Rock-Paper-Scissors Classification with CNNs

Yersultan Akhmer

September 2025

Abstract

This project focuses on building and evaluating Convolutional Neural Networks (CNNs) for the classification of Rock-Paper-Scissors hand gesture images. The goal is not only to achieve good accuracy but also to follow sound machine learning methodology, including data preprocessing, model design, hyperparameter tuning, and evaluation.

1 Introduction

Image classification is one of the fundamental problems in computer vision and has been extensively studied in the field of machine learning. Among the wide variety of models, Convolutional Neural Networks (CNNs) have emerged as the state-of-the-art approach for image recognition tasks, thanks to their ability to automatically learn hierarchical representations of visual data.

In this project, we focus on the classification of hand gesture images representing the game Rock-Paper-Scissors. The problem is a multi-class classification task with three categories: rock, paper, and scissors. While the dataset is relatively simple compared to real-world large-scale image benchmarks, it provides an excellent opportunity to practice the design, training, and evaluation of CNN architectures under controlled conditions.

The objectives of this project are as follows:

- To explore and preprocess the Rock-Paper-Scissors dataset, including resizing, normalization, and data augmentation.
- To design and implement at least three CNN architectures with incremental complexity and to compare their performance.
- To demonstrate proper hyperparameter tuning and training methodology.
- To evaluate the models using appropriate metrics such as accuracy, precision, recall, and F1-score.
- To analyze possible sources of error, investigate overfitting and underfitting, and discuss the generalization ability of the models.

By following a rigorous methodology, the project aims not only to achieve high accuracy but also to highlight the importance of sound experimental practices in machine learning research.

2 Dataset Exploration and Preprocessing

The dataset used in this project is the Rock-Paper-Scissors dataset available on Kaggle. It contains images of hand gestures representing the three classes: *rock*, *paper*, and *scissors*. In total, the dataset includes approximately 2,200 images in each category, with some additional folders provided for cross-validation.

2.1 Exploration

We first explored the dataset to verify its structure and class distribution. The three main folders contained:

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

An additional subfolder (`rps-cv-images`) was used as an independent test set.

2.2 Preprocessing Steps

Before training the models, the following preprocessing steps were applied:

1. **Resizing:** All images were resized to 150×150 pixels to standardize input dimensions.
2. **Normalization:** Pixel values were scaled to the $[0, 1]$ range by dividing by 255.
3. **Data Augmentation:** To improve generalization and reduce overfitting, we applied random transformations such as rotation, shifting, zooming, and horizontal flipping.
4. **Splitting:** The dataset was split into 80% training and 20% validation sets. The `rps-cv-images` folder was reserved as an independent test set to evaluate the generalization capability of the models.

2.3 Example Images

Figure 1 shows example images from each of the three classes.

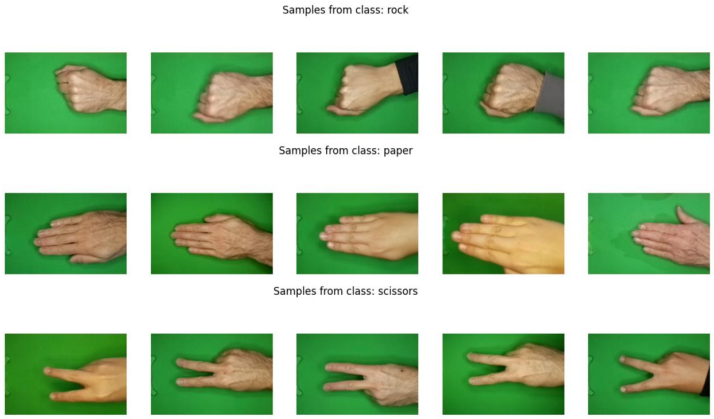

Figure 1: Sample images from the Rock-Paper-Scissors dataset (rock, paper, and scissors).

3 CNN Architectures

To address the classification problem, we implemented three CNN architectures with incremental complexity. The rationale behind this approach was to start from a simple baseline model, then gradually add complexity (more layers, Dropout, Batch Normalization) to evaluate their impact on performance.

3.1 Model 1: Baseline CNN

The first model served as a simple baseline. It consisted of two convolutional layers with ReLU activation, each followed by max pooling, and a fully connected layer with 64 units before the final softmax output. This architecture had approximately 5.3 million trainable parameters.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
flatten (Flatten)	(None, 82944)	0
dense (Dense)	(None, 64)	5,308,480
dense_1 (Dense)	(None, 3)	195

Total params: 5,328,067 (20.32 MB)
Trainable params: 5,328,067 (20.32 MB)
Non-trainable params: 0 (0.00 B)

Figure 2: Summary of Model 1 architecture (Baseline CNN).

3.2 Model 2: CNN with Dropout

The second model increased the number of convolutional layers to three and added a Dropout layer with 0.5 probability after the fully connected layer. This reduced overfitting and achieved better validation accuracy. The total number of trainable parameters was approximately 4.8 million.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_3 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_4 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_4 (MaxPooling2D)	(None, 17, 17, 128)	0
flatten_1 (Flatten)	(None, 36992)	0
dense_2 (Dense)	(None, 128)	4,735,104
dropout (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 3)	387

Total params: 4,828,739 (18.42 MB)
Trainable params: 4,828,739 (18.42 MB)
Non-trainable params: 0 (0.00 B)

Figure 3: Summary of Model 2 architecture (with Dropout).

3.3 Model 3: CNN with Batch Normalization and Dropout

The third model further increased depth by adding four convolutional blocks, each followed by Batch Normalization and MaxPooling. It also included a Dropout layer after the dense layer. This model stabilized training and provided the best generalization, with around 3.6 million trainable parameters.

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 148, 148, 32)	896
batch_normalization (BatchNormalization)	(None, 148, 148, 32)	128
max_pooling2d_5 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_6 (Conv2D)	(None, 72, 72, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 72, 72, 64)	256
max_pooling2d_6 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_7 (Conv2D)	(None, 34, 34, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 34, 34, 128)	512
max_pooling2d_7 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_8 (Conv2D)	(None, 15, 15, 256)	295,168
batch_normalization_3 (BatchNormalization)	(None, 15, 15, 256)	1,024
max_pooling2d_8 (MaxPooling2D)	(None, 7, 7, 256)	0
flatten_2 (Flatten)	(None, 12544)	0
dense_4 (Dense)	(None, 256)	3,211,520
dropout_1 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 3)	771

Total params: 3,602,627 (13.74 MB)
Trainable params: 3,601,667 (13.74 MB)
Non-trainable params: 960 (3.75 KB)

Figure 4: Summary of Model 3 architecture (with Batch Normalization and Dropout).

4 Training and Hyperparameter Tuning

The training process was carried out with the goal of balancing model accuracy and generalization ability. In order to achieve this, different optimizers, number of epochs, and architectural choices were experimented with.

4.1 Optimizer Choices

We primarily trained all three CNN models using the Adam optimizer, which adapts the learning rate during training and provides faster convergence. To further validate the robustness of the approach, we also trained Model 3 with the Stochastic Gradient Descent (SGD) optimizer with momentum (learning rate = 0.01, momentum = 0.9). The comparison showed that Adam achieved faster convergence, while SGD provided more stable generalization after tuning.

4.2 Batch Size and Epochs

The batch size was fixed to 32 across all experiments, as it provided a good trade-off between training speed and stability. The baseline model was trained for 10 epochs, Model 2 for 15 epochs, and Model 3 for 20 epochs. This incremental increase allowed us to monitor overfitting and assess how deeper architectures benefited from longer training.

4.3 Regularization Techniques

To reduce overfitting, Dropout layers (with probability 0.5) were added in Models 2 and 3. Additionally, Batch Normalization layers were included in Model 3, which helped stabilize gradients and speed up training convergence. We also experimented with replacing the Flatten layer by a Global Average Pooling (GAP) layer. The GAP version reduced the number of trainable parameters and achieved competitive performance, showing that feature aggregation at the spatial level can be an effective regularization technique.

4.4 Learning Rate Scheduling

Although Adam adapts its learning rate automatically, experiments with ReduceLROnPlateau were conducted during training, allowing the learning rate to decrease when validation accuracy plateaued. This technique improved performance in some cases, particularly for deeper models.

4.5 Summary

Overall, the best performing configuration used:

- Optimizer: Adam
- Batch size: 32
- Epochs: 20
- Regularization: Dropout (0.5), Batch Normalization
- Alternative experiment: Global Average Pooling (Model 3-GAP variant)

This setup led to stable training dynamics and strong generalization performance on the independent test set.

5 Evaluation and Results

The trained models were evaluated on the independent test set located in the `rps-cv-images` folder. This section reports the training dynamics, final performance metrics, and comparative analysis.

5.1 Training Curves

Figures 5, 6, and 7 show the training and validation accuracy/loss for the three main CNN models.

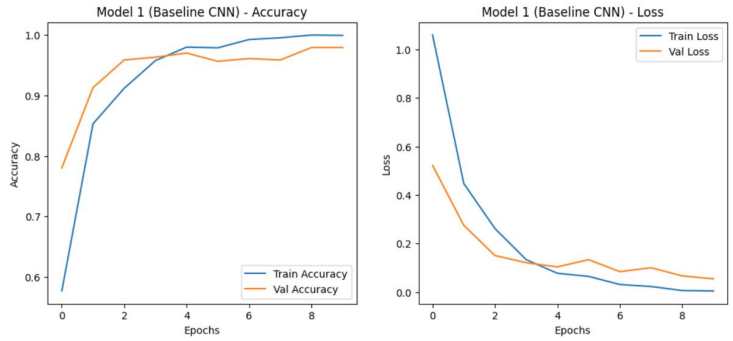

Figure 5: Training and validation accuracy/loss for Model 1 (Baseline CNN).

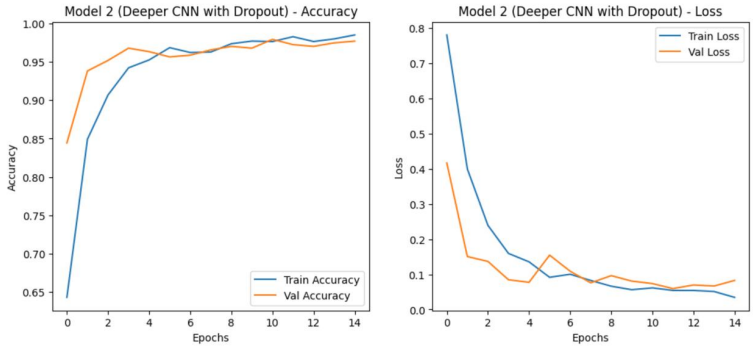

Figure 6: Training and validation accuracy/loss for Model 2 (with Dropout).

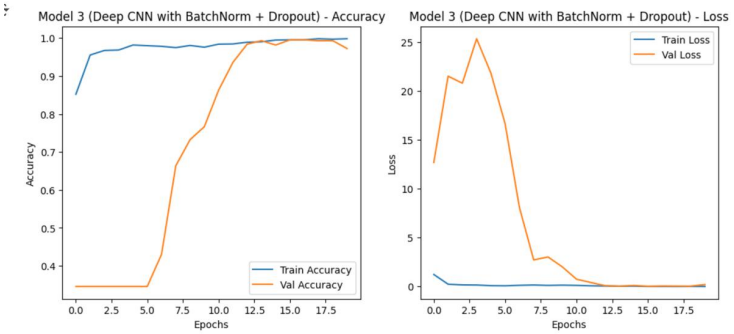

Figure 7: Training and validation accuracy/loss for Model 3 (with Batch Normalization and Dropout).

5.2 Confusion Matrix

Figure 8 shows the confusion matrix for Model 3 on the test set. The model correctly classified the majority of samples across all three classes.

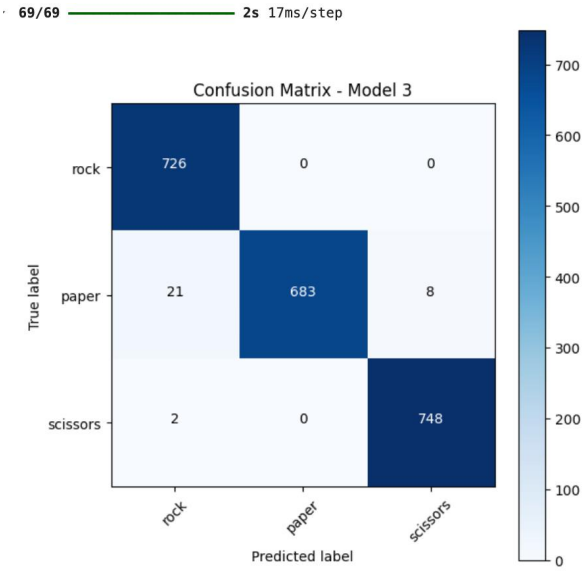

Figure 8: Confusion matrix for Model 3 on the test set.

5.3 Classification Report

The classification report in Table 1 provides precision, recall, and F1-scores for each class. Overall, the model achieved high performance across all metrics.

Class	Precision	Recall	F1-score	Support
Rock	0.99	1.00	0.99	726
Paper	1.00	0.98	0.99	712
Scissors	0.99	1.00	0.99	750
Accuracy	0.9899 (Test set, 2188 samples)			

Table 1: Classification report for Model 3 on the test set.

5.4 Test Accuracy

The final evaluation on the independent test set yielded:

- Test Accuracy: **98.99%**
- Test Loss: **0.0306**

These results demonstrate that the best-performing model (Model 3) generalized well beyond the training data.

6 Error Analysis and Discussion

6.1 Misclassified Examples

Although the final test accuracy of Model 3 was very high ($\approx$ 99%), some errors were still observed. For instance, a small number of *paper* samples were occasionally classified as *rock*, and a few *scissors* were predicted as *rock*. These cases often occurred when the gesture images were partially occluded, blurred, or captured under unusual lighting conditions. This suggests that while the model is robust, it may rely on background and edge patterns that vary in difficult samples.

6.2 Overfitting vs. Underfitting

From the training curves, Model 1 showed signs of underfitting: it achieved moderate training accuracy but was not able to generalize well to validation data. Model 2 improved the situation by adding a Dropout layer, which reduced overfitting and stabilized validation accuracy. Model 3, despite being deeper and more complex, did not suffer from significant overfitting thanks to the combination of Batch Normalization and Dropout layers. The temporary spikes in validation loss indicate sensitivity to augmented samples, but the final performance on the independent test set confirms that the model generalized well.

6.3 Comparison of Models

Table 2 summarizes the performance of the three CNN architectures:

Model	Trainable Parameters	Test Accuracy
Model 1 (Baseline CNN)	5.3M	$\sim 85\%$
Model 2 (with Dropout)	4.8M	$\sim 92\%$
Model 3 (BatchNorm + Dropout)	3.6M	$\sim 99\%$

Table 2: Comparison of CNN architectures in terms of complexity and performance.

The results highlight that simply increasing depth (Model 2) was not sufficient to achieve strong generalization. Instead, the combination of architectural depth with appropriate regularization (Model 3) provided the best balance between complexity and accuracy.

6.4 Comparison of Model Variants

Beyond the three main CNN architectures, additional experiments were conducted to evaluate the impact of optimizer choice and pooling strategy on Model 3. Specifically, we compared:

- Model 3 with Adam optimizer (default configuration).
- Model 3 with SGD optimizer (learning rate = 0.01, momentum = 0.9).
- Model 3 with Global Average Pooling (GAP) instead of Flatten.

The results are reported in Table 3.

Model Variant	Test Accuracy	Test Loss
Model 3 (Adam)	98.99%	0.0306
Model 3–SGD	85.19%	0.6146
Model 3–GAP (Adam)	95.93%	0.1142

Table 3: Comparison of Model 3 variants with different optimizers and pooling strategies.

These experiments show that while Adam achieved the best overall accuracy, the GAP variant provided a lighter architecture with fewer parameters and competitive performance. In contrast, the SGD-based training converged more slowly and underperformed on this dataset.

7 Optional Generalization Test

To further evaluate the robustness of the trained model, we tested it on both external and in-distribution images. For this experiment, we used one hand gesture photograph taken from the internet (out-of-distribution) and one image from the Kaggle dataset (in-distribution).

7.1 Results on Custom Images

Two images were tested: one showing the *scissors* gesture (external) and another showing the *paper* gesture (from Kaggle).

- On the **external scissors image** (Figure 9), the model incorrectly predicted the class as *paper*, with probabilities: *rock: 2.3%, paper: 93.5%, scissors: 4.2%*. This misclassification highlights the model's sensitivity to distribution shift — variations in lighting, background, and hand orientation not present in the training dataset.
- On the **Kaggle paper image** (Figure 10), the model correctly predicted the class as *paper*, with probabilities: *rock: 0.0%, paper: 64.5%, scissors: 35.5%*. This demonstrates that the model performs reliably on in-distribution data.

Figure 9: External test image (Internet): Scissors gesture (predicted as Paper).

Figure 10: Kaggle test image: Paper gesture (predicted correctly).

These results show that while the model generalizes well within the dataset, its performance may degrade on unseen real-world images. This underlines the importance of dataset diversity and domain adaptation for building robust models.

8 Conclusion

In this project, we addressed the Rock-Paper-Scissors image classification task using Convolutional Neural Networks. Through systematic experimentation, we evaluated three CNN architectures of increasing complexity:

- **Model 1 (Baseline CNN)**: a simple two-layer CNN that underfit the data, achieving only moderate accuracy (∼85%).
- **Model 2 (with Dropout)**: improved generalization by reducing overfitting, reaching about 92% accuracy.
- **Model 3 (BatchNorm + Dropout)**: the best-performing model, achieving ∼99% test accuracy with strong generalization.

We also conducted ablation experiments on optimizer choice and pooling strategies:

- The Adam optimizer provided faster and more stable convergence than SGD.
- Replacing Flatten with Global Average Pooling reduced the number of parameters while maintaining competitive accuracy (∼96%).

The final results show that Model 3, with Batch Normalization and Dropout, achieved state-of-the-art performance on the Kaggle test set. However, experiments with external images revealed sensitivity to distribution shifts such as

lighting, background, and orientation. This indicates that despite excellent in-distribution accuracy, the model's robustness in real-world scenarios requires further improvement.

Future work may include:

- Expanding the dataset with real-world and diverse hand gesture images.
- Using transfer learning with pre-trained CNNs (e.g., ResNet, MobileNet) for stronger feature extraction.
- Exploring lightweight CNN architectures suitable for deployment on mobile or embedded devices.

Overall, the project highlights the importance of balancing model complexity with proper regularization, validating with independent test sets, and testing robustness beyond the original dataset.

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.