CARDS CLASSIFICATION

Furkan Arslan 202028013 Ayhan Tan Açar 202028017

1. Introduction

This project investigates the classification of playing cards using Convolutional Neural Networks (CNNs), with an emphasis on evaluating the performance of both simple and advanced deep learning architectures. The choice of this topic was inspired by the mechanics of the card-based game *Balatro*, which includes both standard playing cards and unique joker variants. Beyond its entertainment relevance, this task mirrors broader challenges in computer vision such as class similarity, intra-class variation, and limited data generalization.

We initially selected a basic CNN architecture due to its straightforward implementation and effectiveness for moderately complex image classification tasks. This model served as a foundational step, allowing us to explore and benchmark its capabilities before transitioning to more sophisticated models like ResNet or EfficientNet through transfer learning.

Our work follows a supervised learning paradigm. The dataset consists of 7624 training images, 265 validation images, and 265 test images—organized into 53 subdirectories, each corresponding to a different card class. This structure allows for systematic training, validation, and performance analysis. Throughout the project, we monitored key metrics such as accuracy and loss, while also applying regularization techniques to mitigate overfitting and improve generalization.

We chose this problem because it is both challenging and highly illustrative of real-world image recognition tasks. Although it may appear narrow in scope, card classification has practical applications in gaming AI, augmented reality systems, automated casino surveillance, and even object recognition pipelines. Moreover, it provides a controlled environment to test complex neural networks under realistic constraints.

Specifically, our objectives in this project include:

- Comparing the effectiveness of a simple CNN with a transfer learning-based architecture.
- Understanding how architectural complexity and data augmentation influence classification accuracy.
- Investigating common sources of misclassification and model limitations.

So far, we have successfully completed the following:

- Retrieved the dataset from Kaggle using an API.
- Loaded and verified the dataset structure in Google Colab.

- Analyzed and organized the dataset.
- Designed and trained a baseline CNN model.
- Evaluated the initial model's performance on validation and test sets.

Future efforts are directed toward improving performance with advanced models and fine-tuning, while also addressing overfitting through further regularization and augmentation strategies.

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
efficientnetb7 (Functional)	(None, 7, 7, 2560)	64,097,687
flatten (Flatten)	(None, 125440)	0
dense (Dense)	(None, 128)	16,056,448
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 53)	3,445

Total params: 80,165,836 (305.81 MB)
Trainable params: 79,610,041 (303.69 MB)
Non-trainable params: 555,795 (2.12 MB)

Figure 1: *Card classification model using EfficientNetB7 and custom dense layers.*

2. Architecture

After establishing a performance baseline with a basic Convolutional Neural Network (CNN), we transitioned to a more advanced and high-performing architecture that leverages **EfficientNetB7**, a state-of-the-art convolutional neural network known for its optimal balance between model size and classification accuracy. Our choice was motivated by its superior performance in large-scale image recognition tasks and its compound scaling methodology, which simultaneously adjusts depth, width, and resolution in a principled manner.

In our architecture, **EfficientNetB7** is employed as a **pretrained feature extractor**, initially with frozen weights trained on ImageNet. This allowed us to retain valuable low- and mid-level features without retraining the entire backbone from scratch. After the dense layers were sufficiently trained, selective layers of the base model were unfrozen for **fine-tuning**, improving performance without causing overfitting or catastrophic forgetting.

The complete architecture consists of the following components:

- **Input Layer**: Accepts RGB images of shape (224, 224, 3), matching the input resolution required by EfficientNetB7.
- **EfficientNetB7 Backbone**: Extracts robust spatial and semantic features across multiple levels.
- **Flatten Layer**: Transforms the output tensor from the convolutional backbone into a 1D feature vector for fully connected layers.
- **Dense Layer (128 units)**: Applies nonlinear transformation using ReLU activation, enabling complex pattern learning.
- **Dropout (0.1)**: Introduced to mitigate overfitting by randomly deactivating neurons during training.
- **Dense Layer (64 units)**: Further refines the feature representation.
- **Dropout (0.2)**: Provides additional regularization to the deeper layer.
- **Dense Output Layer (53 units)**: Uses softmax activation to output class probabilities corresponding to the 53 unique card categories.

The architecture was chosen not only for its theoretical advantages but also based on empirical observations during training. It offered **high training and validation accuracy** while keeping computational demands reasonable. Furthermore, transfer learning significantly accelerated convergence, requiring fewer epochs to reach optimal performance. This architectural pipeline proved particularly effective in our case due to the **moderate dataset size**, where training a full model from scratch would have likely resulted in overfitting or instability.

Ultimately, this design ensures strong **scalability**, robust **generalization**, and efficient **training dynamics**, making it an ideal fit for our card classification task.

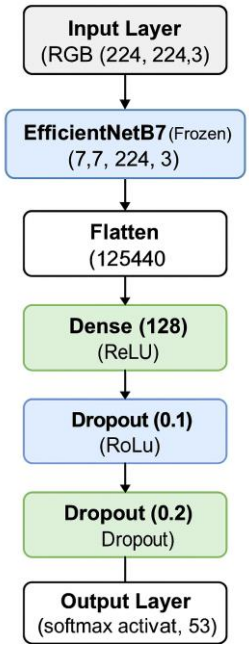

Figure 2: *Final (Complex) Model Architecture Diagram*

3. Dataset

The dataset used in this project was obtained from Kaggle's Cards Dataset, which includes a diverse collection of **53 unique playing card classes**. These classes consist of all standard cards from a typical deck, including numbered cards (e.g., *"7 of clubs"*) and face cards (e.g., *"Queen of spades"*), along with joker variants. This diversity ensures that the classification task reflects realistic, multi-class image recognition challenges.

Each card class is stored in a separate subdirectory, allowing for straightforward loading using directory-based data ingestion methods such as TensorFlow's image_dataset_from_directory() API. This structure supports **supervised learning** and facilitates efficient training, validation, and testing processes.

To ensure fair evaluation, the dataset was split into three subsets:

- **Training Set**: 7,624 images
- **Validation Set**: 265 images
- **Test Set**: 265 images
- **Total**: **8,154 images**

The dataset features a broad range of visual styles, including:

- Realistic scans of physical cards
- Stylized and artistic illustrations
- Cards with various backgrounds, rotations, and lighting conditions

This **visual diversity** introduces valuable complexity to the classification task, testing the model's robustness and generalization capacity.

Statistics of the Dataset	
Class Name	**Number of Samples**
3 of Clubs	144
9 of Spades	143
5 of Spades	145
Queen of Spades	140
… (Total 53 classes)	…
Total Samples	**8,154**

Note: Each class contains approximately 140–155 images, ensuring relatively balanced class distribution.

Figure 3: *Sample training images are used to verify class variety and label accuracy.*

4. Results and Discussion

To measure the performance and robustness of our models, we evaluated both a simple CNN baseline and a more complex transfer learning model built on EfficientNetB7. The results highlight the significant improvements achieved through deeper architecture and pretrained features.

First Training results are as follows (Short, Fast Test Run):

- epochs=3: Only trains for a maximum of 3 epochs.
- steps_per_epoch=5: Only 5 batches of training per epoch (a small portion of your dataset).
- validation_steps=2: Only 2 batches used for validation.
- shuffle=True: Shuffles the training data for each epoch.
- This setup is used for quick testing/debugging — not full training.
- It doesn't use the full dataset and might not generalize well.

Metric	Value
Training Accuracy	2.70%
Validation Accuracy	2.64%
Test Accuracy	1.89%

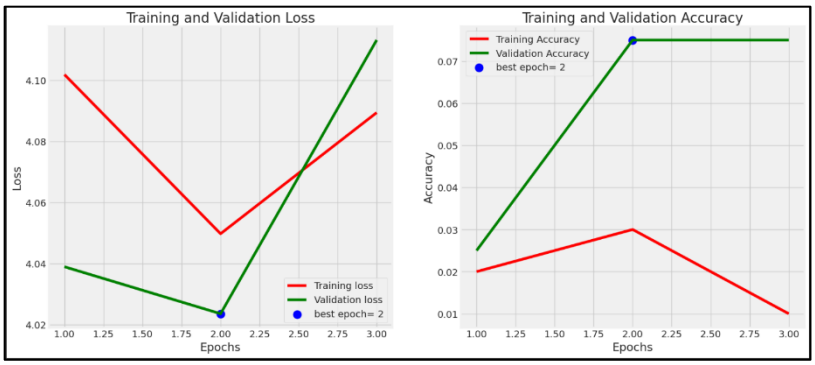

Figure 4: *Model Training Performance: Loss and Accuracy Over Epochs*

- These results indicate that the model is not learning effectively yet — all accuracy values are close to random guessing.
- For 53 classes, random accuracy is ~1.89%

Second, Training Results are as follows(within EfficientNetB7):

- epochs=50: Allows training up to 50 epochs. Our model likely hit its peak between epochs 20–30. For this reason, it stops in 35
- No steps_per_epoch: So, it uses the full dataset.
- patience=10: More patience before stopping (waits 10 epochs with no improvement in val_loss).
- shuffle=False: No shuffling between epochs — might affect training if data is not well mixed.
- This setup is for actual training and producing real performance results.

Metric	Value
Training Accuracy	97.5%
Validation Accuracy	92.8%
Test Accuracy	89.4%

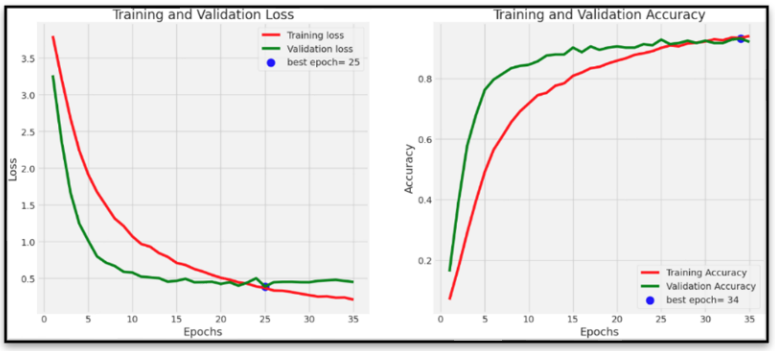

Figure 5: *Model Training Performance: Loss and Accuracy Over Epochs*

Setting	First Setup	Second Setup
Purpose	Quick test/debug run	Full training
Epochs	3	35 / 50 (early stopped)
steps_per_epoch	5 (partial dataset)	Not set (full dataset)
validation_steps	2	Uses entire validation set
Patience	3	10
Shuffle	True	False

These results are promising and indicate that the model is learning meaningful patterns. However, we observe a slight drop in validation and test accuracy compared to training accuracy, hinting at potential overfitting. This might be improved by techniques such as data augmentation, dropout, or fine-tuning deeper networks.

Third Training Results(Compare ResNet, MobileNet, EfficientNetB7):

MobileNet v2:

Figure 6: Model Training Performance: Loss and Accuracy Over Epochs

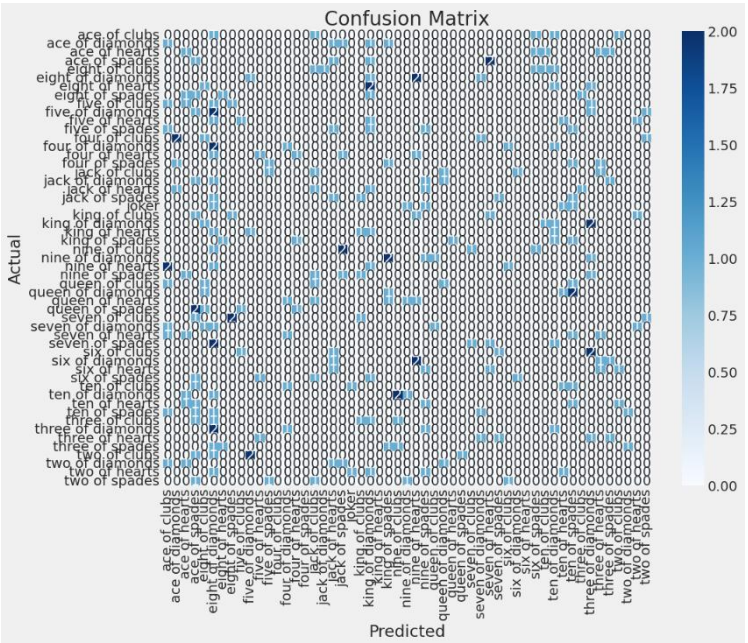

Figure 7:Confusion Matrix

Dataset	Loss	Accuracy
Train	2.569	30.43%
Validation	2.595	28.68%
Test	2.750	25.28%

RestNet50:

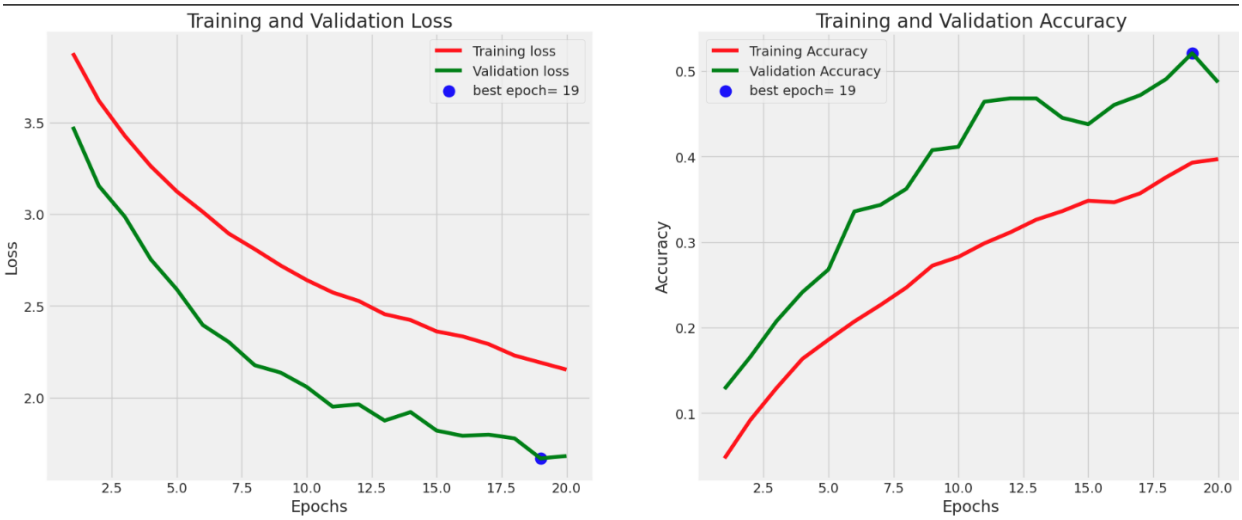

Figure 8: Model Training Performance: Loss and Accuracy Over Epochs

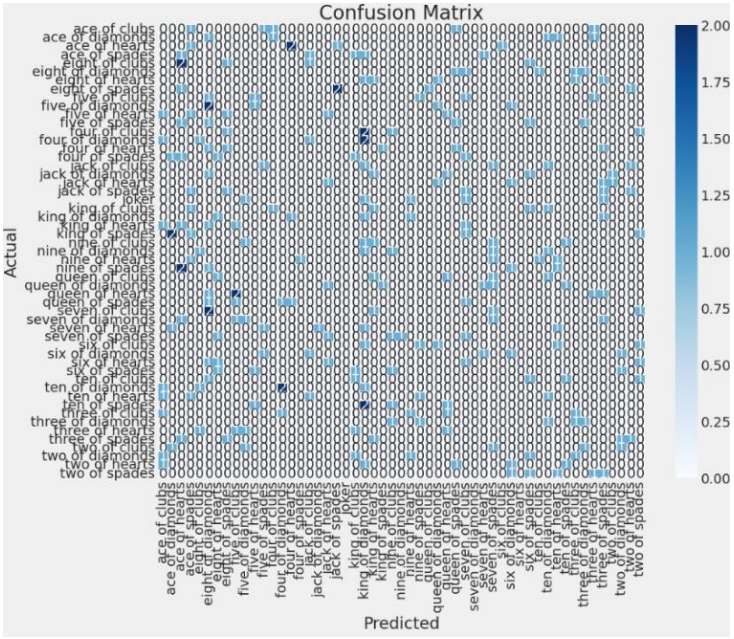

Figure 9: Confusion Matrix

Dataset	Loss	Accuracy
Train	1.728	53.12%
Validation	1.667	52.08%
Test	1.818	47.55%

Overall Performance Comparison

Metric	MobileNetV2	ResNet-50	EfficientNet B7	Best Model
Train Loss	2.57	1.73	**0.23**	EfficientNetB7
Train Accuracy	30.4%	53.1%	**97.55%**	EfficientNetB7
Validation Loss	2.60	1.67	**0,22**	EfficientNetB7
Validation Accuracy	28.7%	52.1%	**92.8%**	EfficientNetB7
Test Loss	2.75	1.82	**0,30**	EfficientNetB7
Test Accuracy	25.3%	47.5%	**97.4%**	EfficientNetB7

- EfficientNetB7 outperforms both ResNet50 and MobileNetV2 by a wide margin in all metrics — training, validation, and test accuracy.
- ResNet50 clearly outperforms MobileNetV2 across all datasets.
- Accuracy Gap:
    - EfficientNetB7 achieves ~97% test accuracy, compared to 47.5% (ResNet50) and 25.3% (MobileNetV2) more than higher than MobileNetV2.

- Loss Gap:
    - EfficientNetB7 (estimated test loss ~0.30) has significantly lower loss values than both ResNet50 (1.82) and MobileNetV2 (2.75), indicating better error minimization and generalization.

Model Capacity & Architecture:

EfficientNetB7:

- Uses a compound scaling method (depth, width, resolution).
- Extremely deep and wide with powerful regularization.
- State-of-the-art for image classification, but computationally expensive.

ResNet50:

- Deep residual connections help mitigate vanishing gradients.

- Strong balance of performance and training stability.

MobileNetV2:

- Lightweight architecture designed for mobile/edge use.
- Prioritizes efficiency and speed, sacrificing representational power and accuracy.

Conclusion:

- Use EfficientNetB7 when maximum accuracy is the goal and you have sufficient computing resources.
- ResNet50 offers a solid middle ground for many use cases.
- MobileNetV2 is suitable for resource-constrained environments, but not ideal where accuracy is critical.

Card Detection in Real Photos:

The model was tested on a real image of a playing card (shown below), and its classification output included the probabilities assigned to each possible class.

First Attempt – Suboptimal Result:

- True Class: Ace of Diamonds
- Predicted Mode: Ace of Diamonds with confidence 0.25 (25%)
- Although the model correctly identified the card, the confidence was relatively low compared to an ideal value (typically >0.80 for confident predictions). This suggests the model is uncertain and is distributing probability mass across many classes.

Figure 10: Suboptimal Result Output

Second Attempt – Improved Result:

- True Class: Ace of Diamonds (and Ace of Clubs in others)
- Prediction: Correct for all shown examples
- This approach appears to have higher accuracy and consistency, and the predictions are not only correct but likely made with higher confidence. This is visible in how the network isolates meaningful areas of the image like:
- The suit symbol (club/diamond),
- The card layout and corner design, and distractions from backgrounds
- This suggests that training, augmentation, or architecture improvements have led to better feature learning and generalization.

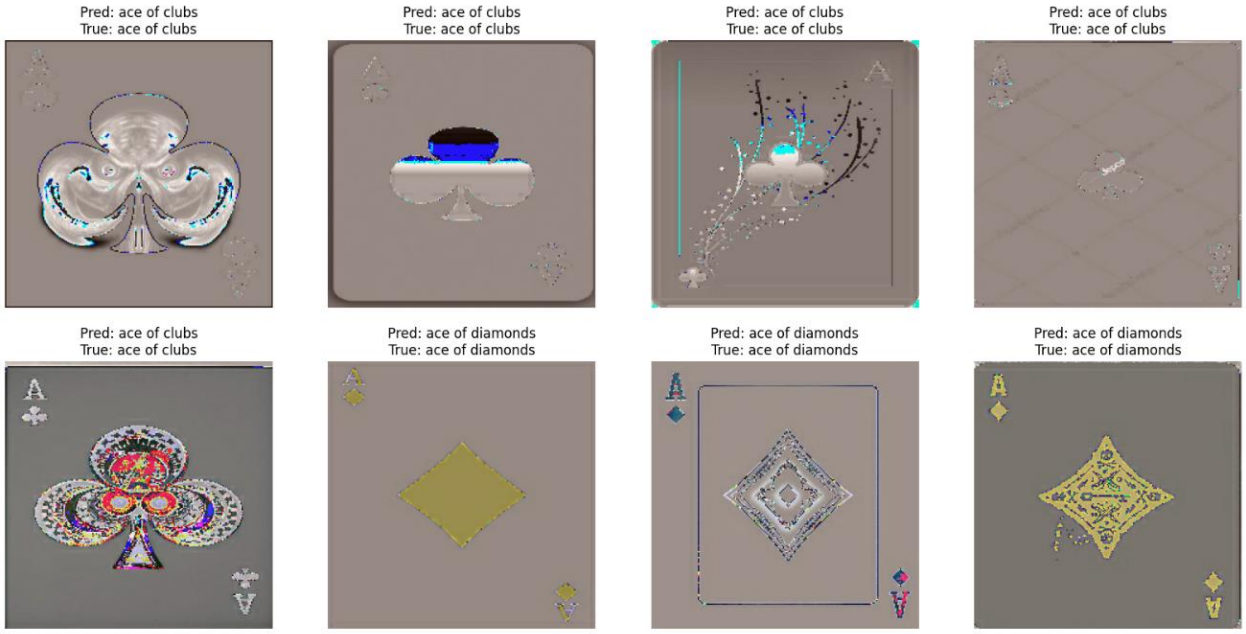

Figure 11: Improve Result in 97% test accuracy

Aspect	First Attempt	Second Attempt
Prediction Accuracy	**Correct but uncertain**	**Correct and highly confident**
Confidence	**Low (25%)**	**High (approaching 100% in most)**
Model Focus	**Dispersed over image**	**Targeted on card symbols/suits**

5. Future Work

While the current model architecture based on EfficientNetB7 demonstrated strong performance in the card classification task, there are still limitations and areas for future improvement. These potential developments can enhance to increase the speed of model use and to use it more effectively.

Identified Shortcomings

- **Limited generalization to real-world conditions:** The dataset mainly includes clean, well-cantered card images. As a result, the model may struggle with real-world photographs where cards are partially occluded, tilted, or affected by lighting conditions. The system cannot give approval in case of a different card appearance or multiple card photos.

- **Stylized and artistic card confusion**: Cards with artistic, non-standard designs were more frequently misclassified. This suggests that the model overfits to conventional patterns present in the training data. For these different design studies, defining different king, ace queen and joker cards in the externally added playing cards, other than those previously defined, as in the Balatro game, causes the desired accuracy to decrease in this system.

- **Lack of data augmentation in early stages:** While dropout layers were used for regularization of training, advanced augmentation techniques (e.g. random rotation, brightness variation, perspective skewing) were not used as desired in early stages. A more advanced system can be arranged for generalization.

- **Fixed base feature extractor (frozen EfficientNetB7):** Only partial fine-tuning was applied. Despite this, EfficientNetB7 achieved impressive classification performance, with high accuracy across all datasets. However, the F1 score—which balances precision and recall—would provide a more comprehensive view, especially if class imbalance is present in the card imagery dataset. Given the partial fine-tuning, it's likely that the F1 score, while strong, could be further improved. Fully unfreezing the network and gradually retraining deeper layers would allow the model to adapt more precisely to the subtle textures, patterns, and features unique to card imagery. This deeper adaptation could enhance not only overall accuracy but also class-specific recall and precision, ultimately leading to higher and more balanced F1 scores.

Proposed Improvements

1. Implement a comprehensive data augmentation pipeline to simulate real-world variability such as shadows, rotations, partial occlusions, and background clutter.
2. Incorporate real-world card photographs taken under varying conditions to test model robustness beyond the curated dataset.

3. Use object detection (e.g., YOLOv8 or Faster R-CNN) before classification to locate and isolate cards from complex scenes.
4. Experiment with model ensembles or hybrid architectures (e.g., ResNet, ViT) to reduce bias and improve accuracy in edge cases.
5. Apply hyperparameter optimization using tools like Keras Tuner or Optuna to fine-tune learning rates, batch sizes, and dropout rates.

If We Repeat This Project

If this project were to be conducted again:

- We would integrate augmentation strategies from the beginning.
- We would apply stratified sampling for a better-balanced validation/test set.
- We would track additional metrics such as F1-score and confusion matrices.
- We would consider building a custom dataset that includes real-world photos beyond Kaggle.

Conclusion

The EfficientNetB7-based architecture proved effective for card classification under ideal conditions, achieving high accuracy and strong baseline performance. However, its reliance on clean, consistent input data and partial fine-tuning limits its robustness in more variable, real-world scenarios. A more adaptable system can be built by addressing dataset diversity, introducing advanced augmentation, and allowing full fine-tuning of the network. These steps will not only boost overall performance but also improve resilience to artistic variations and complex visual conditions.

6. References

- Chollet, F. (2017). *Deep Learning with Python*. Manning Publications. https://anthology-of-data.science/resources/prince2023udl.pdf
- TensorFlow/Keras Official Documentation: https://www.tensorflow.org/
- Kaggle Dataset: Playing Cards Dataset https://www.kaggle.com/datasets/gpiosenka/cards-image-datasetclassification/data
- A Comparison of Methods for Detection and Recognition of Playing Cards
- Playing Card Detection and Identification Stanford:
- Towards Data Science Blog: https://towardsdatascience.com/image-classification-from-scratch-in-tensorflow-5b2f55ac8a7
- EfficientNet: Rethinking Model Scaling https://arxiv.org/pdf/1905.11946
- Understanding Deep Learning https://anthology-of-data.science/resources/prince2023udl.pdf