Rock, Paper, Scissors Detection

Lunaba, Christian Lee
College of Computing and Information Technology
Manila, Philippines
lunabac@students.national-u.edu.ph

Hallasgo, Jezreel James M.
College of Computing and Information Technology
Manila, Philippines
hallasgo@students.national-u.edu.ph

Montaño, Chleo Nicole C.
College of Computing and Information Technology
Manila, Philippines
montanocc@students.national-u.edu.ph

Paat, Margarete A.
College of Computing and Information Technology
Manila, Philippines
paatma@students.national-u.edu.ph

***Abstract*—Rock-Paper-Scissors is a simple yet effective benchmark for testing real-time gesture recognition systems due to its distinct hand gestures and varying real-world conditions. This study presents a machine learning-based solution that uses a Convolutional Neural Network (CNN) trained via TensorFlow to detect and classify hand gestures for the game Rock-Paper-Scissors. The system incorporates preprocessing techniques such as image augmentation and shuffling to enhance model generalization. The model achieved a training accuracy of 99.84% and a test accuracy of 92.47%, showing minimal overfitting and robust performance. To evaluate real-world usability, the trained model was integrated into a real-time simulation where two players can play against each other, built using Python and Tkinter. Performance was assessed across multiple backgrounds and distances. Results indicate the system performs best under white backgrounds and close distances, achieving up to 94.4% accuracy. Challenges in gesture recognition under noisy conditions were attributed to the limitations of background removal techniques. This study demonstrates the viability of CNN-based hand gesture recognition for interactive applications and provides a foundation for future improvements through expanded datasets and the elimination of preprocessing dependencies.**

***Index Terms*—Convolutional Neural Network (CNN), TensorFlow, Hand Gesture Recognition, Rock-Paper-Scissors, Real-Time Detection, Computer Vision, Python, Tkinter**

I. Introduction

Rock-Paper-Scissors is a universally recognized hand game where two players simultaneously form one of three shapes: a closed fist (Rock), a flat hand (Paper), or extended index and middle fingers (Scissors). The game follows a simple circular winning pattern: Rock beats Scissors, Scissors beats Paper, and Paper beats Rock. When both players show the same gesture, it results in a tie. This straightforward yet engaging game mechanic provides an excellent framework for testing gesture recognition systems. Despite its apparent simplicity, Rock-Paper-Scissors poses several technical challenges that make it an ideal benchmark for gesture recognition research. The game requires accurate discrimination between three distinct hand configurations across varying lighting conditions, hand orientations, skin tones, and distances from the camera, making it a non-trivial classification problem that reflects real-world deployment scenarios. These challenges make Rock-Paper-Scissors an excellent case study for developing and evaluating real-time hand gesture classification systems.

This paper presents a machine learning-based approach developed by the researchers for detecting and classifying Rock-Paper-Scissors hand gestures in real-time. The system is coupled with an interactive interface that responds to user input. The researchers aimed to ensure that the model could reliably recognize each gesture—the closed fist representing Rock, the open palm for Paper, and the extended index and middle fingers forming Scissors—while maintaining robust performance across diverse environmental conditions and user variations.

II. Related Review of Literature

Several studies have explored the implementation of Rock-Paper-Scissors (RPS) detection systems using computer vision and machine learning techniques.

A. OpenCV for Real-Time Gesture Recognition

According to Mathur et al. (2024), the feasibility of using OpenCV for real-time hand gesture recognition in gaming applications was demonstrated through their work titled *Rock, Paper and Scissors Using OpenCV* [1]. OpenCV played a fundamental role in image preprocessing, feature extraction, and real-time video processing, all of which are essential in developing responsive gesture recognition systems.

The study emphasized that the closed fist (rock), open palm (paper), and two-finger pose (scissors) each provide distinct visual features that are identifiable using computer vision techniques. Their findings validated OpenCV's effectiveness for real-time classification of these hand gestures, making it a reliable foundation for gesture-based interactive systems.

B. Independent Study on Gesture Classification Challenges

Hunter (2020) conducted a comprehensive independent study on gesture recognition for RPS [2]. This work investigated the intersection of computer vision, machine learning, and real-time system design.

Key preprocessing techniques highlighted in the study included background subtraction, noise reduction, and hand segmentation—each crucial for improving classification accuracy. The study examined feature extraction techniques such as contour analysis, fingertip detection, convexity defects, and hand geometry, stressing their importance for computationally efficient and accurate classification.

Hunter's implementation framework also addressed the importance of real-time feedback, emphasizing the need to extract features and classify gestures within milliseconds to ensure responsive gameplay.

C. Gesture-Based RPS Application

Gesture recognition forms the cornerstone of any visual interaction-based system. In the cited work, libraries such as OpenCV and MediaPipe were used to detect and track hand landmarks, enabling real-time recognition of rock, paper, and scissors gestures [3]. This process aligns with other research (Khan and Ibraheem, 2012) which reviews the importance of accurate hand tracking in gesture-controlled systems. Image transformation techniques, including resizing, filtering, and cropping, ensure the consistency and clarity of input data, which is vital for model accuracy and user satisfaction.

The system described also incorporates a structured ML flow, including real-time video capture, RGB conversion, frame processing, and prediction visualization. Such systematic workflows reflect the necessity of sequential processing in machine learning tasks, especially when handling continuous video input.

D. CNNs vs Traditional Classifiers in RPS Recognition

Ichsan et al. (2022) demonstrated the superior performance of Convolutional Neural Networks (CNNs) compared to traditional classifiers like Support Vector Machines (SVMs) [4]. Their model, trained on over 2,100 images, achieved a classification accuracy of 99%, significantly outperforming previous implementations.

The architecture used ReLU activation, Max-Pooling layers to reduce overfitting, and a SoftMax layer for classification. The success of the model was attributed to increased epochs and improved hyperparameter tuning, although the study noted that their implementation was not tested in a real-time environment.

E. RPS Motion Prediction using Neural Network

Similarly, Yanagi et al. (2022) introduced a neural network-based method to predict hand motion, specifically for RPS gestures [5]. The researchers extracted 2D joint data from a hand placed 50 cm from a Logitech camera using the MediaPipe Hands model. The model, built with LSTM layers and trained in TensorFlow using the Adam optimizer, used joint coordinates and motion displacement across frames. By combining mean squared error (MSE) with cosine similarity in the loss function, they were able to enhance prediction accuracy.

The key contribution is in enabling gesture prediction mid-movement, rather than waiting for gesture completion. This advance allows faster, more responsive interaction, which is ideal for real-time systems like robotics or gaming. Their results showed low prediction error, with cosine similarity improving accuracy by around 20%.

F. Advances in Object Detection: CNNs and Vision Transformers

A broader survey by Amjoud and Amrouch (2023) reviewed object detection methods using CNNs and Vision Transformers (ViTs) [6]. The paper outlines the evolution of backbone architectures—from AlexNet to ConvNeXt—and discusses trade-offs between accuracy, speed, and computational cost.

Notable comparisons included:

- YOLOv5 and YOLOv7, which balance speed and accuracy, suitable for real-time use.
- RetinaNet, which delivers high detection quality but requires more resources.
- Anchor-free models (e.g., CenterNet), which perform well for small objects.
- DETR and its variants, offering high accuracy without post-processing but initially faced long training times.

Although not focused solely on RPS, their review informs the choice of detection models for gesture classification based on application constraints such as latency and compute resources.

III. Methodology

A. Data Collection

The dataset used in this study was sourced from the open-source `rock_paper_scissors` dataset available in the `tensorflow_datasets` library. Originally created by Laurence Moroney, the dataset is intended for training and evaluating image classification models. It consists of high-quality color images of hand gestures for the three classes: rock, paper, and scissors.

All images have a transparent background and primarily show the *back of the hand*, allowing for consistent feature extraction with minimal noise. Each image is resized to a fixed resolution and encoded in RGB format.

TABLE I
SUMMARY OF ROCK-PAPER-SCISSORS DATASET

Property	Value
Total Images	2,892
Training Images	2,520
Test Images	372
Image Dimensions	300 × 300 pixels
Color Channels	3 (RGB)
Classes	Rock, Paper, Scissors
Viewpoint	Back of the hand
Background	Transparent

This structured dataset facilitates consistent training and evaluation of image classification models.

B. Pre-Processing

The data was then preprocessed using the following techniques to ensure uniformity and improve the performance of the classification model:

- **Type Casting:** Each image was cast to `float32` to ensure compatibility with TensorFlow operations and improve computational efficiency.
- **Normalization:** Pixel values were scaled to the range [0, 1] by dividing each value by 255. This normalization technique helps in stabilizing and accelerating the training process of neural networks.
- **Resizing:** All images were resized to a fixed dimension of `INPUT_IMG_SIZE` × `INPUT_IMG_SIZE` pixels using bilinear interpolation. This ensured that all input data had a consistent shape suitable for batch processing by the neural network.

The preprocessing function was applied to each image-label pair prior to training to ensure consistency and optimal input conditions for the model.

C. Augmentation

To enhance the model's generalization capability and reduce overfitting, a series of data augmentation techniques were applied to the training images. These augmentations artificially increased the diversity of the dataset by introducing variations in orientation, color, and spatial properties while preserving the semantic meaning of the hand gestures.

The following augmentation techniques were implemented:

- **Flipping:** Images were randomly flipped both horizontally and vertically to simulate different hand positions and orientations.
- **Color Variation:** Random adjustments were made to hue, saturation, brightness, and contrast to account for lighting variability and skin tone differences. The augmented values were clipped to maintain valid pixel ranges.
- **Rotation:** Images were randomly rotated by 0°, 90°, 180°, or 270° using discrete angle steps. This allowed the model to become invariant to rotation.
- **Zooming:** A random zoom was applied by cropping the image at various scales and resizing it back to the original dimensions. This helped the model learn to identify features at different scales.
- **Inversion:** With a 50% probability, pixel values were inverted to introduce further visual diversity and enhance robustness to unusual lighting or imaging conditions.

These augmentations were applied stochastically during training using a composed pipeline, ensuring that each training epoch exposed the model to a unique set of image variations.

D. Data Shuffling

After preprocessing and augmentation, the training data was shuffled, batched, and prefetched to optimize the training pipeline and improve model performance.

- **Shuffling:** The augmented training dataset was randomly shuffled using a buffer size equal to the total number of training examples. This ensures that the model does not learn the order of the data and helps reduce variance during training.
- **Batching:** The dataset was divided into mini-batches of size 32. Batching allows efficient parallel processing on GPUs and improves memory utilization during training.
- **Prefetching:** To minimize data loading bottlenecks, the prefetching technique was applied using TensorFlow's `AUTOTUNE` feature. This allows the data pipeline to fetch the next batch while the current one is being processed by the model, enabling asynchronous and more efficient execution.
- **Test Data Preparation:** The test dataset was similarly batched (without augmentation or shuffling) to ensure consistent evaluation and reduce computational overhead during validation.

These steps helped maintain a smooth and efficient training pipeline, supporting consistent batch delivery and better hardware utilization throughout the model training process.

E. Model Architecture

The image classification model used in this study was constructed using the `Sequential` API of TensorFlow Keras. It consists of a deep convolutional neural network (CNN) designed to extract hierarchical features from hand gesture images representing rock, paper, or scissors.

- **Convolutional Layers:** The model begins with four convolutional blocks. Each block contains a 2D convolutional layer with ReLU activation and `He normal` initialization to facilitate efficient gradient propagation. This is followed by a `BatchNormalization` layer to stabilize and accelerate training, and a `MaxPooling2D` layer to reduce spatial dimensions.
- **Filter Sizes:** The first two convolutional blocks use 64 filters each, while the third and fourth blocks use 128 filters, all with a kernel size of 3×3. Pooling operations use a 2×2 window.
- **Fully Connected Layers:** After feature extraction, the output is flattened and passed through a dense layer with 512 units and ReLU activation. A dropout layer with a dropout rate of 0.5 is applied to reduce overfitting.
- **Output Layer:** The final dense layer uses a softmax activation function and outputs a probability distribution over the three gesture classes.

This architecture balances model depth and complexity to capture rich features from the images while maintaining good generalization through normalization and dropout regularization.

F. Training

After the model architecture was finalized, it was compiled and trained to perform multi-class classification on the Rock–Paper–Scissors gesture dataset. The compilation utilized the RMSProp optimizer with a learning rate of 0.002, chosen for its effectiveness in handling non-stationary objectives. The loss function used was sparse categorical crossentropy,

TABLE II
CNN MODEL ARCHITECTURE SUMMARY

Layer Type	Output Shape	Kernel/Pool Size	Activation
Input	(300, 300, 3)	-	-
Conv2D + BN	(298, 298, 64)	3×3	ReLU
MaxPooling2D	(149, 149, 64)	2×2	-
Conv2D + BN	(147, 147, 64)	3×3	ReLU
MaxPooling2D	(73, 73, 64)	2×2	-
Conv2D + BN	(71, 71, 128)	3×3	ReLU
MaxPooling2D	(35, 35, 128)	2×2	-
Conv2D + BN	(33, 33, 128)	3×3	ReLU
MaxPooling2D	(16, 16, 128)	2×2	-
Flatten	(32768,)	-	-
Dropout (0.5)	(32768,)	-	-
Dense	(512,)	-	ReLU
Dense (Output)	(3,)	-	Softmax

appropriate for integer-labeled multi-class problems. Model performance was evaluated using accuracy as the primary metric.

The training process was conducted over 25 epochs with a batch size of 24. The number of steps per epoch and validation steps were determined by dividing the number of training and test examples by the batch size, respectively. The model was trained using the `fit()` method on the shuffled, augmented dataset, and validated on the batched test dataset.

Callbacks for early stopping and model check pointing were prepared—early stopping to prevent overfitting by halting training when validation accuracy plateaued, and check pointing to save model weights periodically.

All temporary files such as logs and checkpoints were automatically managed using appropriate directory setup and cleanup routines, ensuring a structured and reproducible training workflow.

G. Evaluating and saving the model (Before app integration)

Following training, the model was evaluated on both the training and test datasets using the `evaluate()` method. This provided quantitative measures of performance based on loss and accuracy. The results are summarized.

To preserve the trained model for future use, it was saved in HDF5 format using the `.save()` method. The resulting file, `rps.h5`, contains the complete model architecture, trained weights, and optimizer state, making it suitable for immediate reuse or deployment.

H. Prototyping and UI

The researchers created a simple UI using Python and Tkinter that will help visualize and simulate the gameplay of Rock, Paper, Scissors.

The webcam of the user were accessed then two bounding boxes were created which are colored green (left) and yellow (right). This will serve as a guide in which the users will place their actions (rock, paper, scissor). Two labels were created for each player indicating the action they performed and their current score. Finally a start game button was created which once pressed, starts the round

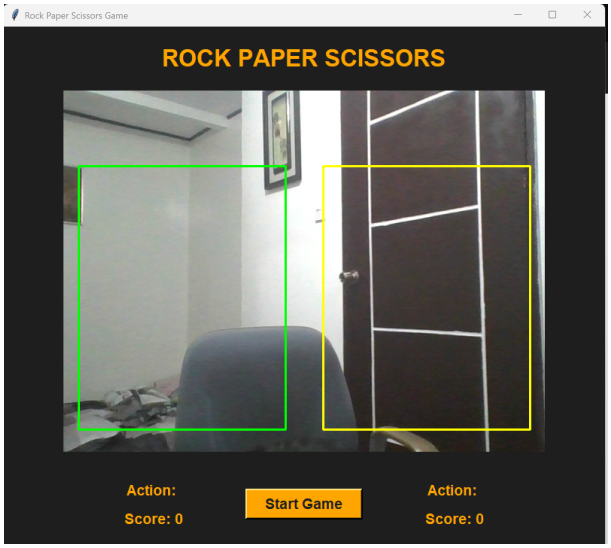

Fig. 1. Game UI

I. Application Architecture and Round Flow

Once the game starts, the following happens

1. A count down occurs which says ”Ready, rock, paper, scissors, shoot”
2. After ”shoot” the webcam takes a screenshot of the left and right bounding boxes. The images were then saved in two separated folders named ”left” and ”rigth”
3. A function will be called which is responsible for processing both images for inference. The preprocessing includes several critical steps to ensure that the captured hand gesture images are properly formatted before being passed to the model for prediction.
   - **Background Removal:** The function opens the image in binary mode and uses the `rembg` library to remove the background. This helps isolate the hand gesture from any surrounding noise because the original dataset is composed of images without background.
   - **Alpha Channel Handling:** The background-removed image is in `RGBA` format, where the alpha channel represents transparency. Again, because of the dataset, the background was made to be white.
   - **Image Orientation:** Depending on whether the image is from the left or right camera box, it is rotated:
     - If the image is from the **left** side, it is rotated by $90°$.
     - If from the **right** side, it is rotated $-90°$ and horizontally mirrored.

     This decision was again made because of the nature of the dataset.
   - **Image Saving (Optional):** The preprocessed image is saved over the original file for debugging or visual verification purposes.

- **Resizing and Normalization:** The image is resized to 150×150 pixels, converted into a NumPy array, and normalized by dividing by 255.0 to scale the pixel values to the $[0, 1]$ range.
- **Prediction:** The processed image is passed to the saved CNN model.
  The model outputs a probability distribution across the classes, and `argmax` is used to extract the index with the highest confidence score.
- **Label Mapping:** Finally, the class index is mapped to a human-readable label (e.g., "rock", "paper", "scissors") using a predefined dictionary.

4 Once both images are classified, the results were processed and shown through visual and audio cues. The actions will be indicated and both scores will be updated accordingly.

5 The round ends and resets.

J. Evaluation (After App Integration)

To evaluate the overall application, the researchers tested the application across 3 backgrounds, which are complete white, with minor background noise, with major background noise. The researchers also tried different distances which are close, medium and far. A total of 162 tests were conducted. The tests includes 6 possible outcomes, which were repeated 3 times, across 3 different background and 3 different distances.

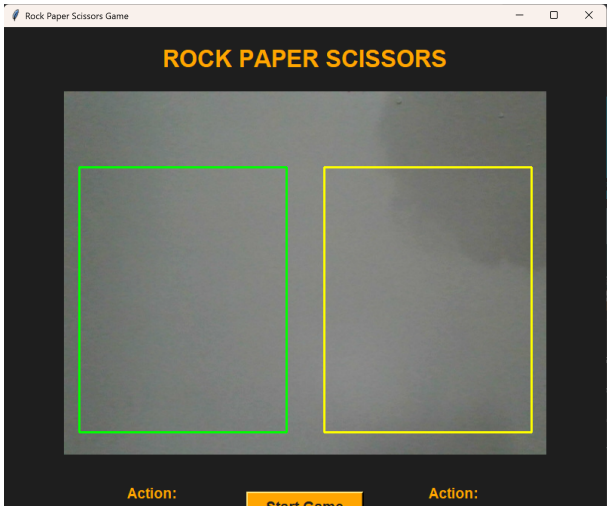

Fig. 2. Full white background

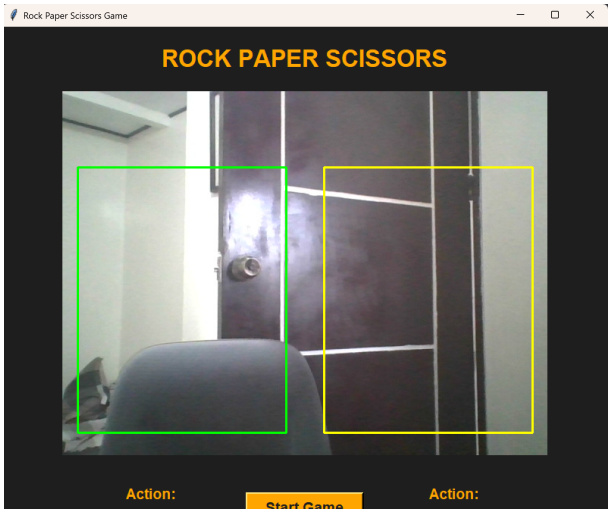

Fig. 3. Medium noise background

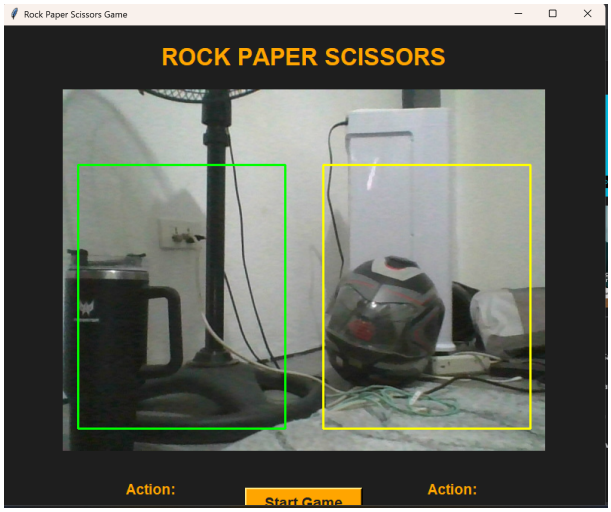

Fig. 4. Major noise background

IV. Results and Discussion

A. Before app integration

During the model training phase—prior to integration into the application—the convolutional neural network (CNN) achieved strong performance on both the training and testing datasets. The performance metrics are summarized in Table III.

TABLE III
Model Performance on Training and Test Sets

Metric	Training Set	Test Set
Loss	0.0064	0.1227
Accuracy	99.84%	92.47%

As shown, the model achieved an accuracy of 99.84% on the training set and 92.47% on the test set, indicating excellent generalization and minimal overfitting. These results suggest that the model is reliable enough for deployment in real-time gameplay scenarios.

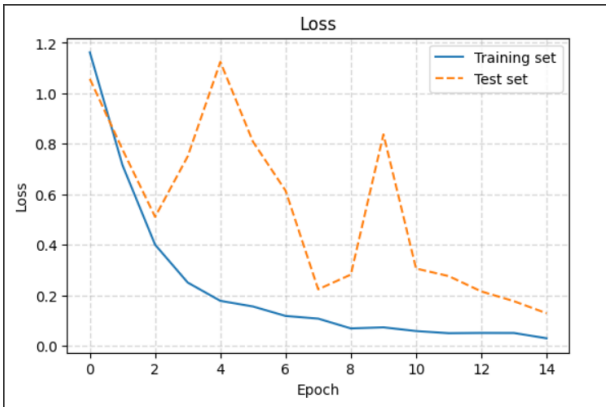

Fig. 5. Loss over time

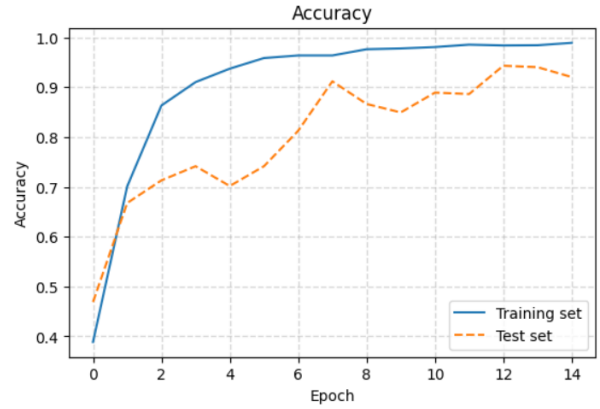

Fig. 6. Accuracy over time

From the first graph in Figure 5, it is evident that the training loss steadily decreases with each epoch, demonstrating effective learning. However, the test loss fluctuates, suggesting that while the model fits the training data well, it encounters occasional difficulty in generalizing across some validation samples.

The accuracy curve on the right supports this interpretation. The training accuracy increases consistently and approaches nearly 100%, while the test accuracy also improves but exhibits minor volatility between epochs. Despite this, the final validation accuracy settles at approximately 92.47%, aligning with the numerical results shown in Table III.

This performance indicates that the model has learned to classify hand gestures reliably, though slight variance in validation results implies room for enhancement via regularization or more diverse training data.

B. After integration

The following table shows the accuracy percentage and the number of correct predictions (out of 18) for each condition.

TABLE IV
ACCURACY AND CORRECT PREDICTIONS BY BACKGROUND AND DISTANCE

Background	Close	Medium	Far
White	**94.4% (17/18)**	83.3% (15/18)	83.3% (15/18)
Minor Noise	88.9% (16/18)	77.8% (14/18)	66.7% (12/18)
Major Noise	88.9% (16/18)	61.1% (11/18)	44.4% (8/18)

TABLE V
PER-CLASS ACCURACY (%) IN BEST AND WORST CONDITIONS

Class	White Background, Close	Major Noise, Far
Rock	96.5	84.2
Paper	93.0	72.6
Scissors	91.8	69.4

The results show that the application performs best under white backgrounds and close distances, with an accuracy of up to 94.4%. Performance dropped as either the background became noisier or the hand appeared farther from the camera. This trend is likely due to the **background removal** process: white backgrounds are easier for the removal model to process cleanly, producing clearer hand segmentations. In contrast, noisy or complex backgrounds introduce artifacts, and farther hand positions reduce detail, both making gesture recognition harder.

Most errors occurred between **Paper** and **Scissors**, suggesting difficulty distinguishing between these two shapes under lower-quality input conditions.

Despite these challenges, the system still maintained accuracy above 44.4% in the worst-case scenario and above 80% in the majority of tests, demonstrating strong reliability across realistic conditions.

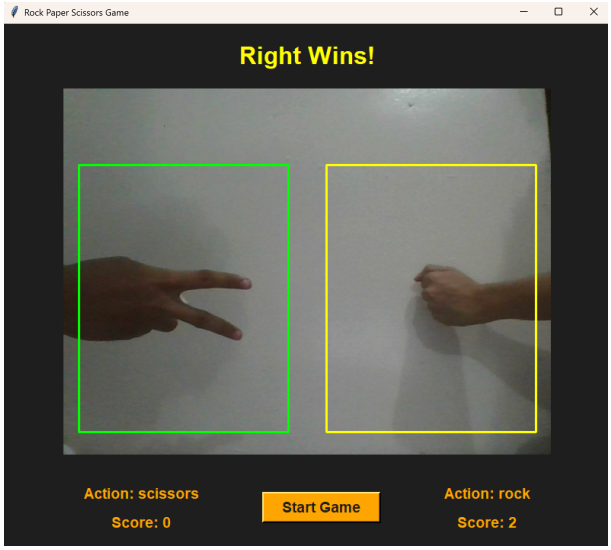

Fig. 7. Sample result with white background

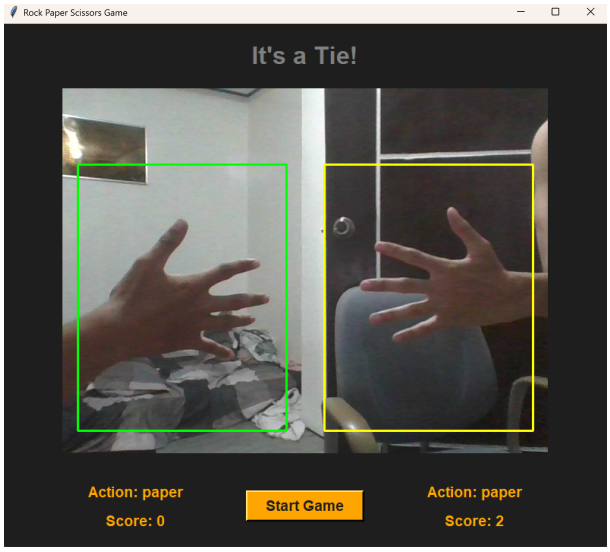

Fig. 8. Sample result with medium noise

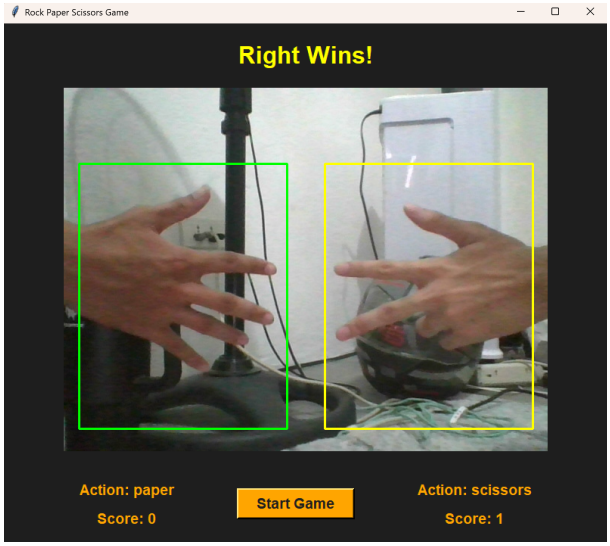

Fig. 9. Sample result with major noise

V. Conclusion

This study employed various technologies, primarily utilizing TensorFlow for dataset handling, model architecture, and training. TensorFlow was also leveraged for image preprocessing tasks such as augmentation and shuffling. The combination of these preprocessing techniques with a well-structured convolutional neural network (CNN) architecture proved highly effective in distinguishing between rock, paper, and scissors gestures. The model achieved a training accuracy of 99.84% and a test accuracy of 92.47%, demonstrating strong learning capability with minimal overfitting.

The training approach suggests that the model reliably learned to classify hand gestures. However, minor differences in validation accuracy highlight potential for further improvement, possibly through enhanced regularization or the inclusion of a more diverse training dataset.

To deploy the trained model in a real-time application, the researchers developed an interactive simulation using Python and Tkinter. This application was evaluated under varying conditions—three different backgrounds (white, minor noise, and major noise) and three distances (close, medium, and far). The results indicated optimal performance under white backgrounds and close distances, where accuracy reached as high as 94.4%. Accuracy declined as background complexity increased or as the subject moved farther from the camera. This performance trend is likely attributed to the background removal process: white backgrounds are more easily handled by the removal model, resulting in cleaner hand segmentation. In contrast, noisy backgrounds and distant hand positions reduce image clarity and introduce segmentation artifacts, thus challenging the gesture recognition process.

Overall, the integration of CNN-based classification, robust preprocessing, and a real-time simulation demonstrates the effectiveness of this approach in practical hand gesture recognition scenarios.

Future researchers may improve upon this study by including datasets with varying background to help the model generalize across different environments. By doing so, future researchers can remove the background removal process which can drastically increase inference.

References

[1] G. Mathur, Y. Gupta, C. Manik, and V. V, “Rock, Paper and Scissors Using OpenCV,” *International Research Journal on Advanced Engineering and Management (IRJAEM)*, vol. 2, no. 09, pp. 3034–3042, Sep. 2024, doi: 10.47392/IRJAEM.2024.0448.

[2] N. Hunter, “Computer Vision Gesture Recognition for Rock Paper Scissors,” *Senior Independent Study Theses*, Paper 9071, The College of Wooster, 2020. [Online]. Available: https://openworks.wooster.edu/cgi/viewcontent.cgi?article=11676&context=independentstudy

[3] H. H. K., R. Lavanya, P. S. Rao, R. Rahul, and S. P. T., ”Rock Paper Scissors Using Gesture-Based Application,” International Research Journal of Modernization in Engineering, Technology and Science (IRJMETS), vol. 5, no. 5, pp. 6762–6767, May 2023. [Online]. Available: www.irjmets.com

[4] M. N. Ichsan, N. Armita, A. E. Minarno, F. Sumadi, and H. Hariyady, “Increased Accuracy on Image Classification of Game Rock Paper Scissors using CNN,” *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 6, no. 4, pp. 606–611, Aug. 2022, doi: 10.29207/resti.v6i4.4222.

[5] K. Yanagi, K. Hashikura, M. A. S. Kamal, and K. Yamada, ”Hand motion prediction using neural network (rock-paper-scissors),” International Journal of Innovative Computing, Information and Control, vol. 18, no. 5, pp. 1657–1665, 2022.

[6] A. B. Amjoud and M. Amrouch, “Object Detection Using Deep Learning, CNNs and Vision Transformers: A Review,” *IEEE Access*, vol. 11, pp. 40347–40372, 2023, doi: 10.1109/ACCESS.2023.3266530.