NeurASP

Leveraging ASP to improve Deep Learning Models

Vedang Gupta (200100166)
Aniket Pokle (20d070011)

Goals

- To demonstrate that reasoning can help to identify perception mistakes by neural networks that violate semantic constraints, which in turn can make perception more robust.
- Implement clingo based sudoku solver.
- Implement a neural network to identify digits in a sudoku grid.
- Implement Anti-Knight Sudoku, and Sudoku-X.
- (Additional) Implemented Sudoku Puzzle Generator.

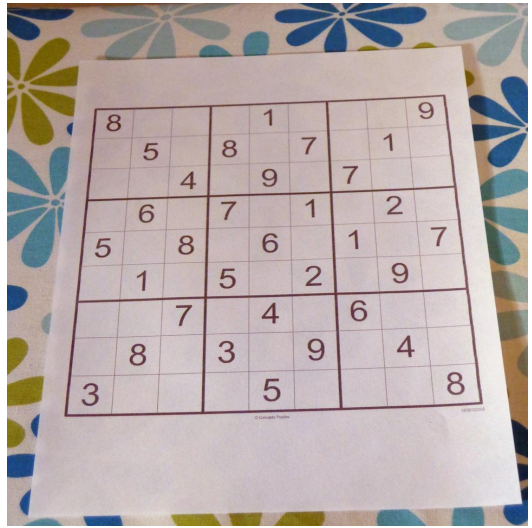
Digit Recognizer and Sudoku Transform

Accomplished

- Trained a CNN model on MNIST dataset to recognize sudoku digits
- Created a pipeline to identify the sudoku grid and recognize the digits in the given sudoku
- Input the recognized puzzle to the Clingo Sudoku Solver

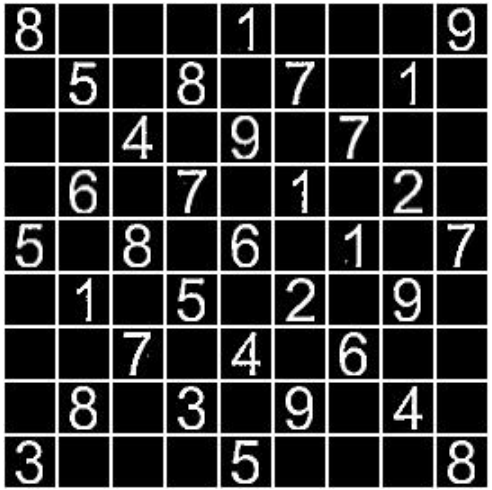
Explaining the pipeline

- The input to the program is a sudoku image clicked by user
- We then use OpenCV to extract and align the sudoku from the input image



Input Sudoku Image

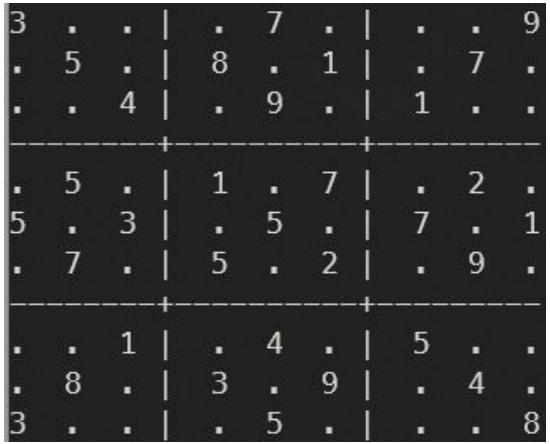
Transformed Sudoku



- Next we get the cells of the sudoku by rescaling it to 450x450 and storing every 50x50 instance of it
- Now if the cell consists of a digit, it is fed to the digit recognition model to identify the digit else a zero is stored to get a 9x9 numpy array of zero's and the identified digits.

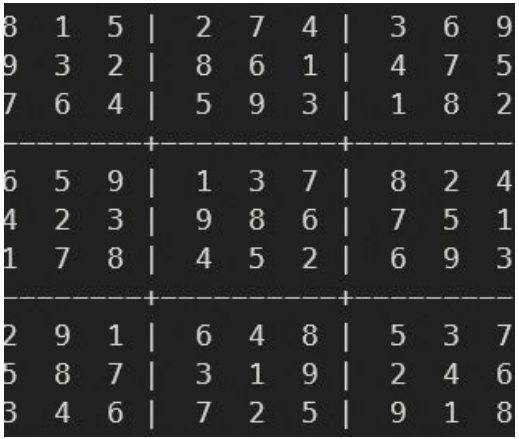
Continued...

- Now from the available array of digits we create an input data file (data.lp) for the logical reasoner (implemented using clingo) to check if any rule for a given sudoku variant is violated



Output Grid w/o logical reasoner

Final Output



- Any recognised digit(s) violating the sudoku rules are dropped from the grid
- After this another data file is created which is given as a input to the clingo sudoku solver
- The output of clingo sudoku solver is then parsed using clingo-python wrapper and reprojected on the terminal

Python Front End and Sudoku Solver

Accomplishments

- Implemented a Sudoku Solver using Clingo.
- Thoroughly tested and verified it for different outputs.
- Used Python-Clingo to integrate the Sudoku Solver with python frontend
- Leveraged Clingo to verify if the identified Sudoku by the model is valid and satisfies all constraints and removed invalid numbers.
- Used Clingo backend to Solve different variants of Sudoku.

Variants of Sudoku and clingo code explained

- First, we check if our digits detected satisfy the sudoku constraints using clingo.
- We have three constraints which are to be satisfied - No same numbers in a row, no same numbers in a column and no same numbers in a 3x3 box.
- These are simple to implement and were done as follows:

```
invalid(R1, C1, X) :- given(R1, C1, X), given(R2, C1, X), R1 != R2.

invalid(R1, C1, X) :- given(R1, C1, X), given(R1, C2, X), C1 != C2.

invalid(R1, C1, X) :- given(R1, C1, X), given(R2, C2, X), R1/3 = R2/3,
C1/3 = C2/3, R1 != R2, C1 != C2.
```

- Here, we are marking the coordinates of the “invalid” numbers, and note that due to clingo’s nature, it will mark both the numbers, but this is what we want since it is impossible to say which is the correct one.

Continued...

- Then, we made a sudoku solver, where we applied above the constraints and the constraint that the given numbers must be part of the solution.
- We also implemented two variants - Sudoku-X (no same numbers on both diagonals) and Anti-Knight Sudoku (no same numbers at knight moves from each other).
- We implemented their solvers and validity checkers with the following extra lines:

```
invalid(R1, C1, X) :- given(R1,C1,N), given(R2,C2,N), |R1-R2|+|C1-C2|=3.
(anti-knight sudoku)
```

```
invalid(R1, C1, X) :- given(R1,C1,N), given(R2,C2,N), R1+C1=8, R2+C2=8, R1 != R2.

invalid(R1, C1, X) :- given(R1,C1,N), given(R2,C2,N), R1=C1, R2=C2, R1 != R2.
(sudoku-x)
```

Possible Improvements

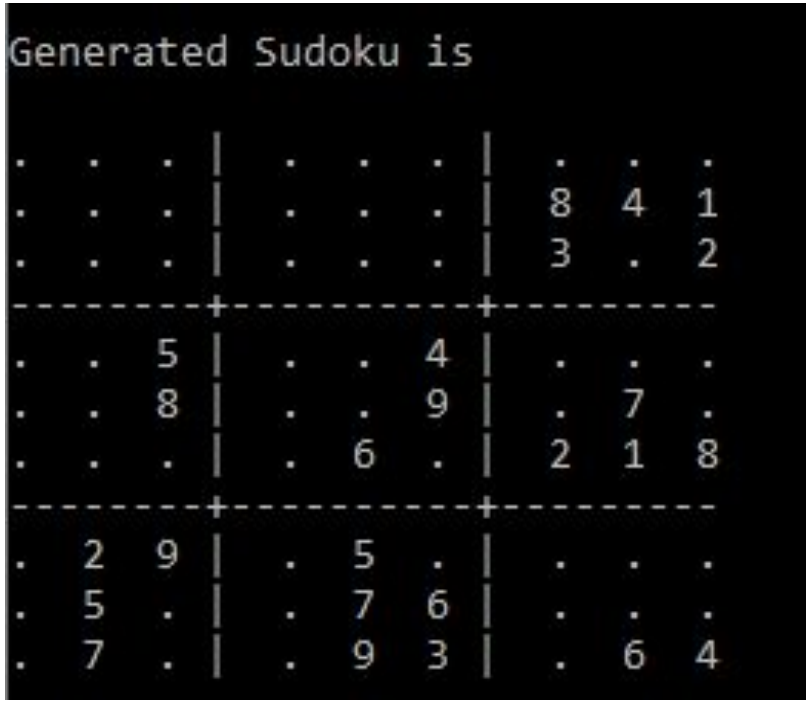
- Instead of removing the invalid numbers altogether, we can take the next best number by extracting the probability vector from the neural network.
- We also note that, there are always two invalid points, so we can choose the best one and discard the other by the probability vector as well.
- Reproject the solved Sudoku puzzle back on the input image.

Additions to the Project - Sudoku Puzzle Generation

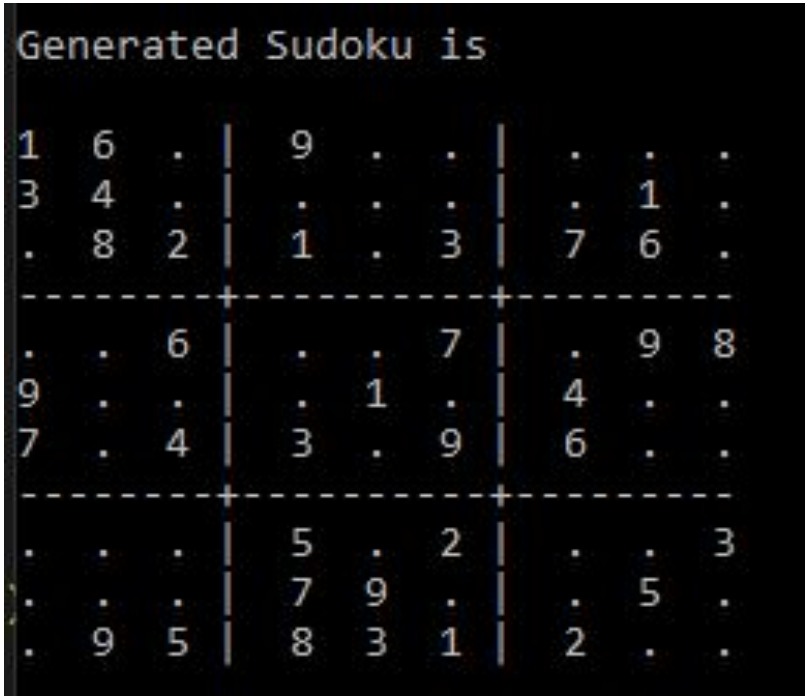
- After the presentation, upon Sir’s suggestion, we considered the problem of generating sudoku puzzles.
- For this, we first use clingo to generate a random solution and then remove numbers to obtain a difficult puzzle.
- For this, we first did the naive approach of generating and randomly removing numbers and then checking if it had a unique solution.
- We would keep increasing the number of cells erased until this was not the case and then print the puzzle.

Sudoku Puzzle Generation - A Heuristic Approach

- The naive randomised puzzle generator is not very effective and creates easy puzzles with about half the cells erased.
- So, we have implemented a more heuristic based generator which traverses the cells row by row and erases cells as long as they give a unique solution.
- This gives us very good and difficult puzzles but is still far from the actual minimum, but that is a NP problem to search.
- We can also use different traversal patterns and difficulty levels by changing constraints on the number of remaining cells and row-column constraints. (not implemented but only easy incremental changes to code required)



Heuristic Approach



Naive Approach

Code and References

Please read the README before executing the code.

Github :

https://github.com/aniketp02/CS449-NeurASP

References:

https://www.ijcai.org/proceedings/2020/0243.pdf

http://zhangroup.aporc.org/images/files/Paper_3485.pdf