UNIVERSITY OF BOLOGNA

Digital Systems Project 2020-2021

Best Move
Recognition of a chessboard and prediction of the best move

Gallesi Pietro
Parigi Luca

23 April 2021

Contents

Keywords

- **Board**: a chessboard intended as a moment of play with any positioning of the pieces.
- Neural Network: neural network. [1].
- **Convolutional Neural Network**: Convolutional neural network, a type of feed-forward artificial neural network in which the pattern of connectivity between neurons is inspired by the organisation of the animal visual cortex [1].
- **Layer**: the foundational building block of a Neural Network[1].
- **Dense**: Layer whose neurons are connected with each neuron of the previous layer[1].
- **Convolutional Layer**: the foundational building block of a CNN [1].

Chapter 1

Introduction

The aim of the project is to implement an application for mobile devices (in particular for Android operating systems) capable of recognising a chessboard from a 2D image and predicting the best move the player can make in that situation.

The recognition of chess pieces and the prediction of the best move were implemented using two neural network models.

Much of the code, including the generation of the models, was written in Python. The program was then implemented with Android Studio, with integration of Python modules through the Chaquopy framework, to be exported to mobile devices.

Figure 1 - 2D chessboard.

The rest of the report is organised as follows.
Chapter 2 presents the tools used to implement the project.
Chapter 3 presents the various steps of the project, together with some implementation details.
In Chapter 4, we look in more detail at the neural network models used. In
Chapter 5, we discuss problems and optimisations.
In Chapter 6, conclusions and possible future developments are presented.

Chapter 2

Tools Used

In this chapter we will take a quick look at the tools used.

2.1 Google Colab and PyCharm

Figure 2 - Colab and PyCharm logos

We used Colab and PyCharm to write code in the Python language.
In particular, Google Colab was particularly useful for the construction of neural networks, the training of models and their evaluation.
With PyCharm, we unified the various Python modules and obtained a first working application.

2.2 Python Libraries

We used several Python libraries for the implementation of our project. In particular:

- **TensorFlow** for creating and training artificial neural network models.
- **TensorBoard** for the evaluation and optimisation of artificial neural networks.
- **OpenCV** to work on the images (in particular the starting chessboard image).
- **Numpy** to work on datasets.
- **Chess-Python for the** graphic visualisation (.SVG) of boards from FEN notation; for the generation (and implementation) of legal moves from a board.

Figure 3 - Library Logos

2.3 Dataset

We used two datasets for training the two models. The first dataset we created using the Python script that recognises the chessboard, generating a dataset that contains numerous images for each possible piece on the chessboard with the respective background or just the background itself.

if no pieces were present. The second dataset contains chessboards represented with matrix vectors and each associated with a value according to the player's situation (i.e. high value if it represents a favourable situation, such as piece advantage or checkmate, and low value if it represents a disadvantageous situation).

2.4 Android Studio

Android Studio is a development IDE for the Android platform. We used it to port the application to mobile devices by generating an APK.

2.4.1 Chaquopy

Chaquopy is a framework that allows to integrate Python components into an Android application. Thanks to it it was possible to import all necessary requirements for the proper functioning of the python modules we developed, and then to integrate the interface created with Android Studio with the python scripts.

Figure 4 - Android Studio and Chaquopy logos

Chapter 3

Outline design and general design choices.

The programme must, from a screenshot image of the 'Chess.com' application, recognise the chessboard present and, by means of an artificial neural network model, recognise the pieces present one by one. A second model, on the other hand, based on the positioning of the pieces will advise the player on the best move to make (within a reasonable time).

Let us look at the various steps into which the project was divided.

3.1 Chessboard recognition

The first step is to recognise the presence of a chessboard in an image. In order to recognise the chessboard within the image, various filters from the OpenCv library are applied to identify the grid lines of the chessboard and subsequently the intersection points of these, which define the vertices of the chessboard squares.

In particular, the following operations were performed on the image:

- **Rescaling of the image**: to avoid other points of interest in the image outside the chessboard being detected during edge detection. The image is cropped relative to the original size, assuming that the chessboard is approximately in the centre of it.

- **Blur** Filter: With this filter, the image is less detailed and at edge detection it is easier to detect only the sharp edges of the cells of the chessboard.
- **Gray Scale**: The image is changed to black and white.
- **Canny** Filter: This filter provided by OpenCV makes it possible to detect the edges of an image, through an initial noise reduction and subsequent calculation of the gradient of each point. Thanks to the calculation of local maxima, the edge of a part of the image is detected.
- **HoughLines**: This function is used to recognise whether there are any straight lines in the image, represented by an angle Theta and a modulus Rho. By filtering these lines for angles between 0 and 180 degrees and with a minimum length threshold for them to be considered chessboard lines, the lines passing through the edges of the chessboard are identified.
- **Lines Intersection**: Through the use of the 'linalg' function provided by numpy, the system given by the horizontal and vertical lines is solved by finding the intersection points representing the cell vertices.

After the 64 cells of the chessboard have been identified, the corresponding 64 images are generated, which must be processed in order to be able to re-know which chess piece is on each of them.

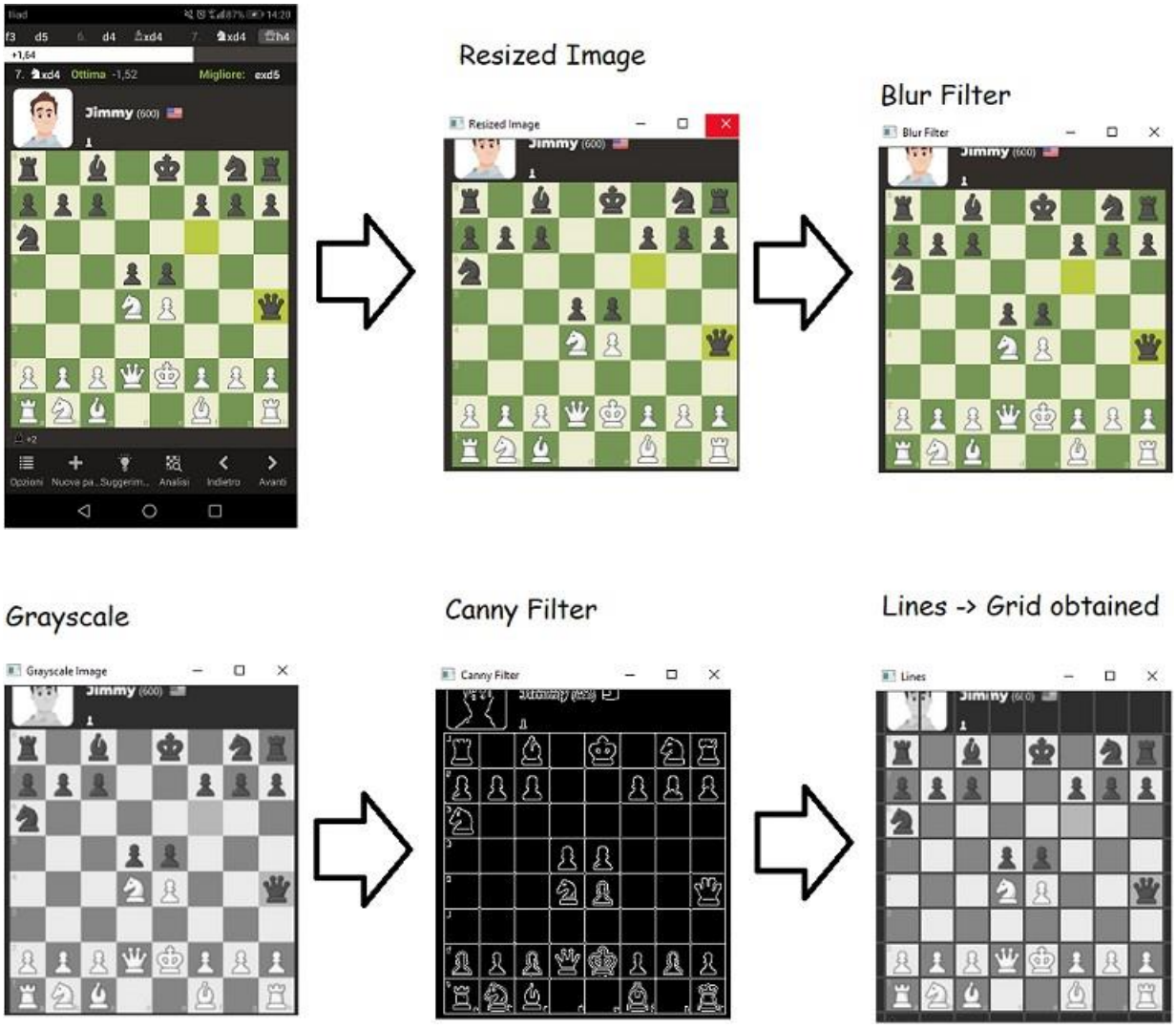


Figure 5 - Operations on the board for its reconsolidation.

3.2 Recognition of parts

The second step consists in recognising one by one the 64 images in which the chessboard has been divided. To do this, we need a model that can predict the chess piece it represents (or possibly an empty square) as input image.
This model was obtained by means of a neural network trained on images generated by us using the Python module previously presented.

The expected output of this Python module is a string containing the situation on the chess board using FEN notation. The Forsyth- Edwards notation is a chess notation used to describe a particular position on the board during a chess game. The purpose of FEN notation is to provide all the information necessary to allow a game to continue from a given position [1].

Fen notation for the initial position:
rnbqkbnr/pppppp/8/8/8/PPPPPP/RNBQKBNR w KQkq - 0 1

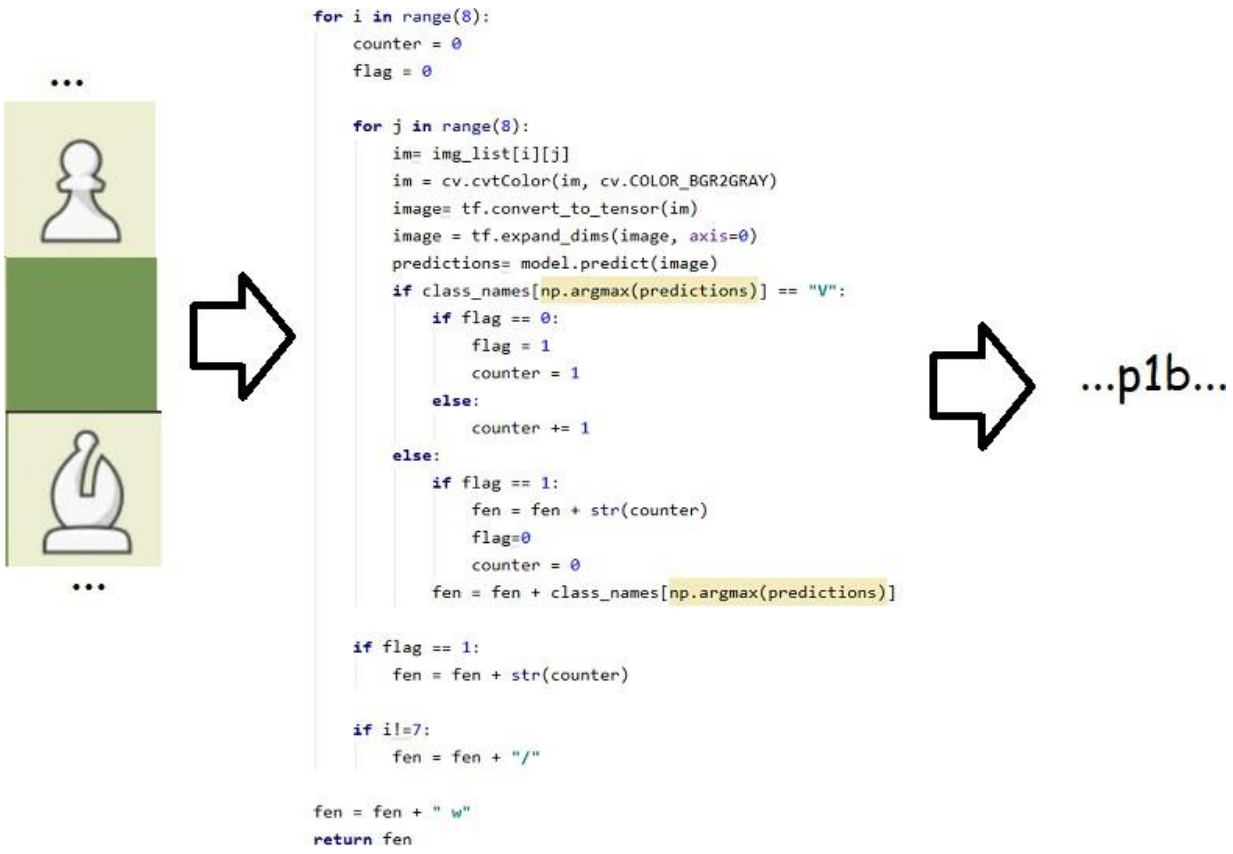


Figure 6 - From an array of images to its FEN notation p = pawn-white , 1 = a blank , b = halo-white

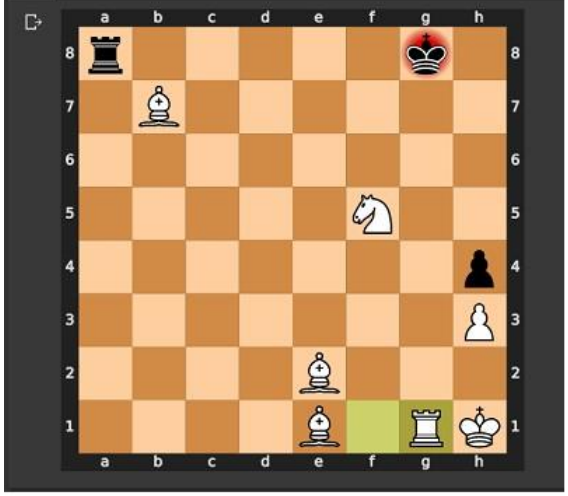
3.3 Prediction of the best move

The third step is to predict the best move the player could make in the situation he is in. The starting situation is represented by the FEN notation obtained from the previous step.

Using the chess-python library it is possible to enumerate all possible legal moves from the current situation and then generate all possible situations on the board based on the move made. The focus of this step is to work out which of these moves is the best. To do this, it is necessary to evaluate the possible situations in search of the one with the highest value.

Genero una board randomica

```
board = random_board()
board
```



Scelta della board con valore maggiore (re non più sotto scacco)

```
move = get_ai_move(board)
board.push(move)   # eseguo la mossa scelta
#print(f'\n{board}')
board
```

```
0.5939614
0.5938124
0.5881175
0.58991075
```

Codice predizione

```
[11] # associa un valore alla board mediante modello
     def predict(board):
       board3d = split_dims(board)
       board3d = numpy.expand_dims(board3d, 0)
       return model.predict(board3d)[0][0]

     # data la board cicla su tutte le possibili mosse legali
     def get_ai_move(board):
       max_move = None
       max_eval = -numpy.inf

       for move in board.legal_moves:
         board.push(move)
         eval = predict(board)
         print(eval)
         board.pop()
         if eval > max_eval:
           max_eval = eval
           max_move = move

       return max_move
```

Figure 7 - Using chess-python

The evaluation of these situations is done by means of a model held by a neural network trained on a dataset containing a large number of chessboards to which each has been associated a value. The chessboards in this dataset are represented by arrays of 14 8x8 matrices of zeros and ones:

- 6 arrays to represent the positions of the white pieces (one for pawns, one for knights, one for bishops, one for rooks, one for the queen, one for the king).
- 6 matrices to represent the positions of the black pieces.
- 2 matrices to represent the pieces under attack (one for white, one for black).

Therefore, every possible future situation on the current chessboard is represented by this array of matrices through a function and evaluated by the model. The one that obtains the best value will be printed on the screen along with the move to be made.

3.4 Application

Let us first see how the various parts of the code work together. Everything is handled by a single class: MainActivity. We have implemented a file chooser that allows the choice of image, the path of which is returned.
The file path is passed to the first Python script (encapsulated within a PyObject thanks to the Chaquopy framework), called board-detection, which works on the image in order to recognise the chessboard.
A second Python script, called piece-classification, will take care of classifying each of the 64 images into which the starting image has been divided, using a neural network model, and generate the FEN notation of the board which will be displayed on the screen.
A third and final Python script, called best-move, will take care of generating all possible legal moves from that board (based on the colour of the player's pieces) and evaluate each board obtained with a CNN model. The move to be made will be shown on the screen (as initial position
- final position) to the board with the best value.

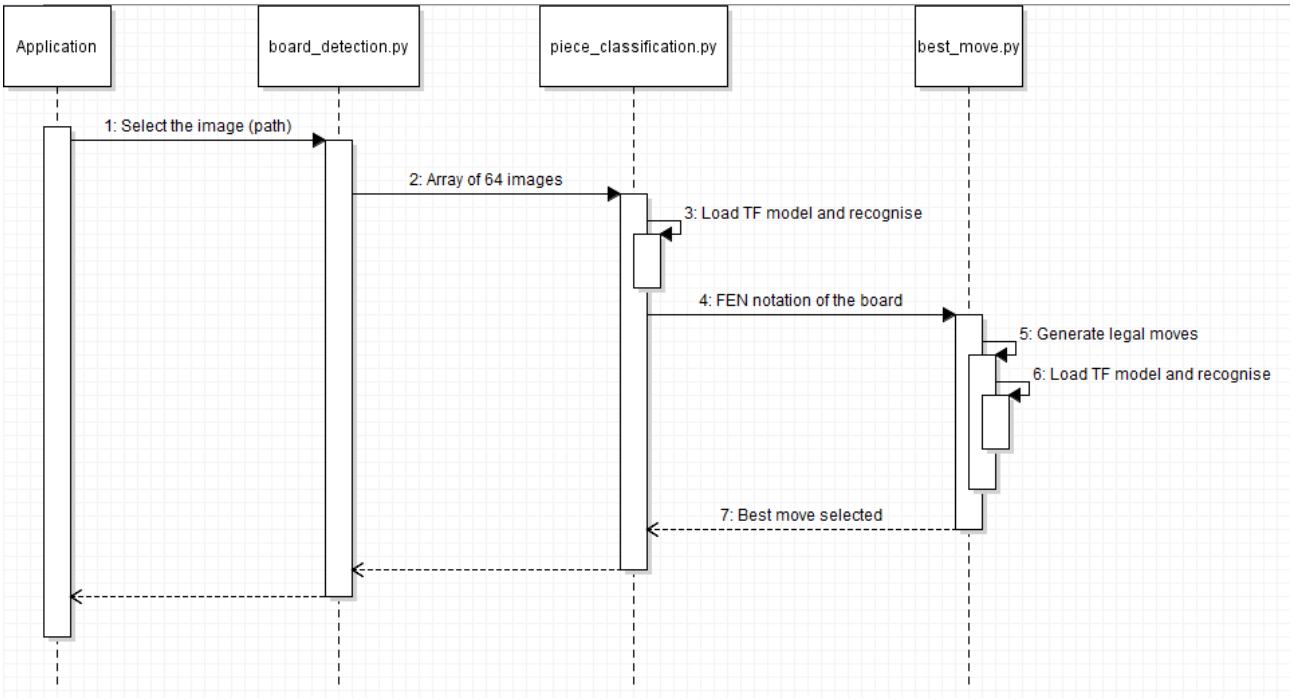


Figure 8 - Sequence Diagram

3.4.1 Operation of the application

Let us now briefly look at the concrete steps and some explanatory pictures of how the application works.

- Highlighted button at the top for selecting the image from the device's file system.
- Image selection.
- A new button for calculating the FEN appears. Before proceeding, it is necessary, if your tokens are black, to select the appropriate checkbox.
- Display of the corresponding FEN notation.
- One last button appears dedicated to the calculation of the prediction.
- The move to be performed as start position - end position is printed.

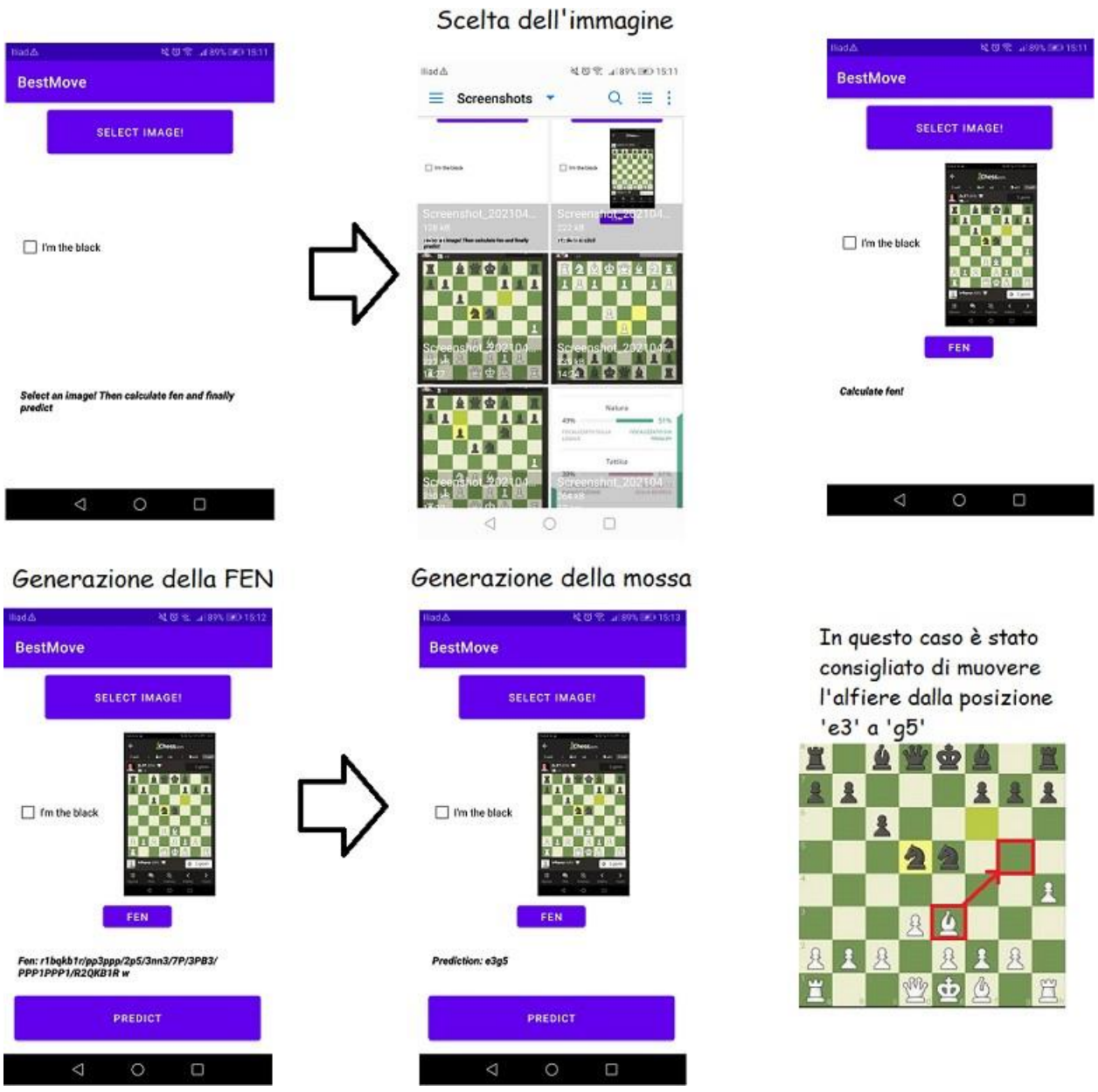


Figure 9 - Application operation, step by step.

Chapter 4

Tensorflow models

As already mentioned, the application makes use of two neural network models: the first, for the classification of images uses a simple Neural Network, while the second for the prediction of the best move consists of a Convolutional Neural Network.

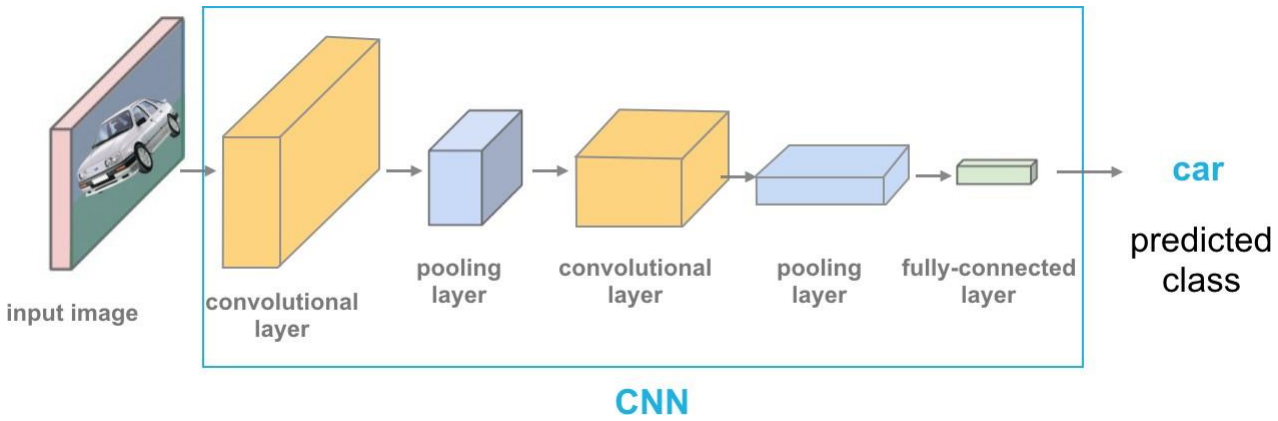


Figure 10 - example of CNN.

4.1 Piece Classification Model

Model towed on dataset containing just 13,000 images, divided into 13 classes, representing the 6 white and 6 black pieces (Bishop,Knight,Pawn,King,Queen,Rook) and the blanks (Void).

For the purpose of this network, a simple network consisting of only four layers proved to be effective:

- 1 Flatten(100,100,1);
- 1 Dense(1024);
- 1 Dense(400);
- 1 Dense(13);

The input consists of the images that were readjusted to a size of 100x100 and with only one greyscale colour value.

More information on the model:

- optimisation function: Adamax.
- batch size: 1200
- number of epochs: 20
- benchmark: loss = *'SparseCategoricalCrossentropy'*

The values achieved after network training for train loss and validation loss are:

TRAIN LOSS = 0.1005 VALIDATION LOSS = 0.1592

4.2 Board Classification Model

Model drawn on a dataset containing boards represented with arrays of 14 8x8 matrices of zeros and ones. Each board is associated with a value according to the question: "is this situation favourable to the player?".
The model consists of:

- 4 2D Convolutional Layers (32 nodes);
- 1 Flatten Layer;
- 1 Dense Layer (64 nodes);
- 1 Dense Layer (1 node).

More information on the

model:

- optimisation function: adam.
- batch size: 2048
- number of epochs: 37
- benchmark: loss = 'mean squared error
- presence of callback functions that terminate the training of the model when there are no more improvements in the controlled parameters.

TRAIN LOSS = 2.78e-4 VALIDATION LOSS = 3.74e-4

Chapter 5

Problems and Optimisations

The focus of this chapter is to show the optimisations made in the various steps of the project. Some optimisation attempts, however, failed and we illustrate the problems.

We first illustrate some values on how the application works on the mobile phone (test performed on Huawei P10 Lite).

- Memory occupation: 1.17GB
- Peak CPU utilisation: 13%.
- Peak memory usage: 379MB

In terms of execution speed on mobile it takes between 20 and 30 seconds for the entire process, from board recognition to move prediction. The application on a computer, on the other hand, is considerably faster: from a screenshot it takes between 5 and 10 seconds to complete.

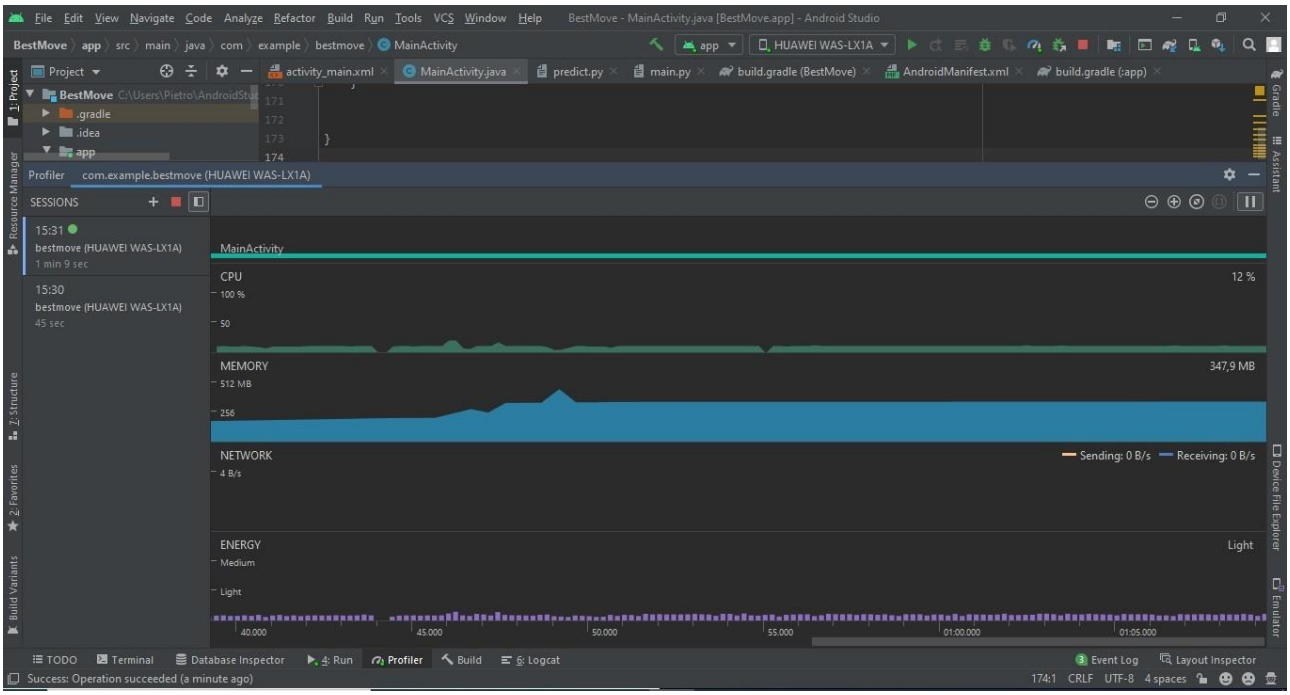


Figure 11 - Mobile device test details, from Android Studio.

5.1 Optimisation: TensorBoard

With regard to the model for the classification of chess pieces, an experimental improvement followed. After numerous tests with different neural and convolutional networks, starting for example with the network provided by Keras VGG16 for the classification of images up to simpler networks, it was found that for the type of very simple images and the redundancy of these, a simple network with only a few Dense layers was sufficient and better to recognise the chess pieces with sufficient accuracy.

After a gradual simplification of the model, up to the choice of layers in the previous paragraph, the increase in batch size was a significant factor in the increase in accuracy. Despite the fact that the accuracy of several models increased to high values and the loss to very low numbers, in the practical application of the model, errors often emerged in the classification of the image, particularly in the case of white pedestrians on a white background.

The choice of the number of images in the dataset and the factors listed above led to a sufficiently accurate network.

Concerning the model related to the attribution of a value to the board according to the advantage offered to the player, the optimisation was done using TensorBoard. We tried various types of neural networks:

- by increasing the number of convolutional layers: from 1 to 4.
- by increasing the number of nodes per L.C. from 32 to 128.
- increasing the number of Dense Layers: from 0 to 2.

The parameter we studied during these tests was always 'loss'. In general, we always observed an improvement with the addition of layers and nodes. In particular the value of validation loss for a model with 4
L.C. and 32 nodes (the one chosen as the final one) is very similar to a model with 3 L.C. and 128 nodes.

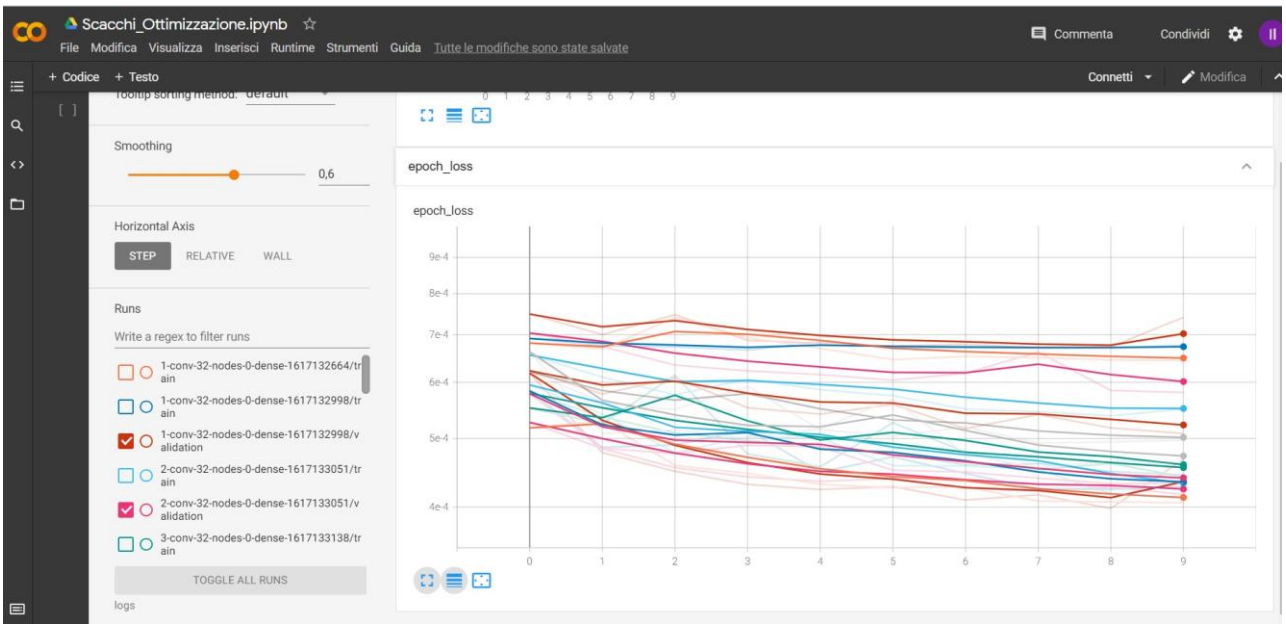


Figure 12 - Example of evaluation using TensorBoard.

5.2 Problem: starting chessboard image

The operation of the application requires a screenshot taken from the device and this leads to a couple of problems that greatly affect the operation of the module that recognises the chessboard.

- The chess application from which the screenshot is taken has a major impact on the result. Our module works well on images taken from the Chess.com application, much less so on others.
- The size and resolution of the screen of the device taking the screenshot. We noticed that this changes the positioning and size of the chessboard. We partially solved this by making changes in the parameter values that deal with the recognition of the chessboard based on the screen size. However, many more tests should be done on different devices. It remains one of the open issues for future development.

5.3 Problem: TensorFlow Lite

TensorFlow Lite is a toolkit whose purpose is to optimise the basic models to make them lighter and faster to make predictions, so that they can also be used on devices with limited computing power (like a number of smartphones). We converted both models, but the result was not satisfactory. The loss in accuracy of both models was, in our opinion, too great to be preferred to the non-optimised model. In particular, the piece classification model only correctly recognised empty squares and pawns. As the starting models were light in terms of memory usage (70MB the first, 3MB the second), we decided to test them with Android Studio using Chaquopy. Both were sufficiently fast in making predictions, particularly in view of the purpose of the application, which does not require answers in real time or in a particularly short time, so we decided not to use the Lite models. Following this implementation choice, the wait for the generation of the FEN notation of the chessboard takes several seconds, while that for the prediction is quite immediate.

5.4 Optimisation: minimax algorithm

Another problem arose: the recommended move sometimes made little sense. We noticed that this was due to the fact that the values of the predictions generated by the board evaluation model were very similar to each other. It was clear, therefore, that more discernment of choice was needed in some cases. We therefore implemented a minimax algorithm in the search for the best choice, also analysing the opponent's possible subsequent moves. This greatly improves the end result by discarding most of the unwise moves, however it takes considerably longer to execute. On PC it takes about 30 seconds, while on mobile it goes up to 6 minutes, which is far too long.

```
function minimax(nodo, profondità)
    SE nodo è un nodo terminale OPPURE profondità = 0
        return il valore euristico del nodo
    SE l'avversario deve giocare
        α := +∞
        PER OGNI figlio di nodo
            α := min(α, minimax(figlio, profondità-1))
    ALTRIMENTI dobbiamo giocare noi
        α := -∞
        PER OGNI figlio di nodo
            α := max(α, minimax(figlio, profondità-1))
    return α
```

Figure 13 - Minimax algorithm

Chapter 6

Conclusions and future developments

In this report, we saw an overview of the implemented application and various details of the process steps.
The application is open to considerable improvement. Possible future developments include a dynamic choice of parameters to handle the recognition of the chessboard from a screenshot, the generation of new patterns
more accurate for TF-Lite tomake execution more fast on devices mobile.

Chapter 7

Bibliography

[1] Convolutional neural network, at en.wikipedia.org

[2] Forsyth-Edwards notation, at en.wikipedia.org

[3] Chaquopy documentation at https://chaquo.com/chaquopy/doc/

[4] Tensorflow documentation at https://www.tensorflow.org/