Rock, Paper, Scissors: CNN for Image Classification

Greta Zehnder

February 12, 2026

Abstract

This project aims to investigate the application of Convolutional Neural Networks (CNNs) to the task of image classification using a Rock-Paper-Scissors (RPS) dataset, with the objective of designing, training, and evaluating multiple deep learning models.

The experimental pipeline is composed of dataset exploration, data preprocessing (which includes train/validation/test splitting, input normalization, and data augmentation), followed by the development of three CNN models (ordered by increasing complexity), and their supervised training and performance evaluation. Finally, a generalization part is carried out to highlight the effectiveness of using CNNs for image classification tasks.

The entire study was carried out in accordance with the official TensorFlow/Keras API documentation.

Declaration

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.

Contents

1 Introduction

Convolutional Neural Networks are widely adopted in image classification tasks because they provide a flexible framework for learning visual patterns directly from image data while supporting different architectural and training choices.

In this context, the development of a CNN can be seen as an exploratory process, in which progressively more complex modeling decisions are introduced and evaluated, moving from simple baseline architectures to manually tuned configurations and, eventually, to more systematic hyperparameter search strategies. Exploring these choices is therefore not only a means of improving performance, but also a way to better understand the sensitivity of a model to different design decisions and training conditions, and to observe how changes in architecture and optimization affect learning dynamics in practice.

This exploratory process is computationally expensive, as training and evaluating multiple configurations requires significant time and resources, which naturally constrains the scope of experimentation and motivates a controlled and incremental evaluation under realistic computational limitations.

2 Data exploration and preprocessing

2.1 Exploratory Data Analysis

The dataset [3] used in this project consists of RGB images representing hand gestures corresponding to the three classes of the Rock-Paper-Scissors game, namely rock, paper, and scissors, and includes a total of 2188 images organized into class-specific subdirectories. The distribution of images across classes is relatively balanced, with 726 images belonging to the rock class, 712 to paper, and 750 to scissors, a property that is particularly relevant for supervised classification tasks, as it allows model performance across classes to be analyzed without strong bias effects.

All images are stored in PNG format and were acquired under controlled conditions, with a uniform green background and consistent lighting and white balance, which limit environmental variability while still preserving meaningful differences in hand shape, orientation, and gesture configuration. An analysis of both absolute class frequencies and relative class proportions confirms the near-uniform representation of the three categories, while a visual inspection of randomly sampled images from each class highlights the presence of intra-class variability related to hand positioning and finger articulation.

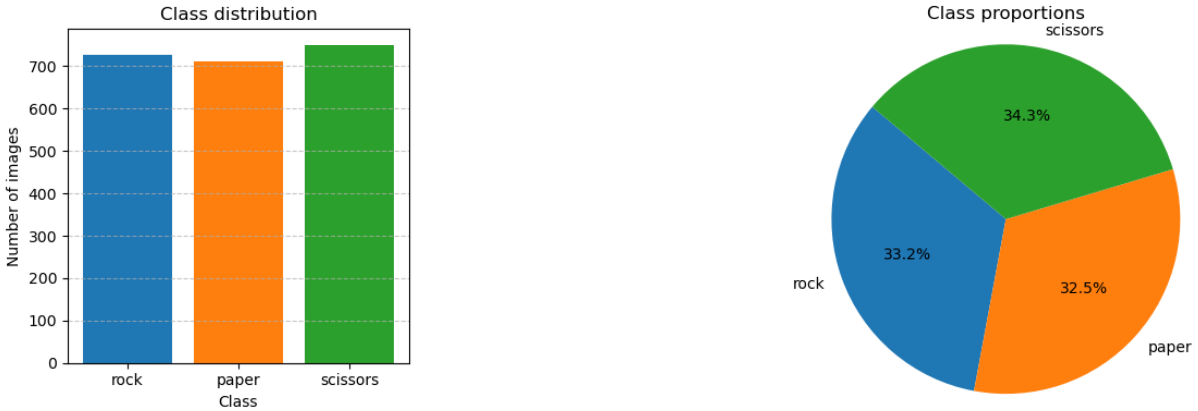

Figure 1: Class distribution (left) and class proportions (right) of the Rock-Paper-Scissors dataset.

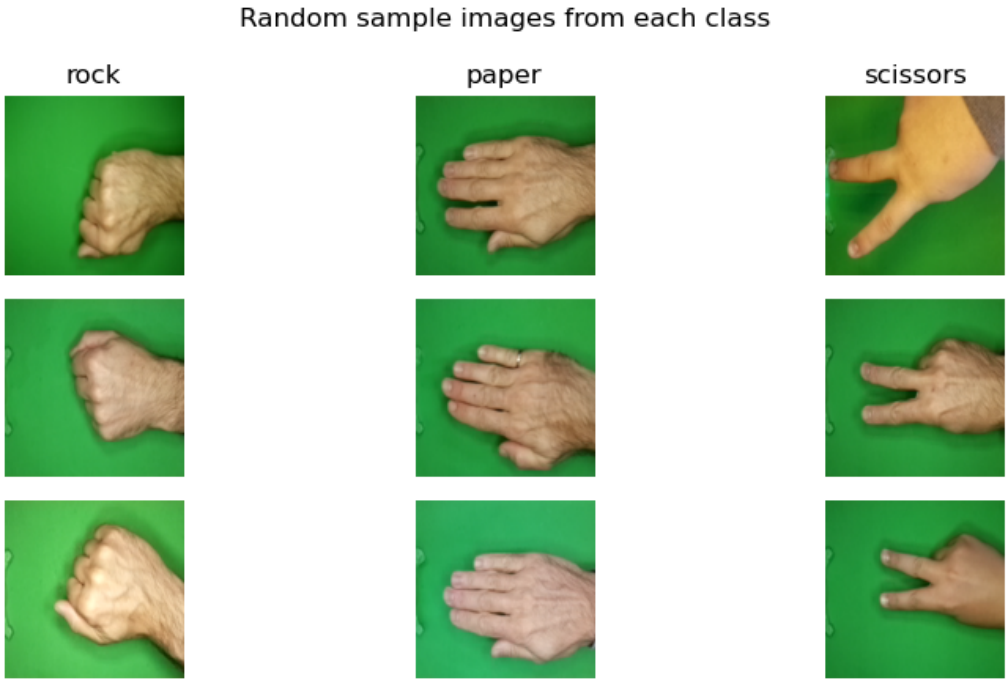

Figure 2: Randomly selected sample images from each class of the Rock-Paper-Scissors dataset.

2.2 Preprocessing

2.2.1 Train, validation and test splitting

The dataset was split into training, validation, and test subsets before model training to enable both hyperparameter tuning and an unbiased final evaluation, with the splitting procedure applied independently to each class using a fixed random seed for reproducibility. In particular, 15% of the images were assigned to the validation set, 15% to the test set, and the remaining samples were used for training, preserving the class distribution across all subsets. The resulting data were then organized into a structured directory hierarchy with separate folders for each split and class, to ensure a clear separation between training and evaluation data.

2.2.2 Normalization

Image normalization was integrated into the data input pipeline by rescaling pixel values from the original $[0, 255]$ range to $[0, 1]$ using the `Rescaling(1./255)` layer provided by the TensorFlow/Keras API [8], so that all images were fed to the network on the same numerical scale. This transformation was applied consistently to the training, validation, and test sets, ensuring that the model received comparable inputs during both training and evaluation and helping maintain stable optimization.

The datasets were created using the `image_dataset_from_directory` utility [9, 2], and normalization was implemented through a mapping operation within the `tf.data` pipeline, which allowed parallel execution and improved data loading efficiency [7]. In addition, caching and prefetching were used to reduce input latency and keep the data flow efficient during both training and testing [6].

2.2.3 Data augmentation

Data augmentation was introduced to improve model robustness and reduce overfitting by exposing the network (during training) to slightly modified versions of the input images while preserving the original class label. In this project, augmentation was implemented using Keras

image preprocessing layers [1], following the workflow recommended in the official TensorFlow documentation [4].

A dedicated augmentation module was defined as a small preprocessing pipeline including random horizontal flipping, small random rotations, and random zoom operations, which are well suited for hand-gesture images as they reflect realistic variations in pose and framing (such as small changes in orientation or distance from the camera) without altering the meaning of the gesture. These transformations were applied only to the training set, whereas the validation and test sets were kept unchanged, so that the evaluation metrics reflect performance on the original data distribution.

The augmentation layers were integrated directly into the model architecture, meaning that they are active during training but automatically disabled during inference, ensuring consistent behavior at deployment while still increasing data variability during learning [5].

References

[1] Keras Team. Image augmentation layers. `https://keras.io/api/layers/preprocessing_layers/image_augmentation/`, 2024.

[2] Keras Team. Keras image data loading api. `https://keras.io/api/data_loading/image/`, 2024.

[3] Laurence Moroney. Rock paper scissors dataset. `https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors`, 2019.

[4] TensorFlow Developers. Data augmentation for images. `https://www.tensorflow.org/tutorials/images/data_augmentation`, 2024.

[5] TensorFlow Developers. Making preprocessing layers part of the model. `https://www.tensorflow.org/tutorials/images/data_augmentation#option_1_make_the_preprocessing_layers_part_of_your_model`, 2024.

[6] TensorFlow Developers. Optimizing data input pipeline: caching and prefetching. `https://www.tensorflow.org/guide/data_performance`, 2024.

[7] TensorFlow Developers. Optimizing data input pipeline: parallel mapping. `https://www.tensorflow.org/guide/data_performance#parallel_mapping`, 2024.

[8] TensorFlow Developers. Rescaling layer. `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Rescaling`, 2024.

[9] TensorFlow Developers. tf.keras.preprocessing.image_dataset_from_directory. `https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image_dataset_from_directory`, 2024.