AI-Powered Card Game System: Combining Computer Vision Recognition with Counting for Strategic Gameplay

1. Introduction

1.1 Problem Statement

Our project deals with several problems faced in card game automation.

1. Real-time Card Recognition: Developing a computer vision system that can accurately recognize playing cards from live video streams with minimal overhead.
2. Multi-Card Tracking and Processing: Creating a foolproof system capable of simultaneously tracking and identifying multiple cards under varying conditions, including different lighting, angles, and partial occlusions.
3. Card Information Extraction: Accurately identifying both the suit and value of cards while handling different orientations and positions on the playing surface.
4. Real-time decision-making system: Building an AI system that can process the recognized card information in real time to provide strategic gameplay recommendations.

1.2 Motivation and why it matters

An AI-powered card game system has significant implications across multiple domains. Firstly, traditional card game learning relies heavily on human instruction, which can be inconsistent and time-consuming. Such a system provides standardized, real-time feedback that accelerates learning and improves skill development. The system's potential also extends to research and game theory applications. Collecting and analyzing gameplay data can contribute to understanding optimal strategies, player behavior patterns, and decision-making processes in blackjack and various card games. This data-driven approach enables the development of more sophisticated training programs.

2. Methodology

2.1 Overview

For our project, we would be doing a comparative analysis between YOLO and Faster R-CNN models for playing card detection. The implementation involved data preprocessing, model training, and performance evaluation to determine the most effective approach for real-time card recognition.

2.2 Dataset Description

We utilized a comprehensive dataset from Kaggle named '*Playing Cards Object Detection Dataset*,' which contained synthetically generated images of playing cards with bounding boxes.

In our code submitted, we did not include the data folder since it is 12GB and too big to upload.. The full code (including the dataset) can be viewed in the GitHub repository.

2.3 Data Preprocessing

Our data augmentation pipeline implemented key transformations to enhance model robustness. We applied rotations at 90-degree increments (90°, 180°, 270°), brightness adjustments ranging from 0.3 (darker) to 1.7 (brighter), and contrast variations from 0.5 (lower) to 2.0 (higher). Additionally, we introduced grayscale conversion and Gaussian blur transformations through the *convert_to_grayscale* and *apply_gaussian_blur* functions, respectively. These preprocessing steps enhanced our dataset diversity and improved model performance across varying real-world conditions.

The dataset's bounding box annotations were originally formatted for YOLOv8, which we converted to R-CNN format as a basis for comparison of the performance of the models. These preprocessing steps enhanced our dataset diversity,

2.4 Training the YOLO

2.4.1 Dataset Preparation

The training dataset consists of playing card images annotated with 52 distinct classes representing each card in a standard deck (2-10, J, Q, K, A across all suits). Images were organized in the YOLO format with corresponding label files containing normalized bounding box coordinates and class indices. The dataset was split into training, validation, and test sets with an 80-10-10 ratio. As mentioned in the data preprocessing section above, data augmentation techniques were applied.

2.4.2 Model Configuration

Base Architecture: YOLOv8s pre-trained on COCO dataset
Input Resolution: 640x640 pixels
Batch Size: 16
Number of Epochs: 150
Device: Apple M1 Max with Metal Performance Shaders (MPS)

2.4.3 Training Process

The training process began with model initialization using pre-trained weights to leverage transfer learning benefits. Throughout training, the model's state was preserved through periodic checkpoints at every epoch, with the best-performing weights retained based on validation metrics. Key performance indicators were continuously monitored, including mean Average

Precision (mAP), precision and recall values, classification accuracy, and individual loss components encompassing bounding box regression, class prediction, and objectness scores.

2.5 Training the Faster R-CNN

2.4.1 Dataset Preparation

The training dataset consists of playing card images annotated with 52 distinct classes representing each card in a standard deck (2-10, J, Q, K, A across all suits)
Images were organized in the YOLO format with corresponding label files containing normalized bounding box coordinates and class indices. The dataset was split into training, validation, and test sets with an 80-10-10 ratio. We then write a program to convert YOLO-style to COCO-style annotations to train our Faster R-CNN model.

2.4.2 Model Configuration

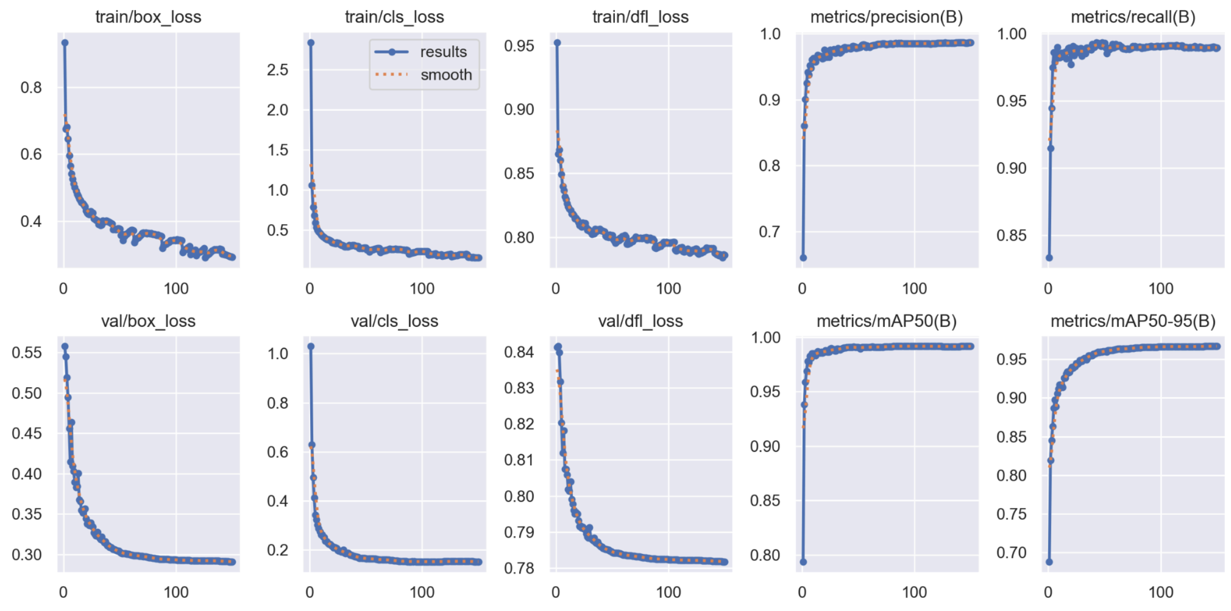
Base Architecture: Faster R-CNN with MobileNet V3 Large backbone and Feature Pyramid Network (FPN), pre-trained on the COCO dataset.
Input Resolution: 640x640 pixels
Batch Size: 16
Number of Epochs: 200
Device: NVIDIA RTX 4070 Laptop GPU (CUDA-enabled)

2.4.3 Training Process

The training process was initialized using pre-trained weights to leverage the benefits of transfer learning. Since MobileNet does not have a built-in method for data augmentation and performing data augmentation requires rewriting the entire JSON file at each epoch, significantly increasing training time, we decided to freeze the earlier layers and fine-tune the later layers to avoid overfitting. Additionally, we reduced the learning rate as the number of epochs increased to reduce overfitting further. During training, the model was saved at every epoch, and the best-performing model (based on validation results) was retained. Furthermore, the training loss and validation accuracies were recorded in a text file, plotted on graphs, and saved as an image for visualization.

3. Results

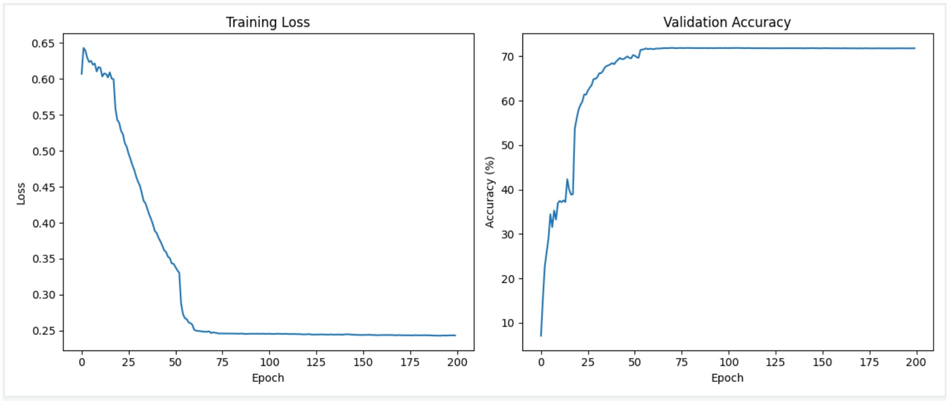
3.1 Yolo Performance



The results indicate:

- Both training and validation losses are decreasing and stabilizing, suggesting good convergence
- Very high precision and recall (~0.98) indicating the model is both accurate and comprehensive in its detection
- Excellent mAP scores (>0.95), suggesting robust performance across different IoU thresholds
- No significant overfitting as validation metrics follow training metrics closely

3.2 RCNN Performance



The results indicate:

- Training Loss: The graph shows a steady decline in loss, indicating consistent convergence
- Validation Accuracy: The validation accuracy improves rapidly in the initial epochs and becomes stable after around the 60th epoch, suggesting the model has learned meaningful representations

3.3 Comparison of the 2 models

3.3.1 Performance Metrics:

- **Accuracy and Precision:** YOLO achieved a higher precision and recall (~0.98) and mAP (>0.95), outperforming Faster R-CNN, which achieved a validation accuracy of ~70%. This suggests YOLO is more accurate in detecting and classifying cards in this dataset.
- **Loss Trends:** YOLO's loss trends demonstrated faster convergence and stability compared to Faster R-CNN, which required more epochs to stabilize.

3.3.2 Training Time and Efficiency:

- **Training Time:** YOLO's training process was faster due to its efficient implementation, whereas Faster R-CNN required more computational time due to its two-stage detection pipeline.
- **Resource Usage:** Faster R-CNN, with its MobileNet backbone, utilized GPU resources efficiently but was slower than YOLO, which is inherently designed for real-time performance.

3.3.3 Suitability for the Task:

- **Real-Time Requirements:** YOLO, being optimized for real-time detection, is more suitable for applications requiring immediate card recognition, such as live gaming or monitoring systems.
- **Accuracy Needs:** Faster R-CNN could still be a viable option for tasks prioritizing accuracy over speed, though it may need further fine-tuning and applying data augmentation to reach YOLO's level of performance.

3.3.4 Flexibility and Implementation:

- **Data Augmentation:** YOLO has built-in support for data augmentation, which simplifies the training process. On the other hand, Faster R-CNN required custom handling of data augmentation, making its setup more complex and taking much longer.
- **Transfer Learning:** Both models leveraged pre-trained weights effectively, but YOLO's plug-and-play nature made it easier to adapt to poker card detection.

3.3.5 Overall Comparison:

- YOLO excelled in terms of speed, precision, and ease of use, making it the better choice for this particular application.
- Faster R-CNN, while slightly slower and less precise, demonstrated strong generalization capabilities and could potentially match YOLO's performance with further optimization.

4. Discussion and Extensions

Our project demonstrated the superior performance of YOLO over Faster R-CNN for playing card detection, achieving higher accuracy and faster processing speeds. The implementation of data augmentation techniques enhanced model robustness across various lighting conditions and card orientations. While both models showed promising results, YOLO's ability to process images in real-time while maintaining high precision makes it particularly suitable for live card game applications needed for games like Blackjack. One limitation we encountered was the use of synthetic data, which might not fully represent real-world playing conditions with factors like glare, shadows occurring in a real-life casino scenario.

When comparing our results with Akkar et al. (2023), both studies utilized similar detection approaches - template matching and bounding box detection. Their implementation achieved 80% accuracy for suit detection and over 50% for number recognition, which aligns with our baseline results, though our augmented dataset and YOLO implementation showed improved performance at around 98% precision and recall.

For future extensions, we could focus on expanding the system's capability to detect and track multiple decks simultaneously, which would enable support for games requiring multiple decks such as casino blackjack.

5. Contributions

S Devi Harshitha:

- Created image transformations for data augmentation
- Created counting algorithm for recommendations
- Writing of all sections of report
- Voiceover for final video

Lim Jian Hong:

- Trained 3 YOLO models
- Created evaluation function for faster R-CNN
- Writing of YOLO results on report
- Created and managed GitHub repository for project
- Edited final video

Joshua Sun:

- Responsible for faster R-CNN
- Trained Faster R-CNN model
- Created card detection program for YOLO

- Writing of Faster-RNN results and model comparisons of report

6. References

Akkar, A., Cregan, S., Cassens, J., Vander-Pallen, M. A., & Mohd, T. K. (2023). Playing blackjack using Computer Vision. Artificial Intelligence and Applications. https://doi.org/10.47852/bonviewaia3202962

Andy8744. (2022, February 9). *Playing cards object detection dataset*. Kaggle. https://www.kaggle.com/datasets/andy8744/playing-cards-object-detection-dataset