Sudoku AI Solver

CSE 847: Machine learning

Project Report

Bhanu Kaushik Kanamarlapudi

kanamar1@msu.edu

1 Introduction

Sudoku is a logic based combinatorial number placement puzzle and it's one of the popular logic-based games. These puzzles are usually found in newspapers and magazines. Solving them has been widely fascinating over the decades. It exploits individuals' abilities for relational reasoning and pattern development. Logic based games have been shown to help delay neurological disorders like Alzheimer's and dementia[11]. Sudoku originated from Japan in the year 1984. It became famous very soon among various age groups. The main objective of the puzzle is to fill the remaining empty squares, using all the numbers between 1-9 exactly once in each row, column and the 3x3 sub grids. Players rely on positional as well as temporal dependencies between the numbers on the board.

To solve sudoku, a certain number of rules to be followed to accept the final output as a valid solution. The most popular rule and search-based approaches to solving sudoku utilize heuristics search algorithms such as back tracking, A* and tree-based search algorithms. These methods incorporation of the rules of sudoku into the solving techniques as heuristics in order to increase the likelihood of convergence. But the convergence of these methods comes at the cost of computational speed[2][3]. Hard level puzzles take longer time to get a valid solution hence requiring more computational resources.[8] applied various constraints with belief propagation models. Other approaches such as membrane computing[10] and Boolean operation [9] have also been proposed to find effective and fast solutions. To overcome this rule based solving approaches, machine learning techniques can be used as the testing time for ML models are usually less than the time needed to run an entire search algorithm. There have been multiple attempts to solve these puzzles using Machine learning techniques. There is a mobile app named Magic Sudoku[3] that solves lightening fast with an accuracy of 99%. Optnet [7] uses a network architecture that integrates optimization in the form of quadratic problems for solving sudoku. This allows the learning algorithm to learn sudoku problems purely from the data with high accuracy. However, it has a cubic complexity to the number of variables and constraints which makes the learning algorithm to have less number of hidden layers in order to guarantee real time predictions. This compromises the accuracy of the predictions made by the learning algorithms. Recurrent relational networks [1] obtain SOTA results by solving 96.6% of the hardest sudoku puzzles.

Since the sudoku puzzles are available online and in newspapers and magazines, it will be easier to take an image and use computer vision techniques to solve the puzzle, thus by passing the manual step of collecting and entering the numbers.

2 Datasets

Two different datasets have been used as part of this project :-

1. MNIST – This dataset contains 60,000 training examples and 10,000 examples for testing. Each example is of dimension 28x28[5]. This dataset contains all types of images like blurred, rotated ect.
2. Sudoku Dataset – This dataset used to train the CNN model is taken from the Kaggle, which was developed using a computer program to generate 1 million valid sudoku puzzles with their solutions by Kyubong Park[12]. It has two columns namely quizzes and solutions. Each row represents an entire sudoku puzzle order left to right and top to bottom with 0's representing blank cells.

3 Methodology

The process starts by training either KNN model or CNN model on MNIST dataset based on user's choice. An image is sent as input. Various computer vision techniques have been used to preprocess the image to identify grids and recognize digits and generate the 9x9 array. This extracted 9x9 grid is sent to CNN learning model which was trained on the Kaggle dataset to solve the sudoku puzzle. Below Figure[1] is the high-level overview of the process.

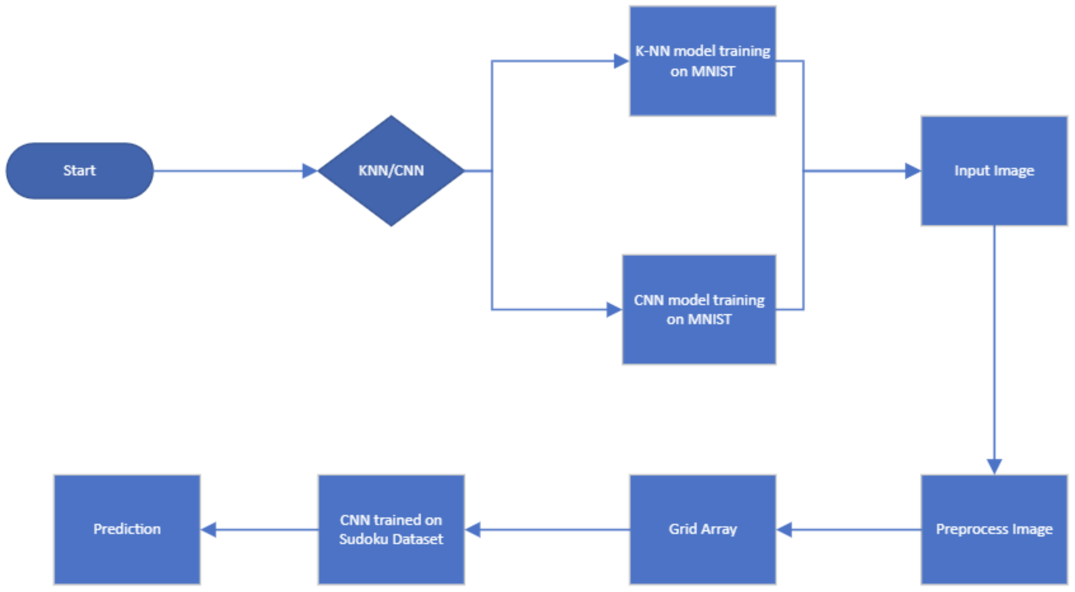

Start
KNN/CNN
K-NN model training on MNIST
CNN model training on MNIST
Input Image
Preprocess Image
Grid Array
CNN trained on Sudoku Dataset
Prediction

Figure [1]: Workflow

3.1 Training KNN/CNN learning models

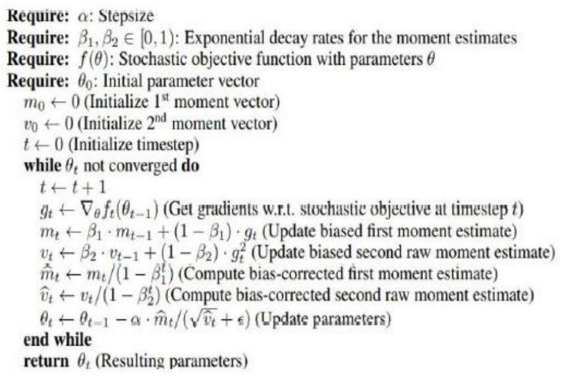
The K-nearest neighbor(KNN) works on the assumption that similar things or data points exist in close proximity to each other. KNN captures the idea of similarity and proximity. A Convolution neural network a.k.a Convnet or CNN assigns importance to objects/aspects present in the image and differentiate one from the other. Both learning models have been trained on MNIST dataset. It is believed that if the learning model achieves high accuracy on MNIST dataset, the chances of the model working on any kind of image is more.

3.1.1 K-Nearest Neighbor

Sklearn package has been used to train the KNN learning model. The model has been run multiple times on various k-values and finally has been set to 3. The data is split into 75% train and 25% test.

3.1.2 Convolution neural network

Tensorflow package has been used to train the CNN learning model. The model consists of 9 layers including 2 2D-Convolution layers. Each of the inner layer uses ‘relu’ and the model uses ‘softmax’ as final layer to generate probability values over the K(1-9) expected values. Cross entropy is used as the loss mechanism. The model utilized the adaptive momentum estimation (Adam) optimizer for learning.



```
**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0,1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)
```

Figure[2]: Adam Algorithm

Adam optimizer uses two moments of gradients namely first moment(m_t) and second moment (v_t). The first moment involves the exponential decaying average of the previous gradients and the second moment involves exponential decaying average of the previous squared gradients. The Adam optimizer then uses the combined averages of previous gradients at different moments to update the parameters better adaptively.

3.2 Computer Vision techniques

Once the learning models are trained, computer vision techniques to the image inputted will be applied. OpenCV has been used to apply these cv techniques to identify the borders and the digits present in the image. Below are the step-by-step techniques used –

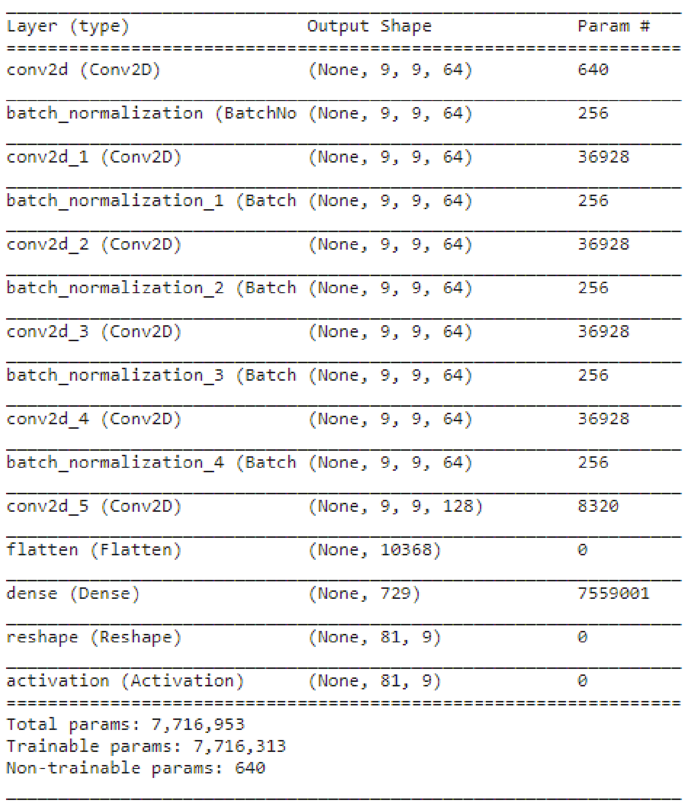
- Gaussian Blur – To reduce the noise and detail
- Adaptive Gaussian Thresholding – to account for different illuminations in different parts of the image.

- Inverting – Making digits and grid lines white and background black
- Dilation – A 3x3 kernel filter has been used to thicken and fill out cracks in the border lines.
- Flood Filling – Board will be the largest connected component, so flood filling from various seed points to find all the connected components. This method was used to address the drawback of the contour approach.
- Eroding – to undo the effects of dilation
- Hough Line Transform – To find all the lines
- Merging – Merge related lines that are close to each other
- Finding the extreme lines – Finding the outer border lines on top, bottom, left and right
- Adaptive Thresholding and Inverting
- Black filling to remove white patches using flood filling.

Once the above steps have been applied on the image, the trained KNN or CNN model is applied in order to identify and recognize the digits in the image and form a 9x9 array.

3.3 Prediction

The dataset collected from the Kaggle is used to train the second convolution neural network. 95% of the data has been for training the model. The below image gives more insights about the model layers and the parameters.

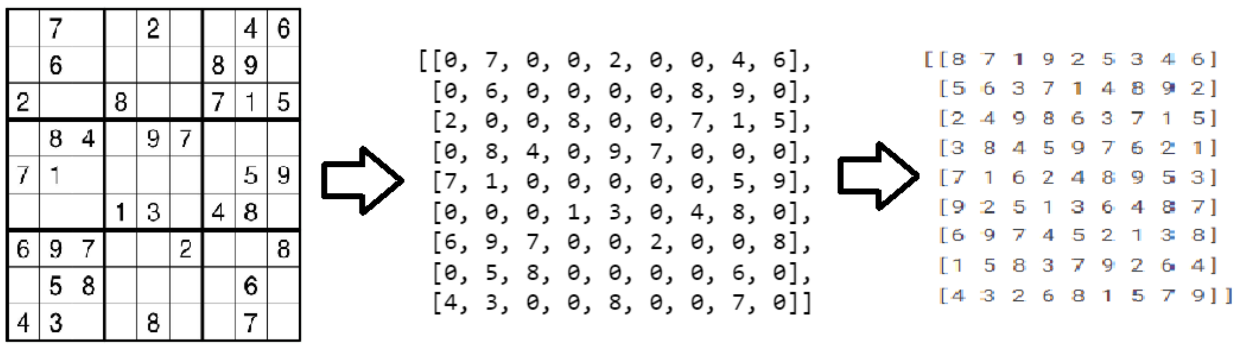


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 9, 9, 64)          640       
_________________________________________________________________
batch_normalization (BatchNo (None, 9, 9, 64)          256       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 9, 9, 64)          36928     
_________________________________________________________________
batch_normalization_1 (Batch (None, 9, 9, 64)          256       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 9, 64)          36928     
_________________________________________________________________
batch_normalization_2 (Batch (None, 9, 9, 64)          256       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 9, 9, 64)          36928     
_________________________________________________________________
batch_normalization_3 (Batch (None, 9, 9, 64)          256       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 9, 9, 64)          36928     
_________________________________________________________________
batch_normalization_4 (Batch (None, 9, 9, 64)          256       
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 9, 9, 128)         8320      
_________________________________________________________________
flatten (Flatten)            (None, 10368)             0         
_________________________________________________________________
dense (Dense)                (None, 729)               7559001   
_________________________________________________________________
reshape (Reshape)            (None, 81, 9)             0         
_________________________________________________________________
activation (Activation)      (None, 81, 9)             0         
=================================================================
Total params: 7,716,953
Trainable params: 7,716,313
Non-trainable params: 640
_________________________________________________________________
```

Figure[3]: CNN Model summary

Like earlier CNN model, all the layers of this model except the last layer uses ‘relu’ as activation mechanism and the last layer uses ‘softmax’ as the activation mechanism. This model uses ‘sparse categorical cross entropy’ as loss mechanism. Similar to previous CNN model, this model too uses Adam optimizer. Metrics such as val_accuracy and val_loss have been used. Techniques such as ReduceLROnPlateau has been used for reducing the learning rate when the metrics stops improving and avoid overfitting. Initial kernel filter sizes have been set to 3x3 to allow the model to focus on solving the 3x3 subgrids. The 1x1 convolution layer used 9 filters, changing the shape of the final dimension such that softmax could output 9 probabilities over labels 1-9. An argmax function is then applied on the output of the softmax to pick the most probable number for the corresponding chosen cell.

The model has been tested with a range of epochs. Since the train dataset is huge, the performance of the model didn’t improve much after 10 epochs. This reduced a lot a training time. Once the model has been trained, the sudoku array generated will be sent as input for the model to predict the solution. A custom function has been written to make predictions until all the 0’s are filled with numbers between 1-9. Figure [4] shows an example of how the above 3 methods.



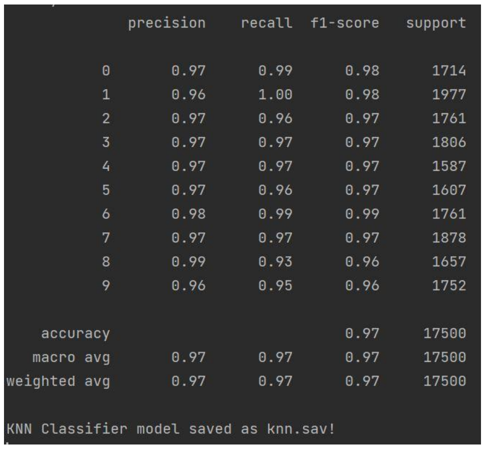
```
[[0, 7, 0, 0, 2, 0, 0, 4, 6],
 [0, 6, 0, 0, 0, 0, 8, 9, 0],
 [2, 0, 0, 8, 0, 0, 7, 1, 5],
 [0, 8, 4, 0, 9, 7, 0, 0, 0],
 [7, 1, 0, 0, 0, 0, 0, 5, 9],
 [0, 0, 0, 1, 3, 0, 4, 8, 0],
 [6, 9, 7, 0, 0, 2, 0, 0, 8],
 [0, 5, 8, 0, 0, 0, 0, 6, 0],
 [4, 3, 0, 0, 8, 0, 0, 7, 0]]
```

```
[[8 7 1 9 2 5 3 4 6]
 [5 6 3 7 1 4 8 9 2]
 [2 4 9 8 6 3 7 1 5]
 [3 8 4 5 9 7 6 2 1]
 [7 1 6 2 4 8 9 5 3]
 [9 2 5 1 3 6 4 8 7]
 [6 9 7 4 5 2 1 3 8]
 [1 5 8 3 7 9 2 6 4]
 [4 3 2 6 8 1 5 7 9]]
```

Figure[4]: Workflow Example

4 Results

The primary metrics used in evaluating the accuracy of the KNN model were recall, precision, f1-score, and support. The macro average and weighted average of the model is 0.97 for all the above metrics. Figure [5][a] gives more details of the classification report of the model. The first CNN model which was used to recognize and identify the digits in the images has been trained with a batch size of 200 and 10 epochs. The model achieved an accuracy of 0.9894 on test dataset. When tested on various images which were preprocessed using the above-described computer vision techniques, the CNN model was able to identify the digits more accurately than the KNN model. The second CNN model was trained on using 950,000 sample and used a batch size of 640. The model was able to achieve an accuracy of 99.2 % on the train set and 87% on the test set samples. Figure [5][b] shows the accuracy of the model against number of epochs.



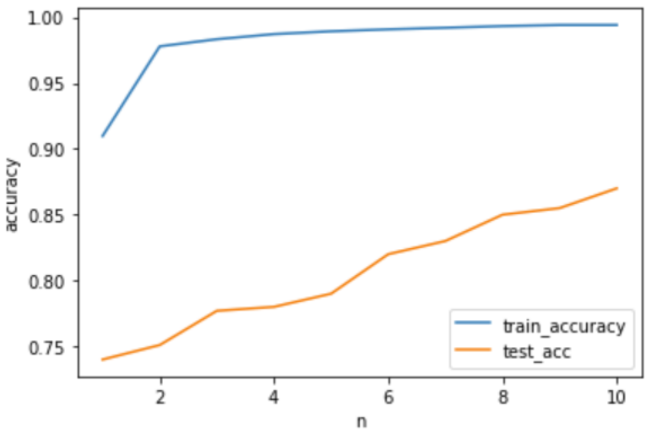
```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      1714
           1       0.96      1.00      0.98      1977
           2       0.97      0.96      0.97      1761
           3       0.97      0.97      0.97      1806
           4       0.97      0.97      0.97      1587
           5       0.97      0.96      0.97      1607
           6       0.98      0.99      0.99      1761
           7       0.97      0.97      0.97      1878
           8       0.99      0.93      0.96      1657
           9       0.96      0.95      0.96      1752

    accuracy                           0.97     17500
   macro avg       0.97      0.97      0.97     17500
weighted avg       0.97      0.97      0.97     17500

KNN Classifier model saved as knn.sav!
```

Figure[5][a]:KNN classfication report



Figure[5][b]:CNN model accuracy

5 Conclusion & Future Work

In this project, I have shown that it is possible to solve sudoku using Machine and Deep learning techniques. Computer vision techniques have been applied to efficiently identify the edges and recognize the digits in an image. I employed the use of K-nearest neighbor learning architecture and CNN architecture to identify and recognize the digits present in the image. The CNN model outperformed KNN model in recognizing the digits. The second CNN model was structured efficiently and consequently performed better at solving sudoku quizzes. A User Interface can be developed to make the application more friendly. The dataset used for training contains more easy and medium level puzzles and fewer hard level puzzles. The model performance can be improved by training the model with more difficult puzzles. Regularization techniques such as L2 can be applied to remove overfitting of the models when trained using more epcohs. Since the steps for solving a sudoku puzzle is sequential and decisions made in previous steps impact the next decisions, Recurrent Neural Networks specifically LSTM models can be used to solve the puzzles. Further, incorporating reinforcement learning techniques along with recurrent relational networks will even allow for higher accuracies. The code for this project can be found at - https://github.com/bhanukaushik/CSE847-Machine_Learning/tree/main/project.

6 References

[1] Palm, Rasmus, Ulrich Paquet, and Ole Winther. "Recurrent relational networks." *Advances in Neural Information Processing Systems* 31 (2018).

[2] Ace.ucv.ro, 2018. Available at http://ace.ucv.ro/anale/2012_vol2/05_Nicolae_Ileana.pdf

[3] Simonis, Helmut. "Sudoku as a constraint problem." *CP Workshop on modeling and reformulating Constraint Satisfaction Problems*. Vol. 12. Citeseer, 2005.

[4] Behind the magic, 2017. Available at https://medium.com/@braddwyer/behind-the-magic-how-we-built-the-arkit-sudoku-solver-e586e5b685b0

[5] Deng, Li. "The mnist database of handwritten digit images for machine learning research [best of the web]." *IEEE signal processing magazine* 29.6 (2012): 141-142.

[6] Adam optimizer, 2014. Available at https://arxiv.org/abs/1412.6980

[7] Proceedings.mlr.press, 2018. Available at
http://www.proceedings.mlr.press/v70/amos17a/amos17a.pdf

[8] Chowdhury, Abu Sayed, and Suraiya Akhter. "Solving Sudoku with Boolean Algebra." *International Journal of Computer Applications* 975 (2012): 8887.

[9] Moon, Todd K., Jacob H. Gunther, and Joseph J. Kupin. "Sinkhorn solves sudoku." *IEEE Transactions on Information Theory* 55.4 (2009): 1741-1746.

[10] Diaz-Pernil, Daniel, et al. "Solving sudoku with membrane computing." *2010 IEEE fifth international conference on bio-inspired computing: theories and applications (BIC-TA)*. IEEE, 2010.

[11] Everyday health, 2021. Available at https://www.everydayhealth.com/senior-health/alzheimers-disease/12-fun-brain-games-adults/

[12] 1 Million Sudoku Games, 2017. Available at https://www.kaggle.com/datasets/bryanpark/sudoku