JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

SUDOKU SOLVER USING COMPUTER VISION AND DEEP LEARNING

Dr.J.B. Jona, Abhishek A, Ashok G

Associate Professor, Student, Student
Department of Computer Applications,
Coimbatore Institute of Technology, Coimbatore, India

\

Abstract: This project presents a comprehensive solution to the problem of solving Sudoku puzzles from natural images using computer vision and deep learning techniques. The process involves image processing, digit classification, puzzle solving, and visualization of the solution. The integration of OpenCV for image processing and Keras for deep learning contributes to a robust system capable of handling varied lighting conditions, angles, and backgrounds in Sudoku puzzle images.

Keywords **- Sudoku, Computer Vision, Deep Learning, OpenCV, Keras, Image Processing, Puzzle Solving, Convolutional Neural Network, Backtracking Algorithm, Pygame.**

I. INTRODUCTION

Sudoku puzzles pose an interesting challenge in computer vision and artificial intelligence. This project aims to develop a Python-based solution that can efficiently solve Sudoku puzzles extracted from natural images. The methodology involves multiple steps, including image processing, digit classification, puzzle solving, and visualization of the solution. The process is to take an image of a sudoku puzzle, extract the puzzle grid, identify and classify digits in each cell, solve the puzzle using a recursive backtracking algorithm, and finally display the solution back on the original image. OpenCV was used for image processing, and Keras for the deep learning component.

II. METHODOLOGY

1. Image Processing

The process begins with capturing an image of a Sudoku puzzle. Utilizing OpenCV, an adaptive threshold is applied to obtain a binary image. The contours of the binary image are computed to identify the main puzzle grid. A perspective transform is then applied to achieve a bird's-eye view of the puzzle grid. Contours within the transformed grid are computed to locate individual cells, and the information about each cell is stored.

2. Digital Classification

For digit classification, a convolutional neural network (CNN) is constructed using Keras. The CNN is trained on a dataset consisting of images of numbers 1 to 9 in various fonts and the MNIST hand-written digits dataset. The model's digit class predictions are used to populate the puzzle grid cells during the solving process.

3. Puzzle Solving

A recursive backtracking algorithm is employed for solving the Sudoku puzzle. The 2D array representing the puzzle grid is passed to an instance of the Sudoku Solver class. The solver returns a solution, even if multiple solutions exist. The solved puzzle is then displayed on the original image.

4. Visualization

Integrate a visualization component to display the solved Sudoku puzzle on the original image, providing users with a visual representation of the solution.

Abbreviations and Acronyms

1. penCV: Open-Source Computer Vision
2. Keras: Keras is an open-source deep learning API written in Python, which runs on top of other popular deep learning frameworks, such as TensorFlow and Theano.
3. CNN: Convolutional Neural Network
4. MNIST: Modified National Institute of Standards and Technology (MNIST) is a large database of handwritten digits widely used for training various image processing systems.
5. Pygame: Pygame is a cross-platform set of Python modules designed for writing video games.

III. IMPLEMENTATION

3.1 Importing Libraries and Modules:

- OpenCV (cv2): Used for computer vision tasks, including image loading and processing.
- Operating System (os): Utilized for checking file existence and handling file paths.
- Matplotlib (plt): Used for plotting and visualizing images.
- NumPy (np): Essential for numerical operations and array manipulations.
- TensorFlow (tf): Deep learning framework for loading and using the trained model.
- Argparse (argparse): Used to parse command-line arguments.
- Custom Modules (sudoku_utils, SudokuSolver): Contains utility functions and the Sudoku solver class.

3.2 Main Function for Sudoku Puzzle Solving:

- Input Validation: Checks if the provided image file exists.
- Image Loading and Preprocessing: Loads the Sudoku puzzle image using OpenCV, converts the color space from BGR to RGB, and resizes the image for display.
- Visualization: Plots and displays the original Sudoku image.

3.3 Loading Trained Model and Making Predictions:

- Loading Trained Model: Loads the deep learning model (CNN) using TensorFlow.
- Image Processing: Utilizes sudoku utils functions to locate grid cells in the image and obtain the 2D array representing the Sudoku puzzle grid.
- Sudoku Solver Instance: Creates an instance of the SudokuSolver class and attempts to solve the puzzle using the solve method.

3.4 Displaying Results:

- Solution Verification: Checks if there are no empty cells (zeros) in the Sudoku grid, indicating a successful solution.
- Visualization of Solution: If the puzzle is solved, generates an annotated image of the solved puzzle using utility functions and displays it.
- Failure Message: If the puzzle is not successfully solved, prints a message indicating that the puzzle could not be solved.

3.5 Command-Line Argument Parsing:

- Command-Line Argument Parsing: Uses the argparse library to parse command-line arguments, including the file paths for the Sudoku image and the trained model.
- Script Execution: Calls the solve_sudoku_puzzle function with the parsed arguments.
- Display Results: Displays the Matplotlib plots showing the original image and, if applicable, the annotated image of the solved Sudoku puzzle.

IV. TRAINING THE DATA

Training the data model involves the following architecture.

- Input Layer: Accepts input images of shape (28, 28, 1), representing grayscale images of size 28x28 pixels.
- Convolutional Layers: Two sets of convolutional layers with ReLU activation, followed by max-pooling layers for feature extraction and spatial reduction.
- Flatten Layer: Flattens the output from the convolutional layers into a 1D array.
- Dropout Layer: Introduces dropout to reduce overfitting during training.
- Dense Layer (Output Layer): A fully connected layer with 9 neurons (assuming digits 1-9) and a softmax activation function for multi-class classification.
- Compilation: Configures the model for training with categorical crossentropy loss, the Adam optimizer, and accuracy as the evaluation metric.

```
def build_model():
    # Define a CNN model for digit classification
    model = keras.Sequential([
        keras.Input(shape=(28, 28, 1)),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(9, activation="softmax")]
    )

    model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

    return model

def main(args):
    data_choice = args['data']
    batch_size = args['batch_size']
    epochs = args['epochs']
    model_save_fpath = args['model_save_fpath']
    exclude_fonts = args['exclude_fonts']

    # Load data depending on user choice
    x_train, x_val, x_test, y_train, y_val, y_test = prep_data.get_data(data_choice=data_choice,
                                                exclude=exclude_fonts)


    # Get a model instance
    model = build_model()
    # Train the model
    print("Starting training...")
    model.fit(x_train, y_train,
            validation_data=(x_val, y_val),
            batch_size=batch_size,
            epochs=epochs)
    print("Training complete")
```

Main Training Function

The training of the main function involves

```
def main(args):
    data_choice = args['data']


    batch_size = args['batch_size']
    epochs = args['epochs']
    model_save_fpath = args['model_save_fpath']
    exclude_fonts = args['exclude_fonts']

    # Load data depending on user choice
    x_train, x_val, x_test, y_train, y_val, y_test = prep_data.get_data(data_choice=data_choice,
                                                exclude=exclude_fonts)


    # Get a model instance
    model = build_model()
    # Train the model
    print("Starting training...")
    model.fit(x_train, y_train,
            validation_data=(x_val, y_val),
            batch_size=batch_size,
            epochs=epochs)
    print("Training complete")
```

- Data Loading: The script loads training, validation, and test data based on user preferences (data_choice) and optional exclusion of fonts (exclude_fonts).
- Model Instantiation: Creates an instance of the CNN model using the build_model function defined earlier.
- Training Configuration: Initiates the training process using the fit method. The training data (x_train and y_train) are used, along with validation data (x_val and y_val) for monitoring performance during training.
- Training Parameters: Configurable parameters include batch size (batch_size), the number of training epochs (epochs), and the file path for saving the trained model (model_save_fpath).

Sudoku Game Module

This Sudoku game implementation provides an engaging and interactive experience, allowing users to play Sudoku puzzles, receive hints, and check their solutions.

```
from sudoku_solver_class import SudokuSolver
import sudoku_utils as sutils


class Button:
    def __init__(self, ui, text, x_pos, y_pos, width, height, enabled):
        self.ui = ui
        self.text = text
        self.x_pos = x_pos
        self.y_pos = y_pos
        self.enabled = enabled

        self.button_width = width
        self.button_height = height

        self.draw()


    def draw(self):


  # Define colours based on clicked and enabled states
        if self.enabled:
            button_bg_colour = "dark gray" if self.check_click() else "light gray"
            fg_colour = "black"
        else:
            button_bg_colour = "light gray"
            fg_colour = "dark gray"


        font = pygame.font.Font("freesansbold.ttf", 18)
        button_text = font.render(self.text, True, fg_colour)
        button_text_rect = button_text.get_rect()
        text_width, text_height = button_text_rect.width, button_text_rect.height

        text_x_offset = (self.button_width - text_width) // 2
        text_y_offset = (self.button_height - text_height) // 2

        button_rect = Rect((self.x_pos, self.y_pos), (self.button_width, self.button_height))

        # Draw the button with rounded corners
        pygame.draw.rect(self.ui.window, button_bg_colour, button_rect, 0, 3)
        # Create a border around the button
        pygame.draw.rect(self.ui.window, fg_colour, button_rect, 2, 3)
        self.ui.window.blit(button_text, (self.x_pos + text_x_offset, self.y_pos + text_y_offset))
```

1. Game State Management (GameState class):

- Handles the state of the Sudoku game, including the board, selected cells, current puzzle, solved puzzle, and various game-related flags.
- Manages puzzle difficulty levels and loading puzzles from text files.
- Integrates a Sudoku solver to provide hints and check user answers.

2. User Interface (UserInterface class):
- Uses Pygame for GUI components, including window setup, event handling, and rendering.
- Manages buttons for actions such as starting a new game, resetting the puzzle, requesting hints, checking answers, showing solutions, and changing difficulty levels.
- Allows the user to interact with the Sudoku board by clicking cells, entering numbers via keyboard input, and using arrow keys for navigation.
- Supports drag-and-drop functionality for loading Sudoku puzzles from images.

3. Button Class (Button):
- Represents interactive buttons in the GUI.
- Handles button drawing, click detection, and visual feedback.

4. Cell Class (Cell):
- Represents individual cells on the Sudoku board.
- Tracks whether a cell is selected, the assigned number, and whether it is a fixed number.

5. Integration with Sudoku Solver (SudokuSolver and sudoku_solver_class):
- Uses a Sudoku solver to validate user answers, provide hints, and check the correctness of entered numbers.

6. Game Loop:
- Runs a game loop using Pygame to continuously update the game state, handle user input, and render the GUI.
- The loop allows for a smooth interactive experience, with events triggered by user actions.

7. Graphics and Color Coding:
- Utilizes Pygame's drawing functions to render the Sudoku board, buttons, and various visual elements.
- Implements color coding to highlight selected cells, indicate correctness, and provide visual feedback to the user.

V. RESULTS

The Output of the sudoku solver is as follows

Success - sudoku solved!

7 6 2 | 3 5 8 | 9 4 1

9 3 4 | 1 6 7 | 2 8 5

5 8 1 | 4 9 2 | 3 7 6

- - - - - - - - - - -

8 1 3 | 6 7 4 | 5 9 2

4 5 9 | 2 1 3 | 7 6 8

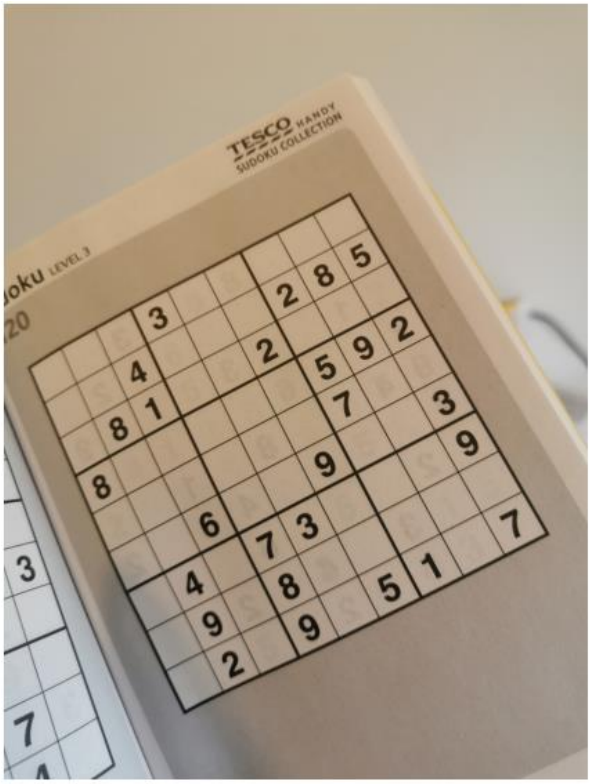
2 7 6 | 5 8 9 | 4 1 3

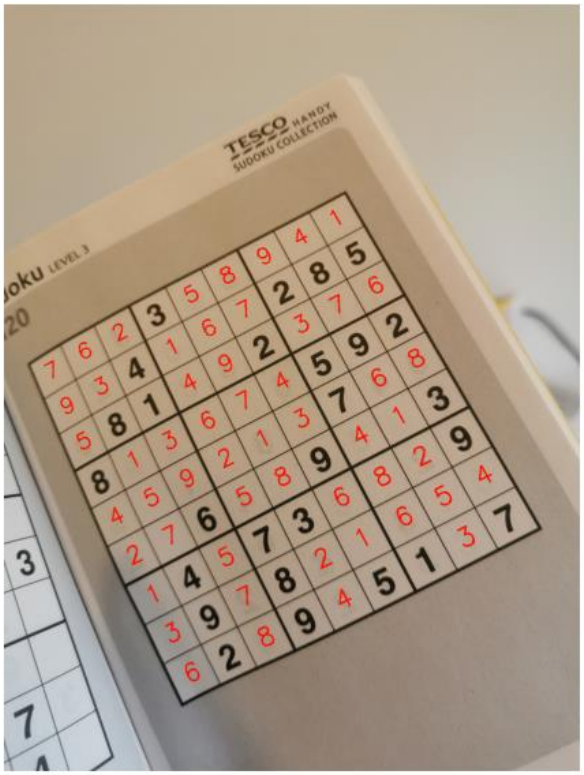
- - - - - - - - - - -

1 4 5 | 7 3 6 | 8 2 9

3 9 7 | 8 2 1 | 6 5 4

6 2 8 | 9 4 5 | 1 3 7





VI. Acknowledgment

The authors would like to express their gratitude to Akshay Gupta from analytics Vidhya for helping me learn about deep learning techniques. Special thanks are extended to Abhishek Sharma for their contribution towards sudoku solving using image processing.

https://machinelearningprojects.net/sudoku-solver/#google_vignette

https://www.analyticsvidhya.com/blog/2021/05/solving-sudoku-from-image-using-deep-learning-with-python-code/

References

[1] https://www.geeksforgeeks.org/sudoku-backtracking-7/
references for learning backtracking algorithms

[2] https://www.coursera.org/projects/create-your-own-sudoku-solver-using-ai-and-python
On creating sudoku solver problem using deep learning and image processing techniques.