Image Classification: Rock–Paper–Scissors

Isabella Cadisco

October 20, 2025

1 Introduction

This project explores multi-class image classification for the game Rock–Paper–Scissors using Convolutional Neural Networks (CNNs). The goal is to automatically recognize hand gestures as rock, paper, or scissors.

CNNs are particularly effective for this kind of task, as they can learn meaningful visual patterns directly from pixel data. The work follows a typical supervised learning workflow, including data exploration, preprocessing, model design, training, and evaluation.

Three CNN architectures with different levels of complexity were developed and compared. All experiments were carried out in PyTorch with GPU acceleration, using the publicly available Rock–Paper–Scissors dataset from Kaggle.

2 Data Exploration and Preprocessing

The dataset employed in this project is the Rock–Paper–Scissors dataset, publicly released on Kaggle. It consists of 2,188 RGB images of human hand gestures corresponding to the three classes: rock, paper, and scissors. Figure 1 shows an example for each class. All the images are in PNG file format, with dimensions 300x200 pixels.

The images are distributed into classes as follows:

- Paper: 712 images
- Rock: 726 images
- Scissors: 750 images

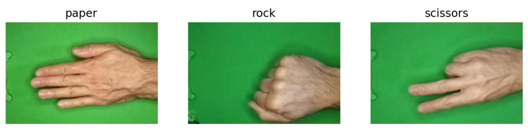


Figure 1: Image examples for each class

Figure 2 clearly shows that the images are balanced across the classes, which makes it an easier task to train a robust classification model.

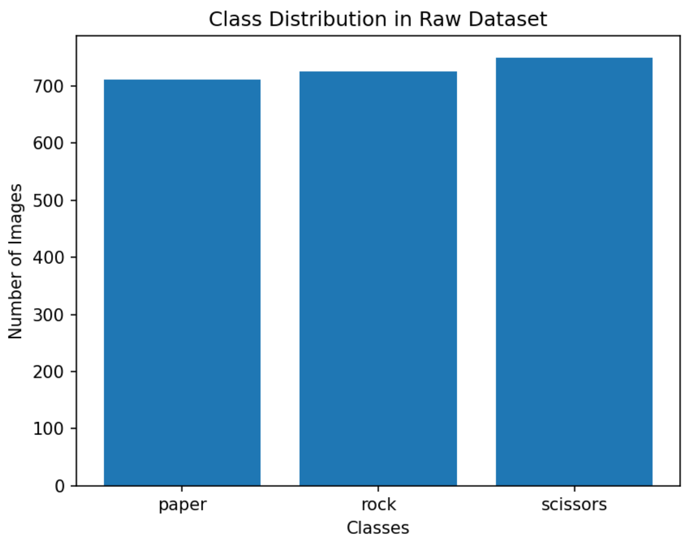


Figure 2: Raw dataset classes distribution

2.1 Data splitting process

The dataset splitting process leveraged the inherent structure of the raw data, which consisted of a separate folder for each class containing all corresponding samples. This organization facilitated the implementation of a balanced partition into training, validation, and test sets. To ensure reproducibility, the splitting procedure makes use of a fixed random seed, with the value of 42.

The resulting splits are described in Table 1 and can be visually inspected in Figure 3.

Table 1: Distribution of images across training, validation, and test sets.

Class	**Train**	**Val**	**Test**	**Total**	**% Train**	**% Val**	**% Test**
Paper	498	107	107	712	69.9%	15.0%	15.0%
Rock	508	109	109	726	70.0%	15.0%	15.0%
Scissors	525	112	113	750	70.0%	14.9%	15.1%
Total	**1531**	**328**	**329**	**2188**	**70.0%**	**15.0%**	**15.0%**

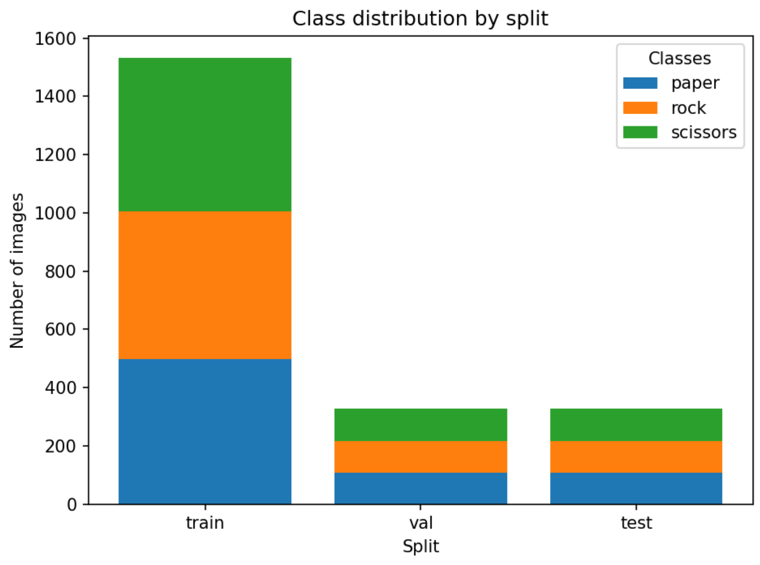


Figure 3: Classes distribution in training, validation and test set

2.2 Image Resizing and Normalization

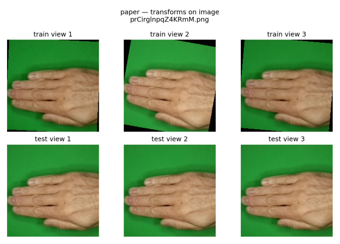
All images were preprocessed through a sequence of transformations implemented using the `torchvision.transforms` module. Each input image was resized such that its shorter side matched the target dimension (set in the configuration file to 192), followed by a center crop operation to obtain square images of fixed size. The resizing used bicubic interpolation with antialiasing enabled, which preserves smooth transitions in color and texture while reducing aliasing artifacts—an important factor when downscaling photographic images, such as hand gestures.

2.3 Data Augmentation Techniques

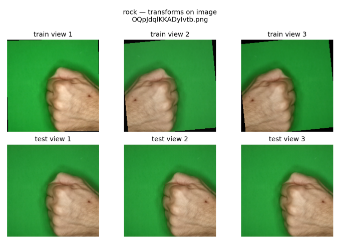
To improve the model's generalization and reduce overfitting, data augmentation was applied dynamically to the training set during data loading. The data loading process was carried out with a fixed random seed, set to 42, to ensure reproducibility. The augmentation pipeline consisted of the following transformations:

- **Random horizontal flipping ($p = 0.5$):** simulates mirrored gestures, effectively doubling the variety of hand orientations seen during training.
- **Random rotation ($\pm 10°$):** introduces small geometric perturbations to account for variations in camera angle or hand alignment.
- **Color jittering (brightness $\pm 10\%$, contrast $\pm 10\%$):** slightly alters illumination conditions to make the model more robust to lighting differences between samples.

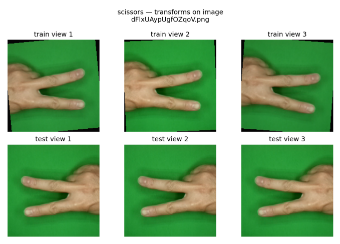
All transformations were chosen to preserve the semantic content of the image—i.e., the gesture class remains identifiable—while introducing controlled variability that enhances the network's ability to generalize to unseen examples. The validation and test datasets, by contrast, were processed with only deterministic transformations (resizing, center cropping, and normalization), ensuring that performance evaluation reflected the model's ability to classify unaltered, real-world images. Figure 4 shows examples for each class of the different treatment for training set and validation, test set.



(a) Transformations for Paper class



(b) Transformations for Rock class



(c) Transformations for Scissor class

Figure 4: Examples of data augmentations applied to each gesture class.

3 Model Design Strategy

3.1 CNN Architectures

Three convolutional neural network (CNN) architectures were implemented to investigate how model capacity and regularization (e.g. Batch Normalization for LargeCNN) influence classification accuracy: `SmallCNN`, `MediumCNN`, and `LargeCNN`. All models follow a modular design composed of three main components: (i) a convolutional feature extractor, (ii) a global adaptive average pooling layer, and

(iii) a fully connected classifier. Each network processes RGB images of dimension $3 \times 192 \times 192$ and outputs a probability distribution across the three gesture classes: *rock*, *paper*, and *scissors*. All architectures use ReLU activations, dropout with rate $p = 0.3$, and are trained using the cross-entropy loss function.

SmallCNN (baseline architecture).

- **Input:** $3 \times 192 \times 192$
- **Feature extractor:**
  - Conv2d(3, 16, kernel_size=3), ReLU
  - MaxPool2d(kernel_size=2)
  - Conv2d(16, 32, kernel_size=3), ReLU
  - MaxPool2d(kernel_size=2)
- **Global pooling:** AdaptiveAvgPool2d(output_size=1)
- **Classifier:**
  - Flatten
  - Dropout($p = 0.3$)
  - Linear(32, 64), ReLU
  - Linear(64, 3)

This lightweight model provides the baseline reference, emphasizing computational efficiency and low variance.

MediumCNN (increased depth and capacity).

- **Input:** $3 \times 192 \times 192$
- **Feature extractor:**
  - Conv2d(3, 32, kernel_size=3), ReLU
  - Conv2d(32, 32, kernel_size=3), ReLU
  - MaxPool2d(kernel_size=2)
  - Conv2d(32, 64, kernel_size=3), ReLU
  - Conv2d(64, 64, kernel_size=3), ReLU
  - MaxPool2d(kernel_size=2)

  - Conv2d(64, 128, kernel_size=3), ReLU
- **Global pooling:** AdaptiveAvgPool2d(output_size=1)
- **Classifier:**
  - Flatten
  - Dropout($p = 0.3$)
  - Linear(128, 128), ReLU
  - Linear(128, 3)

This model increases the number of convolutional layers from 2 `SmallCNN` to 5, enabling a richer feature hierarchy capable of capturing finer textural and shape variations.

LargeCNN (deep regularized architecture).

- **Input:** $3 \times 192 \times 192$
- **Feature extractor:**
  - Conv2d(3, 32, kernel_size=3, padding=1), ReLU, BatchNorm2d(32)
  - Conv2d(32, 32, kernel_size=3, padding=1), ReLU
  - MaxPool2d(kernel_size=2)
  - Conv2d(32, 64, kernel_size=3, padding=1), ReLU, BatchNorm2d(64)
  - Conv2d(64, 64, kernel_size=3, padding=1), ReLU
  - MaxPool2d(kernel_size=2)
  - Conv2d(64, 128, kernel_size=3, padding=1), ReLU, BatchNorm2d(128)
  - MaxPool2d(kernel_size=2)
- **Global pooling:** AdaptiveAvgPool2d(output_size=1)
- **Classifier:**
  - Flatten
  - Dropout($p = 0.3$)
  - Linear(128, 128), ReLU
  - Linear(128, 3)

The addition of batch normalization layers stabilizes gradient propagation and accelerates convergence.

3.2 Training

3.2.1 Training Setup

Activation, and loss.

- **Activation function:** Rectified Linear Unit (`ReLU`) is applied after each convolutional and fully connected layer (except the output). ReLU introduces non-linearity, accelerates convergence, and helps mitigate the vanishing gradient problem.
- **Loss:** `CrossEntropyLoss`, applied to raw logits.

Optimization.

- **Optimizer:** `Adam` on all trainable parameters.
- **Learning rate:** base value $\eta = 10^{-3}$ (constant within each run); values are later explored.
- **Schedulers:** in this experiment, no learning-rate scheduler is employed;

Hardware and software environment.

- **Device:** NVIDIA GeForce GTX 1660 (6 GB), CUDA 12.1 support.
- **Framework:** PyTorch 2.5.1+cu121; Python 3.12.
- **Runtime:** GPU-accelerated local execution; experiments are reproducible on T4-class GPUs with comparable wall-clock time.

3.2.2 Baseline Training

Baseline training experiment. As an initial baseline experiment, the three architectures (`SmallCNN`, `MediumCNN`, and `LargeCNN`) were trained under identical hyperparameter configurations to enable a direct qualitative comparison of learning dynamics. Each model was trained for 10 epochs using the Adam optimizer with a learning rate of 10^{-3}, batch size $B = 64$, and dropout rate $p = 0.3$. Although this approach is not entirely rigorous—since different architectures may require different learning rates or regularization strengths to achieve optimal convergence—it provides an informative first reference for relative training speed and model capacity.

All experiments were executed sequentially on the same GPU (NVIDIA GeForce GTX 1660, 6 GB) to ensure comparable wall-clock measurements. The total training times were approximately:

- `SmallCNN`: $\sim$ 2 minutes
- `MediumCNN`: $\sim$ 2 minutes
- `LargeCNN`: $\sim$ 3 minutes

Training dynamics. Figure 5 shows the training and validation accuracy and loss curves for each architecture. Overall, the results confirm that deeper models achieve higher validation performance in the absence of hyperparameter optimization.

- **SmallCNN:** The model exhibits slow convergence and a relatively limited representational capacity. Both accuracy and loss curves suggest mild underfitting: the training accuracy remains below 0.6.
- **MediumCNN:** The addition of extra convolutional layers substantially improves learning performance, but the dynamics are still not optimal.
- **LargeCNN:** The deeper regularized model reaches high accuracy within the first few epochs. The small gap between training and validation curves, combined with consistently low loss values, suggests that overfitting was successfully avoided despite the higher model capacity.

3.2.3 Explorative Comparison of Network Topologies

To perform a first systematic comparison among the three architectures (`SmallCNN`, `MediumCNN`, and `LargeCNN`), an external K-fold cross-validation was carried out. The purpose of this experiment was not rigorous hyperparameter optimization—which would require the use of a *nested* cross-validation procedure—but rather the estimation of the **expected statistical risk**

$$\mathbb{E}_S\big[\ell_D(A(S))\big]$$

for each architecture, where the expectation is taken over random draws of a training set S of fixed size m.

In this context, A denotes the learning algorithm (here, one of the three CNN architectures) and $\ell_D(A(S))$ the generalization error of the corresponding predictor $A(S)$. K-fold cross-validation provides an unbiased estimate of the expected risk of a fixed predictor trained on a sample of size m, by averaging its empirical error

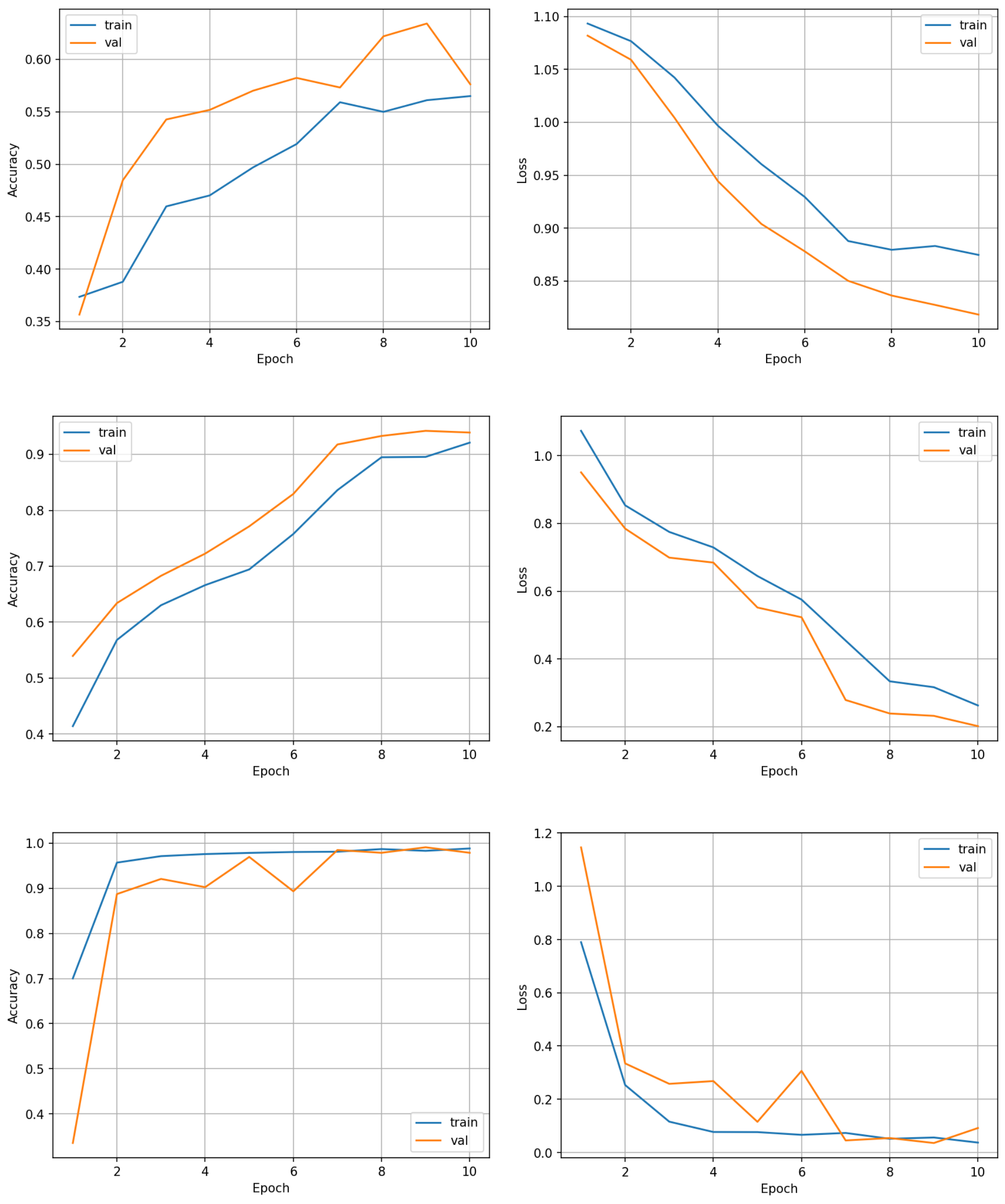


Figure 5: Training and validation accuracy (left) and loss (right) for `SmallCNN` (top), `MediumCNN` (middle), and `LargeCNN` (bottom).

across K folds:

$$\ell_S^{cv}(A) = \frac{1}{K}\sum_{i=1}^{K} \ell_{S_i}\big(A(S_{-i})\big)$$

where S_i and S_{-i} denote the test and training parts of the i-th fold, respectively.

Experimental setup. Each model was trained using the same configuration as in the baseline experiment: 10 epochs, learning rate $\eta = 10^{-3}$, batch size $B = 64$, and dropout rate $p = 0.3$. The cross-validation was performed with $K = 3$ folds on the combined training and validation sets, using stratified splits to preserve class balance across folds.

Results. The mean and standard deviation of the validation accuracy across folds are reported in Table 2. Each value represents an empirical estimate of the expected accuracy of a model trained on a random sample of the same size.

Table 2: Mean and standard deviation of validation accuracy across folds for each architecture.

Model	**Mean Validation Accuracy**	**Std. Dev.**	**Fold Accuracies**
`SmallCNN`	0.632	0.013	[0.645, 0.615, 0.635]
`MediumCNN`	0.791	0.065	[0.806, 0.705, 0.861]
`LargeCNN`	0.974	0.005	[0.973, 0.968, 0.981]

The results show a clear performance improvement as network complexity increases. `SmallCNN` displays limited learning capacity and higher bias, with low variance across folds, whereas `MediumCNN` achieves a substantial gain in accuracy at the cost of increased variance, suggesting a more flexible but less stable fit. `LargeCNN` achieves the highest and most consistent accuracy across folds, with a mean of 97.4% and a small standard deviation (0.5%), indicating strong generalization and robustness to variations in the training set.

From a statistical learning perspective, this experiment can be interpreted as an empirical estimation of the **expected statistical risk** associated with each architecture. While it does not correspond to hyperparameter tuning in the strict sense (which would require nested cross-validation, as described before), it provides an informative approximation of how each model behaves when trained on a typical random sample of fixed size.

3.2.4 Hyperparameter Tuning for Medium CNN

After comparing the three architectures, the `MediumCNN` was selected as the candidate model for hyperparameter tuning. This choice was motivated by its balanced trade-off between predictive performance and computational cost: it achieved substantially better accuracy than `SmallCNN` while being faster to train and less memory-intensive than `LargeCNN`.

Experimental setup. A grid search cross-validation was implemented to explore the effect of learning rate and dropout on validation performance. Specifically, four configurations were tested by varying the learning rate $\eta \in \{5\times10^{-4}, 2\times10^{-3}\}$ and dropout $p \in \{0.1, 0.5\}$, while keeping batch size $B = 64$ and training epochs fixed at 12.

The choice of hyperparameters was guided by empirical considerations. Two learning rates, $\eta \in \{5 \times 10^{-4}, 2 \times 10^{-3}\}$, were selected to investigate the balance between training stability and convergence speed, where the smaller value promotes smoother optimization and the larger one enables faster learning. Similarly, two dropout rates, $p \in \{0.1, 0.5\}$, were tested to assess the trade-off between model regularization and capacity, with the higher dropout providing stronger overfitting prevention at the cost of slower convergence. The batch size ($B = 64$) and number of epochs (12) were held constant to isolate the influence of the learning rate and dropout on validation performance.

Each configuration was evaluated through a stratified 5-fold cross-validation on the combined training and validation sets.

As discussed before, this procedure provides an *optimistic estimate* of the expected generalization risk, since the same folds are used both to select and to evaluate hyperparameters. A statistically sound estimate of the expected risk under hyperparameter optimization would instead require **nested cross-validation**, where an inner CV loop performs hyperparameter selection and an outer loop estimates the risk of the selected predictor. Nevertheless, the present setup offers a computationally efficient and practically informative approximation for exploratory purposes.

Results. Table 3 reports the mean and standard deviation of validation accuracy across folds for each parameter combination.

The cross-validation results indicate that a relatively higher learning rate ($\eta = 2 \times 10^{-3}$) yields better generalization than the lower rate (5×10^{-4}). This suggests that the optimizer benefits from a more aggressive step size under the given normalization and data scale. Furthermore, a mild dropout rate ($p = 0.1$) leads to

Table 3: Grid search cross-validation results for `MediumCNN`. Each value represents the mean validation accuracy across five folds.

Learning Rate	**Dropout**	**Batch Size**	**Epochs**	**Mean Val. Acc.**	**Std. Dev.**
0.002	0.1	64	12	0.959	0.015
0.002	0.5	64	12	0.955	0.017
0.0005	0.1	64	12	0.899	0.064
0.0005	0.5	64	12	0.792	0.095

higher average accuracy and lower variance across folds compared to the stronger regularization ($p = 0.5$), which tends to slow convergence and introduce instability, especially at smaller learning rates.

The best configuration achieves a mean validation accuracy of 95.9% with standard deviation 1.5%, demonstrating both strong generalization and consistent performance across folds. In contrast, configurations with smaller learning rates exhibit larger variance (up to 9.5%), highlighting the sensitivity of convergence speed and final performance to the optimization step size.

From a statistical learning standpoint, these results provide an *approximate* estimate of the expected generalization performance after hyperparameter selection. Although not statistically unbiased (as nested CV would be), this grid search CV effectively identifies a region of stable hyperparameter values and serves as a practical compromise between methodological rigor and computational feasibility.

4 Evaluation and Analysis

4.1 Final Test Evaluation

After selecting the best configuration for `MediumCNN` from the grid search cross-validation ($\eta = 2 \times 10^{-3}$, $B = 64$, $p = 0.1$, 12 epochs), the model was retrained on the full training set (train + validation) and evaluated on the held-out test set, which had not been used at any previous stage of the pipeline.

The test results provide an unbiased estimate of the model's generalization performance on unseen data. Below, precision, recall, and F1-score for each class, along with macro- and weighted-averaged metrics, are specified. The confusion matrix in Figure 6 visualizes the distribution of correct and misclassified samples.

- **Class Paper:**
  - Precision: 1.00

- Recall: 0.94
- F1-score: 0.97
- Support: 107

- **Class Rock:**
  - Precision: 0.98
  - Recall: 1.00
  - F1-score: 0.99
  - Support: 109
- **Class Scissors:**
  - Precision: 0.97
  - Recall: 1.00
  - F1-score: 0.98
  - Support: 113
- **Overall Accuracy:** 0.98
- **Macro Average:**
  - Precision: 0.98
  - Recall: 0.98
  - F1-score: 0.98
  - Support: 329
- **Weighted Average:**
  - Precision: 0.98
  - Recall: 0.98
  - F1-score: 0.98
  - Support: 329

4.2 Analysis of Performance

The final model achieves an overall accuracy of **98.2%**, confirming good generalization on unseen data. All three gesture classes are predicted with high precision

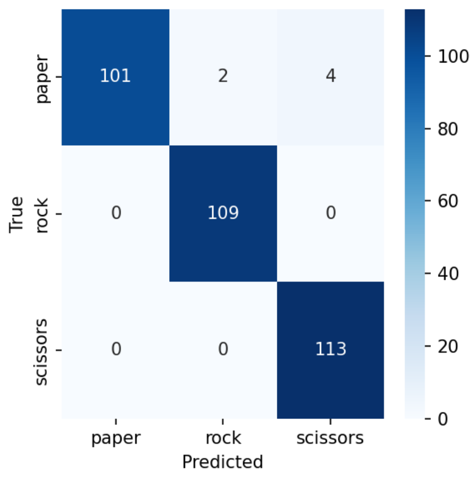


Figure 6: Confusion matrix on the test set for the best `MediumCNN`.

and recall, and the F1-scores are consistently above 0.97. The small difference between macro and weighted averages indicates that the dataset is well balanced and that no class dominates the performance metrics.

The confusion matrix reveals near-perfect classification: the only noticeable misclassifications occur within the *paper* class, where a few samples are predicted as *rock* or *scissors*.

The proposed solution is inspired by aman.ai/primers/pytorch

[1]

[1] 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.