UNIVERSITÀ
DEGLI STUDI
DI MILANO

Building a CNN for Rock-Paper-Scissors Classification

Luca Rigamonti 08920A

20th october 2025

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.

1 Introduction

This project applies classification model known as Convolutional Neural Network (CNN) to distinguishing between images of Rock, Paper, and Scissors gestures. Several different CNN architectures were tested to determine which one performed the best on the input data.
The dataset contains a total of 2188 images corresponding to the 'Rock' (726 images), 'Paper' (710 images) and 'Scissors' (752 images) hand gestures of the Rock-Paper-Scissors game. All image are taken on a green background with relatively consistent ligithing and white balance.

2 Data

All images are RGB images of 300 pixels wide by 200 pixels high in .png format. The images are separated in three sub-folders named 'rock', 'paper' and 'scissors' according to their respective class.
To begin with, the data were unzipped, and the input data were used to create a pandas DataFrame in which each image was labeled according to whether it depicted a rock, paper or scissor's image. Creating a DataFrame allows us to efficiently keep track of the labels and corresponding file names for each image in the dataset.

Moreover, it makes also easier to split the dataest into training, validation,

	paths	label
720	/content/rps-cv-images/rock/T6biQxOCDlM5ad7k.png	rock
721	/content/rps-cv-images/rock/PxixapAFqLmRaP6k.png	rock
722	/content/rps-cv-images/rock/CZMVXF6ReNFTmQu3.png	rock
723	/content/rps-cv-images/rock/qnvJFE0EKgfw4xr8.png	rock
724	/content/rps-cv-images/rock/2O9XPBJRT119drWX.png	rock
725	/content/rps-cv-images/rock/GRUlZRON6TdcmAOe.png	rock
726	/content/rps-cv-images/scissors/xAnfzBYnsnuru9...	scissors
727	/content/rps-cv-images/scissors/0zoQAmDFXehOZs...	scissors
728	/content/rps-cv-images/scissors/CLCnzerdshjQe3...	scissors
729	/content/rps-cv-images/scissors/0Ug54ifXRqqlZS...	scissors

Figure 1: ten-row block of the DataFrame

and testing sets.
Another factor to consider is that I will use a different format for the images; in fact I set the image format to 96x96 pixels to have faster training (the square

shape of the image also makes it easier to manage the feature maps through the convolutional and pooling layers).

	paths	label
720	/content/rps-cv-images/rock/T6biQxOCDlM5ad7k.png	rock
721	/content/rps-cv-images/rock/PxixapAFqLmRaP6k.png	rock
722	/content/rps-cv-images/rock/CZMVXF6ReNFTmQu3.png	rock
723	/content/rps-cv-images/rock/qnvJFE0EKgfw4xr8.png	rock
724	/content/rps-cv-images/rock/2O9XPBJRT119drWX.png	rock
725	/content/rps-cv-images/rock/GRUIZRON6TdcmAOe.png	rock
726	/content/rps-cv-images/scissors/xAnfzBYnsnuru9...	scissors
727	/content/rps-cv-images/scissors/0zoQAmDFXehOZs...	scissors
728	/content/rps-cv-images/scissors/CLCnzerdshjQe3...	scissors
729	/content/rps-cv-images/scissors/0Ug54ifXRqqlZS...	scissors

Figure 2: ten-row block of the DataFrame

3 Models

In this project I will analyze three different models of convolutional neural network (CNN) in order to find which one has the best performance for this specific dataset and problem.
At the begginig of the analysis, I tried to apply different operations in order to find an ideal number of training epochs; in particular, i tried to apply the Early Stopping, that is a piece of code entirely dedicated to creating a "controller" that monitors the training and stops it when performance starts to deteriorate.

```
from tensorflow.keras.callbacks import EarlyStopping
import tensorflow as tf
early_stopping_callback = EarlyStopping(
    monitor='val_loss',
    mode='min',
    patience=5,
    restore_best_weights=True
)
```

Figure 3: Code for Early stopping

In particular, I find different number of epochs for every Model :

- 36 epochs for the Model 1
- 40 epochs for the Model 2
- 50 epochs for the Model 3

However, considering that I wanted the same number of epochs for each model and to keep the computational cost even lower, I decided to limit the number of training epochs to 20.

In this project I tested 3 different CNN models to identify the most effective architecture; in particular, I tested 3 similar models with different combinations of layers and filters. In this section,I will analyze the common features of the three models.

3.1 Common features of the 3 models

- BatchNormalization: Stabilizes learning and reduces convergence time.
- MaxPooling2D((2, 2): It plays a key role in reducing the dimensionality of the data and making the model more robust.
- Dropout(0.3,seed=44) and Dropout(0.4,seed=44): It is a method that prevents overfitting (the model learning too much of the training data by

heart) by randomly deactivating neurons during the training process.Here, I used 0,3 for first Cl and 0,4 for second and third CL.

- Kernel regularizer: Initially, I applied a value of 0.01 but I decided to lower it to 0.005 to improve the model which suffered from overfitting.
- Custom adam: As I did before, I initially used the standard value of adam's learning rate (=0.001) but, to prioritize the stability and accuracy of the model at the expense of training speed, I lowered the value to 0.0003 (in particular for Oscillation Prevention).

In particular, the final dense layer, which provides the final prediction, was configured with three neurons and a softmax activation function (a standard choice for multiclass classification problems with more than two classesa as it returns a probability distribution). For all CNN models, the Adam optimizer was used.

3.2 Model 1

The first model (called CNN1 in the code) contains two convolutional layers (Conv2D), each composed of 64 filters.

```
def cnn1():
  cnn1 = Sequential()
  cnn1.add(Conv2D(64, (3, 3), activation='relu', input_shape=(Width, Height, Channels)))
  cnn1.add(BatchNormalization())
  cnn1.add(MaxPooling2D((2, 2)))
  cnn1.add(Dropout(0.3,seed=44))

  cnn1.add(Conv2D(64, (3, 3), activation='relu'))
  cnn1.add(MaxPooling2D((2, 2)))
  cnn1.add(Dropout(0.4,seed=44))

  cnn1.add(Flatten())
  cnn1.add(Dense(64,activation='relu',kernel_regularizer=regularizers.l2(0.005)))
  cnn1.add(Dropout(0.4,seed=44))

  cnn1.add(Dense(3, activation='softmax'))
  custom_adam = Adam(learning_rate=0.0003)

  cnn1.compile(
        loss='categorical_crossentropy',
        optimizer=custom_adam,
        metrics=['accuracy']
    )

  return cnn1
```

Figure 4: Code for CNN1

As can be seen from the architectural analysis, the CNN1 model is composed of two convolutional blocks using 64 filters each with $3x3$ kernels, supported by Batch Normalization.
the training showed excellent convergence, with a test accuracy of 95,32

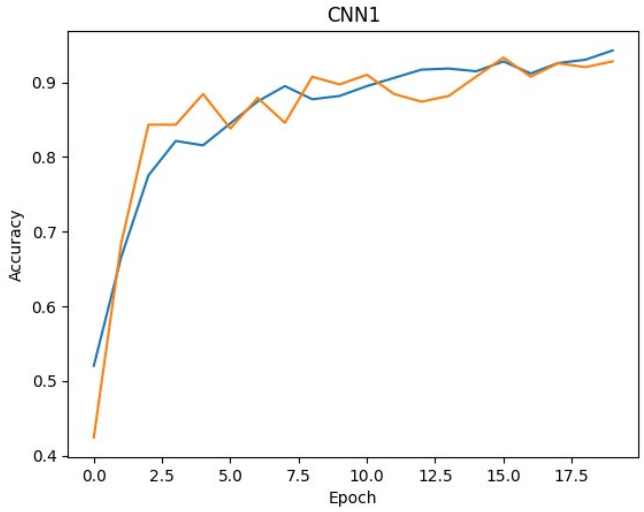

Figure 5: CNN1 Accuracy

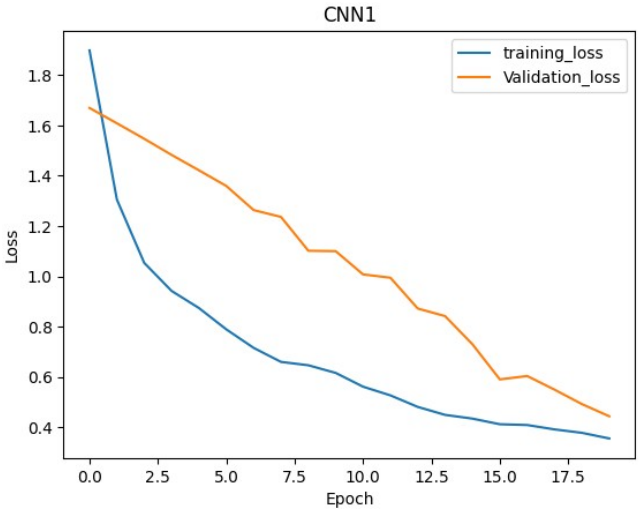

Figure 6: CNN1 Loss

3.3 Model 2

The second model (called CNN2 in the code) contains three convolutional layers (Conv2D), each composed of 32 filters.

As we seen for the first model, the CNN2 model has a similar structure (one more CL but 32 filters); in this case, the model achieved a testing accuracy of 98.36% (higher than CNN1), and a lower testing loss (0.3552, while CNN1 has a

```
def cnn2():
  cnn2 = Sequential()


  cnn2.add(Conv2D(32, (3, 3), activation='relu', input_shape=(Width, Height, Channels)))
  cnn2.add(BatchNormalization())
  cnn2.add(MaxPooling2D((2, 2)))
  cnn2.add(Dropout(0.3,seed=44))


  cnn2.add(Conv2D(32, (3, 3), activation='relu'))
  cnn2.add(MaxPooling2D((2, 2)))
  cnn2.add(Dropout(0.4,seed=44))

  cnn2.add(Conv2D(32, (3, 3), activation='relu'))
  cnn2.add(MaxPooling2D((2, 2)))
  cnn2.add(Dropout(0.4,seed=44))

  cnn2.add(Flatten())
  cnn2.add(Dense(64,activation='relu',kernel_regularizer=regularizers.l2(0.005)))
  cnn2.add(Dropout(0.4,seed=44))

  cnn2.add(Dense(3, activation='softmax'))
  custom_adam = Adam(learning_rate=0.0003)

  cnn2.compile(loss='categorical_crossentropy', optimizer = 'Adam', metrics = ['accuracy'])

  return cnn2
```

Figure 7: CNN2 Accuracy

value of 0.4215), suggesting that, once training is complete, its final predictions tend to be more precise and less uncertain.

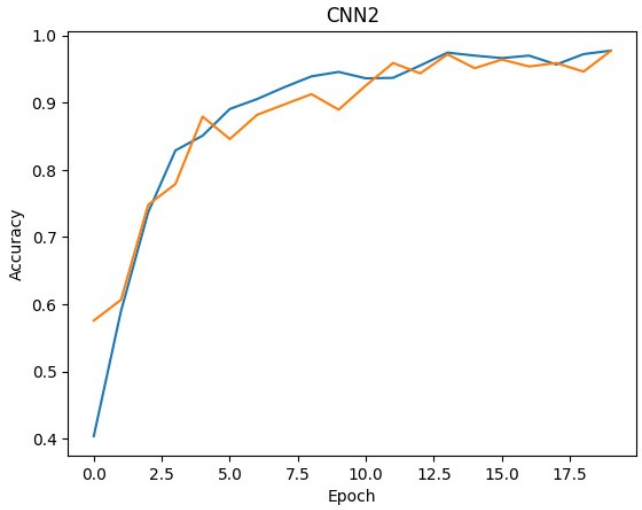

Figure 8: Code for CNN2

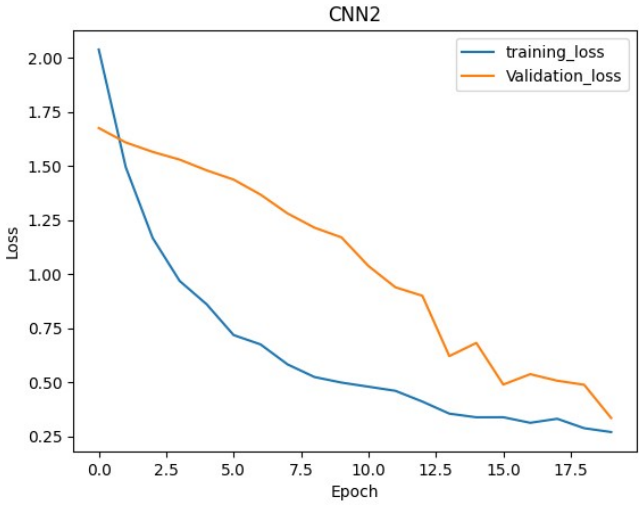

Figure 9: Code for CNN2

3.4 Model 3

In this model we have 3 CL and 64 filters (so, it's a mix of the other models).

```
def cnn3():
  cnn3 = Sequential()


  cnn3.add(Conv2D(64, (3, 3), activation='relu', input_shape=(Width, Height, Channels)))
  cnn3.add(BatchNormalization())
  cnn3.add(MaxPooling2D((2, 2)))
  cnn3.add(Dropout(0.3,seed=44))


  cnn3.add(Conv2D(64, (3, 3), activation='relu'))
  cnn3.add(MaxPooling2D((2, 2)))
  cnn3.add(Dropout(0.4,seed=44))

  cnn3.add(Conv2D(64, (3, 3), activation='relu'))
  cnn3.add(MaxPooling2D((2, 2)))
  cnn3.add(Dropout(0.4,seed=44))

  cnn3.add(Flatten())
  cnn3.add(Dense(64,activation='relu',kernel_regularizer=regularizers.l2(0.005)))
  cnn3.add(Dropout(0.4,seed=44))

  cnn3.add(Dense(3, activation='softmax'))
  custom_adam = Adam(learning_rate=0.0003)

  cnn3.compile(loss='categorical_crossentropy', optimizer = 'Adam', metrics = ['accuracy'])
```

Figure 10: Code for CNN3

this model achieves an accuracy of 96.8% and a Test Loss of 0.278. This means that CNN2 and CNN3 are both excellent models.

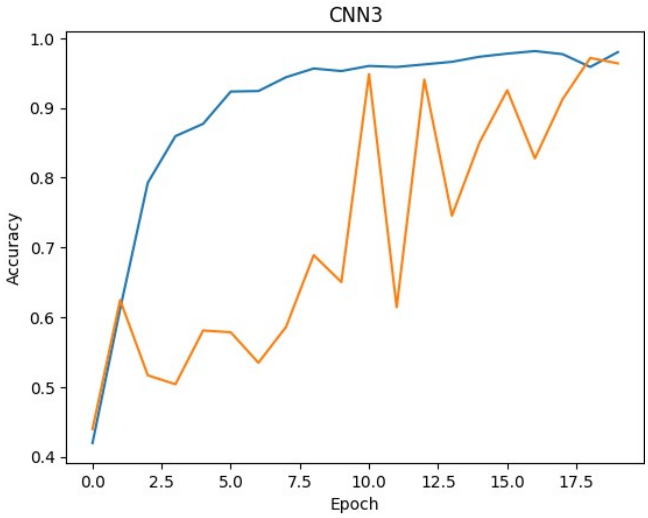

Figure 11: CNN3 Accuracy

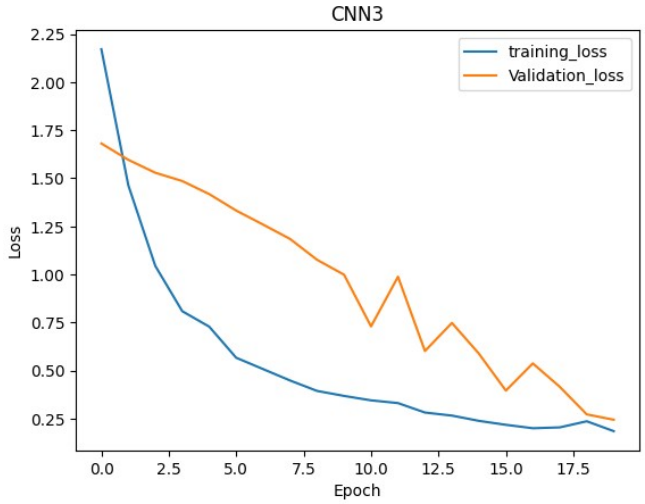

Figure 12: CNN3 Loss

3.5 Results

- CNN2 achieved the highest accuracy, meaning that the model correctly classified the largest percentage of the data in that set
- CNN3 achieved the lowest loss, indicating that, on average, the model's predictions were closest to the true values

In most machine learning scenarios, accuracy is the key metric for determining actual performance on a test set.

Therefore, Test 2 is generally considered the best result because it provides the highest percentage of correct predictions.

3.6 Appendix

Figure 13: Cross validation on CNN3