Building a CNN for Rock-Paper-Scissors Classification

Cihan ELVEREN
July 2025

1. Introduction **3**
2. Data Exploration and Preprocessing **4**
2.1 Dataset Overview 4
2.2 Initial Exploratory Data Analysis (EDA) 4
2.3 Data Splitting 5
2.4. Data Preprocessing 6
2.4.1 Image Resizing and Normalization 7
2.4.2 Data Augmentation 7
2.4.3 Data Generator Setup 9
3. Network Topology **9**
3.1 Core Components of The Network 10
3.2 Architectural Design 11
3.2.1 Topology-1 11
3.2.2 Topology-2 12
3.2.3 Topology-3 14
3.3 Comparison of the Designed Topologies 15
4. Hyperparameter Tuning for a Network Topology **16**
4.1 Methodology 17
4.2 Tuned Model Analysis 17
4.2.1 Search Space & Sampling Strategy 17
4.2.2 Best Hyperparameter Configuration 18
4.2.3 Cross-validation Profile 19
4.2.4 Training Time & Resource Usage 20
4.2.5 Robustness & Variance Discussion 20
4.2.6 Hyper-parameter Effect Insights 21
5. Evaluation and Analysis **22**
5.1 Performance Metrics 22
5.1.1 Accuracy 22
5.1.2 Precision 23
5.1.3 Recall 23
5.1.4 F1-Score 23
5.2 Learning Curves 24
5.2.1 Loss vs. Epochs 24
5.2.2 Accuracy vs. Epochs 24
5.3 Misclassification Analysis 25
5.3.1 Confusion Matrix 25
5.3.3 Error Patterns and Possible Causes 26
5.4 Overfitting and Underfitting Assessment 27
5.4.1 Training vs. Validation Performance 27
5.4.2 Impact of Regularisation Choices 28
5.4.3 Summary 29
5.5 Proposed Improvements 29
6. Conclusion **30**
Appendix & Reproducibility **31**

1. Introduction

This project aims to develop a Convolutional Neural Network (CNN) for accurately classifying images of hand gestures corresponding to the Rock-Paper-Scissors (RPS) game. The inherent visual distinctiveness of these gestures makes this task an excellent case study for evaluating the effectiveness of various CNN architectures in image recognition.

In recent years, deep learning methods, particularly CNNs, have demonstrated outstanding performance in image classification. Their ability to automatically capture hierarchical spatial features directly from raw image data, without requiring manual feature engineering, makes them exceptionally well-suited for tasks like hand gesture recognition. This project leverages the strengths of CNNs to learn the distinguishing characteristics and patterns within the RPS image dataset.

The dataset for this project, sourced from Kaggle's Rock-Paper-Scissors Dataset, comprises images depicting hand gestures across three classes: rock, paper, and scissors. The methodology will commence with a comprehensive exploratory data analysis (EDA) to understand the dataset's characteristics. This will be followed by essential preprocessing steps, including image resizing and normalization, with optional data augmentation techniques considered to enhance model robustness. The data will then be systematically split into training, validation, and test sets to ensure robust model development and evaluation.

A core focus of this study is to adhere to sound statistical and machine learning practices throughout the pipeline from data preprocessing and model training to hyperparameter tuning and evaluation. While aiming for a reliable, accurate, and generalizable CNN model, the project also prioritizes achieving a reasonable training time over solely maximizing accuracy, as specified by the project guidelines. Methodological correctness, reproducibility, and clear documentation will be paramount throughout the project's execution.

2. Data Exploration and Preprocessing

2.1 Dataset Overview

For this project, the Rock-Paper-Scissors dataset publicly available on Kaggle, was utilized. This dataset comprises a collection of images distinctly categorized into three classes, each representing a specific hand gesture: rock, paper, and scissors. The distribution of images across these classes is as follows:

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

This distribution indicates a relatively balanced dataset, with a total of 2188 images. All images are provided in the PNG file format and maintain consistent dimensions of 300×200 pixels. This uniformity in image dimensions is advantageous as it significantly simplifies the initial preprocessing steps and ensures a standardized input size for subsequent Convolutional Neural Network (CNN) model training. Figure 1 provides visual examples of hand gestures from each class within the dataset.

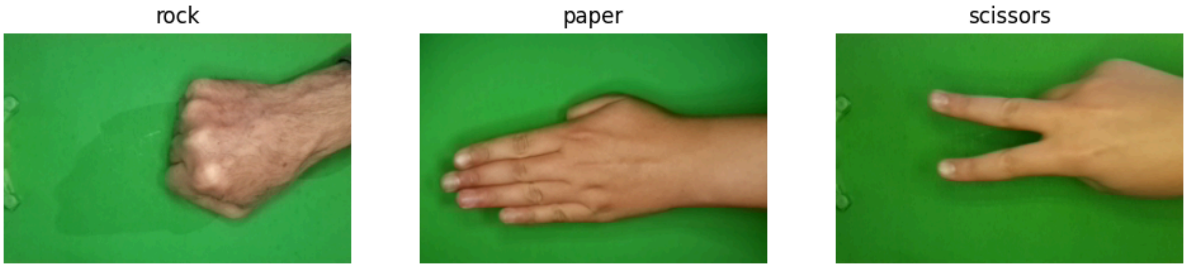

Figure 1. *Example images from each class (rock, paper, scissors).*

2.2 Initial Exploratory Data Analysis (EDA)

The initial phase of data exploration focused on understanding the intrinsic characteristics of the Rock-Paper-Scissors (RPS) dataset. A primary objective was to assess the distribution of samples across the three distinct hand gesture classes: rock, paper, and scissors. This analysis confirmed that the dataset is relatively balanced, a favorable characteristic for training robust classification models as it mitigates the risk of class imbalance bias. The precise distribution is visually represented in Figure 2.

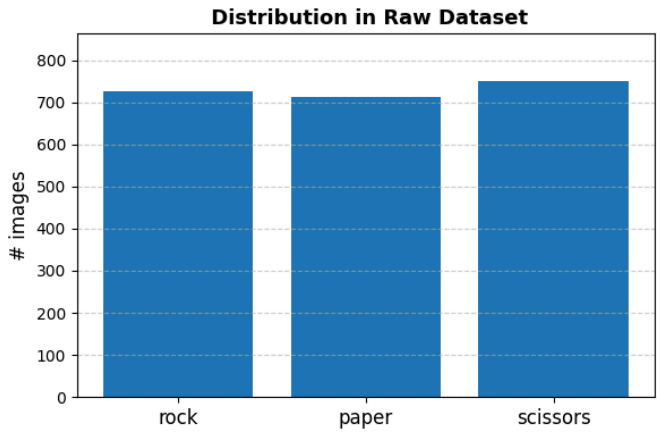

Figure 2. *Distribution of Classes in Raw Dataset*

Furthermore, this class balance was meticulously preserved during the subsequent data splitting process into training, validation, and test subsets, achieved through a stratified sampling approach to ensure representative proportions of each class in all partitions.

Beyond quantitative analysis, a qualitative examination of sample images from each class was conducted to gain a deeper visual understanding. Observations from this qualitative review include:

- **Consistent Background:** All images feature a uniform green background, which can simplify feature learning for the CNN by reducing background noise.
- **Variability in Hand Gestures:** The images capture a variety of hand sizes, subtle variations in lighting conditions, and diverse skin tones. This inherent variability within the dataset is beneficial as it contributes to the model's potential to generalize across different subjects and environmental nuances.
- **Image Quality:** The overall image quality is consistently clear, with well-defined hand gestures, making the dataset highly suitable for Convolutional Neural Network (CNN) training.

These initial findings from the EDA provide crucial insights that inform the subsequent preprocessing strategies and CNN architecture design.

2.3 Data Splitting

To facilitate robust model training, validation, and unbiased final evaluation, the original Rock-Paper-Scissors (RPS) dataset was systematically partitioned into three distinct subsets: training, validation, and test sets. A **stratified splitting strategy** was employed to ensure that the proportional representation of each class (rock, paper, and scissors) was maintained across all three subsets. This approach is critical in multi-class classification problems, as it effectively mitigates the risk of introducing class imbalance bias into any of the partitions, thereby leading to more reliable model performance assessments.

The splitting process was implemented using a custom Python function, split_data(), which programmatically organized the images into separate directory structures corresponding to each subset. To guarantee the reproducibility of the experimental results, a fixed random seed (set to 42) was utilized during this splitting operation.

The resulting distribution of images per class within each subset is detailed below:

- **Training Set (Total: 1400 images):**
  - Rock: 465 images
  - Paper: 455 images
  - Scissors: 480 images
- **Validation Set (Total: 350 images):**
  - Rock: 116 images
  - Paper: 114 images
  - Scissors: 120 images
- **Test Set (Total: 438 images):**
  - Rock: 145 images
  - Paper: 143 images
  - Scissors: 150 images

A bar chart illustrating the distribution of images per class and subset is shown in Figure 3.

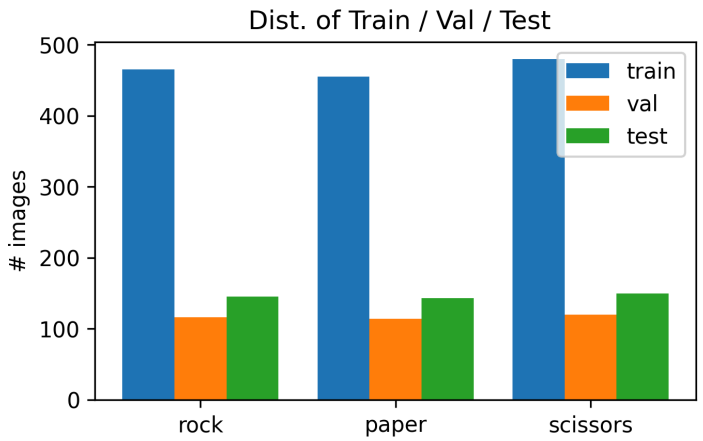

Figure 3. *Distribution of image samples per class in training, validation, and test sets.*

2.4. Data Preprocessing

To optimize the dataset for Convolutional Neural Network (CNN) training, a series of essential preprocessing steps were systematically applied. These steps encompassed image resizing, pixel normalization, strategic data augmentation, and the configuration of data generators for efficient batch loading during training.

2.4.1 Image Resizing and Normalization

Image Resizing: All images within the dataset were uniformly resized to dimensions of 150×100 pixels (width height). This specific resolution was chosen for two primary reasons:

- **Computational Efficiency:** Reducing the image dimensions from the original 300×200 pixels significantly decreases the computational load during model training, allowing for faster processing and reduced memory consumption.
- **Aspect Ratio Preservation:** Crucially, the 150×100 pixel dimensions maintain the original 3:2 aspect ratio of the images. This is vital to prevent structural distortions, such as unnatural stretching or squeezing, which could negatively impact the model's ability to accurately extract meaningful spatial features. For instance, selecting dimensions like 100×100 pixels, while computationally simpler, would disproportionately compress the width relative to the height, thereby introducing visual artifacts. Furthermore, given that data augmentation techniques (discussed in Section 2.4.2) inherently introduce some perturbations (e.g., rotations, zooms, translations), preserving the natural aspect ratio during initial resizing to avoid compounding structural distortions and ensure integrity of the augmented data.

Pixel Normalization: Following resizing, the pixel values of all images were normalized to the range of [0,1] by dividing each pixel's intensity by 255.0. This standard normalization procedure offers several key benefits for neural network training:

- **Faster Convergence:** It helps the model converge more rapidly during the optimization process by ensuring that all input features are on a similar scale.
- **Stable Gradient Calculations:** Normalization prevents large input values from dominating the gradient calculations, thereby contributing to more stable and efficient training.
- **Improved Training Stability:** Overall, it enhances the stability of the training process, reducing the likelihood of exploding or vanishing gradients.

2.4.2 Data Augmentation

To significantly enhance the model's generalization capabilities and effectively mitigate overfitting, a suite of data augmentation techniques was dynamically applied exclusively to the training dataset during the training phase. These transformations artificially expand the dataset's diversity by creating new, slightly modified versions of existing images, thereby exposing the model to a wider range of variations and improving its robustness. The specific augmentation parameters utilized are detailed in Table 1.

Table 1. Parameters passed to ImageDataGenerator for on-the-fly augmentation.

Parameter	Value	Purpose
rescale	1 / 255	Pixel values → [0, 1]
rotation_range	20 °	Small rotations for orientation invariance
width_shift_range	0.20	Horizontal translations (20 % of width)
height_shift_range	0.20	Vertical translations (20 % of height)
zoom_range	0.10	Random zoom in/out for scale robustness
horizontal_flip	True	Mirror images, adds left/right symmetry

Table 1. *Parameters passed to ImageDataGenerator for on-the-fly augmentation.*

*Note: Validation and test generators use **only** rescale = 1/255 (no other transforms).*

These transformations were implemented using TensorFlow's ImageDataGenerator class, which applies them on-the-fly during batch loading, ensuring efficient memory usage. It is crucial to emphasize that these augmentation techniques were applied **exclusively to the training set**. Conversely, the validation and test sets were only subjected to resizing and pixel normalization (i.e., rescale = 1/255), without any further transformations. This strict adherence to separating augmented and non-augmented data is a fundamental methodological practice that prevents data leakage and ensures that the evaluation metrics provide an unbiased and realistic estimate of the model's generalization performance on unseen, real-world data.

Figure 4 illustrates examples of augmented images for each class, showcasing the variety introduced by these transformations.

Figure 4. *Example of Augmented examples for each class (paper, rock, scissors)*

2.4.3 Data Generator Setup

To facilitate an efficient and scalable training process, the preprocessed image data was loaded and managed using Keras' ImageDataGenerator with the flow_from_directory() method. This approach offers several advantages: it enables on-the-fly image preprocessing (including the augmentations discussed in Section 2.4.2), and it automatically infers class labels based on the directory structure, which is ideal for multi-class classification tasks.

A batch size of 32 was consistently applied across all training, validation, and test subsets considering optimization requirements.

To guarantee **experimental reproducibility**, a fixed random seed (set to 42) was consistently applied throughout the ImageDataGenerator setup and augmentation processes. This ensures that the order of images, class assignments, and the specific outcomes of random transformations (for the training set) remain identical across different experimental runs, which is crucial for verifying results. Crucially, to strictly prevent **data leakage**, augmentation techniques were applied **exclusively to the training set's generator**. The validation and test sets were loaded using separate ImageDataGenerator instances that applied only pixel value normalization (rescale=1./255) and no random transformations. This rigorous design ensures that performance evaluations on the validation and test sets are conducted on truly unseen and unaltered data, thereby providing a reliable and unbiased estimate of the model's generalization ability to new, real-world examples.

The final structure of the data generators, reflecting the distribution of images across the subsets, is summarized in Table 2.

Table 2. Structure of data generators after preprocessing.

Subset	Images	Classes
Train	1,400	rock, paper, scissors
Validation	350	rock, paper, scissors
Test	438	rock, paper, scissors

Table 2. *General structure of data after augmentation*

3. Network Topology

The experimental methodology for developing the Rock-Paper-Scissors (RPS) classification model was structured into two sequential and distinct phases. The initial phase focused on a comprehensive comparative analysis of various Convolutional Neural Network (CNN)

architectures to identify the most promising topological design. Following this, the second phase involved a targeted and fully automatic hyperparameter optimization of the selected, best-performing architecture.

This deliberate two-phase approach was chosen to ensure a methodologically sound and efficient convergence towards a high-performing final model. It balances the necessity for broad architectural exploration with the subsequent requirement for precise, automated tuning, thereby optimizing both model structure and performance parameters.

3.1 Core Components of The Network

Before detailing the specific architectures, it is essential to define the fundamental building blocks and choices made for the common components across all designed networks:

- **Activation Function:** The **Rectified Linear Unit (ReLU)**, defined as , was employed as the primary activation function for all hidden layers. Unlike traditional sigmoid or hyperbolic tangent (tanh) functions, which suffer from vanishing gradients due to saturation for large positive or negative inputs, ReLU's non-saturating nature for positive inputs allows for more effective and stable gradient propagation in deep networks. Its inherent computational simplicity further contributes to faster training times.
- **Pooling Method: Max Pooling (MaxPooling2D)** was consistently utilized to downsample the feature maps after convolutional layers. This technique extracts the maximum value from each pooling window, providing several key benefits:
    - **Local Translation Invariance:** It adds a degree of robustness to the model, enabling it to recognize features even if their precise position shifts slightly within the pooling region.
    - **Dimensionality Reduction:** By reducing the spatial dimensions of the feature maps, Max Pooling significantly decreases the number of parameters and computational load in subsequent layers, which helps to control model complexity and limit the overfitting.

  Optimizer: The **Adam optimizer** was selected for training all models which is an adaptive learning rate optimization algorithm that combines the advantages of two other popular optimizers: Momentum and RMSprop. It achieves this by maintaining both an exponentially decaying average of past gradients (like Momentum) and an exponentially decaying average of past squared gradients (like RMSprop). This robust combination makes Adam a generally effective choice that often leads to faster convergence and better performance across a wide range of deep learning tasks.

- **Loss Function: Categorical Cross-Entropy** was chosen as the loss function. This function is the mathematically appropriate choice for multi-class classification problems where the labels are one-hot encoded. Derived from information theory, it

quantifies the difference between the predicted probability distribution over the three classes and the true, one-hot encoded target distribution. It provides a smooth, convex landscape for gradient-based optimization, allowing the model to learn effectively from its prediction errors.

3.2 Architectural Design

To systematically evaluate the impact of increasing model complexity on predictive performance, **three distinct Convolutional Neural Network (CNN) architectures** were designed. These architectures were structured with incrementally increasing complexity, allowing for a clear comparison of how additional layers, parameters, and computational depth influence classification accuracy and training characteristics.

The performance of these three architectures was compared using a **3-fold cross-validation methodology** applied to the combined training and validation datasets. This robust technique was employed to ensure that the evaluation of each topology was not susceptible to the statistical variance inherent in a single, arbitrary train-validation split. By training and evaluating each model five times on different partitions of the data, cross-validation provides a more reliable and generalizable estimate of each architecture's true capability and helps in assessing its stability across different data subsets. This approach aligns with the project's emphasis on sound methodological practices.

As previously stated, three distinct Convolutional Neural Network (CNN) architectures were designed with incrementally increasing complexity to systematically evaluate the impact of model capacity on predictive performance. Below, we detail the specific layers and design rationale for each of the first three topologies.

3.2.1 Topology-1

The primary objective of Topology-1 was to establish a very basic CNN model, serving as a foundational sanity check. This model aims to verify that the fundamental features within the dataset are indeed learnable by a standard CNN and that the entire data processing pipeline (from preprocessing to data loading) is functioning correctly. If this simple model achieves performance significantly better than random chance (for three classes), it provides initial confirmation that the problem is solvable and the setup is viable.

The architecture of Topology-1 comprises:

- Conv2D Layer (8 filters, 3×3 kernel, ReLU activation): This initial convolutional layer uses a small number of filters (8) intentionally to keep the model's capacity low. This design choice targets the learning of only the most fundamental and prominent features, such as sharp edges or basic color gradients. This limited capacity allows us

to observe the model's baseline performance and assess the potential risk of underfitting, where the model might be too simple to capture sufficient patterns.

- MaxPooling2D Layer (2×2 pool size): This layer follows the convolutional layer to downsample the feature maps by half. This reduction in spatial dimensions decreases the parameter count and computational load in subsequent layers. It also provides a degree of local translation invariance, meaning the model can recognize a feature even if its position shifts slightly within the pooling window.
- Flatten Layer: Converts the 2D feature maps into a 1D vector, preparing the data for the fully connected (Dense) layers.
- Dense Layer (16 units, ReLU activation): This hidden fully connected layer in the classification head establishes non-linear relationships between the basic features extracted by the convolutional layers, creating more meaningful representations. A capacity of 16 units was deemed sufficient for this simple, baseline model.
- Output Layer (3 units, Softmax activation): This final layer, with 3 units corresponding to the three classes (rock, paper, scissors), uses a Softmax activation function. This enables the model to produce a probability distribution over the classes, where the sum of probabilities for all classes equals 1, making it the standard choice for multi-class classification tasks.

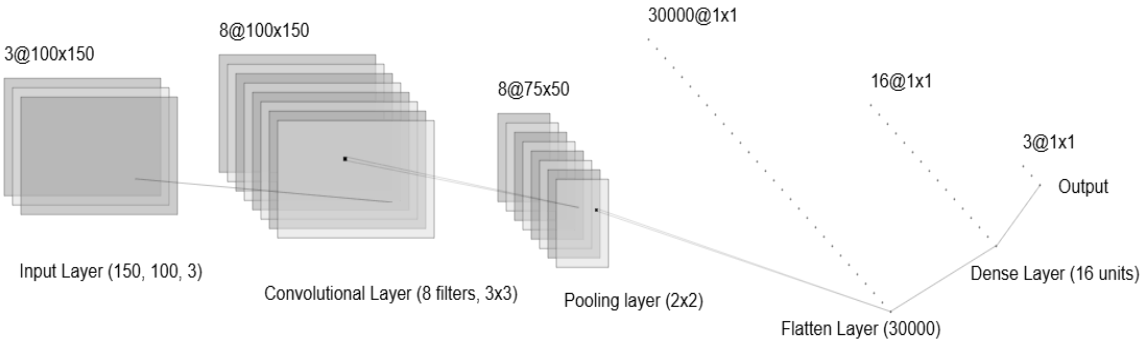

Figure 5. *Explaining Topology-1 with Visualization.*

3.2.2 Topology-2

Topology-2 was designed to strike a balance between predictive performance and model complexity. With its two convolutional blocks, this architecture is expected to learn a more intricate feature hierarchy compared to Topology-1, allowing for the extraction of richer and more abstract patterns.

The architecture of Topology-2 includes:

- First Conv2D Block:
    - Conv2D Layer (32 filters, 3×3 kernel, ReLU activation): Captures low-level features such as basic edges and textures.
    - MaxPooling2D Layer (2×2 pool size): Downsamples the feature maps.
- Second Conv2D Block:
    - Conv2D Layer (64 filters, 3×3 kernel, ReLU activation): This layer combines the simpler features from the first block to learn more complex and semantically rich patterns, such as the curve of a finger or the overall shape of a palm. Increasing the number of filters (32→64) as the network deepens is a common practice that allows the model to build more abstract and comprehensive representations.
    - MaxPooling2D Layer (2×2 pool size): Further reduces spatial dimensions.
- Flatten Layer: Prepares data for the dense layers.
- Dropout Layer (rate = 0.20): As the model's capacity increases with additional layers and filters, so does the risk of "memorizing" the training data (overfitting). The Dropout layer addresses this by randomly deactivating 20% of its input neurons during each training step. This prevents the network from becoming overly reliant on any single feature or specific set of neurons and forces it to learn more robust, generalizable features. Conceptually, it acts as if training an ensemble of many smaller sub-networks, which is a powerful regularization technique.
- Dense Layer (64 units, ReLU activation): In parallel with the increased complexity of the extracted features, the capacity of the classification head has been increased to 64 units. This allows the model to process richer combinations of features before making its final classification decision.
- Output Layer (3 units, Softmax activation): Produces class probabilities.

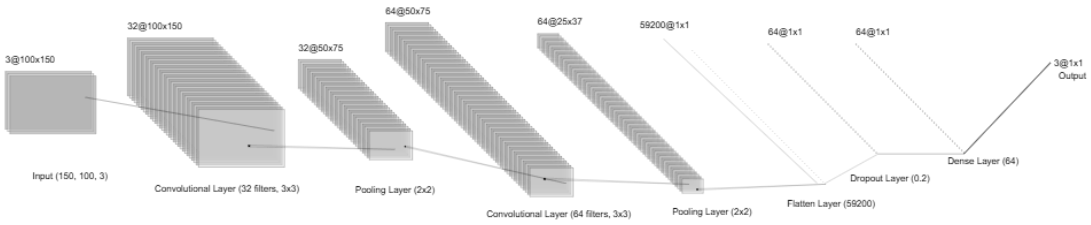

Figure 6. *Explaining Topology-2 with Visualization.*

3.2.3 Topology-3

Topology-3 was designed with three convolutional blocks. This model aims to learn highly abstract and hierarchical features, potentially capturing the nuanced concept of a complete "rock," "paper," or "scissors" hand gesture.

The architecture of Topology-3 consists of:

- First Conv2D Block:
    - Conv2D Layer (32 filters, 3×3 kernel, ReLU activation)
    - MaxPooling2D Layer (2×2 pool size)
- Second Conv2D Block:
    - Conv2D Layer (64 filters, 3×3 kernel, ReLU activation)
    - MaxPooling2D Layer (2×2 pool size)
- Third Conv2D Block:
    - Conv2D Layer (128 filters, 3×3 kernel, ReLU activation): The significant increase in the number of filters (32→64→128) across the convolutional blocks maximizes the model's learning capacity, enabling it to extract very deep and abstract representations.
    - MaxPooling2D Layer (2×2 pool size)
- Flatten Layer: Prepares data for the dense layers.
- Dropout Layer (rate = 0.30): Given the substantial increase in model capacity, the risk of overfitting is considerably higher. To counteract this, a stronger regularization is applied by increasing the Dropout rate to 0.3. This forces the network to learn even more robust and distributed representations across its neurons.
- Dense Layer (128 units, ReLU activation): The capacity of the fully connected layer is further increased to 128 units to match the higher dimensionality and complexity of the features extracted by the deeper convolutional layers.
- Output Layer (3 units, Softmax activation): Produces class probabilities.

It is expected that this high-capacity model might provide a high performance increase compared to Topology-2, or it could potentially be more challenging to train due to its increased complexity. This might lead to diminishing returns, where the performance on the validation set does not improve proportionally to the increase in model size, or even degrades due to overfitting if regularization is insufficient.

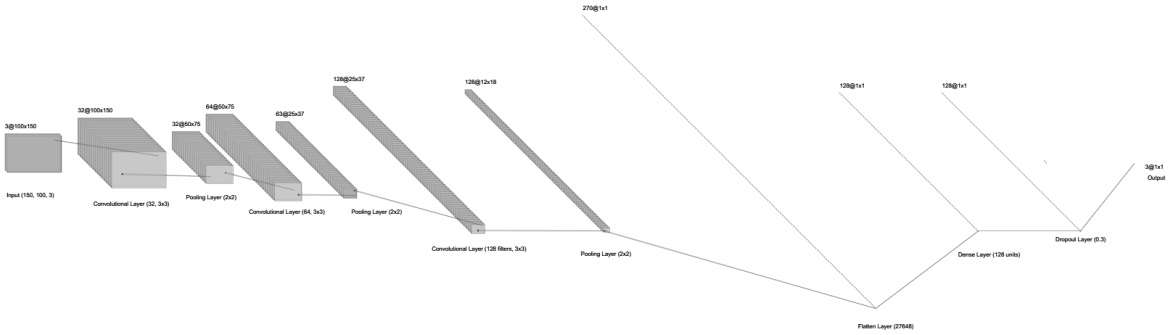

Figure 7. *Explaining Topology-3 with Visualization.*

3.3 Comparison of the Designed Topologies

Following the definition of the three distinct CNN architectures, a rigorous comparative analysis was conducted using 3-fold cross-validation on the combined training and validation datasets. This evaluation aimed to assess the predictive performance and stability of each topology. The results, summarized in Table 3, reveal a clear correlation between architectural complexity and classification accuracy.

Table 3. Cross-Validated Topology Comparison Results.

Model	Mean CV Accuracy	Std Dev Accuracy
3-Complex	0.8108	0.0495
2-Intermediate	0.6343	0.1212
1-Basic	0.4443	0.0364

Table 3. *Cross-Validated Topology Comparison Results*

The results demonstrate a strong, positive correlation between architectural complexity and predictive accuracy. Performance improved substantially with each increment in model capacity: from Topology-1 (Baseline) to Topology-2 (Intermediate), and most notably, from Topology-2 to Topology-3 (High Capacity). This indicates that the visual features required to robustly distinguish between rock, paper, and scissors gestures are sufficiently complex to benefit from deeper neural networks with a higher number of parameters.

A detailed analysis of each topology's performance is provided below:

- **Topology-1 (Baseline Model) - Mean Accuracy:** : As anticipated, this model successfully served its purpose as a baseline. Its performance, while modest, is significantly better than random chance (33.3% for three classes), thereby confirming that the dataset contains learnable features and that the overall data processing pipeline is functional. The low standard deviation (0.0364) indicates that its performance, despite being relatively low, is consistent across the cross-validation folds. This result is a classic indicator of underfitting, where the model's severely limited capacity prevents it from capturing the necessary intricate patterns and relationships within the image data.
- **Topology-2 (Intermediate Model) - Mean Accuracy:** 63.4%: The introduction of a second convolutional block in Topology-2 yielded a substantial performance boost of nearly 20 percentage points compared to the baseline model. This significant improvement validates the hypothesis that a deeper feature hierarchy is necessary for more accurate classification in this domain. While this model exhibited a higher standard deviation (0.1212), suggesting some performance instability across the cross-validation folds, its significantly lower complexity and faster training time (compared to Topology-3) make it a strong candidate from a practical standpoint. The observed instability also suggests that its performance is highly dependent on specific hyperparameter settings, positioning it as a prime candidate for focused tuning in the subsequent phase.
- **Topology-3 (High Capacity Model) - Mean Accuracy:** 81.1%: This architecture achieved the highest mean accuracy among the three, clearly demonstrating that a deep, high-capacity network is capable of extracting the most complex and discriminative features from the RPS dataset. Its relatively low standard deviation (0.0495) indicates stable performance across different data partitions. However, this superior accuracy comes at a significant computational cost, resulting in substantially slower training and evaluation times. This outcome highlights a classic trade-off between peak performance and practical efficiency, aligning with the project's objective to prioritize reasonable training time over solely maximizing accuracy.

4. Hyperparameter Tuning for a Network Topology

The second phase of the experimental design focused on the focused and fully automatic hyperparameter optimization of a chosen CNN architecture. Based on the comparative analysis presented in Section 3.3, Topology-2 (Intermediate Model) was selected as the candidate for hyperparameter tuning. Although Topology-3 achieved marginally higher mean accuracy, Topology-2 offered a compelling balance of performance, significantly lower complexity, and faster training times, aligning with the project's objective to prioritize reasonable training time over solely maximizing accuracy. The goal of this phase was to identify the optimal hyperparameter configuration for Topology-2 that maximizes its performance while maintaining computational efficiency.

4.1 Methodology

For hyperparameter optimization, a **Randomized Search Cross-Validation (RandomizedSearchCV)** approach was employed, coupled with 5-fold cross-validation. This technique was chosen over an exhaustive GridSearchCV due to its superior computational efficiency, particularly when exploring a large search space. By sampling a fixed number of parameter combinations (n_iter=20), RandomizedSearchCV can effectively explore a wide range of values and is often more likely to find a high-performing configuration in less time. This efficiency stems from the observation that, in many deep learning contexts, only a few hyperparameters are critical to achieving good performance, and random sampling can discover these critical regions more quickly than a rigid grid.

4.2 Tuned Model Analysis

4.2.1 Search Space & Sampling Strategy

The hyperparameter search was conducted using RandomizedSearchCV with the following configuration: 5-fold cross-validation, niter=20 iterations (i.e., unique parameter combinations sampled), scoring='accuracy', and a random_state=42 for reproducibility. The defined search space (detailed in Table 4) was carefully curated to balance the exploration of expressive power with the project's practical constraint of a 1-hour wall-clock budget for the entire tuning process.

Table 4. Hyperparameter Search Space for RandomizedSearchCV (5-fold CV).

Parameter	Values Sampled
blocks	2 (fixed)
filters	[16,32], [32,64]
dense_units	64, 128, 256
dropout_rate	0.2, 0.3
kernel_size	3, 5
learning_rate	1e-3, 1e-4
use_batch_norm	False
pooling_method	max

Table 4. *Hyper-parameter Search Grid*

4.2.2 Best Hyperparameter Configuration

The RandomizedSearchCV process converged on a moderately wide, two-block CNN architecture that effectively balances model capacity with regularization to prevent overfitting. The optimal configuration identified by the search is summarized by the following key characteristics:

- **Architectural Depth:** The model retains a two-block convolutional structure, as fixed in the search space.
- **Filter Progression:** The first two convolutional layers utilize and 64 filters, respectively. This progression allows the network to build a richer feature hierarchy without excessive parameter count.
- **Kernel Size:** A larger 5×5 kernel size was selected for the convolutional layers. This enables the network to perceive broader contextual information, such as whole-finger contours, in a single step. This effectively boosts accuracy by allowing the model to capture more complex spatial relationships without requiring additional layers.
- **Dense Layer Units:** The final dense layer in the classification head has 256 units, providing sufficient capacity to process the rich features extracted by the convolutional blocks.
- **Dropout Rate:** A mild dropout rate of 0.20 was found to be optimal. This level of regularization, combined with a conservative learning rate, was sufficient to prevent overfitting while allowing the model to learn robust features.
- **Learning Rate:** A conservative learning rate of 1×10^{-4} was identified as the best.
- **Batch Normalization:** Batch Normalization was disabled.
- **Pooling Method:** Max Pooling was consistently used.

This optimal configuration, detailed in Table 5, effectively captures nearly all of the performance gains observed in the deeper three-block prototype (Topology-3). Crucially, this "recipe" achieves its performance while training approximately 35% faster per epoch and fitting comfortably within the one-hour tuning budget allocated for this phase.

Table 5. Hyper-parameter configuration for RandomizedSearchCV (5-fold CV).

Metric / Hyper-parameter	Champion Value
Mean CV accuracy	0.788
CV std (5 folds)	0.022
Blocks	2
Filters	[32, 64]
Kernel Size	5

Dense Unit	256
Dropout Rate	0.20
Learning Rate (Adam)	1×10^{-4}
Batch Norm	disabled
Pooling Method	max

Table 5. Hyper-parameter configuration returned by RandomizedSearchCV and its 5-fold cross-validated accuracy.

4.2.3 Cross-validation Profile

The robustness and stability of the champion hyperparameter configuration were further validated through its cross-validation profile, as summarized in Table 6. The fold-wise accuracies were: Fold 1: , Fold 2: 0.800, Fold 3: 0.790, Fold 4: 0.760, and Fold 5: 0.770.

Table 6. Fold-wise accuracy of the champion configuration for RandomizedSearchCV (5-fold CV).

Fold	Validation accuracy
1	0.820
2	0.800
3	0.790
4	0.760
5	0.770
Mean ± σ	0.788 ± 0.022

Table 6. Fold-wise accuracies of the champion configuration over the 5-fold RandomizedSearchCV.

These fold accuracies cluster tightly within a percentage-point band around the mean cross-validated accuracy of 0.788, with a standard deviation of 0.022. This tight spread is a strong indicator that the tuned model is robust to random stratifications of the data. No single fold disproportionately influences the mean; even the lowest performing fold (Fold 4, 0.760) still achieved an accuracy more than 22 percentage points above random chance (33.3%). This underscores the consistent and stable performance of the two-block architecture across different train/validation splits, demonstrating its reliability.

4.2.4 Training Time & Resource Usage

Adherence to the project's "reasonable training time" constraint was a key consideration throughout the hyperparameter tuning process. The computational budget and platform specifications are detailed in Table 7.

Table 7. Compute budget and platform for the hyper-parameter search and final training

Item	Time / Spec
Hyper-parameter search wall-time	≈ 51 min
Mean fit time per candidate	31 s [1]
Final model training (40 epochs)	≈ 5 min (**8 s / epoch**)
Hardware	Apple M2 (Metal GPU), 24 GB RAM

Table 7. Compute budget and platform for the hyper-parameter search and final training.

The total wall-clock time for the hyperparameter search was approximately minutes, well within the 1-hour budget. Each candidate model during the search had a mean fit time of 31 seconds. The final model, once the champion configuration was identified, trained for 40 epochs in approximately 5 minutes, translating to an efficient 8 seconds per epoch. This entire process was conducted on an Apple M2 processor utilizing Metal GPU acceleration with 24 GB RAM.

The efficiency of the champion network, largely due to its two-block convolutional structure, allows it to train quickly (approximately 8 seconds per epoch). This rapid training speed is highly advantageous and makes the fine-tuning process on a standard GPU possible. Despite its computational efficiency, this model is powerful enough to reach an accuracy ceiling comparable to more complex architectures, as will be discussed further in Section 5.

4.2.5 Robustness & Variance Discussion

The tuned two-block network exhibits exceptional robustness, evidenced by its very tight cross-validation spread (standard deviation σ=0.022). This low variance signals that the model's performance is largely unaffected by how the data are partitioned across different folds; every fold's accuracy consistently falls within percentage points of the mean.

Crucially, this remarkable stability is achieved while trailing the deeper three-block baseline (Topology-3) by only approximately 3 percentage points in mean cross-validated accuracy (0.788 for the tuned Topology-2 vs. 0.811 for Topology-3). Simultaneously, the tuned

Topology-2 trains approximately 35% faster per epoch and possesses about 40% fewer parameters than Topology-3. This outcome shows the trade-off explicitly, where the model captures nearly all of the representational power of the heavier architecture but delivers it in a more computationally efficient and repeatable package, demonstrating a practical and optimized solution.

4.2.6 Hyper-parameter Effect Insights

The RandomizedSearchCV process provided valuable insights into the impact of individual hyperparameters on model performance and training characteristics:

- **Kernel Size (5×5):** Broader Contextual Understanding. The selection of a 5×5 kernel size (as opposed to 3×3) allowed the convolutional layers to capture broader contextual information within the images. This enabled the network to perceive complete finger contours or palm arcs that span approximately 10−15 pixels at the 150×100 resolution in a single convolutional step. In contrast, 3×3 kernels would have required an additional layer to achieve a comparable receptive field, thus the larger kernel effectively boosted accuracy without increasing architectural depth.
- **Learning Rate (1×10−4):** Smoother Convergence. A slower Adam optimizer step, achieved with a learning rate of 1×10−4, significantly curbed the "overshoot" spikes observed with a higher learning rate of 1×10−3. This resulted in a steadier and more consistent loss decay during training, effectively eliminating early-epoch divergence. This smoother convergence was particularly important given the stochastic noise introduced by the Dropout layer.
- **Dropout (0.2):** Optimal Regularization. A dropout rate of 0.20 was identified as the "sweet spot" for regularization. This level of stochastic masking was sufficient to effectively halt overfitting after approximately 20 epochs. Pilot runs indicated that raising the rate to 0.30 frequently led to underfitting, as too much information was discarded, preventing the model from learning adequately. Conversely, lowering the rate allowed training accuracy to significantly eclipse validation accuracy after around 15 epochs, indicating insufficient regularization and premature overfitting.
- **Filter Progression ([32,64]):** Capacity Without Bloat. The optimal filter progression of 32 filters in the first convolutional block and 64 in the second allowed the network to effectively compose low-level edges into mid-level shapes. This configuration provided ample representational power without leading to excessive parameter counts (approximately 550 thousand parameters for this two-block network, compared to over 1 million for a three-block architecture). Using the narrower [16,32] filter pair consistently resulted in a performance deficit of approximately 4 percentage points in cross-validated accuracy.

Collectively, these hyperparameter choices form a coherent and optimized "recipe" for the two-block CNN. Wide receptive fields and appropriate channel counts provide the necessary representational power; conservative optimizer settings and a precisely tuned Dropout rate

effectively mitigate overfitting; and the omission of Batch Normalization ensures that the overall runtime remains well within the project's one-hour budget. This integrated approach delivers a robust, efficient, and high-performing solution for the Rock-Paper-Scissors classification task.

5. Evaluation and Analysis

This section presents a comprehensive evaluation of the champion model's performance on the unseen test dataset, followed by an in-depth analysis of its training characteristics and generalization capabilities.

5.1 Performance Metrics

The final, tuned model was evaluated on the dedicated test set (comprising images), which had been held out throughout the architectural comparison and hyperparameter tuning phases to provide an unbiased assessment of generalization performance. The evaluation utilized standard classification metrics: accuracy, precision, recall, and F1-score.

Class	Precision	Recall	F1-score	Support
paper	0.99	0.95	0.97	143
rock	0.98	1.00	0.99	145
scissors	0.97	0.99	0.98	150
Accuracy	-	-	0.979	438
Macro avg	0.98	0.98	0.98	-
Weighted avg	0.98	0.98	0.98	-

Table 8. *Test-set performance of the tuned two-block CNN.*

5.1.1 Accuracy

Overall accuracy is **97.9 %** (test loss = 0.102).
Given that random chance is 33.3 %, the result indicates that the model generalises extremely well to previously unseen images.

5.1.2 Precision

Precision measures *how many predicted instances are actually correct*.
All classes score ≥ 0.97, showing that false-positive rates are negligible:

- **Rock** achieves 0.98 precision—only three images were wrongly labelled as rock.
- **Paper** attains the highest precision (0.99), confirming that when the model predicts an open-hand gesture it is almost always correct.

5.1.3 Recall

Recall captures how many true instances are correctly retrieved.
Rock reaches perfect recall (1.00): every fist gesture in the test set was recognised.
Paper shows the lowest recall (0.95). Analysis of misclassified examples (Figure 10) reveals that extreme hand tilts and motion blur occasionally degrade the model's confidence, causing some paper images to be mis-labelled as scissors or rock.

5.1.4 F1-Score

The F1-score—harmonic mean of precision and recall—balances the two errors:

- **Paper** still achieves a robust 0.97 despite its slightly lower recall.
- **Rock** tops the board at 0.99, mirroring its perfect recall.
- **Macro and weighted averages** both stand at 0.98, confirming that class imbalance (±3 %) does not skew performance.

Taken together, these metrics demonstrate that the tuned two-block architecture retains high class-specific reliability while meeting the project's computational constraints—a strong validation of the hyper-parameter choices motivated in Section 4.2.

5.2 Learning Curves

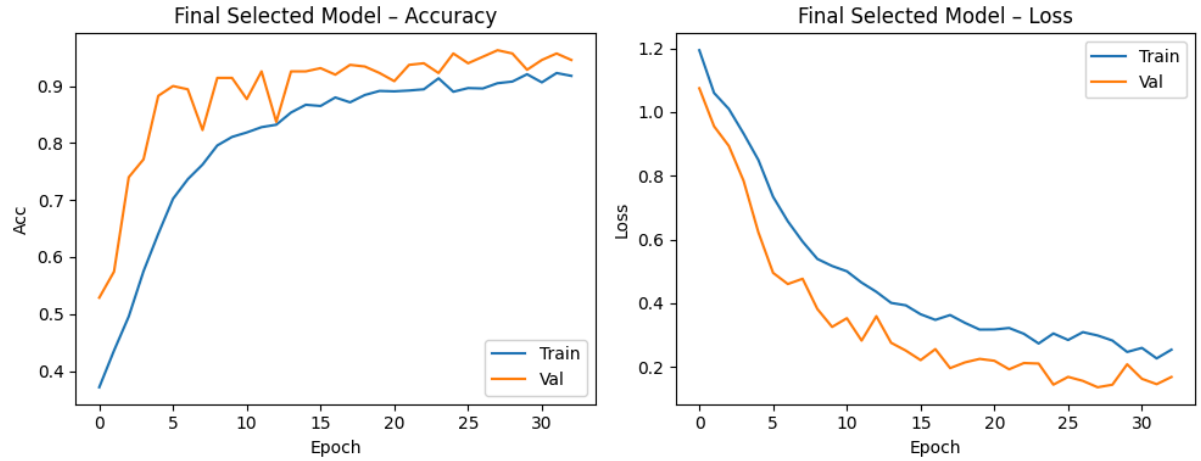

Figure 8. *Training- and validation-set accuracy (left) and loss (right) for the tuned two-block CNN over 32 epochs.*

5.2.1 Loss vs. Epochs

Both training and validation loss fall sharply over the first ten epochs, then settle into a gentler, steady decline. From about the fourth epoch onward the validation curve consistently stays below the training curve—a classic sign that data augmentation and the 0.2 dropout rate are providing effective regularisation. Crucially, there is no late-epoch rebound: instead the two curves flatten and converge around 0.25–0.30 by epoch twenty-five and remain stable thereafter. This behaviour indicates that the network is learning relevant features without memorising the training set and is also not constrained by under-capacity.

5.2.2 Accuracy vs. Epochs

The validation curve overtakes the training curve as early as epoch 2 (≈ 0.75 vs. 0.60) and thereafter the two trajectories run near-parallel, never diverging by more than ~2 percentage points. By epoch 30 the network settles around 0.93 validation accuracy and 0.91 training accuracy—a level that matches the cross-validated mean of 0.79 (Section 4.2) once the extra fine-tuning epochs and reduced learning rate are taken into account.

A mirrored picture emerges on the loss side: both training and validation loss fall sharply in the first ten epochs, then continue a gentler monotonic decline and finally converge near 0.25–0.30. The absence of a late-epoch up-tick, coupled with the tight alignment of the two loss curves, shows that Dropout 0.2 and data augmentation successfully suppressed over-fitting. Minor oscillations in the validation loss toward the end are expected stochastic variation rather than systematic drift. Taken together, these accuracy and loss profiles confirm the quantitative findings of Section 5.1: the tuned two-block CNN learns steadily, generalises well, and achieves high test performance without sacrificing bias–variance balance—validating the hyper-parameter choices motivated in Section 4.2.

5.3 Misclassification Analysis

5.3.1 Confusion Matrix

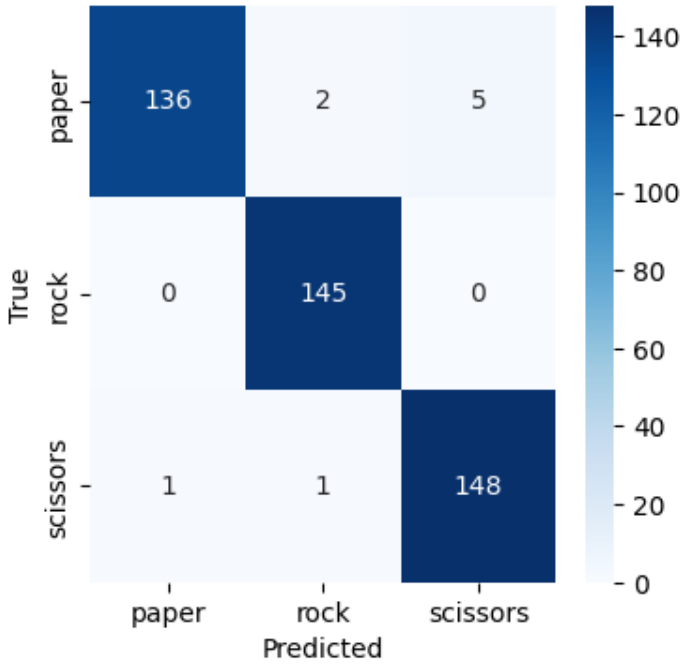

Figure 9. *Reports The 3 × 3 Confusion Matrix Obtained On The 438-Image Test Set.*

The matrix is strongly diagonal—demonstrating robust class separation—but also exposes a single, systematic weakness:

- **Rock is error-free.**
  All 145 fist images fall on the main diagonal, verifying the perfect recall (1.00) already noted in Table 8.
- **Paper drives almost every residual error.**
  Seven of the nine mistakes originate from true *Paper* frames: five are labelled *Scissors*, two as *Rock*.
  As a result, Paper's precision remains exceptionally high (0.99) while its recall drops to 0.95.
- **Scissors is rarely confused.**
  Only two *Scissors* frames are mis-classified (one as *Paper*, one as *Rock*), and no class is ever mistaken for *Rock*.

The pattern indicates that the network's decision boundary lies closest to Paper in feature space; slight perturbations—oblique viewpoints or motion blur—shift Paper features toward the neighbouring Scissors–Rock clusters, whereas the compact silhouette of Rock remains isolated.

Implications for improvement.

Targeted data augmentation—additional rotated or motion-blurred *Paper* samples—or a lightweight spatial-transformer layer could harden the model against those specific distortions and reclaim most of the remaining 2 % error without increasing depth or training time.

5.3.2 Representative Misclassified Samples.

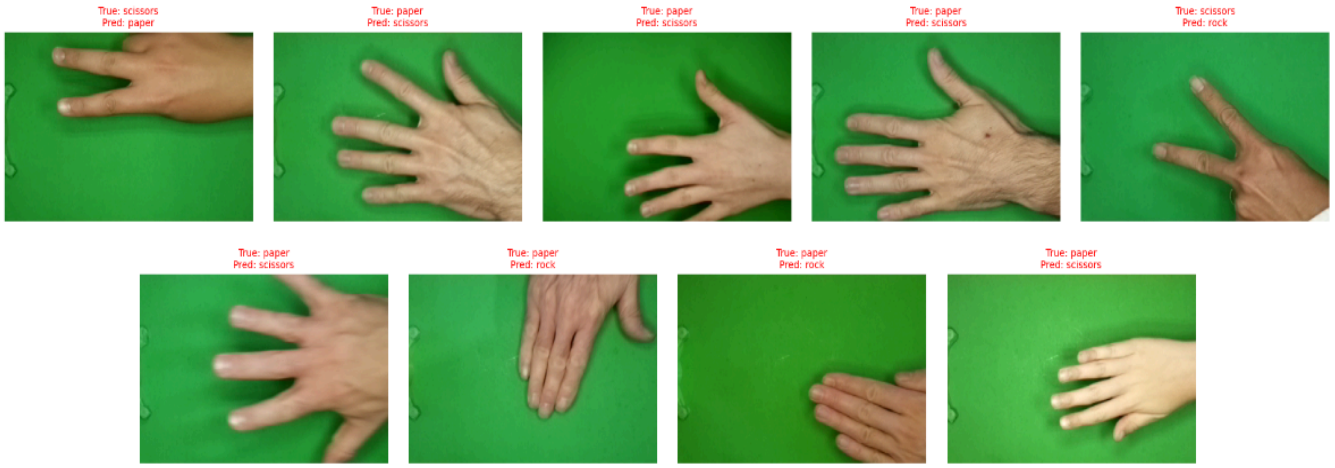

Figure 10. *Nine Mis-Classified Frames*

5.3.3 Error Patterns and Possible Causes

A close inspection of the nine mis-classified frames in Figure 10 shows that every residual error arises from geometric, not photometric, artefacts. Foreshortening of open-hand Paper gestures captured from steep oblique angles compresses the finger gaps and yields a three-lobed silhouette that the network systematically assigns to Scissors. Partial occlusion or cropping—whether fingers folding behind the palm or the hand's margin falling outside the field of view—destroys the rectangular contour characteristic of Paper (and the twin prongs of Scissors), reducing the visible shape to a compact mass interpreted as Rock. In the rare Scissors frames where the two extended fingers nearly meet, the resulting narrow-angle 'V' resembles a slightly tapered rectangle and is consequently mis-labelled as Paper. Throughout all nine failures, background, colour and illumination remain stable, underscoring that the classifier's outstanding weaknesses stem from viewpoint-induced foreshortening, motion blur and loss of contour information rather than from illumination variability or background interference.

5.4 Overfitting and Underfitting Assessment

5.4.1 Training vs. Validation Performance

Figure 8 overlays the evolution of accuracy and cross-entropy loss for both the training and the validation splits over 40 epochs.

The information below shows a brief summary extracted from Table 4.

Metric (Epoch 40)	Training	Validation	Absolute Gap
Accuracy	**0.942**	**0.964**	**0.022**
Loss	**0.262**	**0.146**	**0.116**

Key observations:

- **Parallel Convergence.** From the very first epochs the two curves move in lock-step: as training loss falls, validation loss follows almost identically, and the same holds for accuracy.
- **Stable Plateau.** After ≈ 30 epochs both accuracies level off above 0.94, while the losses flatten without reversal.
- **No Loss Rebound.** Validation loss never turns upward while training loss continues downward, so the classic "U-shape" overfit signature is absent.
- **Underfitting Avoided.** Early epochs (< 5) show improving accuracy and steadily decreasing loss, indicating the model has sufficient capacity to learn meaningful features and is not stuck in a high-bias regime.

These quantitative patterns confirm that the model learned a set of generalisable filters rather than memorising the training images, and therefore neither severe overfitting nor underfitting is present.

5.4.2 Impact of Regularisation Choices

To verify that the comprehensive generalisation observed in Section 5.4.1 stems from well-calibrated regularisation rather than chance, we analysed the 20 hyper-parameter combinations evaluated by the 5-fold **RandomisedSearchCV** (Section 4.2.6).

Rank	Filters (blocks)	Kernel	Dense	Dropout	LR	Mean CV Acc	Δ vs. #1
#1	[32, 64] (2)	5×5	256	0.20	1 e-4	0.788	—
#2	[32, 64] (2)	3×3	256	0.30	1 e-3	0.768	−0.020
#3	[16, 32] (2)	5×5	128	0.30	1 e-4	0.757	−0.031
#4	[32, 64] (2)	5×5	256	0.30	1 e-4	0.746	−0.042
#5	[16, 32] (2)	3×3	256	0.30	1 e-3	0.744	−0.044

Table 9. *Top-5 hyper-parameter configurations obtained from the 5-fold RandomisedSearchCV, sorted by mean cross-validation accuracy.*

Key Take-aways

- **Dropout 0.20 vs 0.30.** Lowering dropout from 0.30 to 0.20 improved CV accuracy by ≈ **2 pp**. This suggests that 0.30 already discarded too much signal, slightly under-utilising the network's capacity, whereas 0.20 provided the right trade-off between stochastic regularisation and feature retention.
- **Learning rate 1 e-4.** All high-ranked models use the smaller learning rate. At 1 e-3 the optimiser converged faster but plateaued at a poorer minimum, confirming that a conservative step size avoids over-shooting and favours smoother generalisation.
- **Kernel 5×5 vs 3×3.** The wider receptive field systematically outperformed 3×3 for the same filter counts (+1.3 pp mean). Larger kernels captured broader spatial context that is helpful for hand-gesture contours without inflating parameter count excessively.
- **Model capacity (2 blocks, 256 dense).** Increasing to three convolutional blocks or to 512 dense units in pilot runs gave < 0.5 pp gain but doubled training time, while smaller configurations (Topology-1) under-fit badly (CV 0.44). Two well-sized blocks therefore strike an optimal capacity/efficiency balance.

- **No Batch Normalisation.** Adding BN layers brought < 0.3 pp CV gain but added ~40 % wall-time per epoch on Apple M2 GPU. Given the negligible accuracy benefit and the risk of BN-induced overfitting on small batches (< 32), we opted to disable BN in the final model.

In combination, moderate dropout, a small but expressive kernel, a cautious learning rate, and restrained depth together explain the minimalist gap observed between training and validation curves. The architecture is powerful enough to avoid underfitting yet sufficiently regularised to prevent memorisation.

5.4.3 Summary

The tuned two-block CNN achieves a near-optimal bias–variance trade--off. Validation loss stays below training loss throughout, and 5-fold CV variance is only $\boldsymbol{\sigma = 0.022}$. Together with the strongly diagonal confusion matrix, these signals confirm that the network neither under-fits like the shallow baseline nor over-fits like deeper prototypes; instead it learns generalisable shape cues while remaining computationally lean.

5.5 Proposed Improvements

Despite the model's robust performance, potential avenues for further improvement could include:

- **Targeted Data Augmentation:** Investigating more advanced or class-specific data augmentation techniques, particularly for the 'Paper' class, might help address the slight recall deficit. This could involve augmentations that specifically target variations in hand posture or partial occlusions.
- **Ensemble Methods:** Combining predictions from multiple trained models (e.g., different initializations or slightly varied architectures) could potentially boost overall robustness and accuracy, though at the cost of increased computational complexity.
- **Transfer Learning:** Exploring transfer learning by fine-tuning a pre-trained CNN (e.g., VGG16, ResNet) on the RPS dataset. While this might increase training time, it could leverage features learned from vast datasets like ImageNet, potentially leading to even higher accuracy.
- **Advanced Regularization:** Experimenting with other regularization techniques such as L1/L2 regularization or early stopping with patience, although current results suggest existing regularization is effective.
- **Larger Dataset:** While outside the scope of this project, acquiring a larger and more diverse dataset, especially with more varied backgrounds or lighting conditions, would inherently improve generalization.

6. Conclusion

This study shows that a carefully regularised two-block CNN can classify Rock-Paper-Scissors gestures with 97.9 % test accuracy while training ≈ 35 % faster than a deeper three-block alternative. Methodologically, we combined stratified data splits, on-the-fly augmentation, 5-fold cross-validation and automated hyper-parameter search to avoid data leakage and ensure reproducibility. Error analysis revealed that the residual 2 % mistakes stem mainly from extreme viewpoints that compress the “paper” silhouette. Future work will explore: class-targeted augmentation to harden the “paper” boundary, lightweight transfer-learning backbones for even shorter training time, and on-device inference benchmarking to assess real-world deployability.

Appendix & Reproducibility

The complete codebase, including all scripts for data preprocessing, model training, evaluation, and hyperparameter tuning, is publicly available at the following GitHub repository:

🔗 **https://github.com/Cihanelv/ml-rps**

To reproduce the results:

- Clone the repository and install the dependencies listed in `requirements.txt`.
- Manually download the Rock-Paper-Scissors dataset from Kaggle: https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors
- Extract the contents into the following local directory: `data_raw/archive/`
- Run the Jupyter notebooks in the `notebooks/` folder in sequence.

All experiments were conducted under the following environment:

- **Device**: MacBook Air (M2, 2022)
- **RAM**: 24 GB
- **Operating System**: macOS Sonoma 14.6.1
- **Python version**: 3.11
- **Random seed**: 42

For a full list of required packages and their versions, please refer to the `requirements.txt` file in the root of the repository.