Machine Learning Report – [RPS] Building a CNN for Rock-Paper-Scissors Classification

Matteo Hasa (56269A)

Abstract

Through the application of machine learning techniques, this report presents the development and evaluation of image classification models for recognizing hand gestures from the game Rock–Paper–Scissors. The analysis explores three convolutional neural network (CNN) architectures of increasing complexity, each trained on a labeled dataset of gesture images from *Kaggle*. The main objective is to design models that can accurately classify gestures from the dataset while ensuring also good generalization to real-time applications.

Objective of the analysis

The purpose of this project is to design and train a custom image classifier capable of recognizing hand gestures used in the game Rock–Paper–Scissors. Unlike approaches that rely on transfer learning or pre-trained networks, the classifier was built entirely from scratch: every step, from dataset preprocessing to network design and training, was implemented manually. The objective was to create and evaluate three distinct convolutional neural network (CNN) classifiers with progressively higher complexity level that can identify hand gestures. The central challenge was to develop those models in order to achieves good accuracy while maintaining reasonable training time and the ability to generalize to unseen images.

A key constraint of the project was keeping the models computationally lightweight, being as efficient as possible, given the size of the dataset and the complexity of the task. This limitation guided most of the design choices, including the architecture depth, the size of the input images, and the preprocessing techniques applied. As a result, the models are intentionally simple but achieve excellent levels of accuracy and works well in real-time applications.

To evaluate their performance, the best model was tested in a real-time webcam tool that predicts the player's gesture live. While not the primary objective of the work, these applications provided valuable testing grounds for robustness, latency, and adaptability in real-world conditions, influencing several design decisions throughout the development process.

Constraint and Limitations

Reaching a reasonable computational time was one of the primary constraint. To respect this constraint, were designed relatively lightweight neural network that could be trained quickly while maintaining good performance.

Real-world variability was another important limitation. In the dataset all the images shows only the hand gesture with green background. Different backgrounds, lighting conditions and objects makes the models struggle a lot to identify the gestures. This resulted in a lot different performance between real-time scenarios and validation set.

The models

Three different models are used in this project: Model A, B and C. Starting from the simplest and then gradually increasing complexity until the Model C which is the final one.

- **Model A**: A lightweight baseline CNN which consists of a single convolutional block, a flattening layer, and small dense layers. Its purpose is to provide a simple, fast reference model with limited accuracy but very low computational cost.
- **Model B**: An intermediate custom CNN composed of three convolutional blocks combined with batch normalization and dropout for regularization, as well as

moderate data augmentation and per-channel standardization. This model strikes a balance between simplicity and stronger accuracy.

- **Model C**: An advanced model using transfer learning with EfficientNetB0 pre-trained on ImageNet and fine-tuned on the dataset. With stronger augmentation and a two-phase training strategy, it achieves the highest performance and robustness, making it suitable for real-time applications.

The Dataset

The dataset, available on Kaggle, contains 727 images of the rock gesture, 713 of paper, and 751 of scissors. Although the three classes are relatively balanced, the slight differences in the number of samples are noticeable during model development. Each image in the dataset depicts a hand gesture displayed against a green background.

Model A

Preprocessing

For Model A, the dataset was split into three subsets following a 70/15/15 ratio: 70% for training, 15% for validation, and 15% for testing. This was done directly using TensorFlow's image_dataset_from_directory, ensuring reproducibility with a fixed random seed.

The preprocessing applied to the images was deliberately kept simple and lightweight. Each image was resized to 64×64 pixels to reduce computational cost while retaining enough information for gesture recognition. The pixel values were then normalized by dividing by 255, transforming them from integers in the range [0, 255] to floating-point values in [0, 1]. This step stabilizes training by ensuring that all inputs are on the same scale.

Finally, the datasets were optimized for training with TensorFlow data pipeline operations:

- **Caching**, to store the preprocessed data in memory and avoid redundant computations.
- **Prefetching**, to ensure that the next batch is prepared while the current one is being processed, thus improving GPU utilization and reducing idle time.

No data augmentation or advanced preprocessing was used in this first model, keeping the pipeline minimal and focused on establishing a working baseline.

Architecture

The architecture of Model A was designed to be minimal and lightweight, with the goal of establishing a simple baseline. It is composed of a small convolutional neural network (CNN) made up of:

- A **convolutional layer** with 16 filters and kernel size 3×3, using the ReLU activation function. This layer extracts local spatial features such as edges and simple patterns from the input images.
- A **max pooling layer** 2x2, which reduces the spatial dimensions by keeping only the most relevant features, lowering the number of parameters and preventing overfitting.
- A **Flatten layer**, which converts the 2D feature maps into a 1D vector to feed into the fully connected layers.
- A **Dense layer** with 32 units and ReLU activation, which combines the extracted features into higher-level representations.
- A **Dense output** layer with softmax activation, producing a probability distribution over the three gesture classes (rock, paper, scissors).

This architecture prioritizes simplicity and speed over complexity. It is not expected to achieve great accuracy, but it is useful as a foundation for understanding the behavior of the dataset and provides a benchmark against which more advanced models can be compared.

Evaluation

The evaluation of Model A was carried out using both the validation and test sets obtained from the initial 70/15/15 split. During training, the model's performance was monitored through accuracy and loss curves.

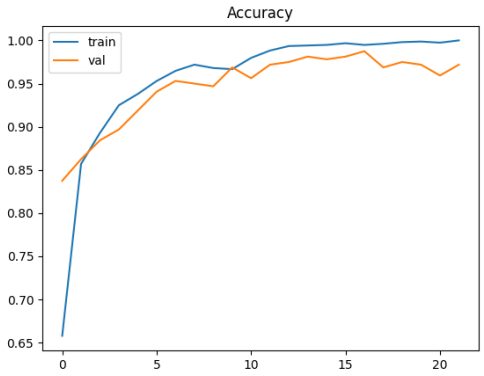

Fig a: Accuracy plot

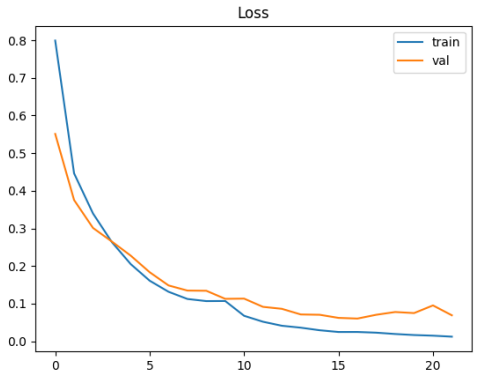

Fig b: Loss plot

These plots show that while the training accuracy curve increases, the validation accuracy stops improving after the fourth epoch. A similar pattern can be observed in the loss curves: the training loss continues to decrease with each epoch, while the validation loss remains flat or even starts to rise. This divergence between training and validation performance is a clear indication of overfitting.

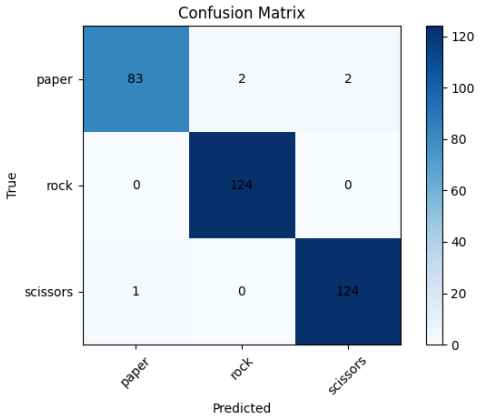

Fig c: Confusion matrix

Looking at the confusion matrix plot, first, it is clear that the three classes are a bit unbalanced, with low values for paper and similar high values for rock and scissors. Then, considering the errors in the prediction of these classes, it can be noticed that the model struggles trying to identify paper gestures, while it is almost perfect in detecting rock and scissors gesture.

In addition to the confusion matrix, a classification report was produced, providing precision, recall, and F1-scores for each class. These metrics confirmed that Model A, while relatively simple, was able to maintain relatively good performance across all categories. Finally, misclassified samples were automatically saved into dedicated folders, allowing a visual inspection of common error patterns, which proved useful for understanding the model's weaknesses.

Model B

Preprocessing

For Model B, preprocessing was designed to ensure a more rigorous treatment of the input data and improve the model's robustness. First of all, the image file paths were **scanned** and associated with their corresponding class labels.

The dataset was then **split** into training, validation, and test sets using a stratified 70/15/15 split, ensuring that each subset maintained the same proportion of rock, paper, and scissors images.

Then, each image was **resized** to 128×128 pixels and converted to float32 values in the [0,1] range.

In addition, **data augmentation** was applied only on the training set to increase variability and simulate real-world conditions. Augmentation included random horizontal flips, small rotations, translations, zooms, and contrast adjustments. These transformations generate new versions of the training images at runtime, simulating a larger dataset, which is helpful to reduce overfitting and improve robustness.

Architecture

The architecture of Model B was designed as an intermediate CNN, more complex than Model A but still lightweight enough to train efficiently. It is structured around three convolutional blocks, each consisting of:

- **Convolutional layers** with increasing filter counts (32, 64, 128) and kernel size 3×3. These layers learn progressively more complex spatial features: from basic edges and textures in the first block to more abstract shapes in the deeper ones.
- **Batch Normalization** after each convolution, ensuring stable activations by normalizing feature maps, which improves convergence speed and prevents exploding/vanishing gradients.
- **ReLU** activation to introduce non-linearity and allow the network to learn richer feature representations.
- **Max pooling layer** (2×2) to reduce spatial resolution, retaining the most salient features while progressively lowering computational cost.

After the three convolutional blocks, the learned feature maps are flattened into a vector and passed to the fully connected classifier:

- A **Dropout layer** (50%) to randomly deactivate half of the neurons, reducing overfitting.
- A **Dense layer** with 128 neurons and ReLU activation, acting as the hidden fully connected layer that combines extracted features into higher-level patterns.
- Another **Dropout layer** (30%) for regularization.
- The output **Dense layer**, with as many units as the number of classes (3) and a softmax activation, producing a probability distribution over rock, paper, and scissors.

This architecture balances depth and regularization: the convolutional stack extracts robust spatial features, while dropout layers and batch normalization improve generalization. The final softmax output provides interpretable class probabilities for prediction.

Evaluation

The evaluation of Model B was performed using the **stratified** 70/15/15 train/validation/test split. The training history showed higher capacity compared to Model A, with accuracy and loss curves reflecting the benefit of the deeper architecture and standardized preprocessing. Validation performance remained close to training performance, although some signs of mild overfitting appeared in the later epochs.

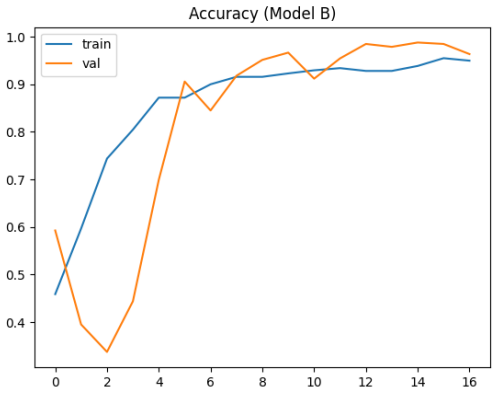

Fig d: Accuracy plot

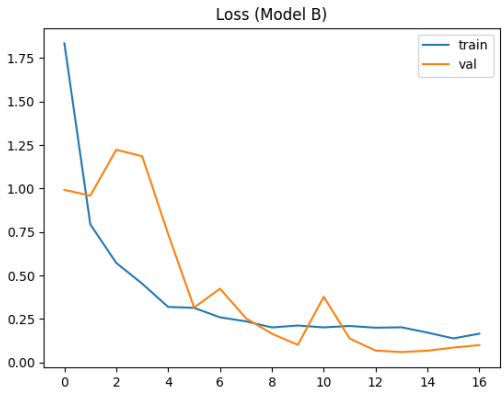

Fig e: Loss plot

On the test set, Model B achieved a higher accuracy than Model A, confirming that the additional convolutional blocks, batch normalization, and dropout improved its generalization ability. The confusion matrix demonstrated balanced classes compared to

the mode A and strong classification across all three classes, though some residual errors occurred, primarily between gestures that share similar outlines.

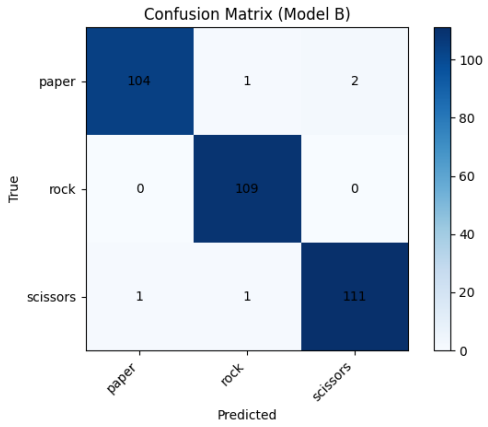

Fig f: Confusion matrix

The classification report provided precision, recall, and F1-scores for each gesture class, indicating great results for each class. Furthermore, incorrectly classified samples were stored in dedicated error folders, which enabled a qualitative inspection of the mistakes. This analysis revealed that errors were often linked to ambiguous or noisy samples rather than to systematic weaknesses of the model.

Model C

Preprocessing

For Model C, preprocessing was designed to support a deeper and **more complex** CNN scheme. All images were resized to 224×224 pixels.

Unlike Models A and B, Model C did not rely on manual normalization. Instead, a custom preprocessing layer was implemented to encapsulate **EfficientNet's** built-in normalization function (preprocess_input). This step scales pixel values into the distribution required by the pretrained backbone, improving stability and compatibility with the network's learned weights.

To improve robustness and encourage generalization, Model C applied stronger data augmentation compared to the previous models. Augmentations included:

- Random **resizing** and **cropping**, ensuring the model learned to recognize gestures from slightly shifted perspectives.
- Random **horizontal flips**, accounting for mirrored gestures.

- Random **rotations**, **translations**, **zoom**, and **contrast changes**, simulating realistic variability in gesture orientation, position, and lighting.

Augmentation was applied **only during training**, ensuring that validation and test sets remained clean and representative of real-world data. This strategy allowed Model C to learn a more diverse feature space without introducing artificial noise in the evaluation phases.

Architecture

Model C employed a transfer learning approach by leveraging EfficientNetB0, a well-established convolutional neural network architecture pretrained on the ImageNet dataset. Instead of building the feature extractor from scratch, the model reused the convolutional backbone to exploit its ability to capture rich visual patterns.

The architecture followed a two-phase training strategy:

- **Phase 1** (head training): the EfficientNet backbone was kept frozen, and only a custom classification head was trained. This ensured that the network learned task-specific features without disrupting the pretrained weights too early.
- **Phase 2** (fine-tuning): the last 50 layers of the backbone were unfrozen and trained with a lower learning rate. This allowed the network to gradually adapt the pretrained filters to the rock–paper–scissors dataset, improving accuracy while preventing overfitting.

The full pipeline of layers can be summarized as follows:

- **Input layer** (224×224×3): receives the resized RGB images.
- Custom **EfficientNet** preprocessing layer: scales and normalizes pixel values using EfficientNet's expected input distribution.
- **EfficientNetB0** backbone (include_top=False): provides a deep feature extractor built from multiple inverted residual blocks with depthwise separable convolutions and squeeze-and-excitation layers.
- **Global average pooling**: compresses the spatial feature maps into a single vector, summarizing the learned information across the image.
- **Dropout layer** (rate = 0.5): prevents overfitting by randomly deactivating half of the neurons during training.

- **Dense output layer** with Softmax activation: produces class probabilities across the three categories (paper, rock, scissors).

This architecture is significantly more complex and powerful than Models A and B. While Model A used a minimal CNN and Model B extended it with multiple convolutional blocks and batch normalization, Model C incorporated a pretrained network designed for large-scale image recognition. As a result, Model C achieved higher accuracy and better generalization, especially when applied to real-time gesture recognition through the webcam interface.

Evaluation

Model C was assessed after the two-phase training process: **first training** the dense classification head while keeping the EfficientNetB0 backbone frozen, and then fine-tuning the last convolutional blocks. The training and validation curves highlighted the effectiveness of this staged approach: in phase one, the model quickly learned to classify gestures with good accuracy, while phase two brought a clear improvement in generalization without excessive overfitting.

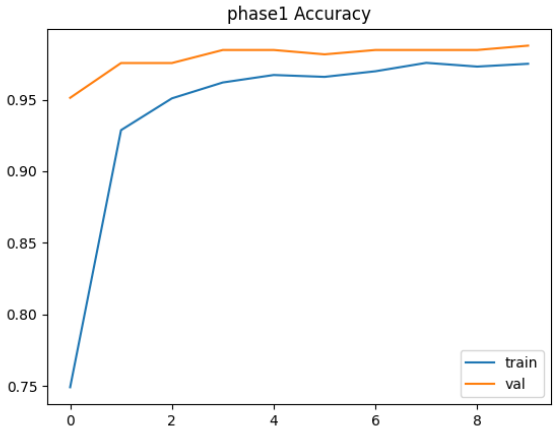

Fig j: Phase 1 accuracy

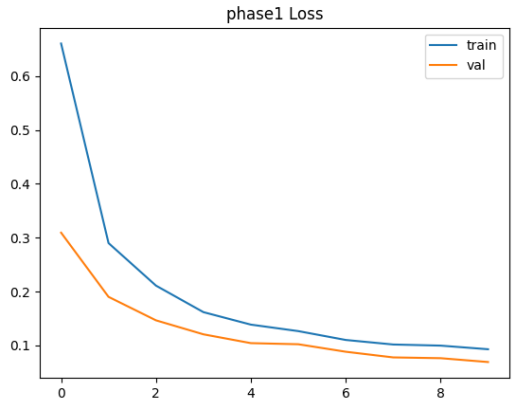

Fig i: Phase 1 loss

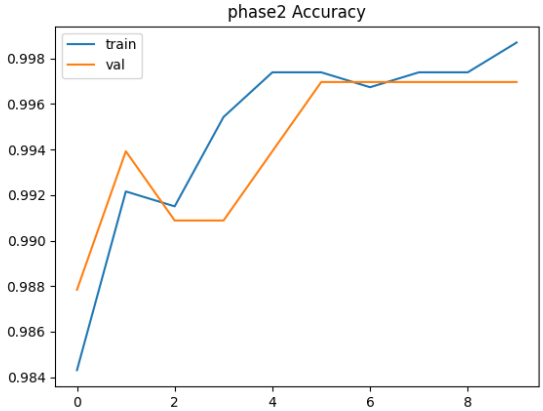

Fig g: Phase 2 accuracy

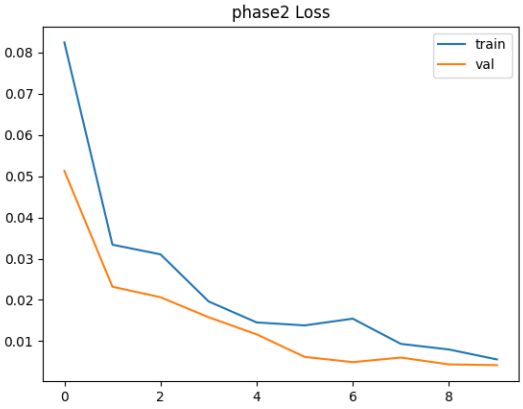

Fig h: Phase 2 loss

On the test set, Model C reached the highest accuracy among all three models, benefiting from the transfer learning setup with EfficientNetB0. The confusion matrix showed zero misclassifications, with 100% of gestures recognized correctly across all categories.

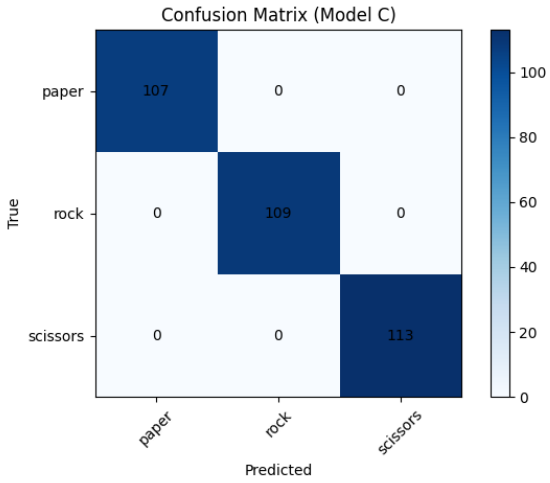

Fig k: Confusion matrix

The classification report confirmed strong precision, recall, and F1-scores for each gesture class, with a particularly consistent performance across the dataset. Overall, Model C demonstrated superior robustness and generalization, making it the most reliable model for real-world applications such as real-time prediction.

Real time prediction

Model C was integrated into a **webcam-based system** to evaluate its performance in real-world conditions. The program automatically detects the presence of a hand in the camera feed and, once stable, captures a sequence of 10 frames that are then analyzed to classify the gesture. The system performs very well when the input consists solely of the hand gesture against a neutral background. However, its performance degrades in the presence of noise, such as when the user's face or body is also visible in the background.

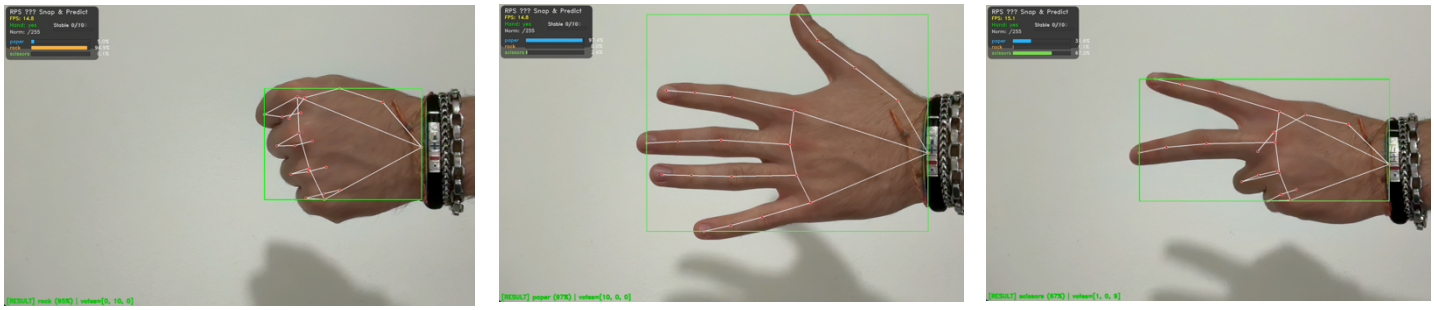

Fig l: Rock-paper-scissor in real time

Conclusions

This project explored the design, implementation, and evaluation of convolutional neural networks for classifying hand gestures from the Rock–Paper–Scissors game.
Three different models of increasing complexity were developed and compared:

- **Model A** provided a lightweight but limited solution, useful mainly for understanding the pipeline.
- **Model B** introduced normalization and deeper convolutional blocks, achieving more stable performance at the cost of slightly higher complexity.
- **Model C**, by fine-tuning a pre-trained backbone, achieved the best overall accuracy and demonstrated strong generalization, but also required a bit greater computational resources.

Beyond offline evaluation, the integration of Model C into a real-time webcam system highlighted both the **strengths** and **limitations** of the models in practical scenarios. The system worked reliably under controlled conditions but faced challenges with background noise and non-ideal lighting, pointing to possible.
In conclusion, this project demonstrates that even with lightweight architectures, it is possible to build functional gesture recognition systems. However, the trade-off between model simplicity and real-world robustness is evident.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.