

Tic-Tac-Toe Image Recognition and Gameplay with Minimax AI

Olaoluwa Malachi - 3787306¹

¹ University of New Brunswick, Saint John, NB E2K 5E2, Canada.
olaoluwa.malachi@unb.ca

Abstract. This project's main objective is to develop an automated system that is capable of identifying Tic-Tac-Toe game states from images, by splitting the board into individual cells, predicting the next best move for the computer making use of Minimax algorithm, and visualizing the results. The AI model is able to classify X, O, and empty cells from board images, allowing the users to see the current state of the game and the next optimal move the computer should make. The core of the project lies in the combination of Convolutional Neural Networks (CNNs) for symbol classification and the Minimax algorithm for strategic decision-making.

Keywords: Minimax, Tic-Tac-Toe, Convolutional Neural Network (CNN).

1 Introduction

Tic-tac-toe, also known as Noughts and Crosses, is one of the most prominent childhood games globally due its simplicity and strategic gameplay. The game involves two players placing different colored or shaped game pieces on a 3×3 grid with X or O marked spaces. However, unlike many other popular board games, such as checker, weiqi (go), and chess, the grid of Tic-tac-toe games makes it accessible anywhere a grid can be drawn, such as dusty windshields, beach sands, napkins, etc. To play the game, each of the two players takes turns to mark an unoccupied position on the grid until the grid is filled, or one of the players align three of their symbols (X or O), horizontally, vertically, or diagonally to win the game.

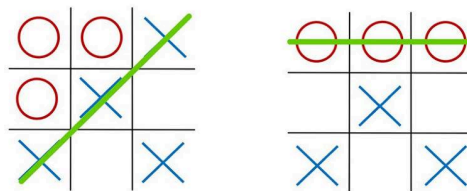


Fig. 1. Winning scenarios in the Tic Tac Toe game board.

Machine Learning (ML, Machine Learning) [1-5] is a fundamental and essential field of image processing and pattern recognition [6, 7] which are now widely implemented in computer science research and entertainment. After identifying a system dynamic patterns, software adapts and optimizes its functionality

to suit user's needs per time. In the regular image recognition algorithms, image preprocessing, feature extraction and classifier are separated from each other [8-11]. Although Tic Tac Toe looks simple to play, it requires the use of combinatorial analysis to get the best out of it. As shown in Table 1 below, There are 255168 game scenarios in Tic Tac Toe game, in which 46080 of them are a draw and 209088 end with O or X winning. By integrating image and pattern recognition into Tic-Tac-Toe to identify patterns on the game grid, such as player moves, potential winning combinations, and defensive strategies, players will have an enhanced new level of interactivity and gameplay.

Table 1. A Detailed analyze of Tic Tac Toe game

	X Wins	O Wins	Overall Solution
5 Moves win	1440	0	0.6%
6 Moves win	0	5328	2.1%
7 Moves win	47952	0	18.8%
8 Moves win	0	72576	28.4%
9 Moves Win	81792	0	32.1%
Total Wins	131184	77904	81.9%
Draw		46080	18.1%
Total		255168	100%

1.1 Problem Statement

Tic-tac-toe is a simple childhood game mostly played by students across the world. Imagine playing Tic Tac Toe with your friend, but instead of taking time to determine the moves, a computer does. However, before this could happen, the computer needs to firstly 'see' the game board and understand each game position. Then, it has to think about the optimal move it can make to help you win or block the opponent from winning. This project is about teaching the computer to do both.

The hard part is making the computer understand what's on the board (is it an X, an O, or empty?). Another challenge is teaching it to choose the best move, just like you would when playing.

1.2 Key Idea

A CNN Model is trained to recognize Tic Tac Toe symbols (X, O, or blank) from images. The CNN acts as the 'eyes' of the computer. Once the board is understood, we use the Minimax algorithm to figure out the best move for the computer. Minimax works by simulating every possible move and its consequences, choosing the one that maximizes the computer's chances of winning while minimizing the chances for the opponent.

2 Literature Review

2.1 Related Works

Tic-Tac-Toe has been largely explored for AI concepts and research due to its simplicity and well-defined rules [12-13]. It has a finite number of possible game states, making it an excellent model for implementing decision-making algorithms like Minimax, Negamax or Bayes estimator. According to Eppes [14], games are the ideal platform and environment for learning more about artificial intelligence, suggesting that simple games like Tic-Tac-Toe are perfect for testing strategies and algorithms in controlled settings. The integration of image recognition in gaming has transformed user interactions. [12, 14] explored how computer vision enhances mobile gaming with features such as AR, gesture recognition, and object detection and how image recognition bridges the gap between physical and digital gaming environments, a concept central to the Tic-Tac-Toe AI model. Convolutional Neural Networks (CNNs), revolutionized the field by enabling machines to recognize patterns and objects in images with high accuracy. This breakthrough laid the foundation for integrating image recognition into mobile apps. Relatively similar technological advancement and gameplay have been seen in games like Pokémon GO and Chess where real-world objects and locations are recognized and integrated into gameplay. [7-9] showcased how image recognition and geolocation can create immersive experiences. While the Tic-Tac-Toe AI model does not rely on AR, its use of image recognition to interpret hand-drawn grids follows a similar principle of merging physical and virtual spaces.

Image recognition, a subset of computer vision used in simulating human image recognition, has seen rapid advancements due to the proliferation of deep learning techniques that gave rise to many proposed image recognition models which involve using mathematics and computer technology to preprocess the obtained target image information and extract the feature [15]. The classifier classifies the image into corresponding categories, and then gets a recognition result from matching with the stored data.

3 Methodology

3.1 Overview

The goal of this research is to build an AI model that recognizes Tic-Tac-Toe grids from images and analyzes the game state to determine the next best move using Convolutional Neural Networks.

The methodology for the Tic-Tac-Toe AI leverages the use of machine learning for image recognition and AI algorithms (Minimax) to simulate decision-making in the game. In this section, we will explore the method used to build Tic-Tac-Toe AI.

3.2 Implementation

For the Tic Tac Toe AI model, we adopt a Convolutional Neural Network (CNN) model trained with datasets consisting of individual cells of the TicTacToe game board. The board image is preprocessed using the OpenCV libraries with the following details:

- **Image Decoding:** For the conversion of raw image bytes, `cv2.imdecode`.
- **Edge Detection:** `cv2.Canny` used for the detection of the Tic Tac Toe grid lines.
- **Contour Detection:** extracts the grids and symbols from the images, `cv2.findContours`.
- **Perspective Transformation:** To align the grid for uniform analysis, `cv2.getPerspectiveTransform`.
- **Thresholding:** for the conversion of image to binary for better symbol detection, `cv2.threshold`.

With these, it processes image data of Tic Tac Toe boards using OpenCV libraries for image handling, TensorFlow for symbol classification via a Convolutional Neural Network (CNN), and NumPy for efficient data manipulation. The system generates a virtual game board by analyzing the classified symbols (X, O, and blank), then determines the next optimal move for O using the Minimax algorithm, a decision-making technique that evaluates possible outcomes to maximize winning chances. Finally, it returns the predicted move and the updated board state as a visualized response.

3.3 Datasets

The Tic Tac Toe AI model was built using a dataset that consists of 494 labeled instances for training, and 120 labeled instances for evaluation, each labeled “Cross”, “Noughts” and “Blank” representing the three possible states of Tic-Tac-Toe grid cells: X, O, and Blank. To ensure consistency in the image resolution, grayscale conversion, and environment noise reduction, the dataset was preprocessed using OpenCV. TensorFlow and Keras were used for designing and training the CNN model, with an architecture optimized for capturing spatial hierarchies in Tic-Tac-Toe grids.

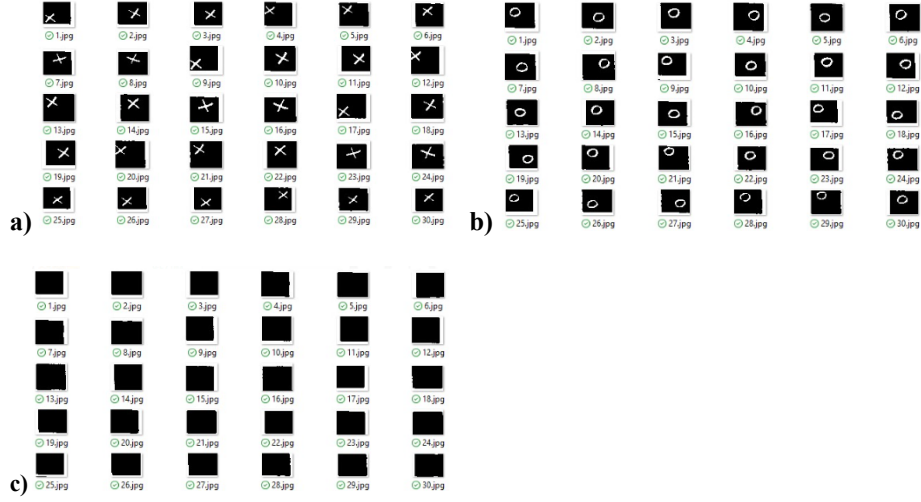


Fig. 2. A snapshot of the dataset for individual cells of the Tic-Tac-Toe board.
a) Cross “X” b) Noughts “O” c) Blank “ ”

3.4 Convolutional Neural Network

Convolutional Neural Network was used for symbol recognition because of its high ability to capture spatial hierarchies and features in images. However, this process of image recognition involves several key steps to ensure accurate analysis and interpretation [16].

Convolutional Neural Network Image Classification

First, image acquisition captures the necessary visual information through optical signals and analog signals, this will provide a form of basis for the dataset that is required for subsequent processing. This step entails the conversion of the signals from input sources such as cameras, image conversion cards, or sensors into electronic data. It can be in the form of a one-dimensional waveform, two-dimensional text, images and so on, while a dynamic video is extracted into single frame images. After this, image preprocessing can be done to transform between images but prepare the raw data for analysis and use. This is the stage that includes techniques like grayscale of color images, image smoothing, noise reduction in the images, contrast enhancement, and edge detection to refine the image and isolate relevant features that are needed. Eliminating the extraneous elements aids in achieving the effect of segmentation of image, and preprocessing ensures that only the best part of the image is analyzed. The following step is feature extraction and selection in which distinctive patterns, shapes, and elements that describe an object are focused on while filtering out irrelevant information based on the principles of the requirements of classification precision and accuracy. However, when analyzing the principle of image recognition technology, we need a more sufficient and more

suitable image recognition technology to achieve an optimal image processing. The design of the classifier is done post feature extraction and selection to create a sample image feature library through training and analysis based on certain defined rules. These are rules to minimize the classification error rate, and of course, ensure accurate categorization. Aiming to achieve this, samples for training, representing a variety in the range of potential variations in the game data, were used.

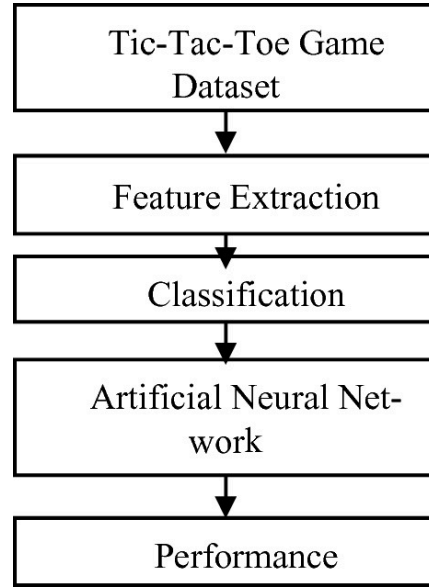


Fig. 3. Architecture of the CNN model for classifying individual cell states in a Tic-Tac-Toe game grid using convolutional layers for feature extraction.

Moreover, the classification performance of the training sample is set to align closely with the characteristics of the images to be identified, ensuring consistency and reliability. Decision Matching is the last step of the image recognition and it leveraged the differences identified during feature integration to classify the extracted features accurately. A suitable classification algorithm analyzes these features quantitatively, and pre-defines the categories within the stored feature database.

The system processes grayscale images resized to a standardized input shape of 32x32x1. The model was trained over 45 epochs with a batch size of 32, ensuring effective learning and generalization for classifying Tic-Tac-Toe symbols.

The Tic-Tac-Toe model assigns numerical values to the individual cell state in order to simplify the analysis and decision-making using one-hot encoding. The values are as follows: 0 for the blank cells, 1 for cells marked with “X”, and 2 for cells marked with “O”, forming the basis for the bot's strategy during gameplay. The classification system (0 for blanks, 1 for “X”, and 2 for “O”) feeds into the decision-making algorithm, ensuring accurate game state evaluations. After complete board

identification, the system analyzes the current state of the game to determine the next best move.

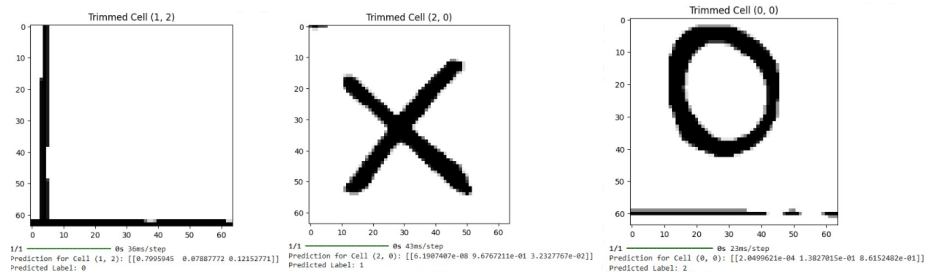


Fig. 4. The classification system (0 for blanks, 1 for “X”, and 2 for “O”), which is mapped using one-hot encoding for implementation.

3.5 Decision-Making Using Minimax Algorithm

The Minimax algorithm aims to properly optimize a game player's decision-making by evaluating all the possible outcomes in a Tic Tac Toe game board and determine the actual best possible move for the AI player. This enhances the gameplay and minimizes potential losses in worst-case scenarios, if any. The terms "maximizer" and "minimizer" depict two players in which one seeks the highest score possible, while the other seeks the least possible score. As a result of this, the name "minimax" comes into place because when one player wins, it automatically means that the other player loses.

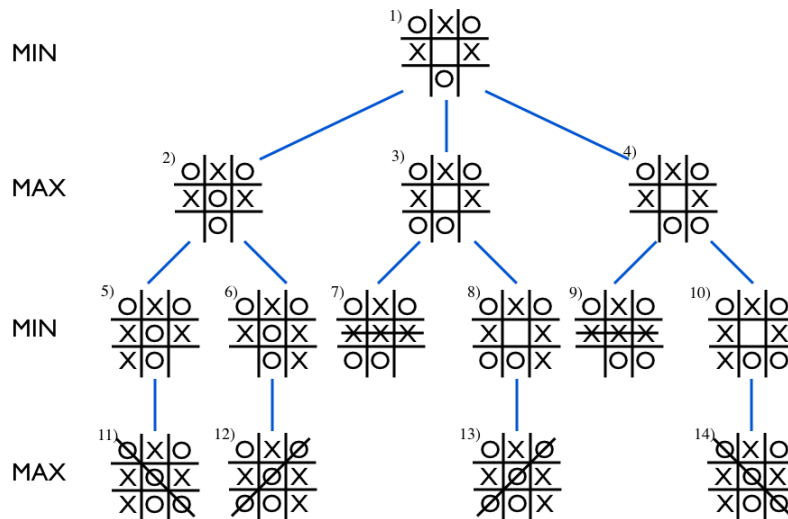


Fig. 5. An illustration of the Minimax algorithm for the TicTacToe gameplay.

The minimax algorithm finds the optimal move for the AI and simulates the other player's move. To do this, a scoring system with numerical values that are computed by NumPy was assigned to each Tic Tac Toe game's outcome.

Table 2. The Minimax Algorithm Scoring System

Cell State	Minimax score	Predicted Move
O	10	O's optimal move
X	-10	Avoid
Tie	0	Neutral

This scoring system guides the bot's decision-making process, for example, a move that will lead to the opponent's win is scored -10 and avoided by the bot, meanwhile, a move that may lead to the bot's win is scored +10 and prioritized by the AI. The algorithm assumes the opponent player is playing the best moves and as a result, it anticipates the next move of the opponents.

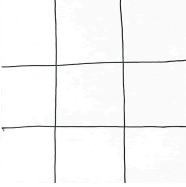
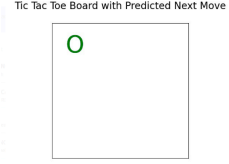
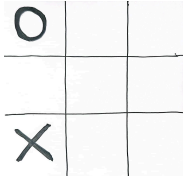
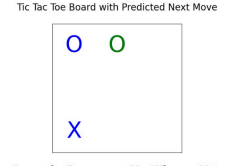
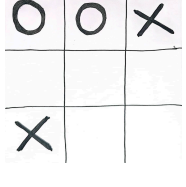

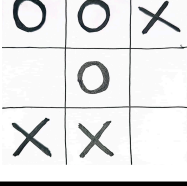
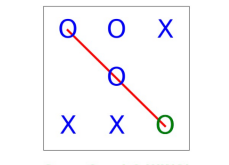
4 Results and Discussion

4.1 Results

In this section we'll discuss the Tic Tac Toe model performance and accuracy in identifying the gameboard and predicting optimal moves during testing. By simulating all possible game states, the AI consistently selected the best moves to maximize its chances of winning or forcing a draw. The training process involved 614 instances that were divided approximately into Training (80%) and Testing (20%) datasets. During the testing phase of implementation, the model achieved an accuracy of 94.3%. Below is an example of a game scenarios of the AI playing as "O" with its suggestions in Green:

In this first scenario, the AI played against a suboptimal opponent and won. This is illustrated in Table 3.

Table 3: Progressive Gameplay Results by the System

Game State/ Turns	INPUT	OUTPUT
1		 <p>Game in Progress. No Winner Yet.</p>
2		 <p>Game in Progress. No Winner Yet.</p>
3		 <p>Game in Progress. No Winner Yet.</p>
4		 <p>Game Over! O WINS!</p>

Different scenarios were used to test the system performance, and it was optimal. In another scenario, the AI demonstrates its defensive ability by predicting the

opponent's optimal next move and selecting a blocking action and as a result the game does not end in a loss, but a tie.

After the board state is identified, the prediction of the best move by the system showed a perfect result. However, in the identification of the board state by model, after splitting into individual cells, based on the model performance evaluation, the accuracy was 94.3%. With a test loss of 25.6% and a training error of 3%.

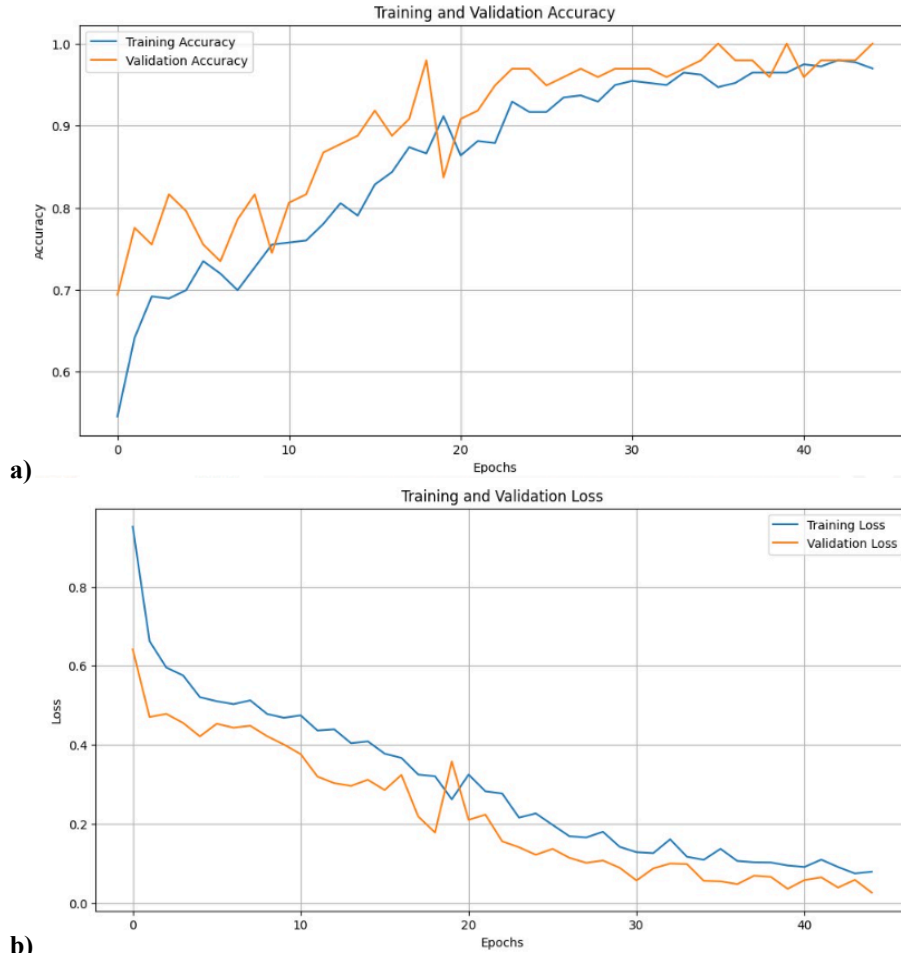


Fig. 6. a) Training and Validation Accuracy. b) Training and Validation Loss.

4.2 Discussion

Based on the analysis and results obtained, the overall performance of the trained CNN showed an accuracy of 94.3% on the test dataset, successfully classifying most images even with variations in lighting and noise. The Minimax algorithm reliably

predicted the optimal move for “O” in all test cases where the board state was accurately classified.

Striking Successes:

The completely trained model gave exceptional performance when assessed with clean images that has clear symbols, and it achieved a near-perfect classification. Also, the minimax algorithm consistently predicted optimal moves in all scenarios, including states involving complex win-block combinations.

Striking Failures:

There are some instances when we observed misclassifications; these happen due to the fact that symbols were poorly drawn or partially obscured by noise. Preprocessing sometimes distorted the “X” symbols or for those having curved strokes, causing them to be misclassified as “O”.

Challenges:

While the implementation of the minimax algorithm for the identified game board was seamless, in the process of image preprocessing, training the model and classification. Some major challenges were encountered. These include:

- Debugging why certain predictions failed
- The time of implementation
- Misclassification of cells data impacting board state prediction
- High Sensitivity to lighting and noisy environment.

5 Conclusion

5.1 Summary

The Tic Tac Toe AI was successfully designed, implemented, and tested with an accuracy of 94.3% achieved, the model was able to recognise and predict from clear images of the game boards. The Tic Tac Toe AI predicted move were in alignment with the expected board states in most cases, showing its ability to classify symbols (“X”, “O”, and blank) effectively and with the integration of the Minimax algorithm, the model makes optimal moves, simulating the needed strategic gameplay. As a result, this project shows the effectiveness and efficiency of Convolutional Neural Network models and algorithmic strategies in building and enhancing intelligent game-playing systems.

5.2 Key Takeaways and Contribution

The project depicts a great demonstration of the successful integration of computer vision and artificial intelligence algorithms to fully classify Tic Tac Toe board states and predict the optimal moves. The use of a CNN for symbol recognition provides high accuracy provided the limited dataset, while the minimax algorithm ensures optimal decision-making. This approach highlights the potential for extending AI-driven solutions to solve real-world problems, especially in applications requiring pattern recognition and strategic analysis.

5.3 Future Works

Future enhancements to this system include an implementation of a live video feed input, used for real-time board state recognition, eliminating the reliance on static image inputs. This could make the system more dynamic and adaptable for practical applications.

Also, the framework can be extended to other board games, incorporating more complex rules and patterns. A considerable improvement in preprocessing techniques to better handle noisy or unclear images and the use of transfer learning could further enhance the robustness of the symbol recognition. Expanding the system to support larger and more intricate boards, such as Connect Four, represents another exciting avenue for development.

6 Bibliography

6.1 References

1. Shi X, Chen Z, Wang H, et al. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting”, 2015:961-997.
2. Xiao H, Rasul K, Vollgraf R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”, 2017:1691-1737.
3. Raccuglia P, Elbert K C, Adler P D F, et al. Machine-learning-assisted materials discovery using failed experiments”, *Nature*, 2016, 533(7601):73.
4. Baydin A G, Pearlmutter B A, Radul A A, et al. Automatic differentiation in machine learning: a survey”, *Computer Science*, 2015(February):451-479.
5. Giusti A, Guzzi J, Dan C C, et al. A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots”, *IEEE Robotics & Automation Letters*, 2017, 1(2):661-667.
6. Couprie M, Bezerra F N, Bertrand G. Topological operators for grayscale image processing”, *Journal of Electronic Imaging*, 2015, 10(10):1003-1015.
7. Mullapudi R T, Vasista V, Bondhugula U. PolyMage:Automatic Optimization for Image Processing Pipelines”, *Acm Sigarch Computer Architecture News*, 2015, 43(1):429-443.
8. Yun J T, Yoon S K, Kim J G, et al. Regression Regression prefetcher with preprocessing for DRAM-PCM Hybrid Main Memory”, *IEEE Computer Architecture Letters*, 2018, 17(2):163-166.
9. Zare F, Ansari S, Najarian K, et al. Preprocessing Sequence Coverage Data for Precise Detection of Copy Number Variations”, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2018, PP(99):1-1.
10. Lei T, Jia X, Zhang Y, et al. Holoscopic 3D Micro-Gesture Recognition Based on Fast Preprocessing and Deep Learning Techniques[C]// 2018:795-801.
11. Moravčík M, Schmid M, Burch N, et al. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker”, *Science*, 2017, 356(6337):508
12. Guerrin C, Aidibi Y, Sanguinet L, Leriche P, Aloise S, Orio M, Delbaere S. When light and acid play Tic-Tac-Toe with a nine-state molecular switch. *Journal of the American Chemical Society*. 2019 Nov 14;141(48):19151-60.
13. Du D.-Z., and Panos M. P., *Minimax and applications*. Vol. 4. Springer Science & Business Media, 1995.
14. Eppes M., “Game Theory — The Minimax Algorithm Explained,”*Towards Data Science*, Aug. 2019, Date Accessed: 2023-03-06. [Online]. Available: <https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1> [Page 10.]
15. Li Longfei. Analysis of Computer Image Recognition Technology Development Status and Prospects [J]. *Digital Communications World*, 2016 (5): 29-29
16. Sun J D, Cui J T, Liu W G, et al. Spatial feature extraction and image retrieval based on entropy [J]. *Systems Engineering & Electronics*, 2006, 28 (6): 791-794.

7 Appendix

7.1 GitHub Repository

This project will be in the following GitHub repository, including the dataset, codes, proposal and report.

<https://github.com/Malachi216/TicTacBot>

7.2 Imported Files

Several files and libraries were imported to enhance the AI's implementation. These include the following Python libraries:

- **TensorFlow and Keras:** For designing, training, and evaluating the CNN model.
- **OpenCV:** For image processing and synthetic data generation.
- **NumPy:** For numerical operations, such as matrix manipulation.
- **Matplotlib:** For visualizing training metrics and board states.

```
import cv2
import numpy as np
from keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Flatten, Dense, Dropout
from keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

Fig. 7. Python libraries imported to support the model development.

7.3 Source Code

Training the CNN

This was done in a Google Colab Environment, and Google Drive was mounted to access the datasets and other relevant input image files.

```
from google.colab import drive
drive.mount('/content/drive') # Mount Google Drive to access datasets
ROOT_DIR = '/content/drive/MyDrive/TicTacToe/data/images' # Update with
your folder path
TRAIN_DIR = os.path.join(ROOT_DIR, 'train')
TEST_DIR = os.path.join(ROOT_DIR, 'test')
input_shape = (32, 32, 1) # Input image size for the model
batch_size = 32
epochs = 45
```

```
def load_img(img_path):
```

```

img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
img = cv2.resize(img, (32, 32)) # Resize to match the input shape
img = np.expand_dims(img, axis=-1)
return img.astype(np.float32)

def one_hot_encode(labels):
    mapper = {'blank': 0, 'cross': 1, 'nought': 2} # Map labels
    encoded = [mapper[label] for label in labels]
    return to_categorical(encoded)

def load_data(root_dir, shuffle=True):
    X, y = [], []
    for root, dirs, files in os.walk(root_dir):
        for class_dir in dirs:
            image_dir = os.path.join(root_dir, class_dir)
            y.extend(np.tile(class_dir, len(os.listdir(image_dir))))
            for img_fn in os.listdir(image_dir):
                img = load_img(os.path.join(image_dir, img_fn))
                X.append(img)
    y = one_hot_encode(y)
    if shuffle:
        data = list(zip(X, y))
        np.random.shuffle(data)
        X, y = zip(*data)
    return np.asarray(X), np.asarray(y)

print('Loading data...')
X_train, y_train = load_data(TRAIN_DIR)
X_test, y_test = load_data(TEST_DIR)
print(f'{len(X_train)} instances for training')
print(f'{len(X_test)} instances for evaluation')

train_val_datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=90,
    shear_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    rescale=1 / 255,
    validation_split=0.2)

train_generator = train_val_datagen.flow(X_train, y_train,
    batch_size=batch_size, subset='training')
val_generator = train_val_datagen.flow(X_test, y_test,
    batch_size=batch_size, subset='validation')

model = Sequential()
model.add(Conv2D(64, (3, 3), input_shape=input_shape, padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3), padding='same'))

```

```

model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.4))

model.add(Dense(3, activation='softmax'))
model.summary()

model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
metrics=['accuracy'])

callbacks = [EarlyStopping(patience=5, verbose=1,
restore_best_weights=True)]

print('Training model...')
history = model.fit(
    train_generator,
    steps_per_epoch=128,
    epochs=epochs,
    validation_data=val_generator,
    validation_steps=32,
    callbacks=callbacks)
print('Evaluating model...')
test_datagen = ImageDataGenerator(rescale=1 / 255)
X_test, y_test = next(test_datagen.flow(X_test, y_test,
batch_size=len(X_test)))

loss, acc = model.evaluate(X_test, y_test, batch_size=batch_size)
print(f'Crossentropy loss: {loss:.3f}')
print(f'Accuracy: {acc:.3f}')
model.save('/content/drive/MyDrive/TicTacToe/TicTacToe_model.h5') # Save
the trained model
print('Model saved to disk.')

import matplotlib.pyplot as plt

# Extracting history details from the training process
history_dict = history.history
# Plot training and validation accuracy
plt.figure(figsize=(12, 6))
plt.plot(history_dict['accuracy'], label='Training Accuracy')
plt.plot(history_dict['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
# Plot training and validation loss
plt.figure(figsize=(12, 6))

```



```

plt.plot(history_dict['loss'], label='Training Loss')
plt.plot(history_dict['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

# Evaluate test performance
test_loss, test_acc = model.evaluate(X_test, y_test,
batch_size=batch_size, verbose=1)
print(f"Test Accuracy: {test_acc:.3f}")
print(f"Test Loss: {test_loss:.3f}")

# Calculate Training Error (final epoch loss)
training_error = 1 - history_dict['accuracy'][-1]
print(f"Training Error: {training_error:.3f}")

# Feedback:
if test_acc > 0.85:
    print("Great performance! The model generalizes well.")
elif test_acc > 0.70:
    print("Good performance, but there's room for improvement.")
else:
    print("The model is underperforming. Consider tuning hyperparameters
or augmenting data.")

```

Implementing Minimax

```

def get_best_move(board):
    """
    Get the best move for 'O' using the minimax algorithm.
    """
    best_score = -np.inf
    move = None
    for m in get_available_moves(board):
        row, col = m
        board[row][col] = 'O'
        score = minimax(board, 0, False)
        board[row][col] = ' '
        if score > best_score:
            best_score = score
            move = m
    return move

def get_available_moves(board):
    """
    Get a list of available moves (empty cells) on the board.
    """
    moves = []
    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                moves.append((i, j))
    return moves

```

```

def check_winner(board):
    """
    Check for a winner in the Tic Tac Toe board.
    Returns 'X', 'O', or 'Tie' if there is a winner or tie, otherwise
    None.
    Also returns the winning line coordinates if applicable.
    """
    # Check rows
    for i in range(3):
        if board[i][0] == board[i][1] == board[i][2] and board[i][0] != ' ':
            return board[i][0], [(i, 0), (i, 1), (i, 2)]
    # Check columns
    for i in range(3):
        if board[0][i] == board[1][i] == board[2][i] and board[0][i] != ' ':
            return board[0][i], [(0, i), (1, i), (2, i)]
    # Check diagonals
    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != ' ':
        return board[0][0], [(0, 0), (1, 1), (2, 2)]
    if board[0][2] == board[1][1] == board[2][0] and board[0][2] != ' ':
        return board[0][2], [(0, 2), (1, 1), (2, 0)]
    # Check for a tie
    if all(board[i][j] != ' ' for i in range(3) for j in range(3)):
        return 'Tie', []
    return None, []

def minimax(board, depth, is_maximizing):
    """
    Minimax algorithm to evaluate the best move for the current player.
    """
    scores = {'X': -10, 'O': 10, 'Tie': 0}

    result, _ = check_winner(board)
    if result:
        return scores[result]

    if is_maximizing:
        best_score = -np.inf
        for move in get_available_moves(board):
            row, col = move
            board[row][col] = 'O'
            score = minimax(board, depth + 1, False)
            board[row][col] = ' '
            best_score = max(score, best_score)
        return best_score
    else:
        best_score = np.inf
        for move in get_available_moves(board):
            row, col = move
            board[row][col] = 'X'
            score = minimax(board, depth + 1, True)
            board[row][col] = ' '
            best_score = min(score, best_score)
        return best_score

```

Visualizing the Board

```
def visualize_board(board):
    for row in board:
        print(' | '.join(row))
    print()

def predict_next_move(board):
    if not any('O' in row for row in board): # Computer plays first
        board[1][1] = 'O'
    else:
        move = get_best_move(board)
        if move:
            board[move[0]][move[1]] = 'O'
    visualize_board(board)
    return board

def preprocess_cell(cell):
    """
    Preprocess a cell image to improve model prediction accuracy.
    - Applies GaussianBlur, thresholding, and resizing.
    """
    # Denoise with GaussianBlur
    cell = cv2.GaussianBlur(cell, (5, 5), 0)

    # Binarize the image (thresholding)
    _, cell = cv2.threshold(cell, 127, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

    # Resize to model input size
    cell = cv2.resize(cell, (64, 64))
    return cell

def initialize_board(img_path):
    """
    Initialize the Tic Tac Toe board from an input image.
    Trims 5% from each side of the cells for better predictions.
    """
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise ValueError(f"Image at {img_path} could not be loaded.")

    # Resize the image for consistency
    img = cv2.resize(img, (300, 300))
    cell_size = img.shape[0] // 3 # Calculate size of each cell
    trim_percent = 0.05 # Percentage to trim from each side
    trim_pixels = int(cell_size * trim_percent) # Convert percentage to
pixels

    board = []

    for row in range(3):
        board_row = []
        for col in range(3):
            # Define the region for each cell
```

```

        x_start, x_end = col * cell_size, (col + 1) * cell_size
        y_start, y_end = row * cell_size, (row + 1) * cell_size

        # Extract and trim the cell
        cell = img[y_start:y_end,
                  x_start:x_end]
        cell = preprocess_cell(cell)
        # Debug: Show each trimmed cell
        plt.imshow(cell, cmap='gray')
        plt.title(f"Trimmed Cell ({row}, {col})")
        plt.show()
        # Preprocess the cell for model prediction
        cell = cv2.resize(cell, (32, 32)) # Resize to model input
size
        cell = np.expand_dims(cell, axis=-1) # Add channel dimension
        cell = cell.astype('float32') / 255.0 # Normalize

        # Predict the class (X, O, or None)
        prediction = model.predict(np.expand_dims(cell, axis=0))
        predicted_label = np.argmax(prediction)
        print(f"Prediction for Cell ({row}, {col}): {prediction}")
        print(f"Predicted Label: {predicted_label}")

        # Map prediction to board symbols
        if predicted_label == 0:
            board_row.append(' ') # Blank
        elif predicted_label == 1:
            board_row.append('X') # Cross
        elif predicted_label == 2:
            board_row.append('O') # Nought

        board.append(board_row)

    return board

test_image_path = '/content/drive/MyDrive/TicTacToe/gameplay14.jpeg'

# Initialize the game board from the image
try:
    board = initialize_board(test_image_path)
    print("Initial Board State:")
    for row in board:
        print(row)
except ValueError as e:
    print(e)

def visualize_board_with_next_move(board):
    """
    Visualize the Tic Tac Toe board with borders, highlight the predicted
next move for 'O',
    and mark the winning row, column, or diagonal in red if applicable.
    """
    # Predict the next move for 'O'
    next_move = get_best_move(board)

```

```

if next_move:
    row, col = next_move
    board[row][col] = 'O'
    # Check for a winner after making the next move
    winner, winning_line = check_winner(board)

    # Create a figure for visualization
    fig, ax = plt.subplots(figsize=(5, 5)) # Adjusted figure size for
compactness
    ax.set_xlim(-0.5, 2.5) # Set limits for the x-axis
    ax.set_ylim(-0.5, 2.5) # Set limits for the y-axis
    ax.set_xticks(np.arange(-0.5, 3, 1), minor=True)
    ax.set_yticks(np.arange(-0.5, 3, 1), minor=True)
    ax.grid(which="minor", color="black", linestyle="-", linewidth=2)
    ax.tick_params(left=False, bottom=False, labelleft=False,
labelbottom=False)

    # Draw the grid and board pieces
    for i in range(3):
        for j in range(3):
            cell_value = board[i][j]
            ax.text(j, 2 - i, cell_value, ha='center', va='center',
fontsize=48, color='blue') # Adjusted font size
            if next_move == (i, j):
                ax.text(j, 2 - i, 'O', ha='center', va='center',
fontsize=48, color='green') # Adjusted font size

    # Mark the winning line if there's a winner
    if winner and winner != 'Tie':
        x_coords = [coord[1] for coord in winning_line]
        y_coords = [2 - coord[0] for coord in winning_line]
        ax.plot(x_coords, y_coords, color='red', linewidth=4)

    # Display the board
    plt.title("Tic Tac Toe Board with Predicted Next Move\n", fontsize=20)
    # Print game status
    status_message = ""
    color = 'black'
    if winner:
        if winner == 'Tie':
            status_message = "Game Over! It's a TIE!"
            color = 'purple'
        else:
            status_message = f"Game Over! {winner} WINS!"
            color = 'green' if winner == 'O' else 'red'
    else:
        status_message = "Game in Progress. No Winner Yet."
        color = 'blue'
    # Display the status message below the board in large font
    plt.text(1, -1, status_message, ha='center', fontsize=24, color=color,
weight='bold')
    plt.show()
visualize_board_with_next_move(board)

```

7.4 Failed Data and Test Cases

These are the examples of images where the CNN image classification failed. The issues were resolved by feeding the model with clear and bright images of clean TicTacToe board sketches.

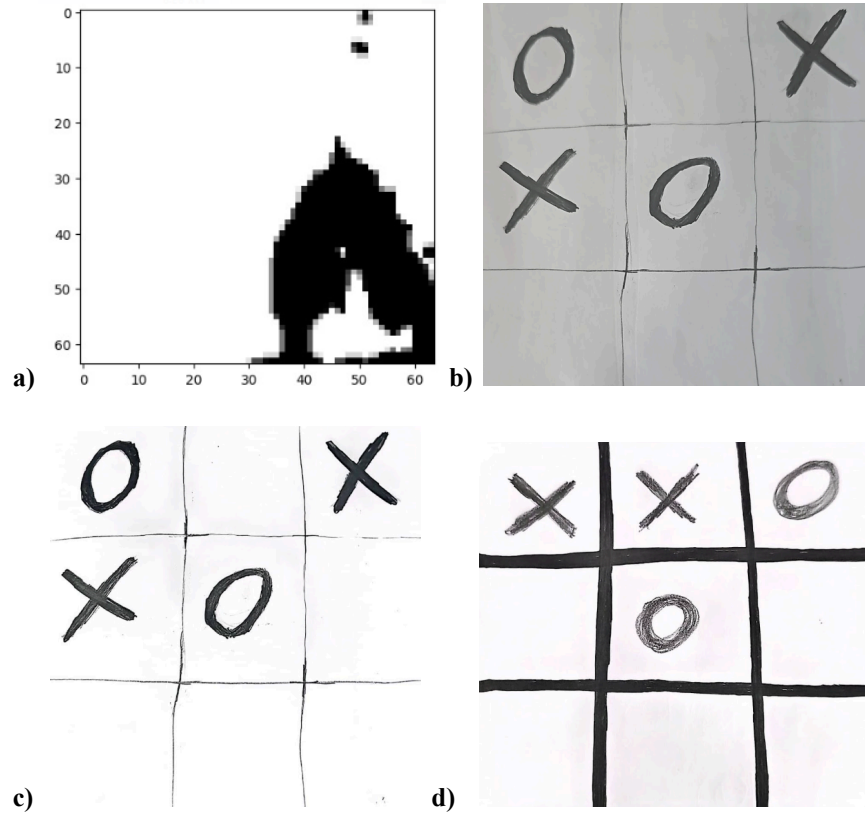


Fig. 8. a) Blank Cells Recognised with shades (Noise). b) Dark Images with poor lighting. c) Too Thin Grids Compared with the cell contents. d) Too Thick Grids Compared with the cell contents.