CS 5388 Project Report: Classification of Cards by Implementing Convolutional Neural Networks (CNNs)

Baris Ozcan
Texas Tech University
bzcan@ttu.edu
R11967437

Nameera Khan
Texas Tech University
namkhan@ttu.edu
R11965493

Namra Khan
Texas Tech University
namrkhan@ttu.edu
R11965577

Sahel Azzam
Texas Tech University
saazzam@ttu.edu
R11814223

Abstract

Card classification from images is a fundamental computer vision task with applications in gaming, automation, and augmented reality. This study investigates the effectiveness of deep learning approaches for automated playing card recognition. This study presents a comparative analysis of two convolutional neural network (CNN) architectures for the classification of standard playing cards from image data which are a custom-designed sequential CNN and a pre-trained EfficientNet-B0 model. The dataset consists of 8,154 high-resolution images uniformly distributed across 53 card classes. The custom CNN model was trained from scratch, incorporating standard convolutional and pooling layers with dropout for regularization. On the other hand, the EfficientNet-B0 model was fine-tuned using transfer learning techniques to use features learned from large-scale datasets. This study shows advantages of transfer learning for card recognition tasks. It offers a reproducible benchmark for future studies in domain-specific image classification.

1. Introduction/Background/Motivation

We taught a computer how to recognize and label different playing cards just by looking at pictures of the card. Our goal was to build a system that can look at an image of a card and correctly tell which card it is, like ”Ace of Spades” or ”3 of Hearts,” using only the picture.

Firstly, we collected many clear images of playing cards. We then trained a model by showing it many of these pictures, along with the correct name for each card.

We built a simpler model from scratch using the Sequential method. This model had fewer layers and was trained entirely on the playing card images we provided. This helped us understand how well a custom-designed network could perform compared to a pre-trained one. Second phase, we used a pre-trained model which is EfficientNet [5][1]. We fine-tuned this model using our playing card images. This means we adjusted the model's existing knowledge to make it better at identifying playing cards. In the scope of our project, we compare their results to discuss methods in terms of classification accuracy and computational cost.

Recent research has explored various approaches to playing card classification. Mittal et al. study to classify and detect how one plays cards specifically using deep learning technique [3]. They developed a CNN that is designed to identify the rank and suit of playing cards by using their image. Synder proposes a approach to classify playing cards [4]. His SIFT-based feature matching technique detect and identify playing cards from static images. This method achieve 98% accuracy under the best possible conditions. However, it has limitations due to the symmetrical features of playing cards. Apiletti et al. proposed a data mining-based decision support system that employs classification algorithms such as decision trees, Naive Bayes, and SVMs [2]. While their core objective is to accurately classify cards based on learned patterns, they specifically focus on the context of poker. These studies collectively illustrate that current methods have evolved from hand-engineered features to deep learning architectures. However, deep learning methods have challenges persist in handling symmetric visual patterns, variable lighting, and real-time classification needs.

Reliable card recognition models have potential to impact amany domains. *Casino surveillance* departments can flag marked or swapped cards in milliseconds, cutting annual losses from cheating events. *Automated card dealers and sorting robots* may gain a robust perception module that tolerates glare and partial occlusion. *Augmented-reality game developers* can embed a classifier for offline playing on mobile devices. the academic community obtains benefits from a publicly accessible benchmark for developing a model to specifically specializes in card recognition tasks.

Data Sheet

- **Purpose:** This dataset was collected to support deep learning models in recognizing and classifying playing card types based on images. It addresses the need for a high-quality image dataset specific to standard playing cards for applications such as object detection, augmented reality, and computer vision-based games. Our group downloaded this dataset from open source to using it in accordance with the purpose of creation.

Composition

- **Instances:** Each instance represents a color image of a single playing card.
- **Total Count:** 8,154 images (7,624 training, 265 validation, 265 testing)
- **Sampling:** The dataset is not a random sample. It is a curated collection where each class of playing card is uniformly represented across the train, validation, and test sets.
- **Instance Content:** JPEG images of resolution 224x224 pixels with 3 color channels (RGB).
- **Labels:** Each image is labeled according to one of the 53 unique card types (including jokers).
- **Data Splits:** Train, validation, and test sets are provided as separate folders with 53 subdirectories each, corresponding to card types.

Collection Process

- **Data Acquisition:** The images were directly captured using a camera.
- **Collection Method:** Our team gathered data from an open-source website named Kaggle.
- **Timeframe:** The dataset was downloaded on March 31, 2025.

Preprocessing, Cleaning, and Labeling

- **Preprocessing:** Each image was cropped to ensure the card occupies over 50
- **Labeling:** Labels were defined based on existing directory structure.
- **Software:** Preprocessing scripts were written in Python.

Uses

- **Current Uses:** Classification and computer vision tasks involving card recognition.
- **Potential Uses:** AR card games, card sorting robots, training for any card classification pipelines.

Distribution

- **Availability:** https://www.kaggle.com/datasets/gpiosenka/cards-image-datasetclassification
- **License:** CC0: Public Domain
- **Restrictions:** Open Source

2. Methodology and Approach

In this section, we describe the key stages of the project, from data collection to model evaluation.

2.1. Data Access and Preparation

To start, we initially handled dataset from Kaggle. Then, for using dataset in colab platform, dataset was a ZIP file called `cards.zip` stored on Google Drive. We mounted Google Drive, extracted the contents of the ZIP file into a folder called `cards_extracted`, and prepared it for further processing.

After that, we defined the paths for the training, validation, and test datasets, which helped us organize the data properly. We made sure to use the correct folders for each stage of data analysis and model training.

2.2. Data Preprocessing

Next, we moved on to data preprocessing to get our images ready for model training. The images were resized to a uniform shape of 200x200 pixels. Resizing helped standardize the input size for both models, making it easier for the network to process.

We also normalized the images by scaling the pixel values to the range [0, 1]. This was done using an `ImageDataGenerator` with the `rescale=1./255` argument. Normalization helps the model learn better and faster by reducing the range of input values.

For loading and preparing the data for training, we used the `flow_from_directory` method. Ultimately, this function allowed us to load images in batches, and it automatically assigned labels to the images based on their folder structure. This step was essential for organizing and labeling the data for both models.

2.3. Model Development

Moving forward, for the image classification task, we worked with two different models which are sequential CNN model which is created by us and pre-trained Efficient Net model that fined tuned by our project.

2.3.1 CNN Model

The first model we developed was based on a Sequential CNN architecture:

- **Conv2D Layers**: The model begins with three convolutional layers: the first with 32 filters, followed by 64 and 128 filters respectively. All use 3×3 kernels and ReLU activation to extract increasingly complex spatial features from the input images of shape (200, 200, 3).

- **MaxPooling2D Layers**: A max pooling layer with a 2×2 window follows each convolutional layer to reduce the spatial resolution.

- **Dropout Layer**: A dropout layer with a rate of 0.5 is applied after flattening to prevent overfitting by randomly deactivating half of the neurons during training.

- **Flatten Layer**: This layer transforms the 2D feature maps into a 1D vector. It serves as a bridge between the convolutional base and fully connected layers.

- **Dense Layers**: A fully connected hidden layer with 128 neurons and ReLU activation is used to capture high-level abstractions. The final output layer consists of 53 neurons (one for each class) with softmax activation. It produces class probabilities.

Our CNN model was trained using the Adam optimizer with a learning rate of 0.0001 which gave better performance compared to alternative values of 0.1, 0.01, and 0.001. The batch size was set to 32 after empirical comparison with a batch size of 64. Also, the model was trained for 10 epochs. These hyperparameters were selected based on heuristic tuning rather than automated search strategies. The training process was implemented by using the GPU of Colab.

2.3.2 EfficientNet-B0-based Model

The second model was based on EfficientNet-B0, a more advanced and efficient architecture. EfficientNet-B0 uses compound scaling, which balances the depth, width, and resolution of the model for optimal performance. We used a pre-trained EfficientNet-B0 model and fine-tuned it for our card classification task.

The model was trained using the Adam optimizer with a learning rate set to 0.001. The loss function used was CrossEntropyLoss which is appropriate for multi-class classification problems. We trained the model for 10 epochs. The training was performed on a GPU in CUDA environment of Colab.

2.4. Model Training

We used the training dataset to train both models for 10 epochs. During training, we monitored the performance of the models using validation data, which helped us assess their generalization ability.

We also tracked key metrics like training and validation accuracy, and loss, by plotting graphs at the end of each epoch. These plots provided a visual understanding of how the models were learning and helped us adjust hyperparameters if needed.

Hyperparameters are adjusted based result of models and our heuristics on these results. Due to limitations in computational power, systematic hyperparameter tuning cannot be conducted.

2.5. Model Evaluation

After training, we moved on to evaluating the performance of both models. We tested each model on the unseen test dataset, which provided an unbiased measure of their accuracy.

Predictions were generated by the models, and we compared these predictions with the true labels. We calculated the final accuracy score for each model. This step allowed us to objectively compare the performance of the CNN model and the EfficientNet-B0-based model.

2.6. Performance Analysis

To dive deeper into the models' performances, we conducted a thorough analysis using various evaluation techniques:

- **Confusion Matrix**: We generated confusion matrices for each model to see prediction errors. These matrices helped us understand how each model performed across different classes.

- **Heatmaps**: The confusion matrices were normalized and visualized as heatmaps, which made it easier to interpret the results.

- **Per-Class Accuracy**: We calculated per-class accuracy to see how well the models performed on individual classes. This allowed us to identify the classes that the models struggled with and those they excelled at.

By analyzing the confusion matrices and per-class accuracy, we identified the worst-performing classes and the best-performing classes for each model. This analysis helped us observe the strengths and weaknesses of our models, which can potentially assist us ahead in improving their performance.

2.7. Reflections and Challenges

At the beginning of this project, we had a clear idea of using deep learning techniques for image classification. We chose to implement two different models which is a traditional CNN and a more modern EfficientNet-B0 because we believed that comparing both would give us a solid understanding of their strengths and limitations on the same dataset. CNNs have consistently proven to be effective for image-based tasks due to their strong feature extraction capabilities. Meanwhile, EfficientNet-B0 is used in the scope of our project because it is a pre-trained model that provide us insights about fine-tuning on card classification. We were confident that using both would help us build a appropriate solution.

What was new in our approach was comparing a foundational model like CNN against a model like EfficientNet-B0 gave us insights into how architectural complexity affects performance on the same task.

We anticipated a few challenges early on. For instance, we expected that working with datasets from two platforms might result in inconsistencies in folder structure, naming conventions, or class distributions. We were also aware of potential overfitting, especially with a relatively small dataset and a powerful model like EfficientNet-B0. Managing computational resources and memory while training a deep network in Colab was another concern.

As we progressed, we faced some of these issues. The CNN model initially overfit the data quite fast, which led us to introduce dropout and monitor validation loss closely. EfficientNet-B0, though more accurate, came with longer training times and higher memory usage, which we managed by reducing the batch size.

Interestingly, the very first thing we tried—the basic CNN model—did work in terms of running and training, but its performance wasn't optimal. It served as a good baseline, but we had to experiment with additional layers and regularization to stabilize its accuracy. Switching to EfficientNet-B0 provided better scores and validated our belief that combining simpler and more advanced models would give us a better overall picture.

While the techniques used are standard in image classification, combining the two data sources and thoroughly comparing a classic CNN with a pre-trained model gave us deeper insights and a more well-rounded evaluation of model performance.

2.8. Data Preprocessing

1. Resizing to 128x128 Pixels: The input images were resized to a fixed resolution of 128x128 pixels to ensure uniformity across all images, allowing the model to process them in a consistent manner.
2. Conversion to Tensor: The images were converted into tensors using transforms.ToTensor(), which automatically scaled pixel values from the range [0, 255] to [0, 1]. This step prepares the images to be fed into the neural network by converting them into a format compatible with PyTorch.

3. Experiments and Results

3.1. Experiment

In this experiment, we used which are sequential CNN model and a pre-trained model. The pre-trained model is in the EfficientNet-B0 architecture. It was initialized with pre-trained ImageNet weights to leverage transfer learning. We modified the model by removing its original classification head and adding a new linear layer with 53 output nodes to match the number of target card classes. This allowed the model to extract high-level features from the input images using the EfficientNet backbone while adapting to the specific multi-class classification task through the new custom head.

Training and Testing Procedures: For each batch, the model performed a forward pass to compute predictions, calculated the loss, and then updated its weights using back-propagation. After each epoch, the model was evaluated on a separate validation set to monitor generalization performance. Metrics such as loss and accuracy were tracked for both training and validation phases.

Cross-Validation or Evaluation Strategies Used: We employed a standard hold-out validation strategy, where the dataset was split into training and validation sets before training began. This allowed us to evaluate the model's ability to generalize to unseen data without additional complexity. Although we did not implement cross-validation techniques like k-fold due to computational constraints, the framework can be extended to include such strategies. Model selection was based on the highest validation accuracy across epochs, ensuring that the most reliable version was preserved.

3.2. Results

As seen in Figure 1, the training and validation curves shows that the model initially learns effectively because

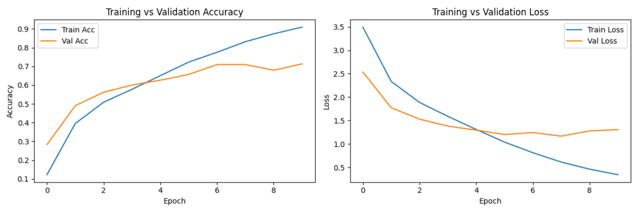

Figure 1. Training versus validation accuracy (left) and loss (right) across epochs for our custom CNN model.

both training and validation accuracy improve steadily during the first several epochs. However, validation loss stops decreasing after epoch 5 and begins to rise slightly. On the other hand, validation accuracy start to increase after epoch 8 until the epoch 10. The difference between validation loss and training loss has potential to cause overfitting problem.

In the Figure 2, for our Baseline CNN model, the confusion matrix consists of classes with lowest accuracy. While some classes such as ten of diamonds, two of hearts, six of clubs, and two of diamonds were classified perfectly, others show significant misclassification issues. Notably, the joker class was entirely misclassified. Several other classes, including three of diamonds, two of clubs, two of spades, and three of spades, also showed confusion with visually similar cards. These misclassifications suggest that the model struggles to distinguish fine-grained differences between similar-looking cards.

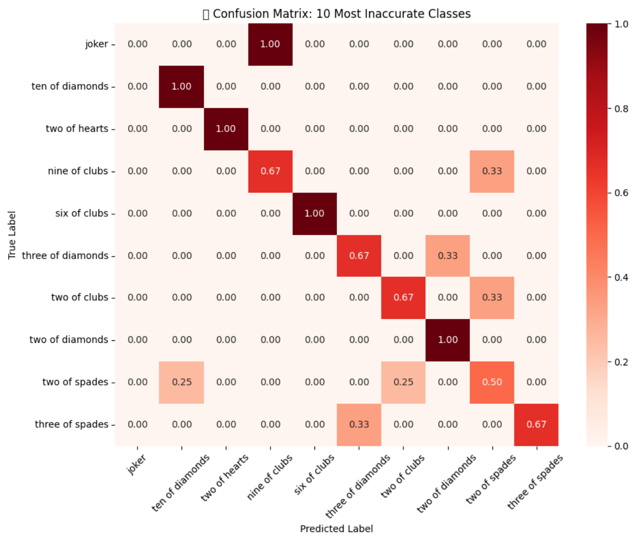

Figure 2. Ten most inaccurate classes for our custom CNN model.

In the Figure 3, best classified class are shown. In this confusion matrix from our Baseline CNN model.This confusion matrix confirms the model's high reliability on many card classes. The few misclassifications are minor and occur between visually related cards. For instance, jack of spades and queen of hearts which were occasionally misclassified as queen of hearts and queen of diamonds, respectively. Similarly, jack of hearts and five of hearts were sometimes confused with each other.

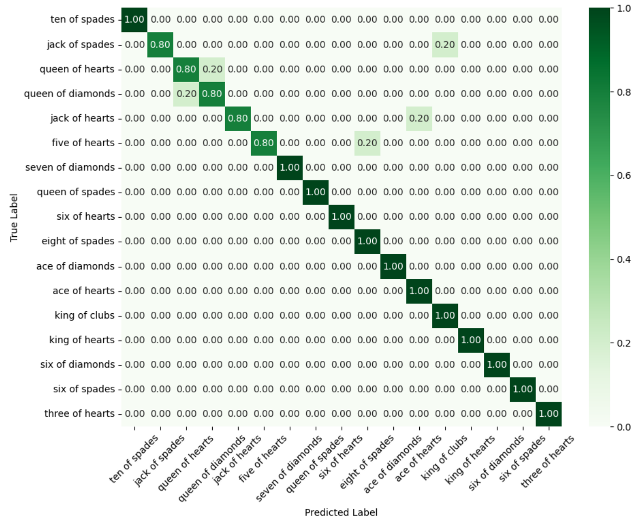

Figure 3. Confusion matrix for the 17 most accurate classes by the our custom CNN model.

As you can see from the two graphs in Figure 4, during the first several epochs the model efficiently learns fundamental card characteristics. The training accuracy grows from 56% to over 90% by epoch 3, while validation climbs to 94%. After epoch 4 both metrics stabilize for training at about 97% and validation at 96%. The loss curves follow same patter. THe training loss comes from 1.57 to below 0.10, and validation loss drops from 0.35 to approximately 0.12 before flattening. A small gap between training and validation after epoch 5 indicates effective generalization.

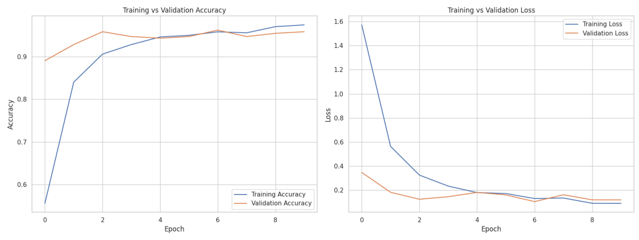

Figure 4. Training versus validation accuracy (left) and loss (right) across epochs for the pre-trained EfficientNet-B0 model.

As you can see from Figure 5, data from the ten most inaccurate classes shows that EfficientNet B0 shows a high level of reliability. Seven out of ten classes (ace of clubs, five of spades, five of diamonds, jack of clubs, six of hearts, three of spades, and ten of spades) consistently receive perfect recognition, and three club cards (five, six, and queen) retain eighty-percent accuracy. All errors occur in the club suit, indicating that the network has correctly identified the suit but occasionally mixed up adjacent ranks.

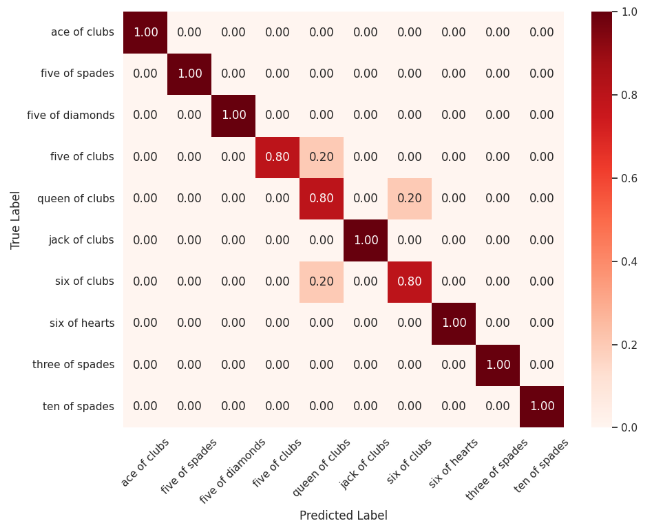

Figure 5. Ten most inaccurate classes for EfficientNet-B0 on the test set.

As you can see from Figure 6, EfficientNet-B0 exhibited perfect accuracy in detecting all 15 most accurate classes of ranks and suits. The test set contained no misclassifications for any cards from Queen of Diamonds among Two of Diamonds. This uniform 1.00 diagonal highlights robust feature separation. It demonstrates model's clear distinction capabilities across multiple card types.

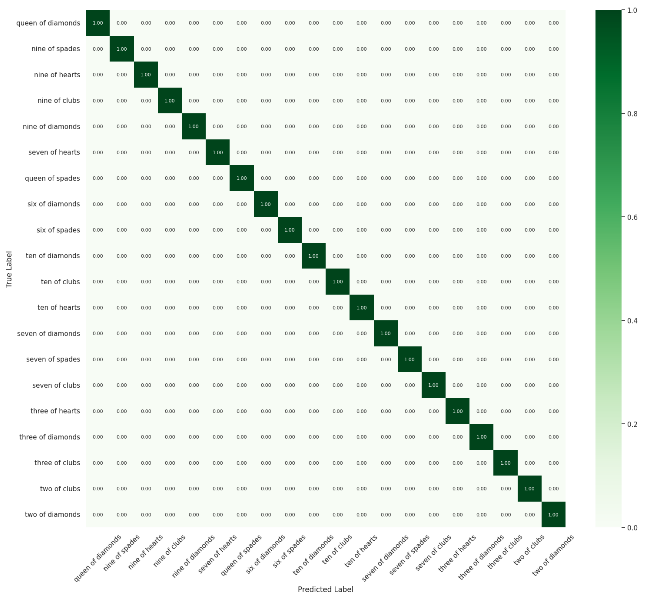

Figure 6. Confusion matrix for the 20 most accurate classes by the EfficientNet-B0 model.

In Table 1, we can see result of our CNN model and Efficient Net model that we fine-tuned. According to evaluation results the Baseline CNN achieved 93% training accuracy.

Model	Training Acc.	Validation Acc.	Test Acc.
Baseline CNN	0.930	0.732	0.691
EfficientNet	0.975	0.959	0.955

Table 1. Accuracy comparison of the two models across training, validation, and test splits.

The validation accuracy reaches 73% and test accuracy reaches 69% respectively. The pre-trained EfficientNet-B0 maintains its high accuracy level throughout every validation phase. The system maintains a constant 95.5 test accuracy level compared to training accuracy at 97.5 across all segments. Transfer learning benefits from ImageNet weights which deliver better features that lead to improved robustness. The transferable features in EfficientNet-B0 reduce overfitting which leads to significant improvements in performance. EfficientNet-B0 looks as the optimal choice for application purposes even it needs high computational power. In short, we can claim that our baseline model suffers from overfitting problem due to insufficient size of training data when compare it pre-trained model.

4. Conclusion

In this project, we addressed the task of classifying standard playing cards using deep learning models based on convolutional neural networks. We implemented and evaluated two different approaches. Our experimental results demonstrated that while the baseline CNN achieved reasonable performance, the EfficientNet-B0 model outperformed its performance.

Based on our analysis which includes confusion matrices and per-class accuracy, we identified specific strengths and limitations of each model. The EfficientNet-B0 model showed strong generalization and robustness. However, its high parameter amount causes misclassifications primarily occurring in visually similar classes. However, we can claim that the EfficientNet-B0 model has power to understand type of cards. Our findings shows the advantages of using transfer learning and pre-trained architectures for domain-specific image classification tasks, especially when dataset size is limited.

Due to computational constraints, we were unable to perform an extensive hyperparameter tuning process in the scope of the project. Key parameters such as learning rate, batch size, dropout rate were selected based on empirical heuristics rather than systematic optimization. In future research, we plan to implement comprehensive hyperparameter tuning strategies such as grid search, random search, or Bayesian optimization. Also, we can use data augmentation strategies by considering underperforming classes based on results of confusion matrix analysis. These improvements can provide more robust and efficient CNN models which are suitable for deployment in real life tasks.

5. Appendix

Name of Team Member	Contributed Aspects	Details
Baris Ozcan	Coordination, Model Development, and Reporting	• Implemented a CNN with three convolutional and pooling layers, followed by dropout and dense layers, using Adam optimizer and categorical crossentropy on 7,624 card images. • Performed heuristic hyperparameter tuning for our Custom CNN model. Specifically, the learning rate and batch size were selected based on empirical observations from a limited set of manual experiments. • Conducted model evaluation through training/validation accuracy and loss plots and confusion matrix analysis. • The literature review was conducted. • Abstract is written. • Conclusion is written. • The project objective and problem definition were determined. • The dataset was described by highlighting key properties in accordance with the ”Datasheets for Datasets” framework. • The methodology and implementation steps were explained. • The anticipated and encountered challenges were documented. • Structured the project workflow and research process by defining tasks and providing coordination between team members to complete tasks. • Reviewed and refined written contributions from other team members for consistency. • Fixed bugs and errors in Python code and LaTeX commands written by other team members. • Final submission files were compiled and cross-checked to ensure consistency and completeness.
Nameera Khan	Testing, Evaluation, Reporting	• Data processing was implemented make data ready use with EfficientNet model • The pre-trained EfficientNet model was fine-tuned using the project dataset to improve classification accuracy. • Challenges related to our approach is documented. • Properties of the pre-trained model were analyzed, including which components contained learnable parameters. • Data visualizations were created to illustrate the performance of the pre-trained model. • The informations about properties of models are written in the report.
Namra Khan	Model Fine-Tuning, Reporting	• Data processing was implemented make data ready use with EfficientNet model • The pre-trained EfficientNet model was fine-tuned. • Challenges related to our approach is documented. • Properties of the pre-trained model were discussed. • Data visualizations were created to illustrate the performance of the pre-trained model.
Sahel Azzam	Model Fine-Tuning, Reporting	• Implemented fine tuning on EfficientNet model. • Figures and tables are was placed in the Latex file. • Figures and a table were interpreted. • Conducted trials to develop a preliminary model to classify cards.

Table 2. Contributions of team members.

References

[1] Maicmi. Train first pytorch model: Card classification. https://www.kaggle.com/code/maicmi/train-first-pytorch-model-card-classification, 2020. Accessed: 2025-05-04. 1

[2] Paulo Martins, Luís Paulo Reis, and Luís Teófilo. Poker vision: Playing cards and chips identification based on image processing. In Jordi Vitrià, João Miguel Sanches, and Mario Hernández, editors, *Pattern Recognition and Image Analysis*, pages 436–443, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. 1

[3] Khushi Mittal, Kanwarpartap Gill, Rahul Chauhan, Manish Sharma, and Gautham Sunil. Playing cards classification and detection using sequential cnn model through machine learning techniques using artificial intelligence. pages 1–4, 04 2024. 1

[4] Daniel Snyder. Playing card detection and identification. 2019. 1

[5] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019. 1