report for the project "Building a CNN for Rock-Paper-Scissors Classification"

Nicholas Fornaroli - matricola 47262A

1 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.

2 Introduction

The main objective of this project is to develop and evaluate three distinct Convolutional Neural Networks (CNNs) for classifying hand gestures corresponding to the Rock-Paper-Scissors game, adhering to sound machine learning practices across all project phases. This report is structured as follows: The first section presents an exploratory data analysis (EDA) and details the image preprocessing techniques applied. The second section introduces the three CNN architectures, including their designs, rationale, hyperparameter selection methods, and evaluation results. The final section assesses the performance of these architectures on a custom-made dataset.

3 data analysis and image pre-processing

3.1 Dataset description

The dataset used for this project is the "Rock-Paper-Scissors" dataset obtained from Kaggle. We have three categories in total in the dataset: paper, scissors and rock, corrisponding to the hand gestures admitted in the game rock paper scissors. The dataset contains 712 images for paper, 750 for scissors, and 726 for rock, totaling 2,188 images, so the class are pretty balanced and so we don't need to employ any techniques to avoid possible biases that can be found in model trained on unbalanced datasets, the balance also allow us to confortbly use the accuracy metric and treat it's value as a meaningfull measure of the model performance. All images are uniformly sized at 200x300 pixels, i decided to resize all the images in order to have smaller images. Initially i opted for the classic 128 x128 size commonly used for classification task but then i opted for a 100x 150 size which have rougly the same size and mantain the aspect ration, which i thought was important in order to reduce the possibility of miscassification.

3.2 Exploratory data analysis and splitting

3.2.1 Splitting

I started by splitting the dataset in a train split, a validation split and a test split in a 70/20/10 ratio. The splitting was done immediatly in order to conduct an exploratory data analysis only on the train set in order to avoid basing eventual image pre-processing techniques on information from the test set that could inflate the performance of the models. the splitting was done ensuring an equal rappresentation of the 3 classes in each split.

3.2.2 Visual inspection and data augmentation

Visual inspection indicates low intra-class variance, with most images presenting the same hand, background, inclination, position, and zoom, suggesting a need for data augmentation to enhance variability and improve model generalization. To address this, at training time, the following data augmentation techniques were applied randomly to the images of the training set to increase the diversity of the dataset: rotation, flipping, zooming and shifting. Additionally, all images were converted to grayscale to reduce computational complexity, simplify further pre-processing and focus the model on geometric hand features, as color information didn't seem essential for this task.

3.2.3 Intensity analysis per class

I wanted to analize the avarage intensity of the images per class in order to see if the were significant differences among them that could bias the models and prevent them from learning to classify the images according to the geometrical pattern of the images, so i created a box plot that diplayed the avarage intenisty of the images for the three classes obtaining the following:

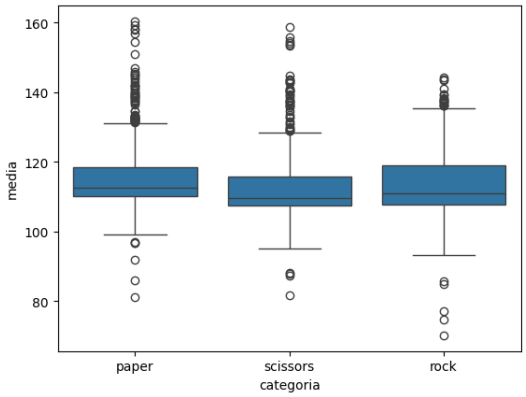


Figure 1: Avarage intensity per class

The boxplot shows that the average image intensity across the three classes is very similar, suggesting no significant brightness bias among categories. As a consequence, brightness normalization was not performed.

3.3 Histogram normalization and contrast enhancement

Leveraging concepts acquired during the Image Processing course, I conducted an analysis of the histograms[1] of the dataset images to assess whether contrast enhancement techniques were required.

Initially, I calculated and plotted the average histogram of each class, obtaining the following:

[1]Given a grayscale image, its histogram is a function that maps each intensity value (0–255) to the number of pixels having that intensity.

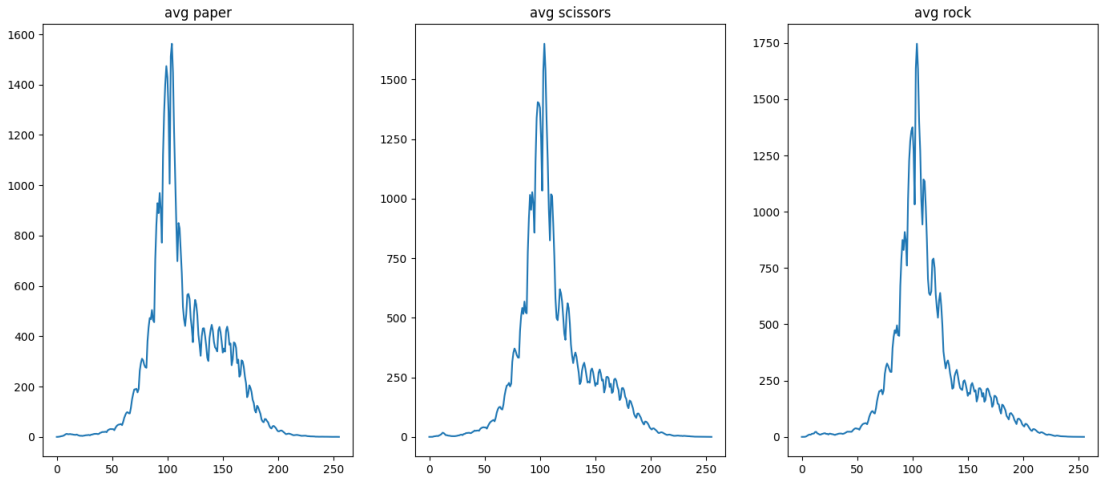


Figure 2: Average intensity histogram per class.

The three average profiles were remarkably similar, with most pixels concentrated in the mid-intensity range. This suggested that a contrast-enhancing technique might help separate the hand from the background, improving feature visibility. The similarity across classes also indicated that the same enhancement procedure could be safely applied to all images, regardless of class.

To further confirm this, I performed an experiment in which I randomly sampled nine images per class, computed their histograms, and plotted them together with the class average histogram. This helped verify whether (i) the histogram of each image was consistent with its class average, and (ii) the overall histogram shapes were similar across classes—justifying the use of a uniform enhancement method. An example of the experiment for the *paper* class is shown below.

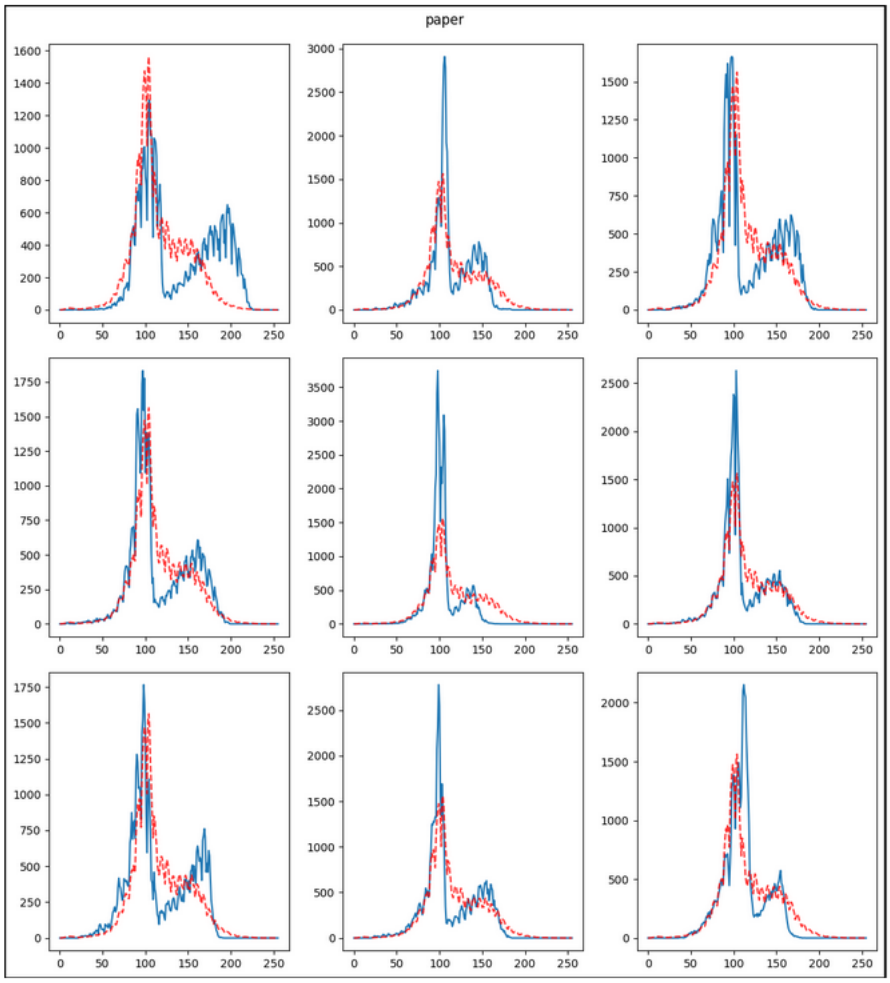
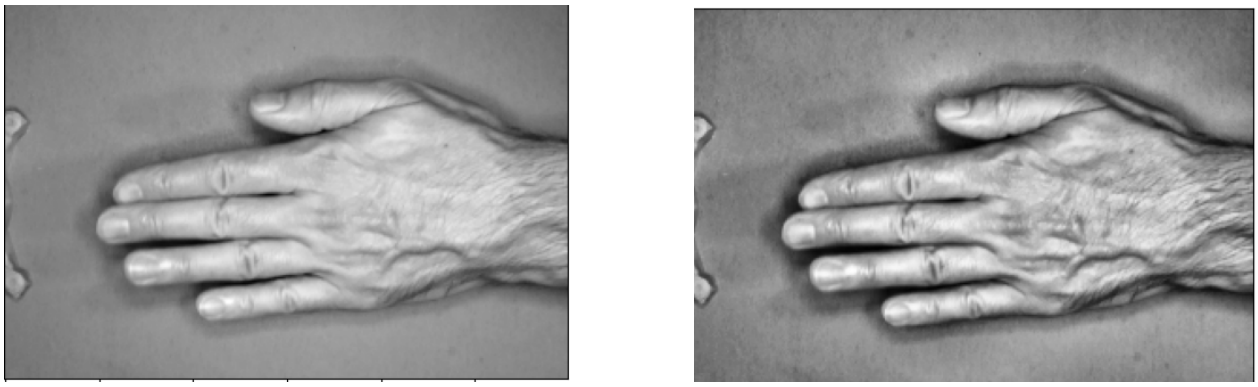


Figure 3: Histograms of nine random images from the *paper* class (blue) with the class average in dashed red.

The results confirmed that the histograms of individual images closely matched their class averages and shared similar profiles across classes. Therefore, I applied the CLAHE [2] (Contrast Limited Adaptive Histogram Equalization) method with the default values of 8x8 tile size and clipping = 2.0 to all images in the dataset for local histogram normalization.

The resulting images exhibited enhanced contrast, with the hands clearly separated from the background and fine features preserved—confirming the effectiveness of this preprocessing step.

[2]CLAHE divides an image into non-overlapping tiles and applies local histogram equalization to each, limiting amplification to prevent noise enhancement. The processed tiles are then smoothly merged to avoid visible boundaries.



(a) unprocessed image

(b) CLAHE processed image

Figure 4: Comparison between processed and unprocessed image

3.4 Normalization

Finally, I normalized the pixel intensities of all images fed to the models by rescaling them to the $[0, 1]$ range. This step ensures numerical stability during training and facilitates faster convergence of the optimization algorithm.

4 Models definition, training, and evaluation

In this section, I describe the three convolutional models developed for this project. For each model, I discuss the architectural choices, the strategies adopted for hyperparameter selection, and the evaluation results.

4.1 Note on reproducibility

To maximize the reproducibility of my experiments, I attempted to control all possible sources of randomness in the code. In particular, I used the instruction **tensorflow.keras.utils.set_random_seed(812)** at the beginning of each notebook to fix the random seed for the NumPy module, the Python random module, and the Keras backend—ensuring consistent weight initialization and data shuffling across runs.

Furthermore, I enabled deterministic GPU operations by calling **tf.config.experimental.enable_op_determinism()**, which makes CUDA-based computations as reproducible as possible. Finally, I ensured that every data-splitting and generator-shuffling operation used a fixed seed, so that the same training, validation, and test partitions were preserved between runs. Despite this precations, meaningfull variations were observed when the experiment where executed on different machines, probably because of the fact that i used a pc with a cuda capable GPU and a M2 mac, which probably used different backend implementations.

4.2 Metrics, early stopping, and loss

In this subsection, I describe the training and evaluation components that were common to all models.

4.2.1 Metrics

The main metric used to evaluate model performance was **accuracy**, defined as the ratio between correctly classified examples and the total number of examples. This metric is appropriate because the dataset is balanced across the three classes. Therefore, accuracy reliably reflects the model's overall ability to classify hand gestures, rather than being biased toward the most represented class.

In addition to accuracy, I also computed class-specific metrics: **precision**[3], **recall**[4], and the **F1-score**[5].

[3]The proportion of correct predictions among all samples predicted as belonging to a given class.

[4]The proportion of samples of a given class that were correctly predicted as such.

[5]The harmonic mean between precision and recall.

These additional metrics allowed for a more detailed evaluation of the model's per-class performance and helped identify possible weaknesses or class imbalances in prediction.

4.2.2 Early stopping

After defining the hyperparameters for each model, I employed the **early stopping** technique during training. This means that training was interrupted when the validation loss stopped improving for a fixed number of epochs (patience). Specifically, I set a patience of 10 epochs, meaning that if the validation loss did not decrease for 10 consecutive epochs, training was halted and the model weights corresponding to the lowest validation loss were restored. This approach reduces overfitting and avoids wasting computational resources once the model stops improving.

4.2.3 Loss function

The loss function used for this task was the **categorical cross-entropy**, defined as:

$$L(\hat{y}, y) = -\log(\hat{y}_i)$$

where $\hat{y}$ is the probability distribution predicted by the model (the output of the softmax layer), and y is the true class label. The index i corresponds to the neuron associated with the correct class. Since the final dense layer of the network contains three neurons (one per class) with a softmax activation, the output $\hat{y}$ is a three-dimensional vector representing the predicted probability of each class.

4.2.4 Adam optimizer

All models were trained using the **Adam** optimizer, one of the most widely used optimization algorithms for deep learning. Adam was chosen because it provides fast convergence, low memory footprint and performs well in a wide range of computer vision tasks.

4.2.5 ReLU activation function

Except for the final output layer, all layers used the **ReLU** (Rectified Linear Unit) activation function[6]. ReLU was chosen because it introduces non-linearity while avoiding the vanishing gradient problem that affects traditional activation functions such as the sigmoid or hyperbolic tangent (tanh) and it is considered the de facto standard in CNN architectures.

4.3 First model: simple one-block CNN

As a starting point, I designed an intentionally simple and compact convolutional neural network (CNN) architecture consisting of:

- One convolutional layer with 32 filters of size 3×3 and ReLU activation.
- One max-pooling layer with a 2×2 window to reduce spatial dimensions.
- One fully connected layer with 64 neurons and ReLU activation.
- One output dense layer with 3 neurons (one per class) and a softmax activation to produce a probability distribution over the classes.

The goal of this first setup was to verify that the preprocessing pipeline, data generators, and training procedures were functioning correctly, and to establish a baseline accuracy against which subsequent, more complex models could be compared.

The model was trained for up to 100 epochs using the Adam optimizer and categorical cross-entropy loss. Early stopping with a patience of 10 epochs was employed to prevent overfitting. The figure below shows the training and validation accuracy and loss curves.

[6] $\text{ReLU}(x) = \max(0, x)$

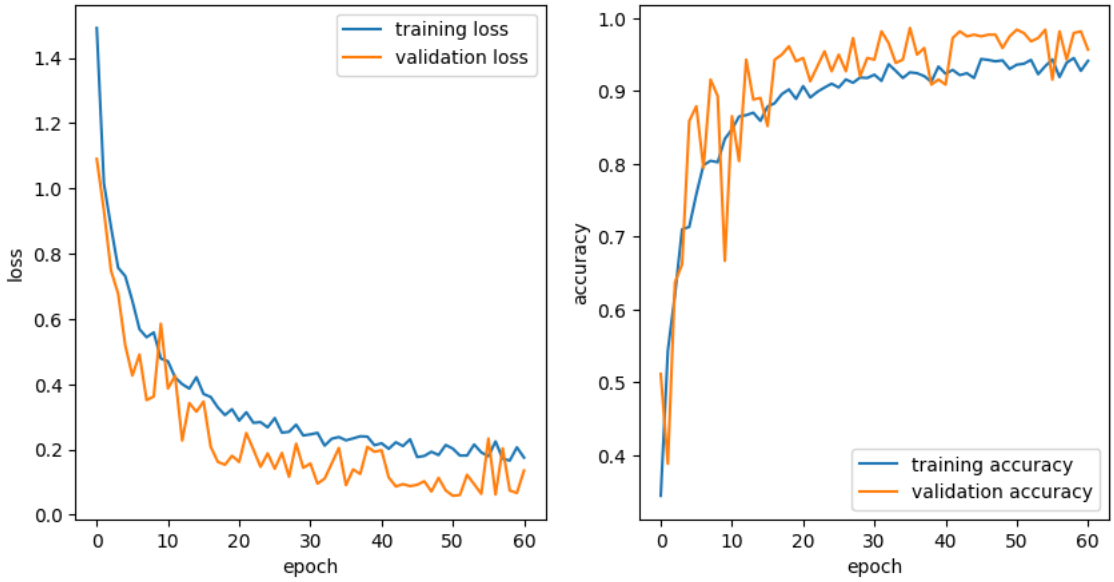


Figure 5: Training and validation accuracy and loss curves for the baseline model.

The training dynamics indicate a rather smooth convergence and good generalization, with no evidence of overfitting: both validation and training accuracy steadily increase and saturate around 98%, while the losses follow a consistent decreasing trend.

On the test set, the model achieved an overall accuracy of **99.54%**. The detailed class-level performance metrics are reported below.

	precision	recall	f1-score	support
paper	0.99	1.00	0.99	71
rock	1.00	0.99	0.99	73
scissors	1.00	1.00	1.00	75

Figure 6: Classification report for the first model on the test set.

Only one misclassification was observed: a rock image incorrectly classified as paper. There was no clear pattern explaining this error.

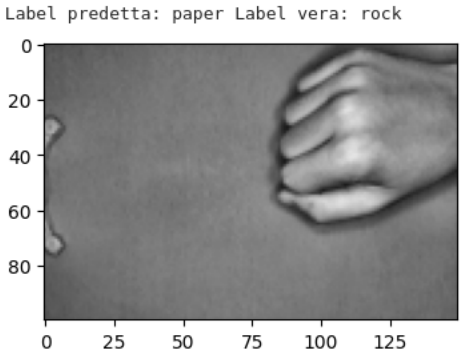


Figure 7: The only error committed by the first model.

Overall, the first model performed remarkably well, demonstrating that the chosen preprocessing steps (grayscale conversion, contrast enhancement, and normalization) were effective. However, the excellent baseline accuracy left little room for measurable improvement with more complex architectures.

4.4 Second model: deeper CNN inspired by LeNet-5

The second model was directly inspired by the first well-known CNN architecture, LeNet-5. It consists of the following layers:

- A convolutional layer with 64 filters of size 3×3 and ReLU activation.
- A max-pooling layer with a 2×2 window.
- A second convolutional layer with 32 filters of size 3×3 and ReLU activation.
- A second max-pooling layer with a 2×2 window.
- A fully connected layer with 128 neurons and ReLU activation.
- A dropout layer with a rate of 0.25.

The inclusion of the dropout layer represents a key modification compared to the original LeNet-5 architecture. During training, dropout randomly deactivates a fraction of neurons in the preceding layer—in this case, each neuron has a 25% probability of being dropped at every iteration. This mechanism prevents the network from relying excessively on specific neurons (a phenomenon known as *co-adaptation*) and encourages different neurons to learn alternative yet useful feature patterns. As a result, dropout helps reduce overfitting and improves the model's generalization ability on unseen data.

4.4.1 Hyperparameter Tuning

For this model, the number of filters in each convolutional layer, the dropout rate, and the initial learning rate were determined through a grid search combined with k-fold cross-validation. The following search space was defined:

Table 1: Hyperparameter Search Space

Hyperparameter	Values
`model__drop_param`	[0.25, 0.5]
`model__learning_rate`	[1e-3, 5e-4, 1e-4]
`model__num_filter_1`	[32, 64]
`model__num_filter_2`	[32, 64]

Every possible combination of these hyperparameter values was evaluated using a 3-fold cross-validation on the training set. The data were split into three equally sized folds, with each class being equally represented. In each run, two folds were used for training and the remaining one for validation. Each configuration was therefore trained and evaluated three times, and the mean validation accuracy across folds was recorded as its performance score. The combination achieving the highest average validation accuracy (`drop_param=0.25`, `learning_rate=5e-4`, `num_filter_1=64`, `num_filter_2=32`) was selected for the final model.

Although the grid was relatively small, this was a deliberate choice to limit computational cost while still exploring representative and commonly used parameter values. For the same reason, each configuration was trained for only 15 epochs. The grid search with cross-validation ensures a systematic evaluation of parameter combinations and provides a robust estimate of generalization performance. Using 3-fold rather than a single validation split allows each sample to serve as validation data at least once, reducing the risk of biased estimates due to a particular data partition.

4.5 Training and Evaluation

After determining the optimal hyperparameters, the model was trained using the Adam optimizer for up to 50 epochs, with early stopping based on validation loss and a batch size of 32. The training and validation curves are shown below.

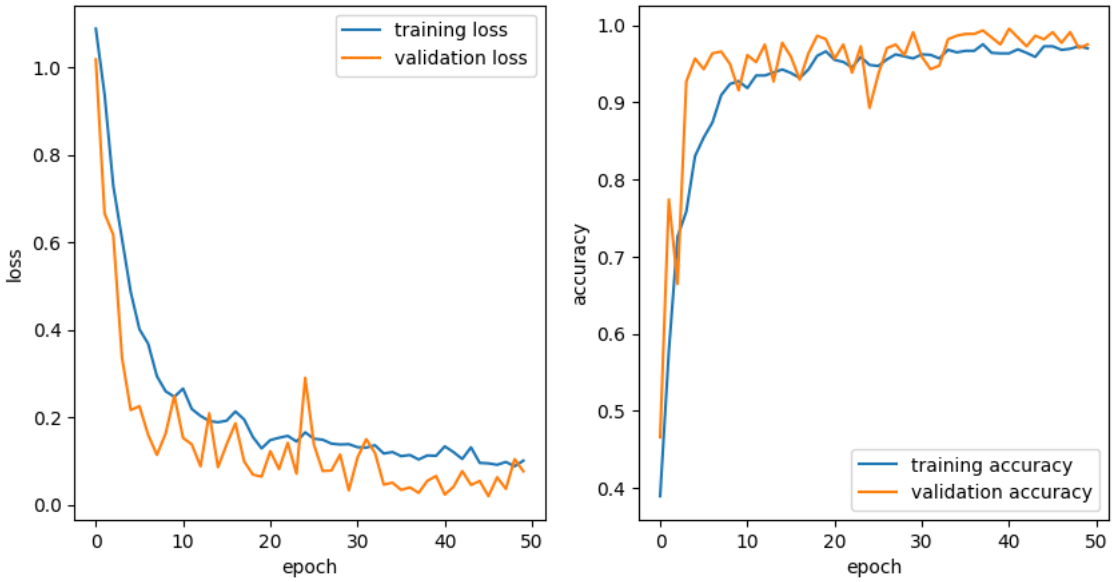


Figure 8: Training and validation accuracy and loss curves for the second model.

Both the training and validation losses show a consistent downward trend, with the model reaching a validation accuracy close to 98% and exhibiting no visible signs of overfitting.

On the test set, the model achieved an overall accuracy of **99.54%**. Class-level metrics are reported in the figure below, showing uniformly high precision, recall, and F1-scores across all classes.

	precision	recall	f1-score	support
paper	0.99	1.00	0.99	71
rock	1.00	0.99	0.99	73
scissors	1.00	1.00	1.00	75

Figure 9: Classification report for the second model on the test set.

As in the first model, only one misclassification occurred: a rock image was classified as paper, the same exact image that was misclassified by the first model.

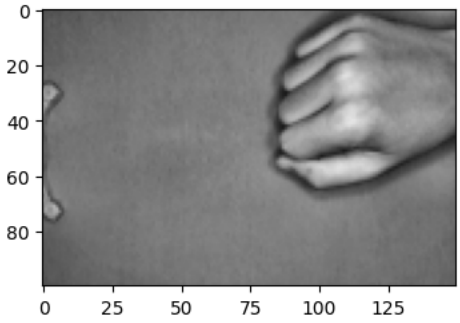


Figure 10: The only error committed by the second model.

Overall, the second model produced results very similar to those of the baseline, confirming that the task's relative simplicity limits the observable benefits of increased model complexity.

4.6 Third model: deep CNN inspired by the VGG architecture

The final and most complex model was inspired by the VGG family of architectures. In this design, each block is composed of two consecutive convolutional layers followed by a single max-pooling layer, and several such blocks are stacked before a final classification head composed of dense layers, culminating in an output layer with n neurons (one per class) and a softmax activation.

The rationale for using two convolutional layers before each pooling operation is that this structure achieves a similar receptive field to that of a single convolution with a larger kernel (e.g., 5×5), but with fewer trainable parameters. Stacking multiple smaller convolutions therefore increases the model's expressive power and allows for the extraction of more complex spatial features while maintaining computational efficiency.

As in the original VGG design, a single dropout layer was retained before the first dense layer to prevent overfitting. However, in this model, the `Flatten` layer—which concatenates all the feature maps into a single long vector—was replaced by a **Global Average Pooling (GAP)** layer.[7] This choice significantly reduces the number of parameters by avoiding a large, fully connected intermediate layer, thereby mitigating overfitting and improving generalization.

Moreover, by averaging across spatial dimensions, the GAP layer encourages the network to detect the presence of features regardless of their position in the image, promoting translation invariance.

Finally, **Batch Normalization** layers were introduced between each convolutional layer and its ReLU activation function with the goals of:

- accelerating the convergence of the training process, and
- making the learning process less sensitive to the initial random weight initialization.

Batch Normalization achieves these benefits by normalizing the activations within each mini-batch, ensuring that the inputs to each layer maintain a stable distribution during training. Although the exact reason why Batch Normalization is effective is still debated, recent hypotheses suggest that it helps by smoothing the loss landscape and stabilizing gradient propagation during optimization.

In particular, the network is composed as follows:

- **Block 1:** two convolutional layers with 64 filters of size 3×3, each followed by batch normalization and ReLU activation, then a max-pooling layer with a 2×2 window.
- **Block 2:** identical to Block 1.
- **Block 3:** two convolutional layers with 256 filters of size 3×3, each followed by batch normalization and ReLU activation, then a max-pooling layer with a 2×2 window.
- **Global Average Pooling (GAP)** layer.
- **Dense layer** with 128 neurons and ReLU activation.
- **Dropout layer** with a dropout rate of 0.5.
- **Output layer** with 3 neurons (one per class) and softmax activation.

4.6.1 Hyperparameter tuning

As with the previous model, several hyperparameters were tuned to optimize performance. Specifically, the search included the number of filters in each convolutional block, the number of neurons in the first dense layer, the dropout rate, and the learning rate.

[7] The Global Average Pooling layer computes the average value of each feature map, effectively summarizing the spatial information into a single representative value per map.

Table 2: Hyperparameter search space for the third CNN model.

Hyperparameter	Search Space	# Choices
`conv_1`	[32, 64]	2
`conv_2`	[64, 128]	2
`conv_3`	[128, 256]	2
`num_units`	[64, 128, 256]	3
`dropout`	[0.25, 0.50]	2
`learning_rate`	[1e-4, 1e-2] (log scale)	Continuous

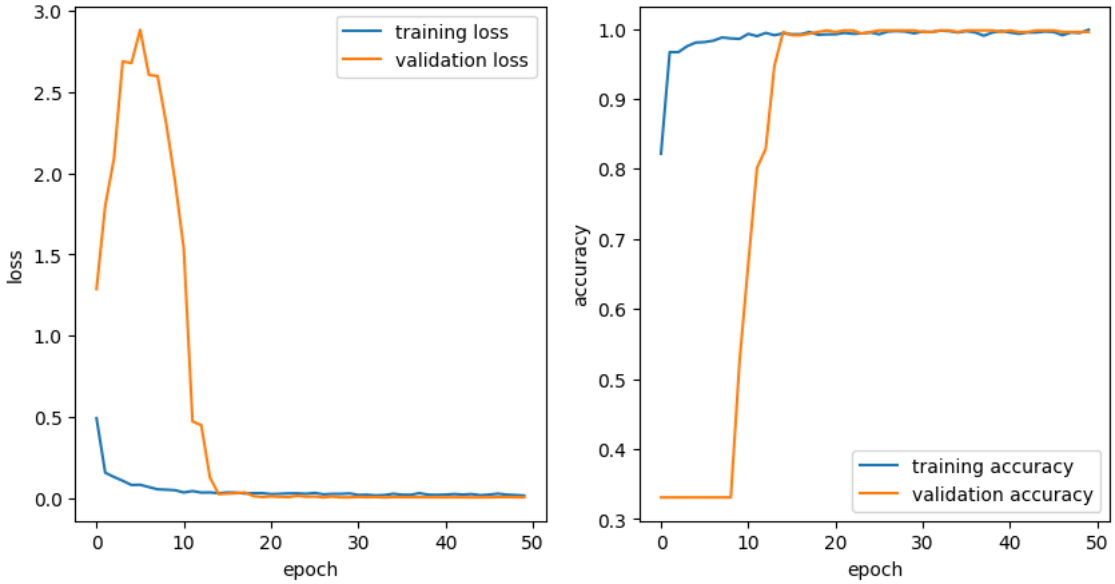
The table above describes the hyperparameter search space considered. It is important to note that, unlike the other parameters, the learning rate was not sampled from a discrete set of values. Instead, for each trial, a value was drawn from a logarithmic distribution within the interval $[10^{-4}, 10^{-2}]$, allowing for a flexible exploration of different learning-rate magnitudes.

Due to computational constraints, a full grid search with k-fold cross-validation was not performed. Instead, a **random search** strategy was adopted, where 25 hyperparameter combinations were randomly sampled from the defined search space and evaluated on the validation split. To reduce the effect of random weight initialization on the results, each sampled configuration was trained three times, and the mean validation accuracy across runs was used as the performance metric. All training runs employed **early stopping** based on the validation loss, which helped prevent overfitting and reduced unnecessary computation. the obtained hyper parameters are shown below:

```
conv_1: 64
conv_2: 64
conv_3: 256
num_units: 128
dropout: 0.5
learning_rate: 0.00027684469877965653
```

4.6.2 Training and evaluation

The final model, built using the hyperparameters selected through random search, was trained from scratch using the **Adam** optimizer and a batch size of 32. Training employed both **early stopping** and the **ReduceLROnPlateau** callback. The latter halved the learning rate whenever the validation loss did not improve for three consecutive epochs. This adaptive scheduling was introduced after preliminary experiments showed irregular and oscillating validation curves, suggesting that the initial learning rate was occasionally too high. Using `ReduceLROnPlateau` helped stabilize the optimization process and produced smoother convergence behavior.



On the test set, the model achieved an overall accuracy of **100%**. The per-class precision, recall, and F1-scores clearly were as follows:

	precision	recall	f1-score	support
paper	1.00	1.00	1.00	71
rock	1.00	1.00	1.00	73
scissors	1.00	1.00	1.00	75

Figure 11: Classification report on the test set.

The training curves and the results on the test set demonstrate an excellent ability to generalize, with no evidence of overfitting or performance degradation. The model exhibits stable convergence, high accuracy, and perfect classification capability across all classes.

5 Domain shift and custom dataset

After training all three models, I saved them and tested their performance on a small custom dataset composed of photos collected from family members. These photos differed significantly from those in the original dataset in several aspects, including lighting conditions, camera angles, hand size and shape (they were mainly female hands), background, and skin tone. The results for the three models are shown below.

	precision	recall	f1-score	support
paper	0.00	0.00	0.00	22
rock	0.33	1.00	0.49	19
scissor	0.00	0.00	0.00	18
accuracy			0.32	59

(a) Results of the third model.

	precision	recall	f1-score	support
paper	0.38	0.82	0.51	22
rock	0.22	0.11	0.14	19
scissor	0.50	0.06	0.10	18
accuracy			0.36	59

(b) Results of the second model.

	precision	recall	f1-score	support
paper	0.35	0.82	0.49	22
rock	0.00	0.00	0.00	19
scissor	0.67	0.11	0.19	18
accuracy			0.34	59

(c) Results of the first model.

As can be seen, the models achieved very low accuracy, essentially equivalent to random guessing among the three classes. This result suggests the presence of a **domain shift**, a situation in which a model trained on data from one domain fails to generalize to data originating from a different domain. A plausible explanation is that, due to their relatively shallow architectures—justified in this context by computational constraints and the limited size of the dataset, which would make deeper models prone to overfitting—the networks were unable to learn sufficiently abstract geometric representations of the gestures. Instead, they likely relied on low-level visual cues such as lighting or local contrast, which are no longer correlated with the target classes in this new domain.