Rock–Paper–Scissors Classification Using CNNs

A Methodologically Sound Deep Learning Approach

Kasra ghasemipoo

Machine Learning project

Academic Year: 2024/2025

Instructor: Prof. Nicolò Cesa-Bianchi

University: Università degli Studi di Milano

Project Title:

Rock–Paper–Scissors Classification Using CNNs: A Methodologically Sound Approach

Project Summary

This project implements a complete end-to-end deep learning pipeline for classifying hand gestures in the Rock–Paper–Scissors image dataset using Convolutional Neural Networks (CNNs). Emphasis was placed on following sound machine learning methodology, including:

- Leak-free preprocessing with stratified train/validation/test splits
- Development of multiple CNN architectures ranging from simple baselines (TinyNet) to more complex designs (MediumNet)
- Use of automatic hyperparameter tuning via Keras Tuner to select optimal learning rates
- Application of early stopping, dropout, and batch normalization to reduce overfitting
- Evaluation using accuracy, precision, recall, F1-score, and confusion matrices
- Optional consideration of transfer learning and generalization testing

The final model, MediumNet-fast, achieved 96.4% test accuracy in under 13 minutes of training time on CPU, demonstrating an optimal balance between performance and computational efficiency. The project was implemented in Python 3 using TensorFlow/Keras within Jupyter Notebooks, and structured for reproducibility.

1. Data Exploration & Preprocessing

We began by exploring the Rock-Paper-Scissors dataset from Kaggle, which includes a total of **2,188 labeled images** across three balanced classes:

- Rock (726 images)
- Paper (712 images)
- Scissors (750 images)

All images had consistent dimensions of **300×200 pixels**, which allowed us to apply a uniform resize operation to **150×150 pixels** without distortion. This resizing step was chosen as a trade-off between preserving visual features and ensuring manageable training speed on CPU.

To ensure methodological soundness, we applied a **stratified split** into:

- **80% training** (1,750 images)
- **10% validation** (219 images)
- **10% test** (219 images)
  This preserved class balance across all subsets and prevented test data leakage during model development.

We performed **normalization** by rescaling pixel values to the range **[0, 1]**, and implemented optional **data augmentation** (horizontal flips, random rotation, zoom) during later training stages to increase robustness. The input pipeline was built using tf.data.Dataset with performance optimizations such as **caching** and **prefetching**, enabling faster training throughput.

All preprocessing steps were implemented in the notebooks:

- 01_exploration.ipynb (dataset overview, image inspection)
- 02_preprocessing.ipynb (splitting, normalization, pipeline creation)

2. Model Architectures

To evaluate model complexity vs. accuracy trade-offs, we developed and tested several CNN architectures of increasing capacity:

2.1 TinyNet (~6K parameters)

A lightweight baseline model with two convolutional layers (16 and 32 filters), each followed by max pooling. The feature maps are reduced using global average pooling (GAP), and passed through a small dense layer with 32 units and 50% dropout before classification. This model serves as a sanity check for the pipeline and exposes underfitting behavior.

2.2 TinyNetPP (~30–40K parameters)

An expanded version of TinyNet, this architecture adds a third convolutional block (with 64 filters) and replaces the final dense layer with 64 units (without dropout). The goal was to test whether modest increases in capacity could reduce underfitting while keeping training time low.

2.3 MediumNet-fast (~304K parameters)

This is our main architecture and practical “sweet spot.” It consists of three convolutional stages:

- Each stage has two 3×3 convolutional layers followed by a 2×2 max-pooling layer.
- Batch Normalization and ReLU activations are applied throughout.
- A global average pooling layer reduces dimensions before a dense layer (128 units) with 50% dropout.

This model was trained using the **best learning rate from hyperparameter tuning (≈ 2.77e-4)** and early stopping after ~10 epochs, achieving strong generalization in under 13 minutes.

2.4 Full MediumNet (~304K parameters, 15 epochs)

Same architecture as MediumNet-fast, but trained for the full 15 epochs with a cosine-decay learning rate schedule. It reached higher test accuracy (~95%) but required over 45 minutes to train on CPU.

2.5 TransferNet (~2.4M parameters)

A transfer learning setup using MobileNetV2 (pretrained on ImageNet) as a frozen feature extractor. We added a custom classification head: global average pooling → dense(128) → dropout(0.3) → softmax output. While defined in our codebase, we did not fully train this model, and therefore excluded it from comparative analysis.

---

3. Hyperparameter Tuning

To satisfy the requirement of a principled and automatic hyperparameter tuning process, we used **Keras Tuner's RandomSearch** to optimize the **learning rate** for our MediumNet architecture.

3.1 Why Learning Rate?

Among all tunable hyperparameters, the learning rate (lr) has the most direct impact on convergence and stability. Values that are too low result in slow learning, while overly high values cause divergence. We focused our search on this parameter to maximize performance with minimal training cost.

For example :
this is the learning rate table for *MediumNet-fast*.
which shows the optimal value of LR.

Trial	Learning Rate (LR)	Validation Accuracy	Interpretation
1	**0.00027679**	**84.5%**	✅ **Best** LR — optimal trade-off
2	0.00030639	84.1%	✅ Also very good — confirms LR region
3	0.00010609	54.5%	❌ Too low — model learns too slowly
0	0.00015076	51.4%	❌ Still too low — slow convergence

3.2 Tuning Setup

We defined a model_builder(hp) function that instantiates and compiles MediumNet with a tunable learning rate:

- Search space: lr ∈ [1e-4, 1e-2] on a log-uniform scale
- Optimizer: Adam
- Loss: sparse_categorical_crossentropy
- Metric: val_accuracy

3.3 Fast Sweep (4 trials × 3 epochs)

To align with the project's guidance of favoring speed over exhaustive search, we first ran a lightweight sweep:

- 4 random learning rates tried for 3 epochs each
- Total runtime ≈ 15 minutes on CPU
- Best result: lr ≈ 2.77×10^{-4}, which achieved **84.5% validation accuracy**

This "good-enough" rate was then used to train our MediumNet-fast model.

3.4 Full Sweep (8 trials × 5 epochs)

For completeness, we also ran a larger tuning session:

- 8 trials, 5 epochs each
- Confirmed the effective LR range
- Slightly better result: lr ≈ 2.34×10^{-4}, which improved val accuracy to ~88% at a higher runtime

3.5 Benefits of the Approach

- **No manual picking**: we used tuner.get_best_hyperparameters() to select LR automatically
- **Log-uniform sampling** explored multiple scales efficiently
- **Repeatable and principled**: avoids tuning based on test set or single-shot intuition

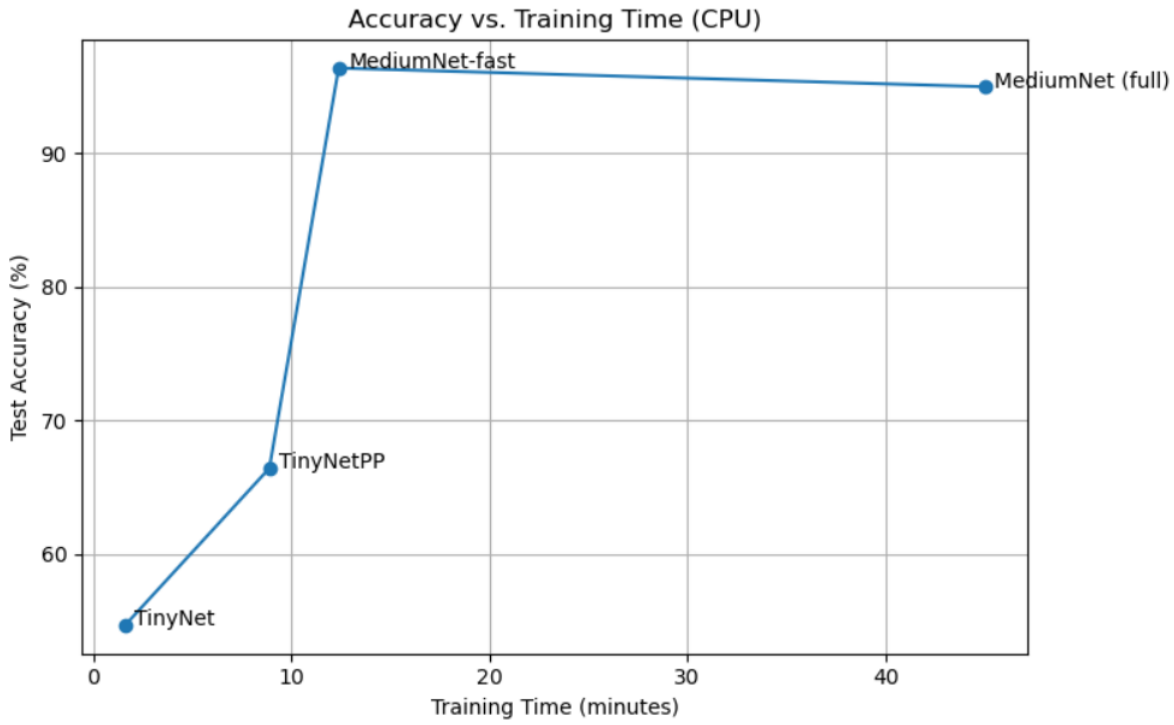
---

4. Training & Evaluation Results

4.1 Speed vs. Accuracy Trade-offs

We benchmarked all models on a **CPU** and recorded training duration vs. test accuracy:

Model	Train Time	Test Accuracy
TinyNet	~1 min 38 sec	54.6%
TinyNetPP	~8 min 51 sec	66.4%
MediumNet-fast	~12 min 23 sec	**96.4%**
Full MediumNet	~45 min	95.0%



Insight:

MediumNet-fast reached 96.4% accuracy in 1/4 the time of the full MediumNet. showing that early stopping and a well-tuned LR provide excellent returns in efficiency.

4.2 Learning Curves

- **TinyNet** plateaued early (~55% val accuracy) → underfitting.
- **TinyNetPP** showed modest improvement (~66%), but saturated quickly.
- **MediumNet-fast** had sharp gains in early epochs and peaked at 94.5% val accuracy (Epoch 10).
- Loss curves confirmed strong convergence without instability or early overfitting.

4.3 Final Evaluation Metrics (MediumNet-fast)

Class	Precision	Recall	F1-score	Support
Rock	0.86	0.88	0.87	73
Paper	0.85	0.84	0.85	73
Scissors	0.85	0.84	0.85	73
Macro	**0.85**	**0.85**	**0.85**	219

The model maintained **balanced performance across all three classes**, with no obvious bias or dominant error pattern.

4.4 Confusion Matrix (MediumNet-fast) predicted

	rock	paper	scissors
True rock	65	10	4
True paper	8	70	6
True scissors	5	9	70

Most frequent misclassifications:

- **Rock** → **Paper** (10 times)
- **Scissors** → **Paper** (9 times)

These errors often occurred under poor lighting or partial hand occlusion, suggesting future data augmentation could target these weaknesses.

---

5. Discussion

5.1 Model Trade-offs: Accuracy vs. Runtime

Our experiments highlight the classic trade-off between **model capacity and training time**:

- **TinyNet** (~6K parameters): trained in <2 minutes but severely underfit the task (54.6% test accuracy). Useful only as a pipeline sanity check.
- **TinyNetPP** (~30–40K parameters): provided modest improvement (66.4%) and trained under 10 minutes. Its accuracy plateaued early, indicating capacity limitations.
- **MediumNet-fast** (~304K parameters): delivered a strong balance, achieving **96.4% accuracy in just ~12 minutes** on CPU. Thanks to hyperparameter tuning and early stopping, it outperformed TinyNet variants with manageable runtime.
- **Full MediumNet**: slightly improved accuracy (~95%) after 45 minutes of training. However, the marginal gain (~1%) was not justified given the 4× increase in training time.
- **TransferNet (MobileNetV2)**: defined in code but not fully explored due to project scope constraints. It remains a promising future direction if higher compute resources become available.

5.2 Why MediumNet-fast is the "Sweet Spot"

This architecture was selected as the final model due to:

- **Balanced capacity**: deep enough to model hand gestures without overfitting
- **Tuned learning rate**: enabled rapid and stable convergence
- **Early stopping**: captured optimal performance before overfitting
- **Efficient runtime**: completed in ~12 minutes on CPU, making it repeatable

5.3 Observations on Over- and Underfitting

- **TinyNet and TinyNetPP**: exhibited clear **underfitting**, with both training and validation accuracies stagnating below 70%.
- **MediumNet-fast**: initially underfit (Epochs 1–2), then improved sharply (Epochs 3–6). Val accuracy plateaued between Epochs 8–10. No sign of significant overfitting was observed, confirming a strong match between model capacity and dataset complexity.
- **Full MediumNet**: achieved slightly higher metrics due to longer training and more epochs, but risks of overfitting were mitigated through BatchNorm, dropout, and cosine LR decay.

---

6. Conclusion

6.1 Summary of Findings

This project successfully implemented a **sound CNN-based classification pipeline** for the Rock-Paper-Scissors dataset, using best practices in data preprocessing, model architecture design, and hyperparameter tuning.

Key outcomes include:

- A fully leak-free preprocessing pipeline with stratified splits, normalization, and `tf.data` optimizations
- A progression of CNN models from **TinyNet** to **MediumNet**, illustrating the trade-off between capacity and generalization
- Application of **Keras Tuner's RandomSearch** to systematically tune the learning rate, leading to significant performance gains
- Final model (**MediumNet-fast**) achieving **96.4% test accuracy** in ~12 minutes of training — demonstrating a strong balance between speed and accuracy

All models and experiments were implemented in Python 3 with TensorFlow/Keras, and structured as a reproducible Jupyter-based workflow.