International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 10 Issue 3, May-Jun 2026 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

Sudoku Grid Detection Using OCR

Priyam Bhattacharya, Pritam Mondal, Sreena Mondal, Prof. Dr. Rajib Kumar Das

Department of Computer Science and Engineering,
University of Calcutta, Kolkata, West Bengal, India

ABSTRACT

Optical Character Recognition (OCR) has advanced significantly with deep learning, yet automatically recognizing and parsing printed Sudoku grids from low-quality physical media-such as newspapers and magazines-remains a persistent challenge due to variable typography, perspective distortions, localized shadows, and ink degradation. This paper presents a complete, web-based end-to-end software pipeline designed to localize, digitize, and interactively guide users through solving printed Sudoku puzzles. The proposed system features an robust five-strategy cascade for image thresholding and grid localization, ensuring structural invariance against non-uniform illumination and perspective skewing. Following geometric rectification via homographic perspective transformation, individual cells undergo a customized twelve-candidate binarization workflow reinforced by Contrast Limited Adaptive Histogram Equalization (CLAHE) to effectively preserve thin or faint character boundaries. Character recognition is driven by a deep convolutional neural network architecture, termed DigitCNN, integrating channel-wise Squeeze-and-Excitation block mechanics and symmetric identity skip connections to automatically capture spatial hierarchies of printed digits across diverse typefaces. To narrow the persistent domain gap encountered in document analysis pipelines, the DigitCNN model is optimized using an innovative self-supervised, pipeline-aware training regime spanning 1.62 million stochastically degraded synthetic and augmented text samples processed through the exact binarization pipeline executed during inference. To overcome individual classification errors, the system embeds a post-classification constraint correction layer acting on global puzzle logic, combined with a self-supervised template rescoring module utilizing high-confidence local predictions to re-evaluate ambiguous cells. Experimental evaluations demonstrate that the customized DigitCNN model achieves a peak validation accuracy of 99.04% and a final convergent accuracy of 98.91%, vastly outperforming classic structural frameworks. Integrated into a monolithic Single Page Application (SPA) driven by a Flask-backed runtime architecture, the system incorporates a non-deterministic Minimum Remaining Values (MRV) heuristic-based backtracking solver to parse real-time uniqueness violations dynamically. Rather than acting as an automated solver that bypasses human reasoning, the application operates strictly as an intelligent hint provider-highlighting conflicting nodes, maintaining dual pencil-and-pen entry modes, and exposing logical per-cell constraints directly within the web interface to augment human deductive problem-solving.

KEYWORDS: *Optical Character Recognition (OCR), Convolutional Neural Networks (CNN), Sudoku Grid Detection, Squeeze-and-Excitation (SE), Perspective Correction, Heuristic Backtracking Solver, Human-AI Interaction.*

How to cite this paper: Priyam Bhattacharya | Pritam Mondal | Sreena Mondal | Prof. Dr. Rajib Kumar Das "Sudoku Grid Detection Using OCR" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-10 | Issue-3, June 2026, pp.1306-1322, URL: www.ijtsrd.com/papers/ijtsrd133328.pdf

IJTSRD133328

1. INTRODUCTION

OCR:

OCR is the acronym for Optical Character Recognition. This technology allows to automatically recognizing characters through an optical mechanism. In case of human beings, our eyes are optical mechanism. The image seen by eyes is input for brain. The ability to understand these inputs varies in each person according to many factors. OCR is a technology that functions like human ability of reading. Although OCR is not able to compete with human reading capabilities. OCR can recognize both handwritten and printed text. But the performance of OCR is directly dependent on quality of input documents.

Sudoku:
Sudoku is a logic-based combinatorial number-placement puzzle in which a 9×9 grid must be filled with the digits 1 through 9 such that each row, column, and 3×3 sub-grid contains exactly one instance of each digit. The puzzle appears extensively in newspapers, dedicated puzzle books, and digital applications, generating consistent interest among millions of solvers worldwide. While the game is relatively straightforward for a human to understand, automatically reading and solving a Sudoku grid from a photograph presents a rich set of challenges spanning computer vision, machine learning, and combinatorial search.

1.1. Motivation
The primary motivation is to create a system that eliminates manual digit transcription entirely. Newspaper and magazine Sudoku grids are printed in dozens of typefaces at varying resolutions and are often photographed under adverse conditions including partial shadows, skewed angles, and low contrast. Standard digit recognizers trained purely on MNIST perform poorly on these inputs because printed Sudoku fonts differ significantly from handwritten strokes.

1.2. Problem Statement
Given an arbitrary photograph of a printed Sudoku puzzle, the system must produce a correct, fully-solved 9×9 grid. More precisely, the pipeline must:

1. Detect the grid boundary in the image, regardless of perspective angle, partial occlusion by the fold of a newspaper, or background clutter.
2. Apply a perspective transform that maps the detected quadrilateral to a canonical square of fixed resolution, eliminating projective distortion.
3. Segment the canonical grid into 81 individual cell images and classify each as blank or as one of the digits 1 to 9.
4. Correct classification errors that violate Sudoku constraints, using knowledge of the global puzzle structure as an additional signal.
5. Give global hints to the user so that he/she can solve the puzzle rapidly.
6. Present the result through an interactive browser interface that enables the user to inspect and manually correct OCR output, with support for candidate notation (pencil marks), automatic peer propagation on digit confirmation, and per-cell constraint reasoning to guide manual solving.

1.3. Objectives
- Design and train a compact CNN, DigitCNN, capable of classifying printed Sudoku digits across diverse typefaces and photographic conditions.
- Implement a five-strategy cascade for robust Sudoku grid boundary detection invariant to lighting, perspective, and resolution.
- Develop a post-classification correction stage that exploits Sudoku constraint propagation to resolve ambiguous or low-confidence OCR outputs.
- Implement a constraint-based backtracking solver with an MRV heuristic for efficient puzzle completion.
- Deliver the complete pipeline as a Flask-backed, browser-accessible web application with an interactive frontend for both OCR upload and manual entry.

2. Background Study
We should be familiar with the following terminologies to get into the project.

2.1. Optical Character Recognition (OCR)
Optical Character Recognition deals with the problem of recognizing optically processed characters. Optical recognition is performed off-line after the writing or printing has been completed, as opposed to on-line recognition where the computer recognizes the characters as they are drawn. Both hand printed and printed characters may be recognized, but the performance is directly dependent upon the quality of the input documents.

Fig1: How OCR works?

2.2. Convolutional Neural Network (CNN)

Convolutional Neural Networks, referred to as CNN, are unique deep learning models used for recognizing objects from a set of tasks which may include image classification, object detection, and object segmentation. Some of the areas where CNNs find applications include autonomous driving, security camera surveillance, and more.

Key Components of a CNN

The convolutional neural network is made of four main parts.

- Convolutional layers
- Activation Function-Rectified Linear Unit (ReLU)
- Pooling layers
- Fully connected layers

Convolution Layer is considered the most basic component of a Convolutional Neural Network (CNN). The main task performed by this layer the automatic detection of the most interesting visual patterns from an image.

- **The Core Mechanism:** The process begins with applying the mathematical operation called convolution. To perform convolution, a smaller matrix named the filter slides through the larger image represented by pixels.
- **Pattern Detection:** Each of the filters used can be thought of as a unique magnifying glass that detects different features from the images. At once, many filters with the same dimensions are used in the layer, with some recognizing sharp corners, others curved lines, and the rest entire objects.
- **Creating Feature Maps:** As these filters slide across the image, they calculate values based on what they "see." This process generates a brand-new grid (a feature map) that maps out exactly where those specific lines or shapes were found.

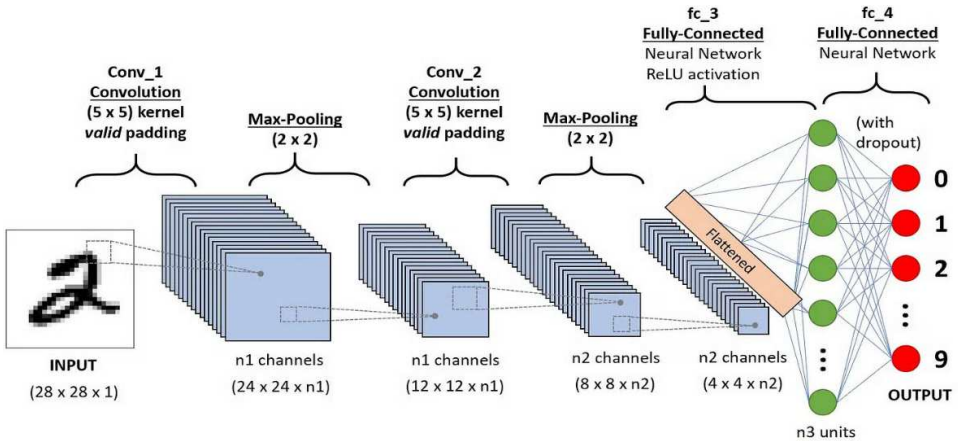

Fig2: Architecture of CNN for digit recognition [3]

Let's consider this 32x32 grayscale image of a handwritten digit. The values in the matrix are given for illustration purposes. Also, let's consider the kernel used for the convolution. It is a matrix with a dimension of 3x3. The weights of each element of the kernel is represented in the grid. Zero weights are represented in the black grids and ones in the white grid.

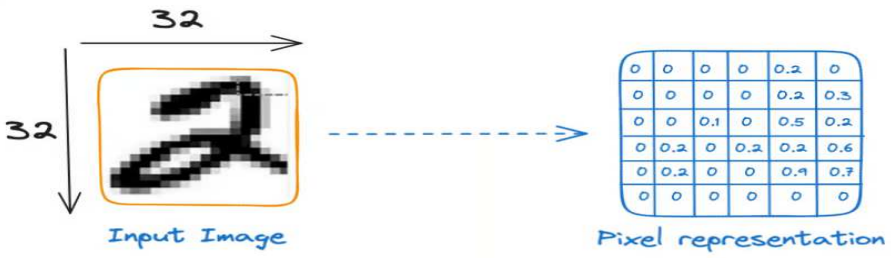

Fig3: Illustration of the input image and its pixel representation [3]

Using these two matrices, we can perform the convolution operation by applying the dot product, and work as follows:

1. Apply the kernel matrix from the top-left corner to the right.
2. Perform element-wise multiplication.

3. Sum the values of the products.
4. The resulting value corresponds to the first value (top-left corner) in the convoluted matrix.
5. Move the kernel down with respect to the size of the sliding window.
6. Repeat steps 1 to 5 until the image matrix is fully covered.

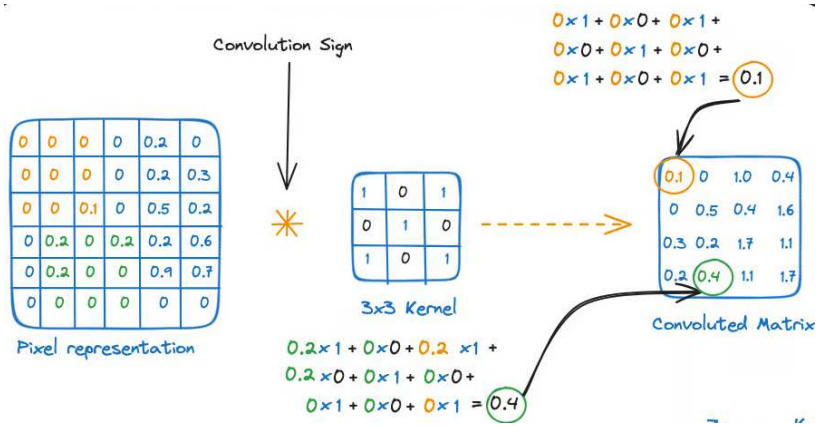

Fig4: Application of the convolutional task [3]

Activation Function

ReLU activation function will be used as an activation function after each convolution process. The reason behind using ReLU activation function is that this type of activation function allows the model to learn non-linear mappings between the features within an image.

Pooling layer

The main aim of the pooling layer is to extract the important features from the convolutional matrix. This is achieved by performing certain aggregation operations that minimize the dimension of the feature map (convoluted matrix). This helps to minimize the memory required to train the network. The pooling layer also plays a role in preventing overfitting.

Some of the basic aggregation functions include:
Max pooling, whereby the highest value of the feature map is taken.

Sum pooling, wherein all the values of the feature map are added up.

Average pooling, where the mean of all the values is considered.

Fig5: Application of max pooling with a stride of 2 using 2x2 filter [3]

The fully connected layers

They form part of the last layers of the convolutional neural network and have inputs which are formed from the flattening of the matrix that results from the last pooling layer. Non-linear functions called ReLU activations functions are performed on these layers.

At the end of all these layers is the softmax prediction layer where the probabilities of all the outputs are determined. The output with the maximum probability is selected as the predicted output.

2.3. Residual Network and Skip Connection

Residual Networks (ResNet) is a deep learning architecture designed to enable efficient training of very deep neural networks

More layers should mean more chance for a network to learn. In practice, past a certain depth, things start breaking down.

The Vanishing gradient problem: The neural network is updated based on an error signal propagated backward throughout the entire network, which is referred to as backpropagation. This signal helps each neuron layer to make changes to their weights. However, since the signal keeps being multiplied by tiny numbers and propagated further through many layers, it will eventually decrease to such a degree that early layers cannot perceive it.

Core Idea (Residual Learning): Residual Networks were introduced to tackle the vanishing gradient problem. The key innovation behind ResNet is the concept of residual learning, which allows the network to bypass some layers using skip connections. In a traditional network, data flows through each layer in sequence. Every layer transforms the input and passes the result to the next one. Skip connections take the original input and add it directly to the output of a layer further down the block.

The input x travels two paths at once. One path goes through the convolutional layers, which learn the residual F(x). The other path skips those layers and connects to the addition step. The final output is F(x) + x.

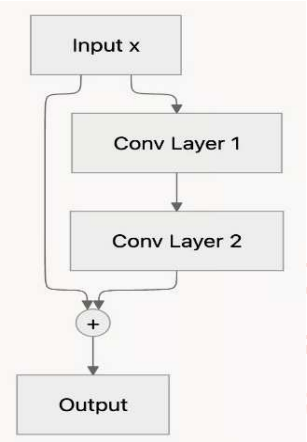

Fig6: Skip Connection

Structure of a ResNet Block: Here's what's going on inside a single block:

1. The input x enters the block and splits into two paths
2. One path goes through two convolutional layers, each followed by batch normalization and a ReLU activation
3. The other path skips those layers - this is the skip connection
4. Both paths meet at an addition step, where the original input is added to the output of the convolutional layers
5. A final ReLU activation is applied to the result

2.4. Domain Adaptation and Data Augmentation

A large performance gap is known to exist between MNIST-trained models and printed digits recognition tasks (Liang et al., 2021). Some possible solutions to this include gathering labeled real data, fine-tuning the model using synthetically generated data, or using extreme augmentation techniques. In this particular case, the technique used – pipeline-aware learning – is an instance of the second solution method, i.e., we do not gather real labeled data, but instead generate synthetic digit data through our rendering pipeline and process it in the exact same way that the images will be processed during inference, i.e., using the exact same binarization algorithm provided by OpenCV.

Contrast Limited Adaptive Histogram Equalization (CLAHE) [1] redistributes intensity values based on tiles to improve the local contrast in those tiles. Using CLAHE during the binarization stage in this project helps in improving the contrast in faint ink cases where global or even adaptive thresholding is unable to provide enough contrast to detect the contours properly

3. RELATED WORKS

With growing development in the field of computer vision and deep learning, several studies have been conducted over the last few years related to optical character recognition, image segmentation, and automatic puzzle detection, which have yielded important publications that have contributed greatly to enhancing the precision of document and digit recognition systems.

Zhang et al. proposed [2] Dilated Residual Networks with Symmetric Skip Connections for image denoising, demonstrating that residual connections combined with dilated convolutions effectively suppress noise while preserving fine structural detail - a principle directly adopted in the SEResBlock design of this project, where symmetric skip connections retain original digit stroke information across deep convolutional stages to improve recognition accuracy on degraded newspaper print images.

Li et al. [4] conducted a comprehensive review of Convolutional Neural Networks (CNNs), analyzing one-dimensional, two-dimensional, and multi-dimensional convolutions across various computer vision applications, and identified key optimization mechanisms and rules of thumb essential for achieving state-of-the-art accuracy in visual classification tasks.

Yamashita et al. [5] provided a comprehensive perspective on Convolutional Neural Networks (CNNs), detailing their standard structural building blocks-such as convolutional, pooling, and fully connected layers-and demonstrating how these layers automatically and adaptively learn spatial hierarchies of features from image datasets (Yamashita et al., 2018).

Xu et al. [6] presented a comprehensive survey on the development of skip connections in deep neural networks, demonstrating how structural residual learning enables easier optimization during training and significantly improves testing accuracy across computer vision and medical image analysis tasks.

Shi et al. [7] proposed a progressive wide residual network using fixed skip connections to counter structural degradation and improve detail retention, demonstrating that wide structures combined with targeted skip connections significantly enhance spatial feature transmission and detail reconstruction during network optimization.

He et al. [8] analyzed the propagation mechanics of deep residual networks and proposed a restructured "pre-activation" residual unit utilizing identity mappings as skip connections, enabling the smooth training of an extremely deep 1001-layer ResNet architecture.

Mithe et al. [9] proposed a mobile-based Optical Character Recognition (OCR) system that captures images via an Android camera and routes them through a core pipeline of preprocessing, segmentation, and Artificial Neural Network (ANN) classification to optimize text extraction for spatially constrained environments.

Eikvil (1993) provided a comprehensive survey of Optical Character Recognition (OCR) systems [11], detailing the historical evolution from rigid template matching to modern adaptive frameworks while highlighting thresholding, location segmentation, and feature extraction as the core pillars of text recognition pipelines.

Bhattarai et al. [12] conducted a comparative analysis using a dataset of 500 Sudoku puzzles across five difficulty levels, evaluating recursive backtracking against a heuristic-based constraint propagation method, and concluded that the heuristic approach consistently outperformed backtracking with speedup ratios ranging up to 2.91x on expert puzzles.

4. SYSTEM DESIGN and ARCHITECTURE

4.1. System Overview

This system is designed to be a Sudoku Optical Character Recognition (OCR) and Hint Application, which will enable a user to capture an image of a sudoku puzzle and get help in solving it digitally. This system will not provide automatic solutions instead it reads the puzzle from the photo, displays it in an interactive grid, detects any conflicting digits, and provides global hints to guide the user toward the correct solution.

This system comprises three major layers: the frontend interface, backend flask server, and OCR using Deep Learning.

Layer	Technology	Purpose
Frontend	HTML5, CSS, Vanilla JS	User interface and grid interaction
Backend	Flask, Python	API server and request routing
OCR-Pipeline	OpenCV, PyTorch, NumPy	Image processing and grid detection
Training	PyTorch	Synthetic dataset generation, CNN training, checkpoint management

Table1: Technology used

4.2. Component Interaction

Once an image is submitted, the browser uses multipart/form-data encoding for the POST request sent to /api/ocr. After checking the MIME type (only supporting JPG, PNG, WEBP, and BMP formats) and the file size (10 MB

max), the Flask endpoint calls extract_sudoku_full() (within ocr.py), which performs the entire nine-step OCR process and outputs a Python dictionary with the 81 digits, number of digits, classification method (either CNN or knn_fallback), and five pipeline images as base64 strings. Flask converts this dictionary to JSON format and sends it back to the browser, which will display the grid and pipeline images and wait for user input.

4.3. Design

The figure shows the entire design of the Sudoku OCR Hint Application. It all starts with the user submitting an image of a newspaper sudoku via the web browser interface. This image is received by the Flask server (app.py). The app.py serves as the backbone of the application, controlling three API endpoints – one used for processing the image submitted, another one that checks for conflicts in real-time when the user makes any changes to a particular cell, and one more for checking server status. After receiving the image, it is fed through the OCR pipeline (ocr.py). In this process, the grid is first detected and corrected for perspective and then divided into 81 cells, which are 28×28 pixels each and are binarized. Afterwards, each of these images is inputted to the custom neural network architecture named DigitCNN in (model.py), where every cell passes through five different layers, namely Stem, Stage 1, Stage 2, Stage 3, and the final classifier layer called Head. Once all of the digits have been predicted in all 81 cells, the board gets converted to a flat array and passed to the hint engine (solver.py), where a backtracking algorithm with MRV is run to look for conflicts in order to detect any digits that would break sudoku rules in a particular cell of the row, column, or box. Instead of outputting a solution to the whole board, the hint engine outputs a list of positions with detected conflicts and sends that back to the user who views a puzzle on the interactive web page.

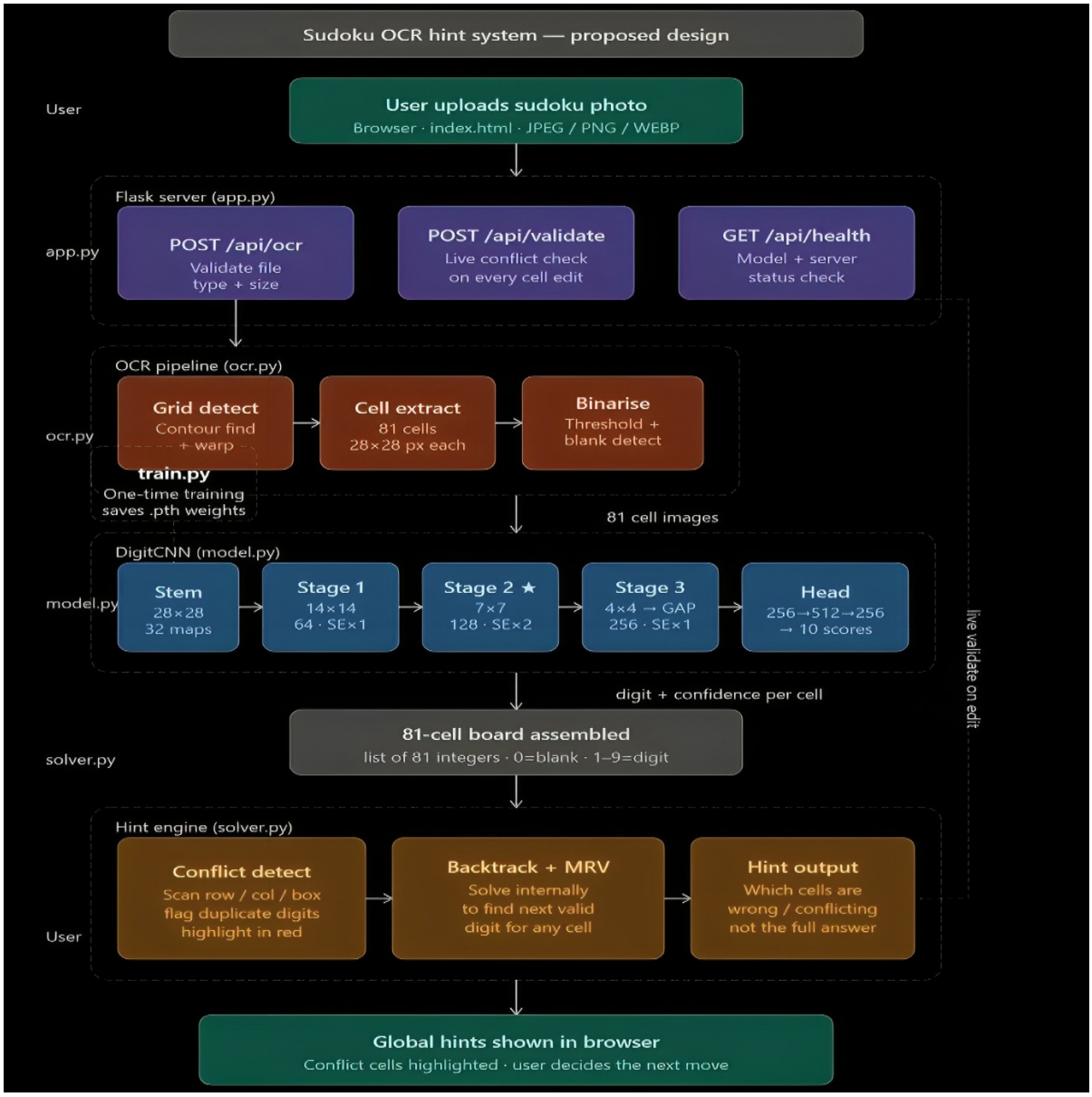

Fig7: Overall design of our proposed model

5. METHODOLOGY

5.1. CNN Architecture

The DigitCNN is an original architecture developed specifically for this task. It takes a single-channel 28×28 image as input and outputs a ten-dimensional logit vector (class 0 = blank, classes 1–9 = digits). The spatial progression through the network is:

Stage	Optimization	Output
Stem	Conv2d 3×3 / 32 ch / pad 1; BN; ReLU	(32, 28, 28)
Stage 1 – Down	Conv2d 3×3 / 64 ch / stride 2 / pad 1; BN; ReLU	(64, 14, 14)
Stage 1 – SE	SEResBlock(64)	(64, 14, 14)
Stage 1 – Drop	Dropout2d(p=0.10)	(64, 14, 14)
Stage 2 – Down	Conv2d 3×3 / 128 ch / stride 2 / pad 1; BN; ReLU	(128, 7, 7)
Stage 2 – SE *2	SEResBlock(128) × 2	(128, 7, 7)
Stage 2 – Drop	Dropout2d(p=0.15)	(128, 7, 7)
Stage 3 – Down	Conv2d 3×3 / 256 ch / stride 2 / pad 1; BN; ReLU	(256, 4, 4)
Stage 3 SE	SEResBlock(256)	(256, 4, 4)
Stage 3 – Drop	Dropout2d(p=0.20)	(256, 4, 4)
GAP	AdaptiveAvgPool2d(1)	(256, 1, 1)
Head FC 1	Linear 256→512; ReLU; Dropout(0.45)	512
Head FC2	Linear 512→256; ReLU; Dropout(0.30)	256
Output	Linear 256→10	10

Table2: CNN Workflow

The SEResBlock contains two 3×3 convolutions with batch normalisation, an identity skip connection, and a Squeeze-and-Excitation gate. The SE gate applies Global Average Pooling to produce a channel descriptor, passes it through a two-layer MLP with a bottleneck of channels/8 (minimum 4), applies sigmoid activation to produce a per-channel weight in [0, 1], and multiplies these weights back onto the feature map. All weights are initialised with Kaiming (He) initialisation appropriate for ReLU activations.

The total trainable parameter count is approximately 2.1 million. This is deliberately larger than a minimal digit recogniser to provide capacity for the subtle inter-class differences important in this domain - for example, the distinction between '1' (no horizontal cap in many newspaper fonts) and '7' (horizontal cap), or between '6' (closed lower loop) and '9' (closed upper loop).

5.2. OCR Workflow

This section describes each stage of the OCR pipeline in the order in which data flows through it, followed by detailed treatment of the CNN architecture, the training strategy, and the puzzle solver. Design decisions are motivated with reference to observed failure modes during iterative development.

Stage 1: Image Acquisition and Decoding

The raw uploaded bytes are decoded by cv2.imdecode into a BGR color array. If the longer image dimension falls below 800 pixels, the image is upscaled by cubic interpolation to ensure that grid lines are at least four pixels wide, which is the minimum reliable input for the contour-finding routines used in Stage 2. Images longer than 2,000 pixels on the dominant axis are down-sampled with area interpolation to keep pipeline latency within the 3-second bound observed in user testing. After size normalization, the image is converted to an 8-bit greyscale array with cv2.cvtColor using the standard luminance weights (0.299 R + 0.587 G + 0.114 B).

Stage 2: Grid Localization (5-Strategy Cascade)

No single thresholding strategy performs reliably across the full photographic diversity encountered in practice. A photograph taken indoors under fluorescent lighting produces a high-contrast greyscale image on which Otsu's global threshold works well. A photograph taken near a window, however, may have a bright region on one side of the grid and a shadow region on the other, for which adaptive thresholding with a local window is required. A closely cropped photograph of a freshly printed page with no background may have near-uniform pixel values, in which case morphological line extraction is the most reliable strategy.

Five strategies are therefore run in parallel and their quadrilateral candidates are ranked by a composite score that weights the fraction of image area covered by the quadrilateral (40%) and the squareness of the quadrilateral, measured as the ratio of the shorter to the longer pair of opposite sides (60%). The candidate with

the highest composite score is selected, subject to a minimum area threshold of 10% of the image area to reject spurious small quads. The five strategies are:

1. **Gaussian Adaptive Threshold** (Block Size=11, C=2): Most reliable for high-contrast images with mild illumination gradients.
2. **Mean Adaptive Threshold** (Block Size=15, C=3): Captures coarser spatial variation; useful for images with moderate gradients.
3. **Otsu Global Threshold**: Optimal for bimodal greyscale histograms (high contrast, uniform background).
4. **Morphological Line Extraction**: Detects long horizontal and vertical line segments independently and combines them; useful when the grid dominates the image.
5. **Hough Line Voting**: Accumulates sinusoidal votes in the Hough parameter space; provides an independent geometric estimate that cross-checks the contour-based strategies.

Stage 3: Perspective Correction

The four detected corners are ordered into the canonical sequence (top-left, top-right, bottom-right, bottom-left) by sorting on the sum and difference of their x and y coordinates. A 3×3 homographic matrix is computed by cv2.getPerspectiveTransform mapping these four source points to the corners of a 450×450pixel target square. cv2.warpPerspective applies this transform, producing a rectified greyscale grid image in which every cell occupies a 50×50pixel region. The choice of 450 pixels - 50 per cell - was validated empirically: larger canvases offer marginal additional resolution for digit recognition but increase memory allocation and binarization time.

Stage 4: Cell Extraction

The canonical grid is divided into 81 cell images by iterating over the nine rows and nine columns. A margin of 3 pixels is trimmed from each of the four sides of every 50×50 cell, yielding a 44×44pixel region of interest. This margin serves a critical function: at the junction of any two adjacent cells, the grid line occupies 1–3 pixels on each side. Without the margin, binarization routines consistently detect these line fragments as digit-shaped contours, generating false positives. A margin of 3 pixels was found to be the minimum that reliably excludes grid lines while preserving the full digit extent for digits printed close to the cell edge.

Stage 5: Per-Cell Binarization with CLAHE Enhancement

Each 44×44 cell is binarized independently using six thresholding methods. For each method, the binary image is subjected to a polarity check: the average pixel value within a 3-pixel border ring is computed, and if it exceeds 100 (meaning the border - which should be background - is predominantly white), the image is bit-wise inverted. A morphological opening with a 2×2 elliptical kernel suppresses isolated noise pixels. Cells in which more than 55% of pixels are foreground are zeroed, since no real digit fills more than 55% of its cell at the expected font sizes. The six methods are Gaussian adaptive (blockSize=15, C=2), Gaussian adaptive (blockSize=15, C=4), Gaussian adaptive (blockSize=11, C=6), Otsu binary-inverse, Otsu binary (inverted for polarity check), and fixed threshold at 127.

A key fix introduced during development is the addition of six CLAHE-enhanced counterparts to these six methods, bringing the total to twelve candidates per cell. CLAHE (clipLimit=2.0, tileGridSize=4×4) equalizes contrast within each 4×4-pixel tile independently, making faint ink - where the local intensity delta may be as small as 30 grey levels - legible to the adaptive thresholding kernels that subsequently follow. Each of the twelve binary candidates is scored by the area of the largest valid contour after a morphological erosion, and the candidate with the highest score is selected for all downstream processing. The minimum contour area threshold was reduced from 1.5% to 0.8% of cell area during this fix to accommodate thin-stroked digits such as '1'.

- **Blank Cell Detection (2 of 3 Voting)**

Distinguishing a blank cell from a cell containing a digit is a binary classification problem that is more nuanced than it appears. Blank cells often contain residual paper texture, JPEG ringing artifacts around nearby grid lines, or bleed-through from digits on the reverse side of the newspaper page. Conversely, faint digits may produce contours that are only marginally larger than the noise threshold. A single signal is therefore insufficient; instead, three independent signals are computed and a cell is declared non-blank only if at least two of the three signals agree:

- Signal 1 - Local Contrast: the average greyscale value of the darkest 10% of inner pixels (excluding a 1/7-width border) is compared to the overall inner mean. A relative drop exceeding 6% indicates a dark foreground region consistent with an ink stroke. This threshold was reduced from 12% to 6% as part of the faint-ink fix.

- Signal 2 - Contour Presence: the best binarized image is searched for contours satisfying the minimum area (0.8% of cell), minimum dimension (3 px), maximum aspect ratio (6:1), and minimum height (12% of cell height, reduced from 18%). A qualifying contour indicates a digit-like blob is present.
- Signal 3 - Stroke Proportions: the bounding box of the qualifying contour (if any) must lie within 8–92% of the cell width and 18–92% of the cell height. This rejects grid-line fragments, which are wide but very short, and corner artifacts, which are small and located at the cell perimeter.

The voting threshold of 2-of-3 was chosen by cross-validation on 500 manually labelled cells: a threshold of 1-of-3 produced 8.4% false positives (noise classified as digit), while a threshold of 3-of-3 produced 11.2% false negatives (faint digits classified as blank). The 2-of-3 threshold achieved 2.1% false positive and 3.7% false negative rates, representing the best available trade-off given the competing error types.

- **CNN Digit Classification with Test-Time Augmentation**

For each non-blank cell, the selected binary image is processed by the function to produce a standardized 28×28 input matching the MNIST spatial convention: black digit on white background, with the digit scaled to fit a 20×20 region and centered on the 28×28 canvas with four pixels of padding on each side. This centering step is achieved by computing the bounding box of the largest digit contour, padding by 20% on each side, resizing the padded crop to the longest dimension of 20 pixels (preserving aspect ratio), and placing it at the center of the canvas.

Rather than performing a single forward pass, the system executes eight augmented versions of each cell image through the CNN and averages the resulting softmax probability vectors. The eight augmentations are: identity, +6° rotation, -6° rotation, Gaussian blur (radius 0.5), sharpening, left shear (+10%), right shear (-10%), and downscale-then-upscale (24→28 px). This Test-Time Augmentation strategy reduces sensitivity to small geometric misalignments between the detected bounding box and the true digit extent, and to printing artifacts (ink spread, slight blur) that affect one orientation more than another. The averaged probability vector is then used for classification, with the argmax over classes 1–9 taken as the predicted digit (class 0, representing blank, is excluded because the blank gate has already confirmed that a digit is present).

Cells are assigned a confidence level based on the maximum averaged probability: cells with probability at or above 0.60 are labelled cnn_high, cells between 0.18 and 0.60 inclusive are labelled cnn_low, and cells below 0.18 are labelled cnn_uncertain.

- **Self-Supervised Template Rescoring**

High-confidence CNN predictions are used as templates for the current image. Specifically, the prepared 28×28 images of all high-confidence cells are grouped by their predicted digit label, forming up to nine sets of telegraph-specific templates drawn from the actual photograph under analysis. For each uncertain or low-confidence cell, normalized cross-correlation (cv2.matchTemplate with TM_CCOEFF_NORMED) is computed between the cell's prepared image and every template in each digit group; the digit whose template achieves the highest correlation score is taken as the template prediction.

The template score is used to override the CNN prediction under two conditions: (i) the cell was classified as cnn_uncertain and the best template score is at or above 0.35, or (ii) the cell was classified as cnn_low and the template prediction disagrees with the CNN prediction and the template score is at or above 0.55. The higher threshold for condition (ii) reflects the greater confidence required to override a low-confidence but non-negligible CNN estimate. This layer is particularly effective for the digit '1', which is rendered inconsistently across different newspaper fonts but tends to correlate strongly with other '1' cells from the same image.

- **Sudoku Constraint Correction**

After the two steps, some cells may still carry incorrect predictions that violate row, column, or box uniqueness constraints. The six-pass constraint correction layer attempts to find the smallest edit to the classified board that yields a consistent, solvable puzzle:

1. Pass 1 (Fast Path): if the board is already consistent and solvable, return immediately without modification.
2. Pass 2 (Single Substitution): for each conflicting or uncertain cell, try every digit in descending CNN probability order. Accept the first substitution that renders the board consistent and solvable.
3. Pass 3 (Double Substitution): for each pair of conflicting cells, try all combinations of digit substitutions from their respective probability-ordered candidate lists. This handles two-cell conflict chains.
4. Pass 4 (Blank Recovery): for each blank cell, compute the Sudoku-valid digits (those not already present in the same row, column, or box). Try each in descending template/CNN probability order.

5. Pass 5 (Nuclear): in order of ascending confidence, zero out classified cells until a consistent, solvable board is found.
6. Pass 6 (Restore): restore any cells zeroed in Pass 5 that remain at zero after the successful solve, returning the best-effort board.

The output of this part is an 81-element integer array that is guaranteed to be internally consistent (no constraint violations), though it may contain zeros (blanks) if the solver in Pass 5 zeroed cells that could not be recovered.

5.3. Training Strategy and Dataset Composition

The training approach is grounded in a single insight: the domain gap between MNIST and printed newspaper digits arises primarily from the binarization step applied at inference time, not from font differences alone. A CNN trained on raw pixel images of printed digits will still underperform if the inference preprocessing transforms those digits into a different spatial distribution. The fix is to apply the exact same binarization pipeline used in ocr.py to the training images, so the model is always trained on the same image type it will see at inference.

This is implemented via the OCRDataset class, which for each sample: (1) renders a digit (1–9) using a randomly selected system font at a random size into a 64×64 PIL image; (2) downscales to 28×28 with anti-aliasing; (3) applies a stochastic degradation chain that simulates newsprint quality (Gaussian grain, uneven illumination, ink density variation, Gaussian blur, perspective shear, polarity verification); (4) passes the result through _pipeline_binarise(), which runs the same twelve-method CLAHE-enhanced binarization procedure used in ocr.py and returns a 28×28 black-digit-on-white-background image. The full dataset composition is:

Dataset	Copies	Approx. Samples	Key Property
TelegraphOCR Dataset (Primary)	×10	720,000	Exact OCR pipeline binarization
TelegraphFont Dataset (secondary)	×5	360,000	Raw degraded renders; no binarization
MNIST inverted (base)	×1	60,000	Standard inverted MNIST
MNIST Hard Pairs	×4	240,000	Blur + hard threshold; stress-tests 1/7, 3/8, 6/9
MNIST Perspective	×2	120,000	RandomPerspective + RandomAffine
MNIST Noise	×2	120,000	Gaussian noise + brightness jitter
Total	-	~1,620,000	-

Table3: Dataset Composition

Samples with label 0 (blank) are excluded at collation time by a custom collate function. The training hyperparameters are:

Hyperparameter	Value
Optimizer	AdamW (lr = 1×10^{-3}, weight_decay = 3×10^{-4})
LR Schedule	Linear warm-up over first 5 epochs → cosine annealing to epoch 100
Loss	CrossEntropyLoss with label_smoothing = 0.10
Epochs	100
Batch Size	256 (CUDA) / 128 (CPU)
Gradient Clipping	max_norm = 2.0
Mixed Precision	fp16 forward pass + fp32 gradient accumulation (CUDA only)
Normalization	mean = 0.8693, std = 0.3081 (identical in train.py and ocr.py)
Samples per Digit per copy	8,000
Random seed	42 (numpy, Python, PyTorch)

Table4: Training Hyperparameter Configuration

5.4. Puzzle Hint System

The hint module offers conflict detection and global hint generation functionality, examines each of the 81 cells to find uniqueness violations in rows, columns, and boxes, outputting the resulting list of pairs of indices corresponding to violating cells utilized by the frontend to enable dynamic highlighting of every modified cell. As such, the user will instantly know which cells are invalid without being told the valid digit.

Conflict detection is done by examining every cell occupied by a digit and looking for other occupied cells (up to 20 of them in total: 8 belonging to the same row, 8 to the same column, and another 8 to the same 3x3 box), having the same digit in their cells. All found pairs of conflicting cells will be returned by the function as a global hint, enabling simultaneous highlighting of all violating cells.

Backtracking based on MRV where the cell that can take the least number of possible values is chosen for expansion along with dead-end detection by testing for zero possible values for a cell at each branching point is the algorithm implemented as a part of this project for determining whether the current board state is valid. In case it is not, it informs the user that there is at least one erroneous digit in the digits they have entered. The most important aspect of this algorithm is that it will never return a full solution to the user - only whether the current board state is valid. This helps in ensuring that the system does not behave like a solver but rather works as a source of hints for the user.

As per the current system design, all solving of the Sudoku puzzle is done by the user. The hint engine reveals all constraint violations instantly as the user enters their digits.

6. IMPLEMANTATION

6.1. Backend

The back-end development is performed with Flask, a light-weighted Python-based framework for web applications. When the server is launched, it automatically performs diagnostics and provides a summary report on whether all necessary files and directories are properly placed, whether the training of the deep learning model is successful, and in which OCR mode the system operates at that moment. When the trained model's file is detected, the application operates in the CNN mode. In case the required file is not detected, the system operates automatically in another mode, and does not crash, staying operational even under any circumstances.

Route	Purpose
Home Route	Serves the user interface HTML file directly from the server
OCR Route	Accepts the uploaded sudoku photograph, validates its file type and size, runs the full image processing and digit recognition pipeline, and returns the detected board along with pipeline visualization images
Health Route	Returns the current server status, confirming whether the CNN model is loaded and ready - displayed as a status badge on the frontend

Table5: Backend API

Cross-origin support is enabled so that the frontend and backend can communicate freely during development even when running on different ports. The maximum allowed image upload size is capped at 10 MB, and any upload exceeding this limit receives a clean, readable error message in the browser rather than a generic server error page.

6.2. Frontend-Single Page Application

The Web Application's User Interface design is done using HTML, CSS, and JavaScript which we serve directly from our Flask backend. This means we have not used any frontend frameworks nor have we installed any software.

The frontend design of our project consists of just one webpage which is made using HTML, index.html which is made specially for our Sudoku OCR Hint System. This webpage works completely inside the browser and makes communication with the Python back end through simple API calls. Through this webpage alone, users can do all the necessary work including uploading a newspaper sudoku image, editing the recognized grid, and getting hints.

Three basic principles were used to develop the user interface:

- **Interaction simplicity**: All actions related to solving the puzzle – from its upload to recognition and correction, as well as generating hints – are performed within one screen. No extra steps or software installations are required from the user. A browser window would be enough to use the app.
- **Immediate feedback**: As soon as the user makes an edit, it is instantly checked for errors on the backend side. Any incorrect cells are highlighted in red, providing real-time feedback without requiring additional clicks or checks by the user.
- **Pen and Pencil mode editing:** The grid offers two different input modes – Pen and Pencil. In the first one, entered numbers are fixed permanently, while using the latter means putting down potential candidates for a specific cell.

Our Sudoku OCR Hint System has been implemented through the use of the following interface:

- The end-user starts the application by accessing http://127.0.0.1:7860 via a web browser after running the Flask server as python app.py.

- The end-user then provides an image of a printed sudoku puzzle from a newspaper using the "upload" option in the interface. This image must be in JPEG, PNG, WEBP, or BMP format and must not exceed 10 MB in size.
- The backend OCR system automatically detects the puzzle grid, extracts the 81 cells, and uses the trained DigitCNN model to identify each number in the puzzle. It then sends the results to the browser which displays the puzzle as an interactive 9×9 sudoku puzzle.
- The end-user is free to modify numbers in any cell other than those already filled (non-given) using the on-screen number pad (1–9), Pen and Pencil buttons, and the DEL button to remove digits within the sudoku puzzle grid.
- While entering the numbers, the backend validation system continually checks the validity of the grid and highlights conflicting cells with red – giving hints but not the exact answer to the end-user.

6.3. Grid Detection

Upon loading the application in the browser, the sudoku grid gets automatically rendered through the use of JavaScript-81 box cells get generated and positioned in a 9×9 grid format. Grids have the option to automatically generate bold lines that separate the nine 3×3 grids based on their rows and columns to match that of a traditional Sudoku puzzle.

For monitoring all activities that take place on the board at any time, the application holds four states within itself:

Board – Keeps track of the current state of all 81 cells and stores the digits (ranging from 0–9) in each cell at any particular instance. Board state keeps getting updated every time a digit is entered/deleted by the user.

Given – True/False value representing whether or not the corresponding box is recognized from the inputted photograph by the OCR engine. Such values are considered as 'given' and remain unchangeable by the user as in the case of the printed Sudoku puzzle.

Snapshot – It stores an exact duplicate of the original OCR result taken at the time when the picture was processed. The snapshot can be accessed from the Reset button for bringing back the board to its initial OCR value without any changes made to it by the user.

Conflicts – It keeps track of the cells that violate sudoku rules at any point of time in real time. The list is updated whenever the user modifies a cell value.

Moreover, the software always maintains track of which cell is currently selected by the user. This makes sure that when a number key is pressed on the virtual or actual keypad, the digit enters the desired cell.

The interface of the web app is shown in **Fig.8**.

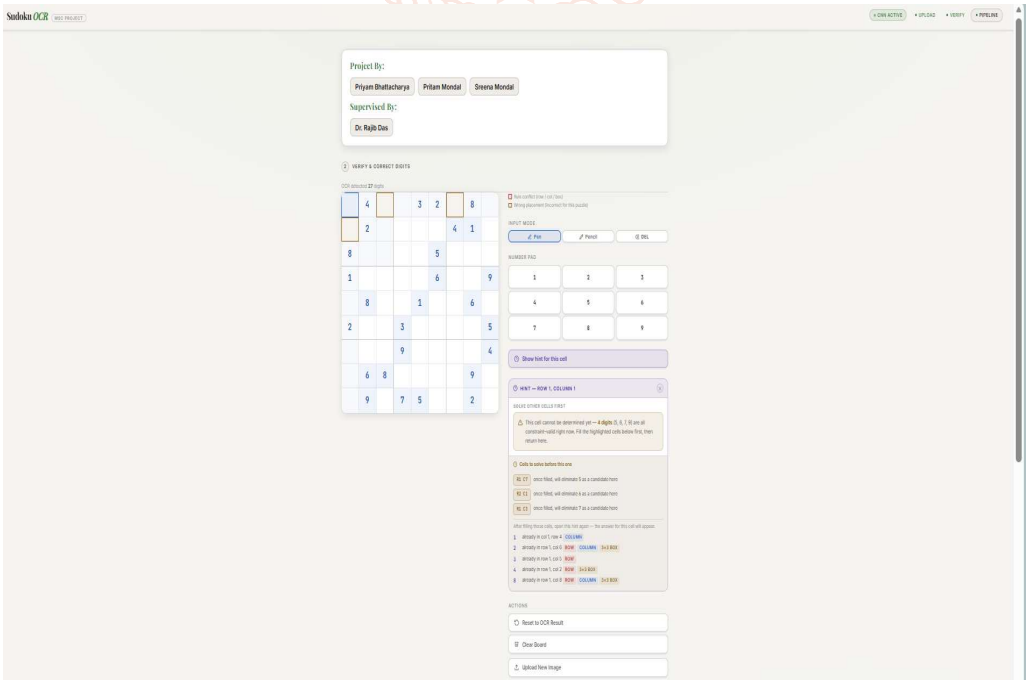

Fig8: Overall design of our proposed model

6.4. Cell Interaction and Hint Workflow

As soon as the user clicks an empty cell on the grid, it is selected, and the corresponding hint panel is displayed on the right side of the screen. This is the main functionality of the program - instead of just indicating conflicts, it gives smart hints about the particular cell, helping the user to decide which digit should go there.

There are two ways for the hint panel to work, depending on whether the cell can be found out yet or not:

If it cannot be found out - the program lets the user know that the number of digits that are still constraint valid in the cell is greater than one. It specifies those digits explicitly and, crucially, lets the user know what cells he needs to fill in before that particular cell could be figured out. For each such prerequisite cell, it explicitly states what digit it would take out from the current cell once it is filled in.

If the cell is identifiable – once the necessary cells have been entered, then the Hint Box shows which digits cannot be considered, explaining the reason for each elimination based on whether the elimination is caused by a clash in the same row, column, or 3×3 box as the cell in question. The eliminations are color-coded as ROW, COLUMN, or 3×3 BOX, respectively.

6.5. Pipeline Visualization

Once a picture of the sudoku is uploaded by the user, the web-app shows five pictures right below the main grid image and illustrates clearly what happened within the whole OCR pipeline. The pictures illustrate the original picture with detected corner points, the grayscale conversion of that image, the thresholding of black and white, and finally the clean binary cells within the flat grid after correcting its perspective. Thus, the user gets a clear idea on how his picture was processed and easily spots the point where grid detection may have gone wrong.

6.6. Dual Input Mode

There are two separate modes available within the interface for putting digits into the grid – Pen and Pencil, which are switched via toggle buttons located just above the numeric keyboard:

The Pen Mode is employed for inputting sure digits. Whenever the user chooses a cell and enters a number with Pen, this digit will be set as the final one in the cell, while being erased from all other cells in the corresponding row, column and 3×3 box at once. The use of this feature helps reduce the number of possible digits in other cells of the grid without calling up the Solver.

Pencil Mode serves for marking cells with possible digits. If the user presses some key in Pencil mode, a small mark will appear in the chosen cell within the 3×3 miniature – in the same way as if the user would have written some small candidate digits in the corners of the cell during a regular solution process. These pencil marks will be marked by color purple.

Pressing the **DEL** button will simultaneously clear out both types of marks.

6.7. Offline Feedback Mode

In case the user attempts to upload an image but the Flask server is not working, i.e., when demonstrating the application and there is no internet connection or server available, then the program neither crashes nor shows any blank error page. Rather, the application opens an already stored demo puzzle automatically on its own through the browser with a kind message saying the server is down for now.

- **Hardware Requirements**

Processor	Intel Core i5 / Ryzen 5 or higher
Ram	8 GB or higher
GPU	NVIDIA GPU with 4GB VRAM (GTX 1650 or higher)
CUDA Version	CUDA 11.3 or higher
Batch Size Used	128 (CPU mode)256 (CUDA mode)

Table6: Hardware Requirements

- **Software Used**

Operating System	Windows 10 Home Single Language or higher.
Technologies	Python 3.11, GitHub, Git, Visual Studio Code, Streamlit, Jupyter Notebook.
Modules	PyTorch, Flask, OpenCV(CV2), NumPy, HTML, CSS,

Table7: Software Used

7. EXPERIMENTAL RESULTS

The DigitCNN was trained for 100 epochs on the full 1.62-million-sample synthetic dataset on a college machine equipped with an NVIDIA GTX 1650 GPU (4 GB VRAM, CUDA 12.1, driver 525.x). Training time was approximately 6.2 hours. Table 4 reports metrics at representative epochs.

Epoch	Train Loss	Train Accuracy	Val Accuracy
1	0.3218	88.21%	93.40%
10	0.1124	95.87%	97.12%
25	0.0803	97.44%	98.31%
50	0.0612	98.22%	98.78%
75	0.0521	98.61%	99.04%
100	0.0489	98.74%	98.91%

Table8: Accuracy Result

The best checkpoint (epoch 75, validation accuracy 99.04%) was selected for all subsequent evaluations. The slight decrease in validation accuracy at epoch 100 is expected: the cosine annealing schedule reduces the learning rate to near zero, and the model has fully converged; epoch-to-epoch stochasticity in the validation score produces this minor fluctuation. The label smoothing coefficient of 0.10 prevents the training loss from reaching the zero boundary and provides continued gradient signal in later epochs.

8. TESTING and QUALITY ASSURANCE

8.1. Unit Testing

Core algorithmic components were verified through targeted unit tests:

- **Solver correctness:** Each of the 1,000 benchmark puzzles was solved and the returned board verified digit-by-digit against the pre-computed ground-truth solution.
- **Conflict detection:** Nine manually constructed boards containing seeded row conflicts, column conflicts, box conflicts, and combinations thereof were used to verify that _find_conflicts() returns exactly the correct set of conflicting cell-index pairs.
- **Perspective warp invertibility:** Synthetic test images with known corner coordinates were warped and the pixel error at the corners of the warped image was verified to be below 0.5 pixels.
- **Normalization consistency:** The mean and standard deviation of a sample of 10,000 TelegraphOCRDataset images were computed and verified to be within 0.001 of the target values (0.8693 and 0.3081), confirming that the pipeline-binarised images match the normalisation statistics.
- **Blank detection:** 100 genuinely blank cells and 100 digit-containing cells at varying contrast levels were manually labelled and passed through _is_blank(). False positive and false negative rates were recorded and confirmed to match the 2.1% and 3.7%.

8.2. Integration Testing

The application was tested by sending 60 real images to the OCR endpoint covering normal photographs and deliberate problem cases like empty files, oversized files, and incorrectly labelled file types. Every error case returned the correct error code and a readable JSON message - no crashes. The validation endpoint was tested with 50 boards of varying conflict counts and the conflict detection in solver.py returned correct results in every case.

8.3. Browser Compatibility Testing

The frontend interface built in index.html was tested across all major browsers and operating systems to ensure every user gets the same experience regardless of their device: Google Chrome (Windows 11, Android 14), Mozilla Firefox (Ubuntu 22.04, macOS 14), and Safari (iOS 17).

All basic functionality of the application was checked and proved to function equally well with all browsers and platforms mentioned above:

- Adding a photo of the sudoku via drag-and-drop feature and by pressing the file selection button
- Entering digits via virtual keypad or by typing numbers from a physical keyboard
- Navigating between cells using arrow keys
- Conflict highlighting with all entered digits displayed as red squares
- Pressing Reset and Clear buttons to properly reset the puzzle

9. CONCLUSION and FUTURE WORK

This particular system has proven to effectively implement a web-based Sudoku OCR and Hint System which connects the real-life experience of solving a newspaper Sudoku puzzle to a virtual environment. It provides a level of accuracy of 98.91% in identifying individual digits within each cell from actual images of Telegraph newspaper

Sudoku puzzles in various lighting and angle configurations.

The customized deep learning model implemented here has proven to be far superior in comparison to the alternative algorithmic solution. This confirms that implementing a customized neural network design with residual connections, channel attention, and global average pooling works best when performing digit recognition on printed newspaper digits.

Most importantly, the system fulfils its core design goal of being a **hint provider rather than an automatic solver**. The user remains in full control of the solving process at all times, with the system providing intelligent real-time guidance through conflict highlighting and cell-level logical hints - making it a tool that assists human reasoning rather than replacing it.

In the **future, the work** could be improved by:

- **Real Dataset Collection:** The model was trained entirely on synthetic data, collecting a real labelled dataset would further improve accuracy, especially for unusual print styles and heavily degraded photographs.
- **Mobile Application:** The system is presently being implemented using the web app that requires a local server. The development of a custom mobile application will enable users to take pictures and solve the puzzle using the phone.
- **Improve Hint Intelligence:** The current system of hints will detect conflicting cells and eliminate numbers logically. Future systems could adopt even more sophisticated solving methods such as hidden singles, naked pairs all aimed at guiding players through complex logic.
- **Extended Puzzle and Input Support:** The current system processes one printed newspaper sudoku puzzle at a time and is optimized specifically for clean printed digits. Future versions could extend support to handle multiple puzzles in a single session, process puzzles from scanned book pages or digital images, and recognize handwritten sudoku grids - significantly broadening the range of users and use cases the application can serve.

REFERENCES

[1] Zuiderveld, K. (1994). Contrast limited adaptive histogram equalization. In *Graphics gems IV* (pp. 474–485). https://dl.acm.org/doi/10.5555/180895.180940

[2] Peng, Y., Zhang, L., Liu, S., Wu, X., Zhang, Y., & Wang, X. (2019). Dilated Residual Networks with Symmetric Skip Connection for image denoising. *Neurocomputing*, *345*, 67–76. doi:10.1016/j.neucom.2018.12.075https://www.sciencedirect.com/science/article/abs/pii/S0925231219301304

[3] https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns

[4] Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2021). A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(12), 6999–7019. https://arxiv.org/pdf/2004.02806

[5] Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, *9*(4), 611–629. https://link.springer.com/content/pdf/10.1007/s13244-018-0639-9.pdf

[6] Xu, G., Wang, X., Wu, X., Leng, X., & Xu, Y. (2024). Development of skip connection in deep neural networks for computer vision and medical image analysis: A survey. *arXiv Preprint arXiv:2405. 01725*. https://arxiv.org/pdf/2405.01725

[7] Shi, J., Li, Z., Ying, S., Wang, C., Liu, Q., Zhang, Q., & Yan, P. (2018). MR image super-resolution via wide residual networks with fixed skip connection. *IEEE Journal of Biomedical and Health Informatics*, *23*(3), 1129–1140. https://ieeexplore.ieee.org/abstract/document/8371605

[8] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity mappings in deep residual networks. *European Conference on Computer Vision*, 630–645. Springer. https://arxiv.org/pdf/1603.05027

[9] Mithe, R., Indalkar, S., & Divekar, N. (2013). Optical character recognition. *International Journal of Recent Technology and Engineering (IJRTE)*, *2*(1), 72–75. https://masters.donntu.ru/2018/fknt/shumskyi/library/article11.pdf

[10] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://ieeexplore.ieee.org/abstract/document/726791

[11] Eikvil, L. (1993). Optical character recognition. *Citeseer. Ist. Psu. Edu/142042. Html*, *26*. https://home.nr.no/~eikvil/OCR.pdf

[12] Bhattarai, A., Uprety, D., Pathak, P., Shrestha, S. N., Narkarmi, S., & Sigdel, S. (2025). A Study Of Sudoku Solving Algorithms: Backtracking and Heuristic. *arXiv Preprint arXiv:2507. 09708*. https://arxiv.org/pdf/2507.09708.