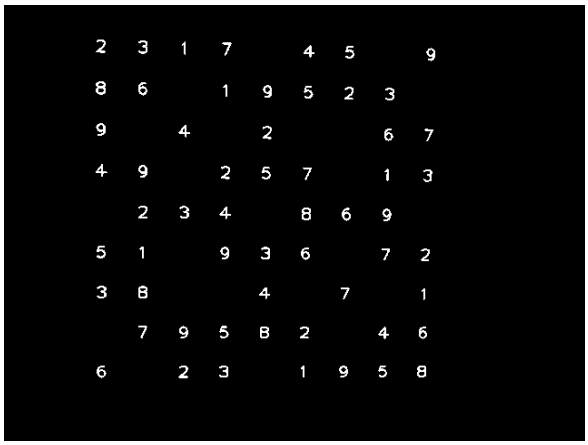

Figure 11: Reversed solved Sudoku image in gray scale

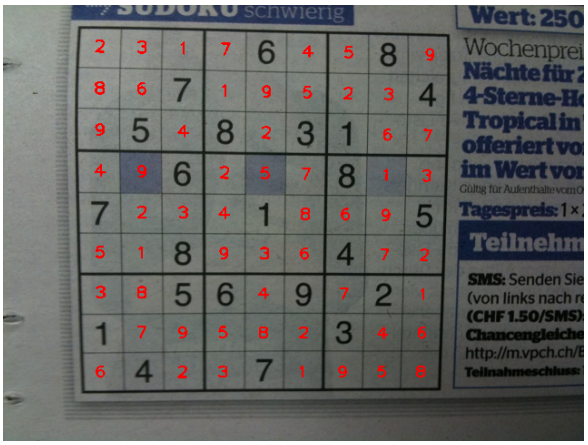

Figure 12: Original input image with remapped solved Sudoku

3 Data Generation

3.1 Overview

The data generation of our project plays a pivotal role in equipping our deep learning models with the ability to accurately recognize and classify digits within Sudoku grid cells. To ensure a robust and diverse training set, we employed a systematic approach in creating sample images containing digits from 1 to 9.
Each image is standardized to a size of 40x40 pixels, featuring a white background with a black digit. To enhance the versatility and adaptability of our models, we introduced variations in positions, fonts, and font sizes across the generated images. This deliberate diversification aims to expose the models to a wide array of scenarios, promoting resilience and accuracy in digit classification.
In total, we generated an extensive data set comprising 175,000 images per digit. To assess the models' generalization capabilities, we implemented a 95% train/test split, resulting in approximately 1.5 million images for training and 80,000 images for testing. This sizable data set ensures the models are well-equipped to handle the intricacies of Sudoku puzzles and exhibit robust performance across diverse scenarios.
Figure 13 and 14 show sample images of our generated training and test data sets.

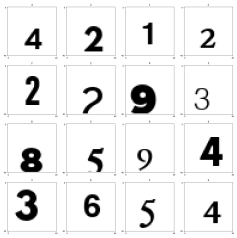

Figure 13: Generated Digit Samples - Training

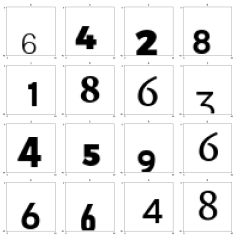

Figure 14: Generated Digit Samples - Testing

3.2 Data preprocessing

Before training our deep learning models, we preprocess the generated training and test sample images to eliminate blurry pixels and normalize pixel values between 0 and 1. This consistent preprocessing approach aligns with the methodology used in the cell preprocessing of the Sudoku solving algorithm, promoting optimal model performance and accurate digit recognition.
Figure 15 and 16 display the identical sample images as before, but this time they are free of blurriness and have been normalized.

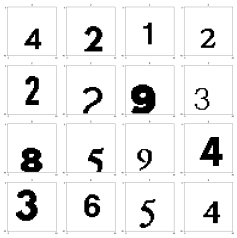

Figure 15: Preprocessed Digit Samples - Training

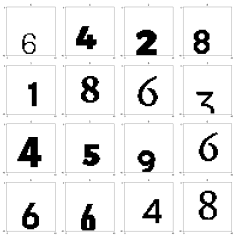

Figure 16: Preprocessed Digit Samples - Testing

4 Models

In this upcoming section, we delve into the foundations of our digit recognition models, each designed to leverage distinct feature types for optimal performance. To illustrate the diversity in feature extraction, we have generated uniform sample digit images,

maintaining consistent positions, fonts, and font sizes across digits 1 to 9. These uniform samples serve as a canvas to showcase the unique characteristics captured by each feature type.

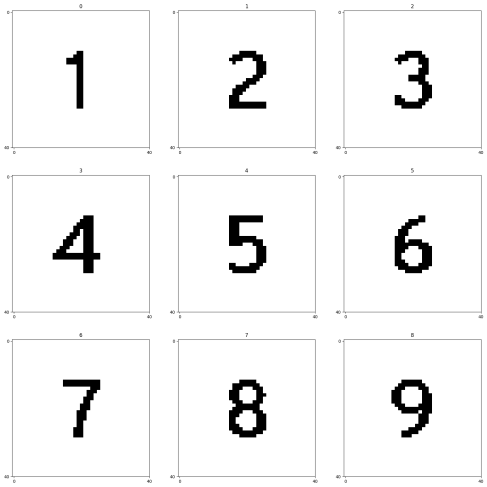

Figure 17: Feature visualization sample digits

Our training efforts were dedicated to extracting three distinct features from the images. The models we employed belong to the category of fully connected neural networks, characterized by their basic architecture with input, hidden, and output spaces.
For all our trained models, we selected the CrossEntropyLoss, a fitting choice for multi-classification tasks such as ours, where the goal is to model probabilities across the digits 1 to 9. To optimize each model, we utilized the Adam optimizer as well as linear learning rate scheduler, ensuring effective convergence during the training process.

4.1 Pixel Based

This model employs a straightforward approach to feature extraction. Simplifying the process, it takes an array of size (1600,) as the input layer—this represents the flattened and normalized image containing 0's and 1's.
In Figure 18, you can observe the flattened sample digit images. It's worth noting that the x-axis has been shortened to the specific range where black pixels exist, and black and white pixels have been inverted for enhanced visualization. The graphs clearly depict slight differences in the extracted features for each pixel that the model is capable

of learning.
The model is based on following architecture:

$$\begin{aligned}
&\text{Linear Layer 1:} && \mathbb{R}^{1600} \to \mathbb{R}^{240} \\
&\text{Linear Layer 2:} && \mathbb{R}^{240} \to \mathbb{R}^{120} \\
&\text{Linear Layer 3:} && \mathbb{R}^{120} \to \mathbb{R}^{80} \\
&\text{Linear Layer 4:} && \mathbb{R}^{80} \to \mathbb{R}^{9} \\
&\text{Activation Function:} && ReLU : \mathbb{R} \to \mathbb{R} \begin{Bmatrix} x & , x \geq 0 \\ 0 & else \end{Bmatrix}
\end{aligned}$$

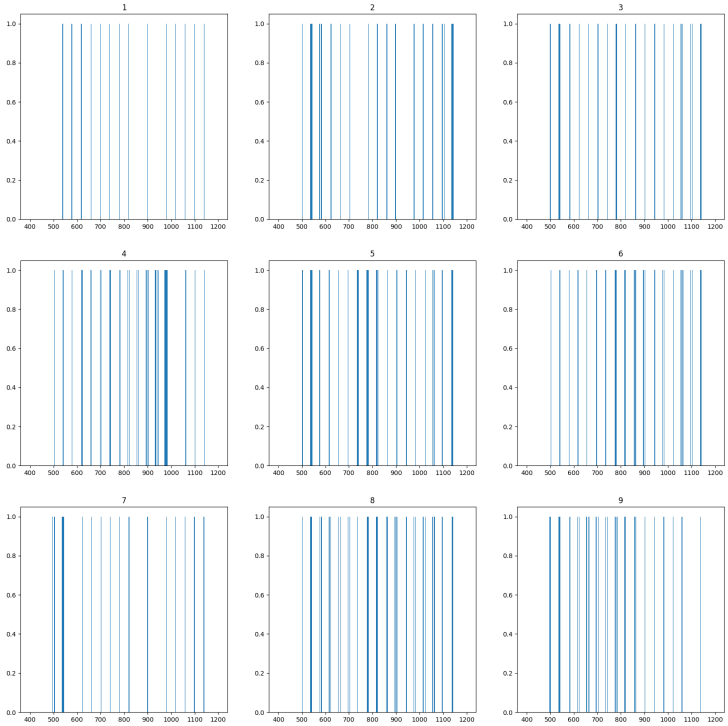

Figure 18: Feature visualization of pixel transformed sample digits

In the training phase, the initial learning rate was set to 0.0001 experiencing a 40% reduction from the initial value over 8 epochs, thanks to a linear learning rate scheduler.

Following 31 epochs of training, the achieved metrics include a training loss of 0.00027, a test loss of 0.00028, a training accuracy of 100%, and a test accuracy of 100%. These outcomes underscore the model's ability not only to learn from the training data but also to generalize features effectively, resulting in exceptional performance on the test data.

4.2 Pixel Density

This model employs a more intricate approach to feature extraction, centering on individual pixel densities for each row and column of the image. Pixel densities are determined by summing up the pixel values along rows and columns and subsequently dividing the result by the number of pixels in the respective row or column. An essential note is the inversion of pixel values, a crucial step in generating the density of black and not white pixels across each row and column.
The model is based on following architecture:

$$\begin{aligned}
&\text{Linear Layer 1:} \quad \mathbb{R}^{80} \to \mathbb{R}^{240} \\
&\text{Linear Layer 2:} \quad \mathbb{R}^{240} \to \mathbb{R}^{120} \\
&\text{Linear Layer 3:} \quad \mathbb{R}^{120} \to \mathbb{R}^{80} \\
&\text{Linear Layer 4:} \quad \mathbb{R}^{80} \to \mathbb{R}^{9} \\
&\text{Activation Function:} \quad ReLU : \mathbb{R} \to \mathbb{R} \left\{ \begin{array}{ll} x & , x \geq 0 \\ 0 & else \end{array} \right\}
\end{aligned}$$

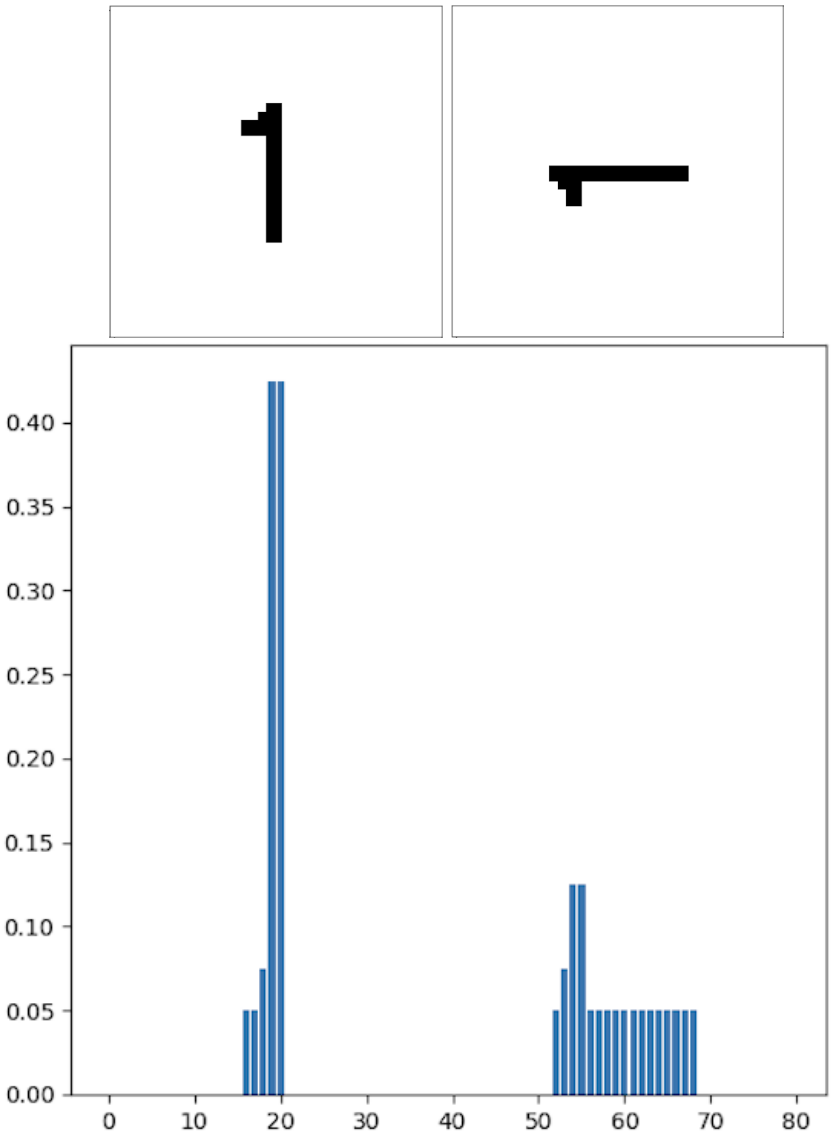

Figure 19: Feature visualization example of pixel density with number 1

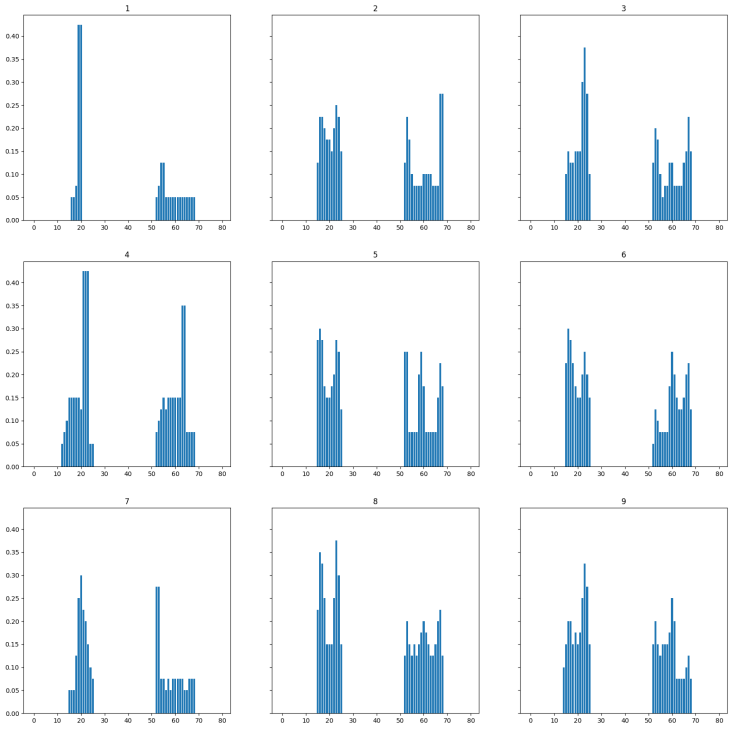

Figure 20: Feature visualization of pixel density of transformed digit data

During the training process, the initial learning rate was established at 0.01, experiencing a 20% reduction from the initial value over 8 epochs with the assistance of a linear learning rate scheduler. Following 10 epochs of training, the attained metrics reveal a training loss of 0.0006, a test loss of 0.0006, a training accuracy of 99.99%, and a corresponding test accuracy of 99.99%.

4.3 Image Gradient Density

The final model developed for this report focuses on a more computationally intensive approach to feature extraction. Initially, the image gradient for each pixel in the black-and-white image is computed using SobelX and SobelY filters and convolution. The outputs of these filters represent the changes in pixel values in the x and y directions, allowing for the calculation of the combined gradient a pixel is facing, commonly referred to as the image gradient as illustrated in 21. To determine the image gradient

densities, the number of pixels facing a specific direction is divided by the total number of pixels having a gradient. It's worth noting that, for simplicity, the calculation of this density was restricted to the four main directions: up, down, left, right, and their four intermediate directions: up-right, up-left, down-right, down-left.
The model is based on following architecture:

$$\begin{aligned}
&\text{Linear Layer 1:} && \mathbb{R}^8 \to \mathbb{R}^{60} \\
&\text{Linear Layer 2:} && \mathbb{R}^{60} \to \mathbb{R}^{40} \\
&\text{Linear Layer 3:} && \mathbb{R}^{40} \to \mathbb{R}^{20} \\
&\text{Linear Layer 4:} && \mathbb{R}^{20} \to \mathbb{R}^{9} \\
&\text{Activation Function:} && ReLU : \mathbb{R} \to \mathbb{R} \left\{ \begin{array}{ll} x & , x \geq 0 \\ 0 & else \end{array} \right\}
\end{aligned}$$

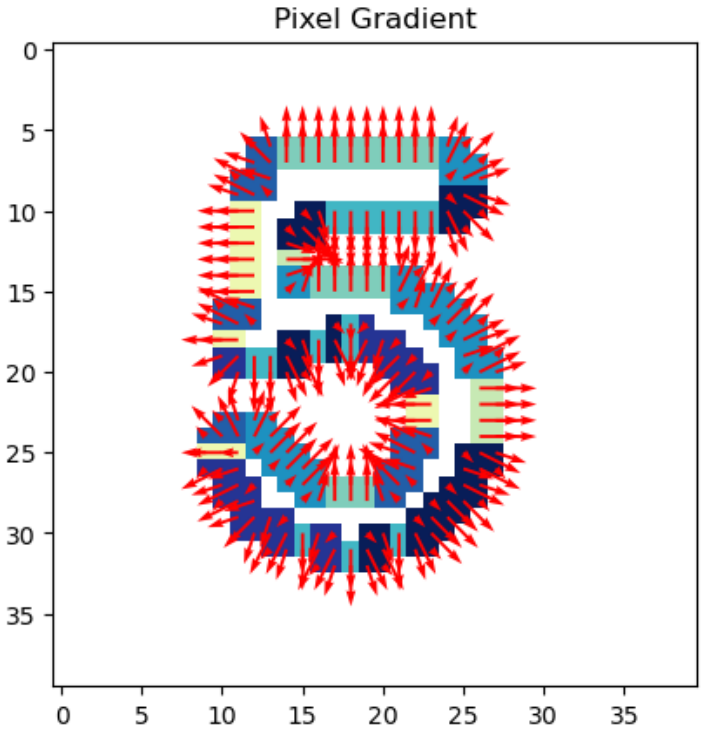

Figure 21: Feature visualization of image gradient for a sample digit 5

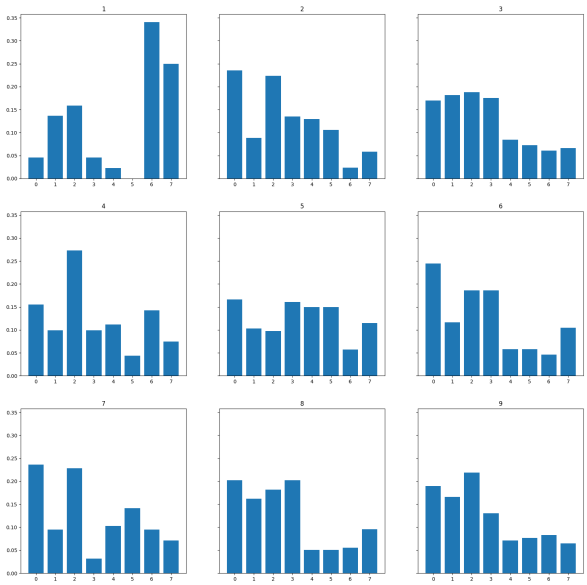

Figure 22: Feature visualization of image gradient density transformed sample digits

The training commenced with an initial learning rate of 0.001, which experienced a 20% reduction from the initial value after 80 epochs, facilitated by a linear learning rate scheduler. Upon reaching 100 epochs, the metrics obtained consist of a training loss of 0.0154, a test loss of 0.0167, a training accuracy of 99.63%, and a test accuracy of 99.62%.

5 Experiments

5.1 Training Performance

To assess the efficacy of the models mentioned earlier, we utilized the generated test dataset to construct confusion matrices for digits 1 through 9. These matrices reveal the count of accurate classifications made by the model, along with statistical information on the frequency and specific digits associated with incorrect classifications.

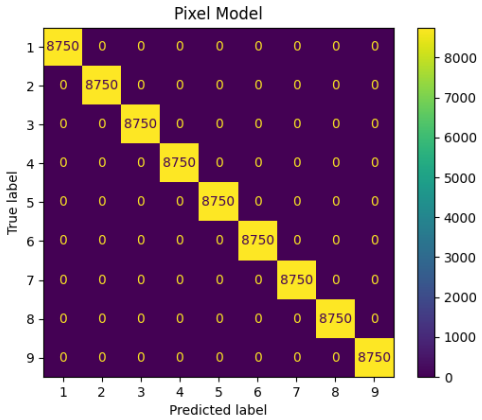

Figure 23: Performance of Pixel Model on generated digits test data

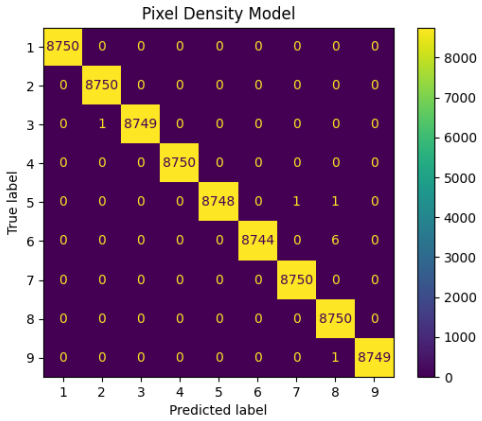

Figure 24: Performance of Pixel Density Model on generated digits test data

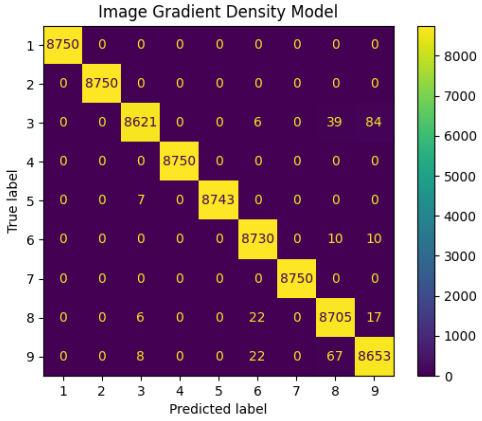

Figure 25: Performance of Image Gradient Density Model on generated digits test data

5.2 Application Performance

Given that the preceding figures solely offer insights into the conclusive training state of the diverse models on the generated data, a secondary experiment was devised to assess their practical performance along with that of the solver.
For this experiment, 55 test images sourced from the GitHub repository *sudoku dataset* were analyzed to gauge the solver's effectiveness in real-world scenarios. The solver functions, up to and including the classification phase, were employed, and the resulting classification output was compared to the actual values within the Sudoku grid of the test images.
This testing yields metrics akin to those presented earlier, showcasing the classification performance of each model, along with the efficacy of the initial digit detection based solely on contour analysis. Consequently, instead of representing the digits 1 to 9, the Figures 26, 27 and 28 all encapsulate information about the digit 0, which internally represents an empty cell.

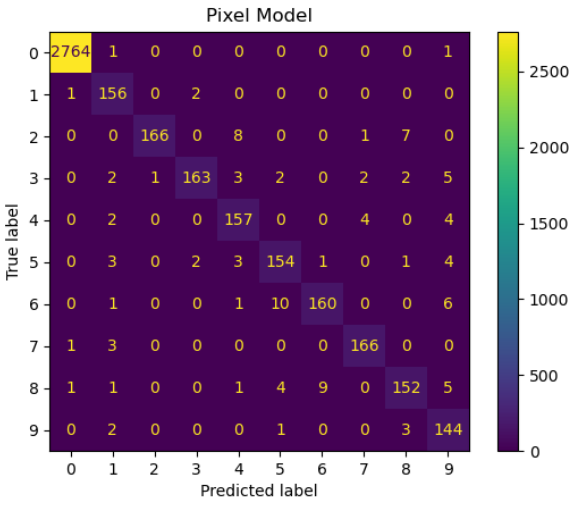

Figure 26: Performance of Pixel Model on Sudoku test data

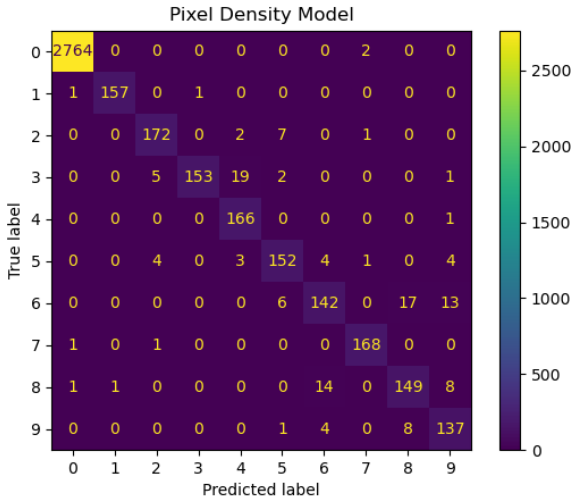

Figure 27: Performance of Pixel Density Model on Sudoku test data

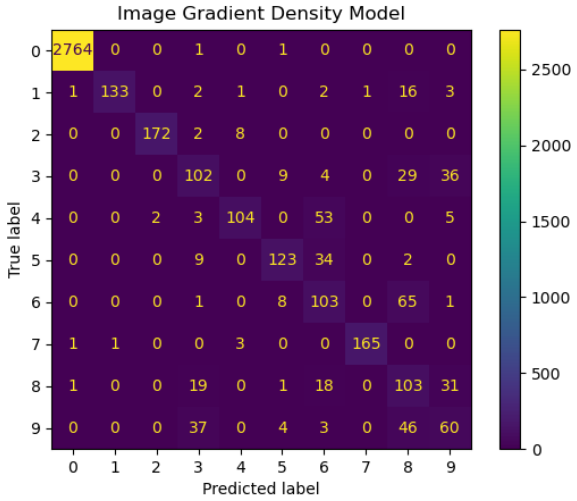

Figure 28: Performance of Image Gradient Density Model on Sudoku test data

5.3 Results

The PixelModel demonstrated superior performance in both generated digits and actual image data, achieving a final application score of 93.85% in cell classification and solving 20 Sudokus. In comparison, the PixelDensity Model followed with a 93.29% classification score and 8 solved Sudokus. The ImageGradientDensity Model lagged behind with a cell classification accuracy of 85.68% and no Sudoku solutions.

5.3.1 Deep Learning Classification

Although the models perform quite well, there is a significant gap between the model performance on the generated images and the real sudoku cell images as we can see in the Figures 5.1 and 5.2. In general, generating images can't capture all the special features and noises a photography of a sudoku printed on paper has. Although we apply some techniques to reduce the noise and unwanted artifacts in the sudoku images, there are still some peculiarities in the application images that make them different to the generated data. Besides this difference in training and application data, we now want to discuss other possible weaknesses of the models and the feature engineering and dive deeper into our prediction process.

Image Gradient Density Model

Looking at the Image Gradient Density Model, we can see that there are some pairs of numbers which are susceptible for misclassification such as $(6, 8)$ i.e. an eight gets misclassified as a six. This can be see in Figure 29.

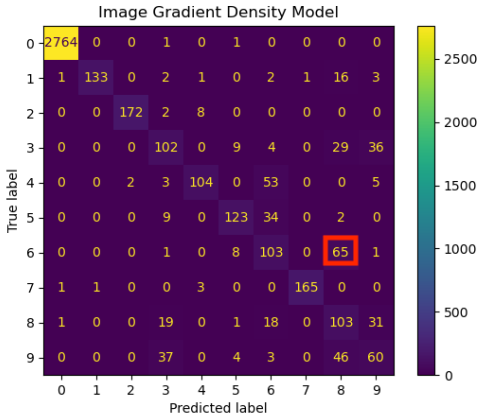

Figure 29: Performance of Image Gradient Density Model on Sudoku test data. The number six is misclassified as an eight for 65 times.

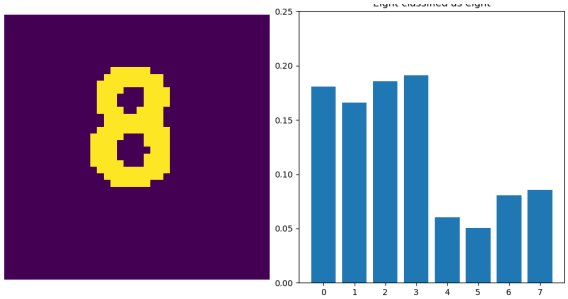

Figure 30: The number eight and its density.

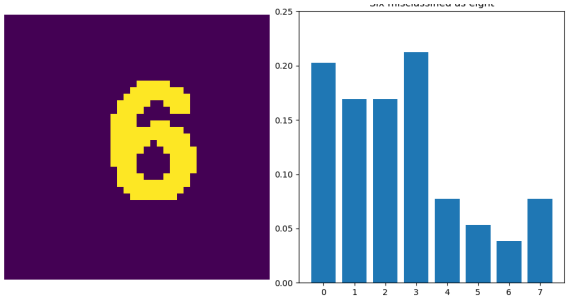

Figure 31: This six is misclassified as eight. We can see the similarities in the densities of a six and an eight.

In the Figures 30 and 31, we can see an example of the structural similarities of the densities of eights and sixes. Under the assumption that similar inputs to a machine learning model produces similar outputs, this can be one reason for a bad prediction performance and an inappropriate feature engineering.
To further examine this, we can look at all misclassified examples and see their mean gradient densities. These mean densities can be compared to the mean densities of the corresponding numbers.
For example, we will look at all sixes that hat been misclassified as eights and build the mean density of their densities. This mean density can be compared with the mean densities of all eights that have been correctly classified via Mean Squared Error.

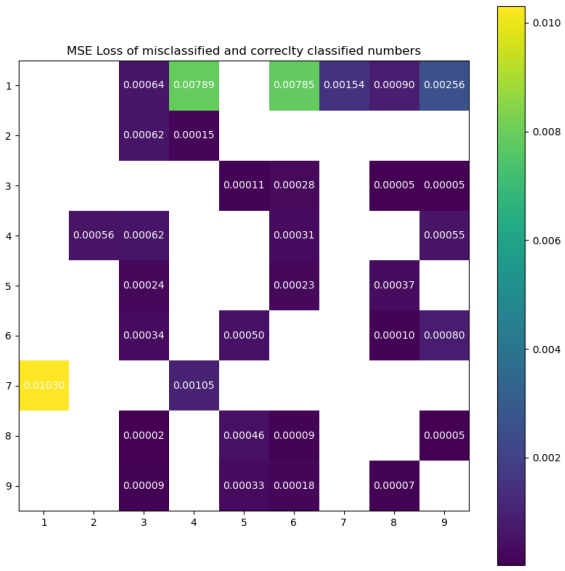

Figure 32: The Mean Squared Error of misclassified and correctly classified image gradient densities. The empty cells correspond to either numbers y that never have misclassified as x or classes of numbers that are compared to themselves (diagonale).

We can see, that a low MSE in the showcased matrix in Figure 32(similar gradient densities) correspond to a high number of misclassifications shown in Figure 28.

Pixel Density Model

For some pairs, the observed problem of 5.3.1 can also be seen looking at the Pixel Density Model. One example is again the pair $(6, 8)$.

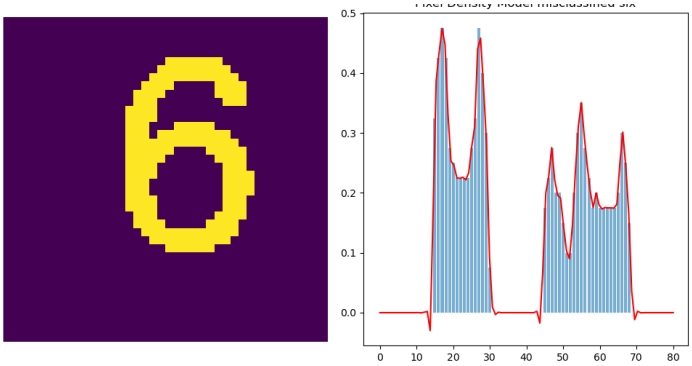

Figure 33: The number six that has been classified as an eight.

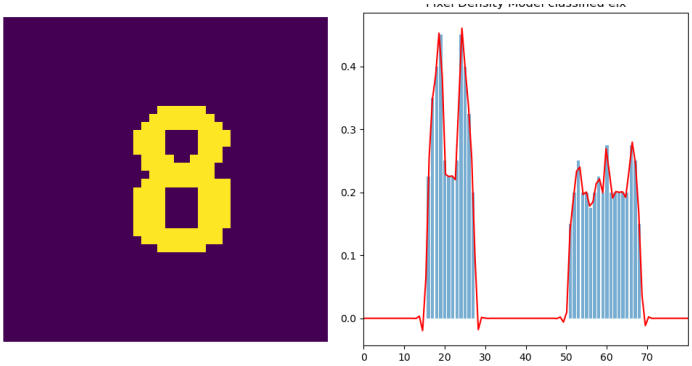

Figure 34: The number eight and its pixel density.

Figures 33 and 34 show one example of the structural similarities that can be seen in the data of the Pixel Density Model. Although this might be a problem to our model, there are some pairs such as $(3, 4)$ that are misclassified quite often but don't show this kind of similarity at all. One example of this phenomenon can be seen in figures 35 and 36.

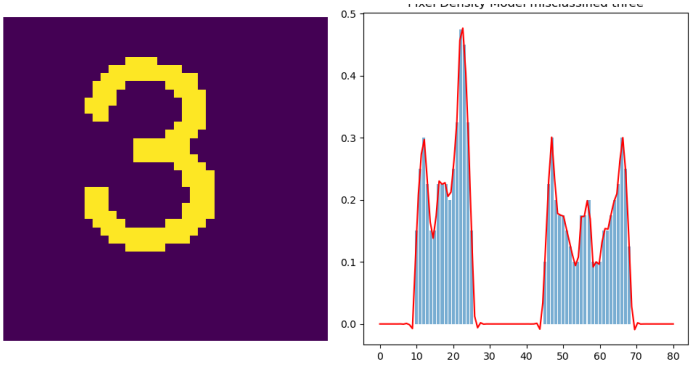

Figure 35: The number three that has been classified as an four.

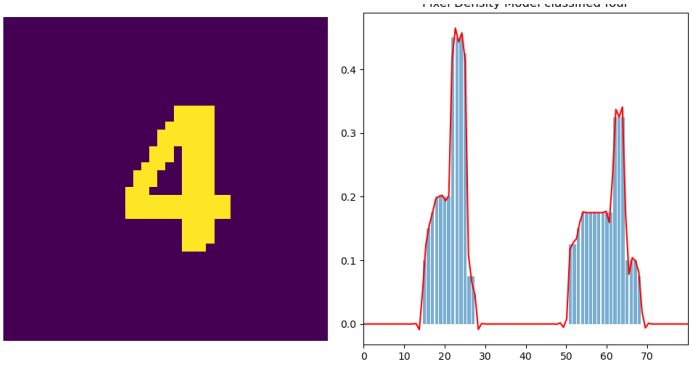

Figure 36: The number four and its pixel density.

Pixel Model

Analyzing observations for the final model proves challenging. The pixel model relies on a feature space comprising 1600 input features, rendering it difficult for humans to discern the factors influencing the model's classification decisions. This challenge is particularly pronounced for our trained model, given its commendable final application accuracy exceeding 93%, with only a minimal number of misclassifications.
Furthermore, the complexity of the feature space in the pixel model demands a more nuanced investigation to understand its decision-making process. Exploring techniques such as feature importance analysis or visualization methods specific to high-dimensional spaces could shed light on the critical factors influencing the model's predictions. Additionally, further research into optimizing the model's interpretability, even in the context of a substantial number of input features, could enhance our understanding of its performance and aid in refining the model for potential future applications.

5.3.2 Empty Cell Predetermination

Upon reviewing the confusion matrices, it is evident that only 4 out of 2766 truly empty cells were misidentified. In addition, 3 non-empty cells were erroneously labeled as empty. This discrepancy results in a total error rate of 7 out of 2769 for the empty cells, representing approximately 0.25%. Despite this minimal error rate compared to the overall success of the deep learning approach in cell classification, it remains essential to conduct a detailed examination of these misclassifications of empty cells, as this step holds critical importance in the preprocessing phase setting the groundwork for our deep learning models by extracting the cells needing a classification solely based on contour analysis.
In the following figure 37 the first four truely empty cells that were incorrectly identified as filled cells are depicted. The first column shows the original image at its original size of 100x100 pixels. Next to it, in the second row, is the thresholded 100x100-pixel image, where the largest contour is sought and classified as noise or a number based on thresholding methods. In columns 3 and 4, both images (original and thresholded) are displayed again, but this time the image was scaled to a size of 40x40 pixels before applying any processing, representing the actual procedure used by the algorithm. The respective pairs also include the calculations of the threshold values, which ultimately determined whether the cell contains a number or if the black pixels in the image are to be considered noise.

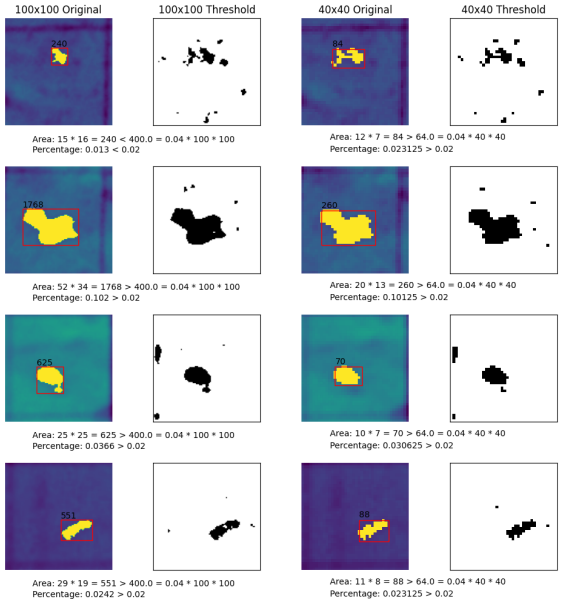

Figure 37: Truely empty cells classified as a digit

The reason for misclassification in the first image is identified as the downsizing per-

formed in advance. By scaling from 100x100 pixels to 40x40 pixels, two relatively large noise points are merged into a much larger noise point. This leads to an incorrect representation of the actual noise points in the smaller image, resulting in the image being classified as a number. Had the check been performed on the original 100x100 pixel image, this error would not have occurred, achieving a better performance of the preliminary check accordingly.
For the last three images in both the original 100x100 pixel image and the reduced 40x40 pixel image, noise detection fails. However, a possible solution approach for these three images can be analyzed as well. Although the noise is too large to be filtered out using an area check, it is observed that the aspect ratios of the contours surrounding the different noise points do not conform to the norms for numbers. Either the noise contours are too small in height to be considered as a number, or the ratio between width and height is not applicable for front facing digits as the height of the contour always has to be larger than the width. A possible solution would be to also apply width and height examinations and not solely rely on area checks.
Not only did the algorithm mistakenly label empty cells as numbers, but it also falsely identified three cells containing numbers as empty. Figure 38 provides a comparable visualization to the previous one, now featuring a third column that displays the thresholded image before applying the border removal function. Examining the last images for each original image reveals that after applying the border removal function the resulting image only contains black pixels. This effect is also evident in the second image, where every white pixel in the images is linked to a white pixel at the edge of the corresponding image. As the border removal function turns every white pixel connected to a white edge pixel to black, it results in a black output image lacking discernible contours. This misleads the algorithm into assuming the image is empty, despite the presence of a number.

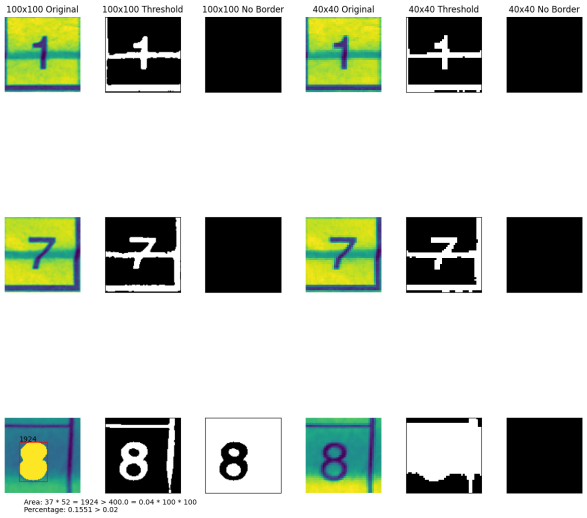

Figure 38: Truely filled cells classified as empty

One way to address this issue may be by using an adaptive threshold. So far, only a simple binary threshold has been used to compress pixel values in the image to 0 and 1. By using an adaptive threshold before the binary threshold, the actual contours of the numbers in the image can be better highlighted against the background. This allows for the removal of possible connecting pixels to the edge, leaving only the actual number with potential noise, which can be recognized and extracted as the largest contour, provided it passes the previously applied area check.
Following figures show the result of adding the previously mentioned changes. By adding width and height checks, an adaptive threshold and performing the predetermination on the full sized 100x100 pixel image the falsely classified empty cells in both directions can be reduced to every cell being correctly classified as containing a digit or being empty, showcasing that by adding these small details the performance of the predetermination can be improved. Note that the coded changes only represent a quick fix and need further examination to be claimed fully reliable.

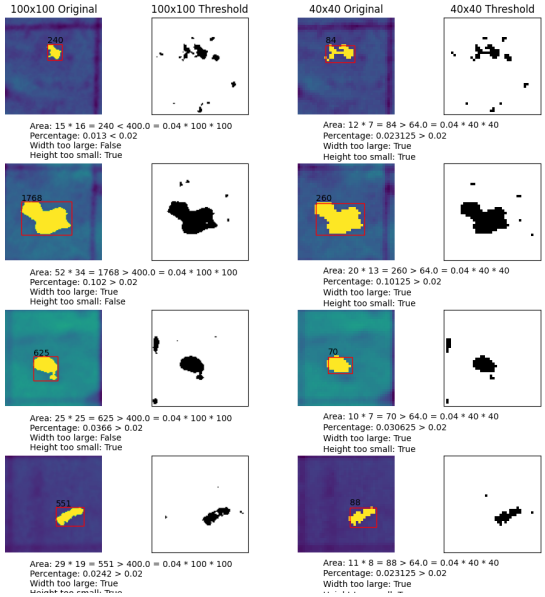

Figure 39: Truely empty cells classified as a digit (solved)

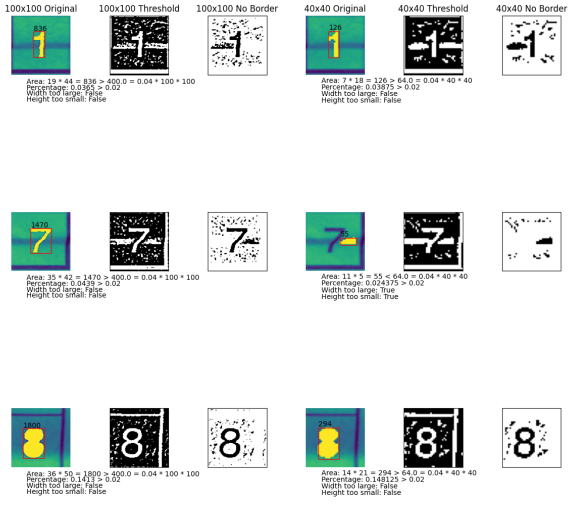

Figure 40: Truely filled cells classified as empty (solved)

5.3.3 Final Conclusion

The initial non-machine learning components, involving the Sudoku grid extraction, individual cell extraction, and empty cell determination, exhibited remarkable performance utilizing basic computer vision techniques such as contour analysis. However, challenges arose with the integration of deep learning models. The use of basic feed-forward networks, as opposed to convolutional networks known for superior image classification, may have contributed to limitations. Additionally, reconsidering more sophisticated feature extraction methods or combining various types to create a more comprehensive ensemble model could enhance overall performance. The pivotal issue in the deep learning classification phase was the selection of training data. Creating a dataset with manually generated images of printed digits may have been an inadequate assumption, and exploring alternative approaches, such as augmenting extracted and preprocessed cell images for training, could potentially yield better results. These assumptions necessitate further analysis and implementation for validation.
Despite not achieving optimal performance, our solver demonstrated success in solving Sudokus, as evidenced by the highest score achieved with over 20 images. This underscores the effectiveness of breaking down the task into independent subtasks, each simplifying the preceding step, ultimately leading to successful results.
Through this project, we gained valuable insights into computer vision, deep learning, collaborative work, and problem-solving. We express our gratitude to Professor Nguyen Duc Dung for guiding this project and the Computer Vision lecture at Hanoi University of Science and Technology.