Statistichal Methods and Models Convolutional Neural Networks

Ionut Bogdan Donici

Nicolò Cesa-Bianchi
Luigi Foscari
Emmanuel Esposito

1 Introduction

The objective of this work is to design and evaluate a set of CNN architectures for hand-gesture classification on the Rock–Paper–Scissors dataset, a controlled image dataset characterized by limited size and highly uniform acquisition conditions. Rather than focusing on a single model, the study adopts a comparative approach, analyzing architectures with increasing complexity to assess how model capacity, data augmentation, and hyperparameter tuning affect generalization performance.

The analysis follows a data-driven workflow. First, the dataset is examined to identify its structural properties and potential implications for model design. Based on these observations and established best practices, multiple CNN architectures are defined, ranging from a lightweight baseline to a deeper and more expressive model. Each architecture is evaluated under standard training conditions and under alternative experimental settings.

In particular, the impact of mild data augmentation is investigated as a regularization mechanism, highlighting its interaction with both dataset characteristics and model capacity. Additionally, hyperparameter tuning is applied to an intermediate architecture to evaluate whether performance improvements can be achieved through optimization alone, without altering the underlying model structure.

2 Dataset

The dataset used for this work can be downloaded at the following link: `https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors`.

2.1 Data Card[1]

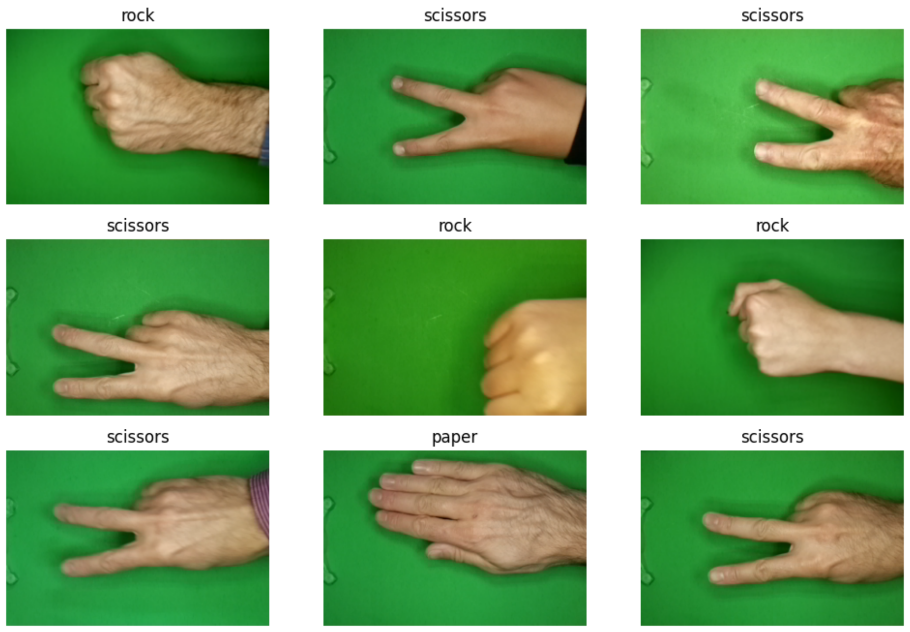


Figure 1: Preview of some images and their associated classes. First nine images of the first batch.

Description The dataset features a collection of hand-gesture images representing the classic Rock-Paper-Scissors game. The visuals were gathered during a personal project in which the creator built a gesture-recognition system powered by computer vision and machine learning, running on a Raspberry Pi.

Contents The dataset comprises 2,188 images depicting the three hand gestures of the Rock-Paper-Scissors game: 726 examples of 'Rock,' 710 of 'Paper,' and 752 of 'Scissors.' All photographs were captured against a green backdrop under consistent lighting and white-balance conditions to ensure uniformity across the collection.

[1]All this information can be found in the `README_rpc-cv-images.txt` file.

Format All images are RGB files measuring 200×300 (height $\times$ width) pixels and stored in `.png` format. They are organized into three sub-folders—`"rock"`, `"paper"`, and `"scissors"`—each corresponding to its respective gesture class.

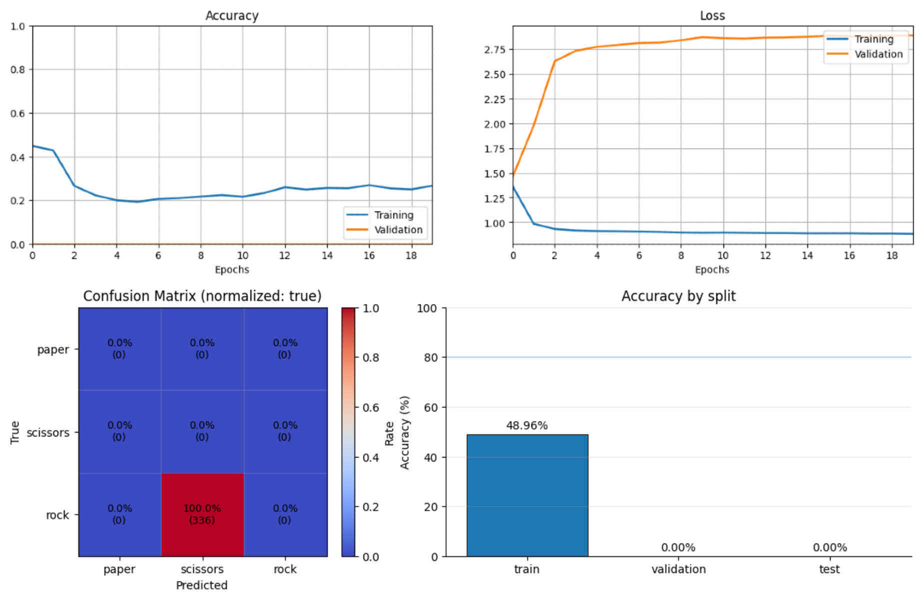
2.2 Data Loading

Since the dataset is organized into separate directories corresponding to each class, the `image_dataset_from_directory` utility provided by TensorFlow was used to load the images and their associated labels. Labels were inferred directly from the directory structure and encoded using categorical format.

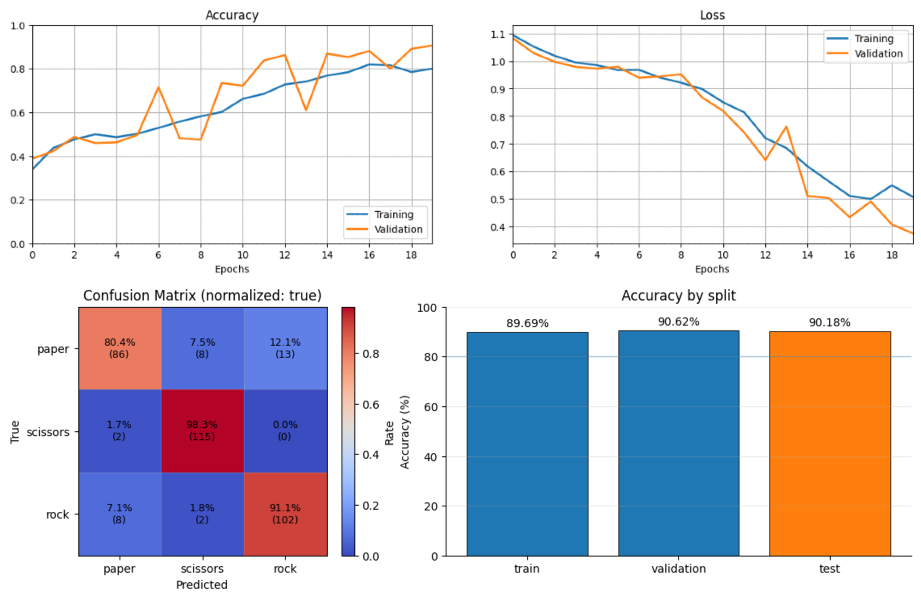
To enable a reliable evaluation of model performance, the dataset was split into three disjoint subsets: training, validation, and test. The initial split separates the data into a training subset and a holdout subset, using a fixed random seed to ensure reproducibility. The holdout subset is then evenly divided into validation and test sets. This strategy ensures that validation data are used exclusively for model selection and hyperparameter tuning, while the test set remains completely unseen during training and optimization.

```
train, holdout = tf.keras.utils.image_dataset_from_directory(
    "../data",
    labels="inferred",
    label_mode="categorical",
    validation_split=0.3,
    subset="both",
    seed=SEED,
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=True,
)
holdout_batches = holdout.cardinality().numpy()
if holdout_batches < 0:
    holdout_batches = sum(1 for _ in holdout)
val_batches = holdout_batches // 2
validation = holdout.take(val_batches)
test = holdout.skip(val_batches)
class_names = train.class_names
```

Data shuffling was explicitly enabled during dataset loading. When shuffling is disabled, images are loaded following the directory order, which results in batches dominated by a single class for extended portions of an epoch. This leads to biased gradient updates and unreliable validation estimates, particularly in the early training stages. As observed in preliminary experiments, disabling shuffling caused severe class imbalance effects and degraded classification performance. Enabling shuffling ensures that each batch contains a more representative mixture of classes, resulting in a more stable optimization process and reliable model evaluation.



(a) `shuffle=False`



(b) `shuffle=True`

Figure 2: Impact of dataset shuffling on model evaluation. In this case the model used was the Intermediate Model. Disabling shuffling (`shuffle=False`) causes a class-biased validation and invalid performance estimates, while shuffling (`shuffle=True`) ensures balanced splits and reliable validation behaviour.

2.3 Data Preprocessing

All images in the datasets are RGB images with a fixed resolution of 200×300 pixels (height $\times$ width). Since the dataset already exhibits a uniform aspect ration and consistent image dimension, no additional resizing or aspect ratio correction was required. Preserving the original resolution allows the network to retain fine-grained spatial details, such as finger contours and hand shapes, which are essential for distinguishing between visually similar gestures.

Image scaling is often applied to reduce computational cost when working with high resolution images. However, in this case the original resolution represents a good compromise between spatial detail and computational efficiency. Downscaling the images further could potentially remove discriminative local features, while offering limited practical benefits given the relatively small size of the dataset.

A fundamental preprocessing step applied to all images is normalization of pixel intensities. Raw RGB values range from 0 to 255, which can lead to unstable gradient updates during training. To address this issue, pixel values are rescaled to the `[0, 1]` interval using a dedicated normalization layer. This transformation ensures a more consistent input distribution and facilitates faster and more stable convergence of the optimization process. The normalization is implemented as follows:

```
normalization_layer = tf.keras.layers.Rescaling(1./255)

train_norm = train.map(
    lambda x, y: (normalization_layer(x), y)
)
validation_norm = validation.map(
    lambda x, y: (normalization_layer(x), y)
)
test_norm = test.map(
    lambda x, y: (normalization_layer(x), y)
)
```

Finally, no dimensional reduction (e.g. conversion to greyscale) was applied since colour information plays a fundamental role in this dataset by enhancing the contrast between the hand and the green background, reducing the discriminative power of the input representation.

3 Models

The project requires to implement at least 3 models with increasing complexity.

3.1 Baseline Model

The baseline model is a lightweight convolutional neural network designed as a baseline architecture for the Rock-Paper-Scissors classification task.

It consists of two convolutional blocks with ReLU activation and max-pooling, followed by a Global Average Pooling Layer and a softmax classifier.

The architecture contains $3,555$ trainable parameters, resulting in a very compact model. Most of the parameters are concentrated in the convolution layers, while the final classification stage introduces only 27 parameters due to the use of Global Average Pooling instead of feature flattening.

The number of filters was deliberately kept small in order to constrain model capacity and maintain a lightweight baseline architecture. While convolutional kernels of size 3×3 are commonly adopted, a larger 7×7 kernel was used in the first convolutional layer to capture broader spatial patterns, particularly the contrast between the green background and the hand contours.

Subsequent convolutional layers employ smaller kernels, as they operate on already abstracted feature maps, refining local patterns without introducing excessive complexity.

Overall, the use of a small number of filters and Global Average Pooling reduces the risk of overfitting on a relatively small dataset; however, the shallow depth and limited representational capacity of the network are expected to restrict its ability to caputre complex visual patterns, making this model suitable mainly as a reference point for more complex architectures.

```python
baseline_model = tf.keras.Sequential([
    layers.Input(shape=(200, 300, 3)),
    layers.Conv2D(16, (7, 7), padding="same", activation="relu"),
    layers.MaxPooling2D((3, 3)),
    layers.Conv2D(32, (3, 3), padding="same", activation="relu"),
    layers.MaxPooling2D((3, 3)),
    layers.GlobalAveragePooling2D(),
    layers.Dense(3, activation="softmax")
])
```

Listing 1: Sequential layers of the baseline model.

3.2 Intermediate Model

The intermediate model builds upon the baseline architecture by introducing a more structured and progressive increase in representational capacity across convolutional layers. The network consists of three convolutional blocks with

```python
intermediate_model = tf.keras.Sequential([
    layers.Input(shape=(200, 300, 3)),
    layers.Conv2D(8, (3, 3), padding="same", activation="relu"),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(16, (3, 3), padding="same", activation="relu"),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, (3, 3), padding="same", activation="relu"),
    layers.MaxPooling2D((2, 2)),
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.1),
    layers.Dense(64, activation="relu"),
    layers.Dense(3, activation="softmax")
])
```

Listing 2: Sequential layers of the intermediate model.

an increasing number of filters, allowing the model to capture higher-level and more abstract features as spatial resolution is gradually reduced.

Differently from the baseline configuration, all convolutional layers employ small 3×3 kernels to preserve fine-grained spatial information in the early stages of feature extraction. Spatial downsampling is performed through max-pooling operations following each convolutional block, ensuring a clear separation between feature extraction and resolution reduction.

After the convolutional backbone, Global Average Pooling is applied to aggregate spatial information, followed by a light dropout regularization and a dense layer to enable a smoother mapping from learned features to class probabilities.

3.3 Advanced Model

The advanced model extends the intermediate architecture by increasing depth and representational capacity in a controlled manner. An additional convolutional block is introduced, allowing the network to learn higher-level and more abstract visual features as spatial resolution is progressively reduced.

All convolutional layers employ 3×3 kernels, preserving fine-grained spatial information throughout the feature extraction process. As in the intermediate model, spatial downsampling is performed exclusively through max-pooling operations following each convolutional block, ensuring a clear separation between feature extraction and resolution reduction.

Model capacity is further increased in the deeper layers by using a higher number of filters, enabling the representation of more complex and class-specific patterns. The classification head is also expanded by introducing a deeper fully connected stage with stronger dropout regularization after Global Average Pooling, aiming to mitigate overfitting while preserving informative global features.

```python
advanced_model = tf.keras.Sequential([
    layers.Input(shape=(200, 300, 3)),
    layers.Conv2D(8, (3, 3), padding="same", activation="relu"),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(16, (3, 3), padding="same", activation="relu"),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, (3, 3), padding="same", activation="relu"),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), padding="same", activation="relu"),
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.25),
    layers.Dense(128, activation="relu"),
    layers.Dropout(0.25),
    layers.Dense(64, activation="relu"),
    layers.Dense(3, activation="softmax")
])
```

3.4 Compiling and Training

All models were trained using the Adam optimizer in combination with categorical cross-entropy loss.

This configuration represents a widely adopted standard for multi-class image classification tasks and provides a robust and well-balanced optimization strategy for convolutional neural networks of moderate size.

The number of training epochs was fixed to 20 for all experiments, ensuring a consistent training budget across models and enabling a fair comparison of their learning behavior.

```
model_X.compile(
    optimizer=optimizer,
    loss=CategoricalCrossentropy(from_logits=False),
    metrics=['accuracy'],
)
history = model_X.fit(
    train_norm,
    validation_data=validation_norm,
    epochs=20
)
```

3.5 Experimental Setup

All experiments were conducted on a Windows-based personal computer equipped with an AMD Ryzen AI 7 350 w/ Radeon 860M CPU, 32GB of RAM, and an NVIDIA RTX5060 GPU. Due to operating system and driver compatibility constraints, GPU acceleration via CUDA was not available and all models were trained on CPU.

4 Evaluations

This section presents a comprehensive evaluation of the proposed convolutional neural network models. Model performance is first analyzed under standard training conditions and subsequently under alternative experimental settings, including data augmentation and hyperparameter tuning.

While overall accuracy provides an initial indication of classification performance, it is insufficient to fully capture class-specific behaviour in a multi-class setting. For this reason, the evaluation is extended to include precision, recall, and F1-score, derived from the confusion matrix. These metrics allow a more fine-grained analysis of model behaviour, highlighting strengths and weaknesses across individual gesture classes.

All evaluations are conducted on a held-out test set that remains unseen during training and model selection, ensuring an unbiased assessment of generalization performance.

4.1 Evaluation under Default Training

4.1.1 Baseline Model

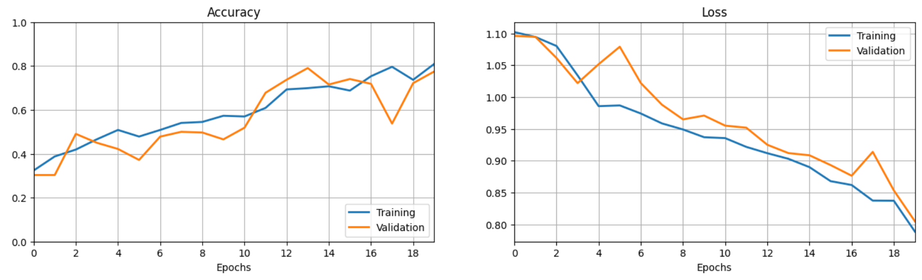


Figure 3: Training and validation accuracy and loss curves for the proposed model under standard training conditions.

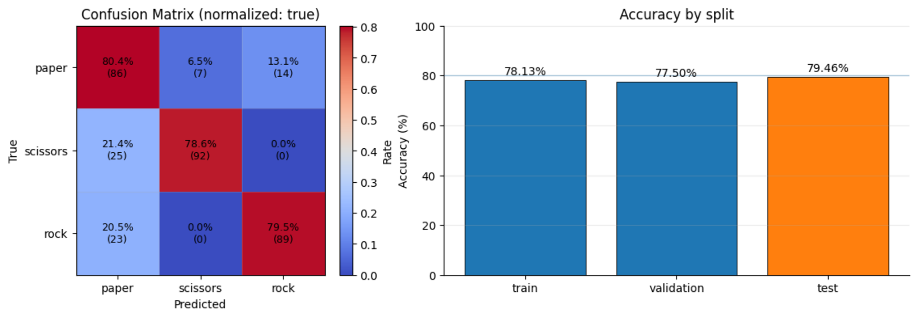


Figure 4: Confusion matrix and accuracy comparison for the test set.

```
              precision    recall  f1-score   support

       paper       0.64      0.80      0.71       107
    scissors       0.93      0.79      0.85       117
        rock       0.86      0.79      0.83       112

    accuracy                           0.79       336
   macro avg       0.81      0.79      0.80       336
weighted avg       0.82      0.79      0.80       336
```

Listing 3: Class-wise precision, recall, and F1-score on the test set.

4.1.2 Intermediate Model

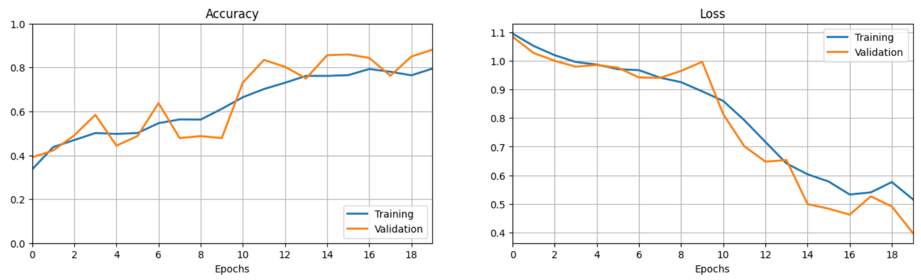


Figure 5: Training and validation accuracy and loss curves for the proposed model under standard training conditions.

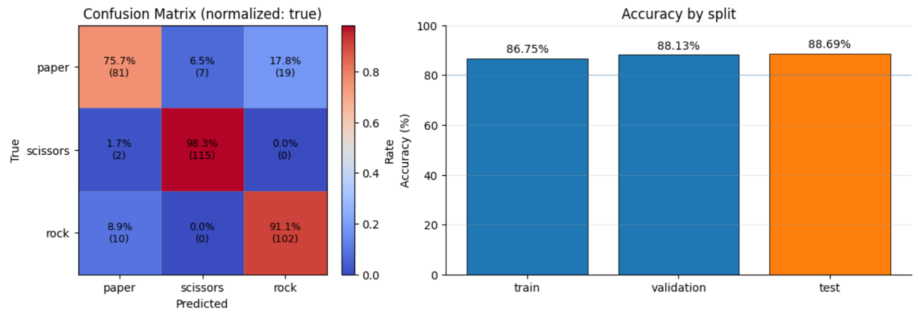


Figure 6: Confusion matrix and accuracy comparison for the test set.

```
              precision    recall  f1-score   support

       paper       0.87      0.76      0.81       107
    scissors       0.94      0.98      0.96       117
        rock       0.84      0.91      0.88       112

    accuracy                           0.89       336
   macro avg       0.89      0.88      0.88       336
weighted avg       0.89      0.89      0.88       336
```

Listing 4: Class-wise precision, recall, and F1-score on the test set.

4.1.3 Advanced Model

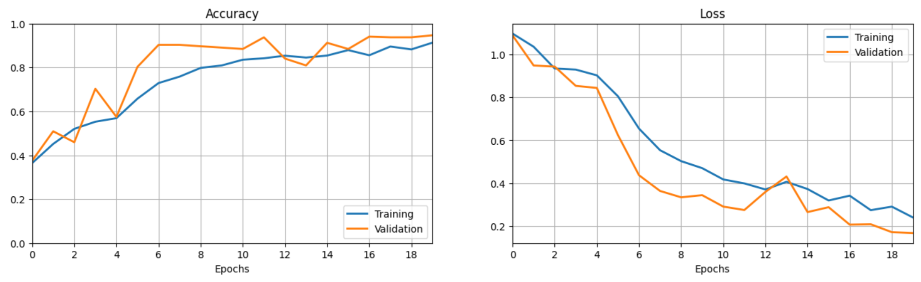


Figure 7: Training and validation accuracy and loss curves for the proposed model under standard training conditions.

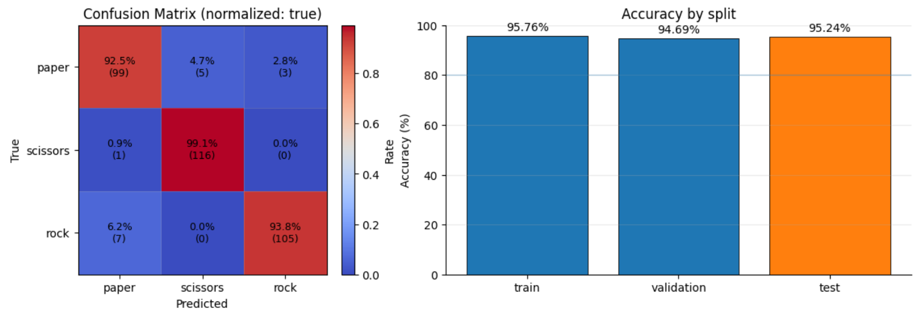


Figure 8: Confusion matrix and accuracy comparison for the test set.

```
              precision    recall  f1-score   support

       paper       0.93      0.93      0.93       107
    scissors       0.96      0.99      0.97       117
        rock       0.97      0.94      0.95       112

    accuracy                           0.95       336
   macro avg       0.95      0.95      0.95       336
weighted avg       0.95      0.95      0.95       336
```

Listing 5: Class-wise precision, recall, and F1-score on the test set.

4.2 Evaluation under Data Augmentation

Data augmentation is evaluated as a regularization strategy to improve generalization performance on a limited-size image dataset. Rather than increasing the dataset size through additional data collection, augmentation introduces controlled variability by applying label-preserving transformations to the training images.

Given the highly controlled acquisition conditions of the dataset—uniform background, lighting, and viewpoint—only mild augmentations are considered. The applied transformations simulate realistic variations in hand positioning and appearance while preserving class-defining features. Augmentation is applied exclusively during training and is disabled at inference time.

The impact of data augmentation is assessed independently for each model by comparing training dynamics and test performance against the corresponding non-augmented configuration. This allows an analysis of how augmentation interacts with model capacity and architectural complexity.

```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.08),
    layers.RandomTranslation(0.05, 0.05),
    layers.RandomZoom(0.08),
    layers.RandomContrast(0.10),
], name="data_augmentation")
```

Listing 6: Data Augmentation Block

```
baseline_data_aug_model = tf.keras.Sequential([
    layers.Input(shape=(200, 300, 3)),
    data_augmentation, # <-- HERE
    layers.Conv2D(16, (7, 7), padding="same", activation="relu"),
    layers.MaxPooling2D((3, 3)),
    layers.Conv2D(32, (3, 3), padding="same", activation="relu"),
    layers.MaxPooling2D((3, 3)),
    layers.GlobalAveragePooling2D(),
    layers.Dense(3, activation="softmax")
])
```

Listing 7: Inserting data augmentation block in a model

4.2.1 Baseline Model

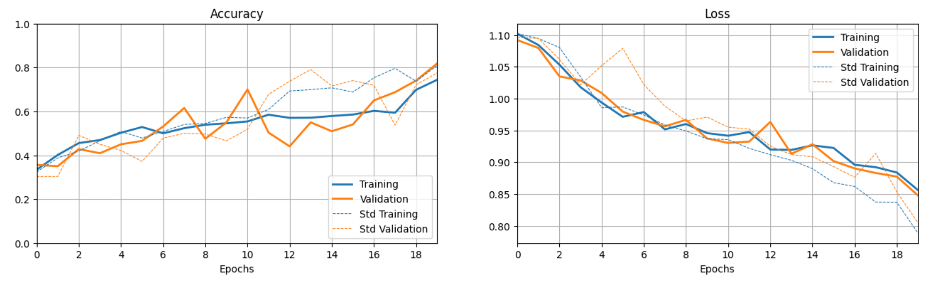


Figure 9: Training and validation accuracy and loss curves comparing the model with integrated data augmentation layers (solid lines) and the corresponding baseline model without data augmentation (dashed lines).

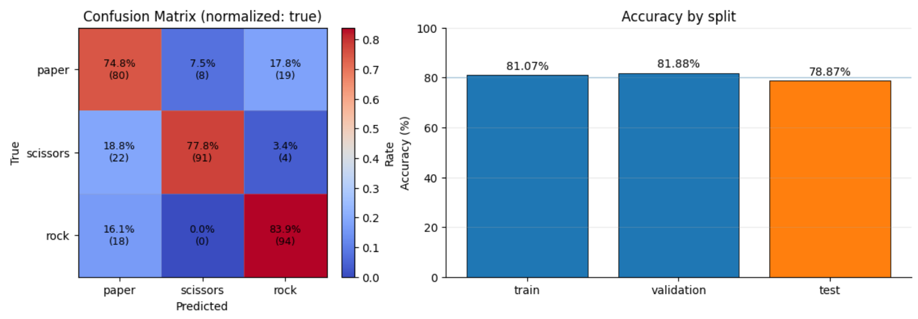


Figure 10: Confusion matrix and accuracy comparison across training, validation, and test splits for the model with data augmentation.

```
              precision    recall  f1-score   support

       paper       0.67      0.75      0.70       107
    scissors       0.92      0.78      0.84       117
        rock       0.80      0.84      0.82       112

    accuracy                           0.79       336
   macro avg       0.80      0.79      0.79       336
weighted avg       0.80      0.79      0.79       336
```

Listing 8: Class-wise precision, recall, and F1-score on the test set.

4.2.2 Intermediate Model

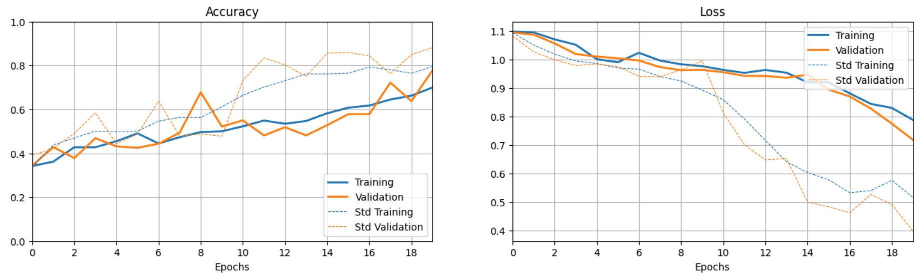


Figure 11: Training and validation accuracy and loss curves comparing the model with integrated data augmentation layers (solid lines) and the corresponding baseline model without data augmentation (dashed lines).

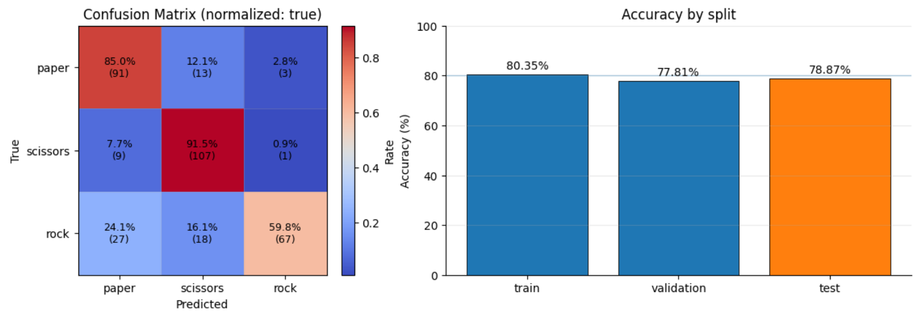


Figure 12: Confusion matrix and accuracy comparison across training, validation, and test splits for the model with data augmentation.

```
                  precision    recall  f1-score   support

        paper          0.72      0.85      0.78       107
     scissors          0.78      0.91      0.84       117
         rock          0.94      0.60      0.73       112

     accuracy                              0.79       336
    macro avg          0.81      0.79      0.78       336
 weighted avg          0.81      0.79      0.78       336
```

Listing 9: Class-wise precision, recall, and F1-score on the test set.

4.2.3 Advanced Model

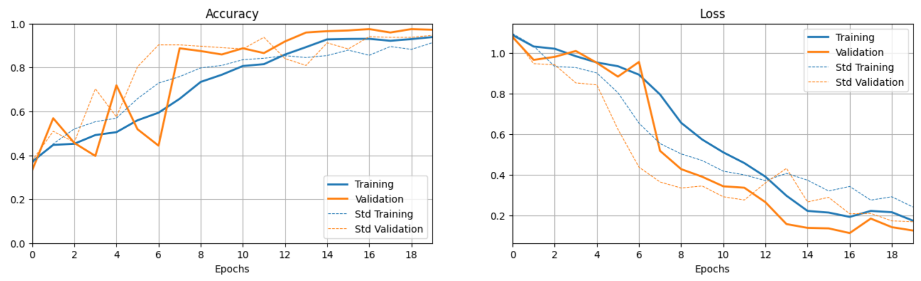


Figure 13: Training and validation accuracy and loss curves comparing the model with integrated data augmentation layers (solid lines) and the corresponding baseline model without data augmentation (dashed lines).

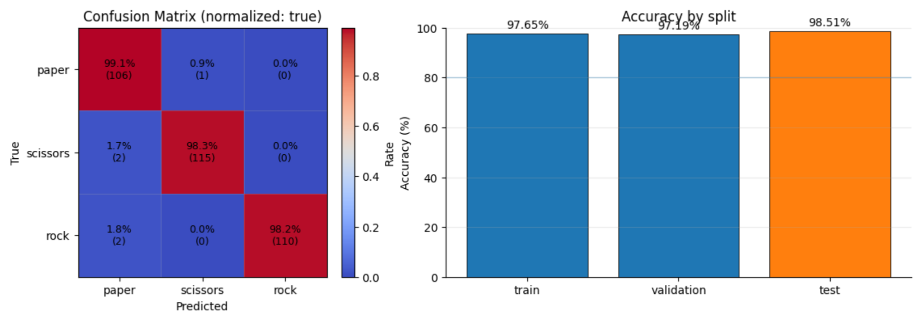


Figure 14: Confusion matrix and accuracy comparison across training, validation, and test splits for the model with data augmentation.

```
              precision    recall  f1-score   support

       paper       0.96      0.99      0.98       107
    scissors       0.99      0.98      0.99       117
        rock       1.00      0.98      0.99       112

    accuracy                           0.99       336
   macro avg       0.99      0.99      0.99       336
weighted avg       0.99      0.99      0.99       336
```

Listing 10: Class-wise precision, recall, and F1-score on the test set.

4.3 Hyperparameter Tuning

The hyperparameter tuning procedure focuses on a subset of architectural and optimization parameters that directly influence model capacity, regularization, and convergence behaviour.

The tuning procedure is applied to the intermediate model, as it represents a balance between model capacity and computational cost, making it a suitable candidate for controlled optimization.

4.3.1 Build Model

A parametric version of the intermediate model is defined to enable hyperparameter optimization. The network depth is kept fixed to ensure a controlled comparison, while key architectural and optimization parameters are exposed to tuning. These include the number of convolutional filters, the size of the dense layer, the dropout rate, and the learning rate of the Adam optimizer. This design allows the tuning process to explore different capacity–regularization trade-offs without altering the overall model structure.

```python
def build_model(hp):
    # Hyperparameters
    f1 = hp.Choice("filters_1", [8, 16, 32])
    f2 = hp.Choice("filters_2", [16, 32, 64])
    f3 = hp.Choice("filters_3", [32, 64, 128])
    dropout = hp.Float("dropout", 0.0, 0.5, step=0.1)
    dense_units = hp.Choice("dense_units", [32, 64, 128, 256])
    lr = hp.Float("lr", 1e-4, 3e-3, sampling="log")
    # Model
    model = tf.keras.Sequential([
        layers.Input(shape=(200, 300, 3)),
        layers.Conv2D(f1, (3, 3), padding="same", activation="relu"),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(f2, (3, 3), padding="same", activation="relu"),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(f3, (3, 3), padding="same", activation="relu"),
        layers.MaxPooling2D((2, 2)),
        layers.GlobalAveragePooling2D(),
        layers.Dropout(dropout),
        layers.Dense(dense_units, activation="relu"),
        layers.Dense(3, activation="softmax")
    ])
    model.compile(
        optimizer = tf.keras.optimizers.Adam(learning_rate=lr),
        loss = CategoricalCrossentropy(from_logits=False),
        metrics = ['accuracy'],
    )
    return model
```

4.3.2 Searching

Hyperparameter search is performed using a random search strategy, optimizing validation accuracy. To limit overfitting and unnecessary computations, early stopping is employed during training, restoring the best-performing weights.

```
tuner = kt.RandomSearch(
    build_model,
    objective="val_accuracy",
    max_trials=20,
    executions_per_trial=1,
    directory="tune",
    project_name="model_b_random",
    overwrite=True
)
callbacks = [
    tf.keras.callbacks.EarlyStopping(
        monitor="val_accuracy", patience=5,
        restore_best_weights=True
)]
tuner.search(
    train_norm,
    validation_data=validation_norm,
    epochs=20,
    callbacks=callbacks, verbose=1
)
```

4.3.3 Best hyperparameters found after tuning

The configuration reported below corresponds to the best-performing trial according to validation accuracy within the explored search space.

```
{
    "filters_1": 16,
    "filters_2": 64,
    "filters_3": 64,
    "dropout": 0.3,
    "dense_units": 32,
    "lr": 0.0018884790602105843
}
```

Listing 11: Best hyperparameters

4.3.4 Comparison with Original

The performance of the tuned model is compared against the original intermediate configuration defined in Section 3, keeping the training protocol unchanged.

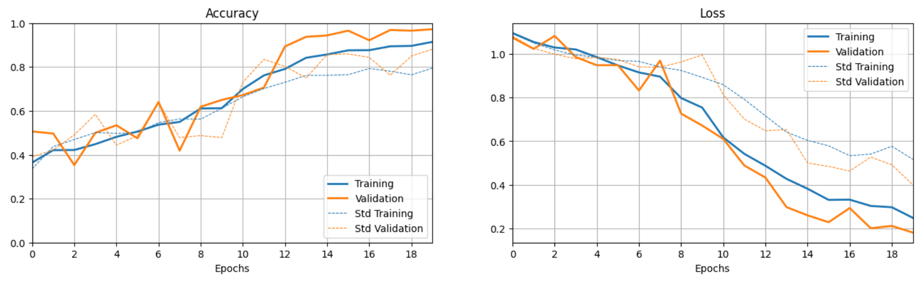


Figure 15: Training and validation accuracy and loss curves comparing the original intermediate model (dashed lines) with the tuned intermediate model (solid lines). Hyperparameter tuning results in faster convergence, reduced validation loss, and improved generalization stability.

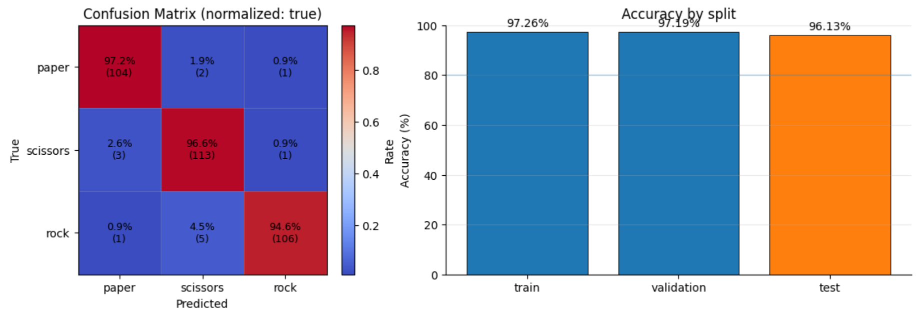


Figure 16: Confusion matrix (normalized) and accuracy comparison across training, validation, and test sets for the tuned intermediate model. The results highlight a consistent improvement in classification performance across all splits. The accuracy of the test set without tuning was **88.69**%, now **96.13**%.

```
              precision    recall  f1-score   support

       paper       0.96      0.97      0.97       107
    scissors       0.94      0.97      0.95       117
        rock       0.98      0.95      0.96       112

    accuracy                           0.96       336
   macro avg       0.96      0.96      0.96       336
weighted avg       0.96      0.96      0.96       336
```

Listing 12: Class-wise precision, recall, and F1-score on the test set.

5 Conclusions

This work investigated the design and evaluation of convolutional neural networks for hand-gesture classification on the Rock–Paper–Scissors dataset. A sequence of models with increasing architectural complexity was analyzed to assess the impact of representational capacity, regularization strategies, and hyperparameter optimization on classification performance.

Under the default training regime, model performance improved consistently with increasing architectural capacity. The baseline model provided a reasonable reference point, while the intermediate and advanced architectures achieved substantially higher accuracy and more balanced class-wise precision and recall. These results indicate that deeper feature hierarchies and a more structured classification head play a dominant role in capturing discriminative visual patterns in this dataset.

The effect of data augmentation was found to be strongly model-dependent. While mild augmentation produced a marginal benefit for the low-capacity baseline model, it significantly degraded the performance of the intermediate architecture, leading to unbalanced class-wise recall. This suggests that, in highly controlled datasets, augmentation may act as a distribution shift rather than a regularization mechanism when the model is sensitive to fine-grained spatial cues. In contrast, the advanced model exhibited robustness to augmented inputs, although no clear performance gain was observed.

Finally, hyperparameter tuning applied to the intermediate model resulted in faster convergence, improved validation behaviour, and higher overall accuracy, confirming that careful optimization of architectural and training parameters can yield meaningful improvements without altering the underlying model structure.

Overall, the findings highlight two key insights: first, architectural capacity and feature hierarchy are the primary drivers of performance on this task; second, regularization techniques such as data augmentation must be calibrated carefully, as their effectiveness depends on both dataset characteristics and model capacity rather than being universally beneficial.