UNIVERSITA' DEGLI STUDI DI BOLOGNA

Progetto di Sistemi Digitali 2020-2021

Best Move
Riconoscimento di una scacchiera e predizione della mossa migliore

Gallesi Pietro
Parigi Luca

23 Aprile 2021

Contents

Parole chiave

- **Board**: una scacchiera intesa come momento di gioco con un qualsiasi posizionamento dei pezzi.
- **Neural Network**: rete neurale. [1].
- **Convolutional Neural Network**: rete neurale convoluzionale, un tipo di rete neurale artificiale feed-forward in cui il pattern di connettività tra i neuroni è ispirato dall'organizzazione della corteccia visiva animale [1].
- **Layer**: ovvero il blocco costruttivo fondante di una Rete Neurale[1].
- **Dense**: Layer i cui neuroni sono collegati con ciascun neurone del layer precedente.[1].
- **Convolutional Layer**: ovvero il blocco costruttivo fondante di una CNN [1].

Chapter 1

Introduzione

Scopo del progetto è implementare una applicazione per dispositivi mobile (in particolare per sistemi operativi Android) in grado di riconoscere una scacchiera a partire da una immagine 2D e predire la mossa migliore che il giocatore può eseguire in quella situazione.

Il riconoscimento dei pezzi della scacchiera e la predizione della mossa migliore sono stati implementati mediante due modelli di reti neurali.

Gran parte del codice, compresa la generazione dei modelli, è stata scritta in Python. Il programma è stato poi implementato con Android Studio, con integrazione dei moduli Python mediante framework Chaquopy, per essere esportato su dispositivi mobili.

Figura 1 - Scacchiera 2D.

Il resto della relazione è organizzato come segue.
Il Capitolo 2 presenta i tool utilizzati per l'implementazione del progetto.
Il Capitolo 3 presenta i vari step del progetto, insieme ad alcuni dettagli sull'implementazione.
Nel Capitolo 4 vediamo più nel dettaglio i modelli di reti neurali utilizzati.
Nel Capitolo 5 parliamo di problemi ed ottimizzazioni.
Nel Capitolo 6, infine, vengono presentate le conclusioni e possibili sviluppi futuri.

Chapter 2

Tool utilizzati

In questo capitolo vedremo velocemente i tool utilizzati.

2.1 Google Colab e PyCharm

Figura 2 - Loghi Colab e PyCharm

Abbiamo utilizzato Colab e PyCharm per scrivere codice in linguaggio Python.
In particolare Google Colab è stato particolarmente utile per la costruzione delle reti neurali, il training dei modelli e la loro valutazione.
Con PyCharm abbiamo unificato i vari moduli Python e ottenuto una prima applicazione funzionante.

2.2 Librerie Python

Abbiamo utilizzato svariate librerie Python per l'implementazione del nostro progetto. In particolare:

- **TensorFlow** per creazione e training dei modelli di reti neurali artificiali.
- **TensorBoard** per la valutazione e l'ottimizzazione delle reti neurali artificiali.
- **OpenCV** per lavorare sulle immagini (in particolare l'immagine della scacchiera di partenza).
- **Numpy** per lavorare sui dataset.
- **Chess-Python** per la visualizzazione grafica (.SVG) delle board a partire dalla notazione FEN; per la generazione (ed attuazione) delle mosse legali a partire da una board.

Figura 3 - Loghi Librerie

2.3 Dataset

Abbiamo utilizzato due dataset per l'allenamento dei due modelli. Il primo dataset lo abbiamo creato mediante lo script Python che riconosce la scacchiera, generando un dataset che contenesse numerose immagini per ogni possibile pezzo sulla scacchiera con il rispettivo sfondo o solamente quest'ultimo

se non fosse stato presente alcun pezzo. Il secondo dataset contiene delle scacchiere rappresentate con vettori di matrici e ad ognuna associata un valore in base alla situazione del giocatore (quindi valore alto se rappresenta una situazione a lui favorevole, come vantaggio di pezzi o scacco, mentre valore basso se rappresenta una situazione svantaggiosa).

2.4 Android Studio

Android Studio è un IDE per lo sviluppo per la piattaforma Android. L'abbiamo utilizzato per portare l'applicazione su dispositivi mobile, generandone un APK.

2.4.1 Chaquopy

Chaquopy è un framework che permette di integrare delle componenti Python in un'applicazione Android. Grazie ad esso è stato possibile importare tutti i requisisti necessari per il corretto funzionamento dei moduli python da noi sviluppati, e successivamente integrare l'interfaccia creata con Android Studio con gli script python.

Figura 4 - Loghi Android Studio e Chaquopy

Chapter 3

Progetto di massima e scelte progettuali generali.

Il programma deve, a partire da un'immagine screenshot dell'applicazione di "Chess.com", riconoscere la scacchiera presente e, mediante modello di rete neurale artificiale, riconoscere uno ad uno i pezzi presenti. Un secondo modello, invece, in base al posizionamento dei pezzi consiglierà al giocatore la miglior mossa da eseguire (in un tempo ragionevole).

Vediamo i vari step in cui è stato diviso il progetto.

3.1 Riconoscimento della scacchiera

Il primo step consiste nel ricoscere la presenza di una scacchiera in un'immagine. Per poter riconoscere la scacchiera all'interno dell'immagine vengono applicati diversi filtri dalla libreria OpenCv per individuare le linee della griglia della scacchiera e successivamente i punti di intersezione di queste, che definiscono i vertici dei quadrati della scacchiera.

In particolare sono state effettuate le seguenti operazioni sull'immagine:

- **Ridimensionamento dell'immagine**: per evitare che nei passaggi successivi, durante la edge detection, fossero individuati altri punti di interesse nell'immagine al di fuori della scacchiera. L'immagine viene ritagliata in relazione alla dimensione originale, supponendo che la scacchiera si trovi circa al centro di essa.

- **Blur Filter**: attraverso questo filtro l'immagine risulta meno dettagliata e al momento della edge detection risulta più semplice individuare solo i bordi netti delle celle della scacchiera.
- **Gray Scale**: l'immagine viene modificata in bianco e nero.
- **Canny Filter**: questo filtro fornito da OpenCV permette di rilevare i bordi di un'immagine, attraverso una riduzione iniziale del rumore e successivamente al calcolo del gradiente di ogni punto. Grazie al calcolo dei massimi locali viene individuato il bordo di una parte dell'immagine.
- **HoughLines**: questa funzione serve a riconoscere se nell'immagine siano presenti delle linee rette, rappresentate da un angolo Theta e da un modulo Rho. Filtrando tali linee per angoli compresi tra 0 e 180 gradi e con una minima soglia di lunghezza perché possano essere ritenute linee della scacchiera, vengono individuate le linee passanti per i bordi di questa.
- **Lines Intersection**: attraverso l'uso della funzione "linalg" fornita da numpy, viene risolto il sistema dato dalle linee orizzontali e verticali, trovandone i punti di intersezione che rappresentano i vertici delle celle.

Dopo l'individuazione delle 64 celle della scacchiera, vengono generate le 64 immagini corrispondenti che dovranno essere processate per poter riconoscere quale pezzo dell scacchiera si trovi su ognuno di essi.

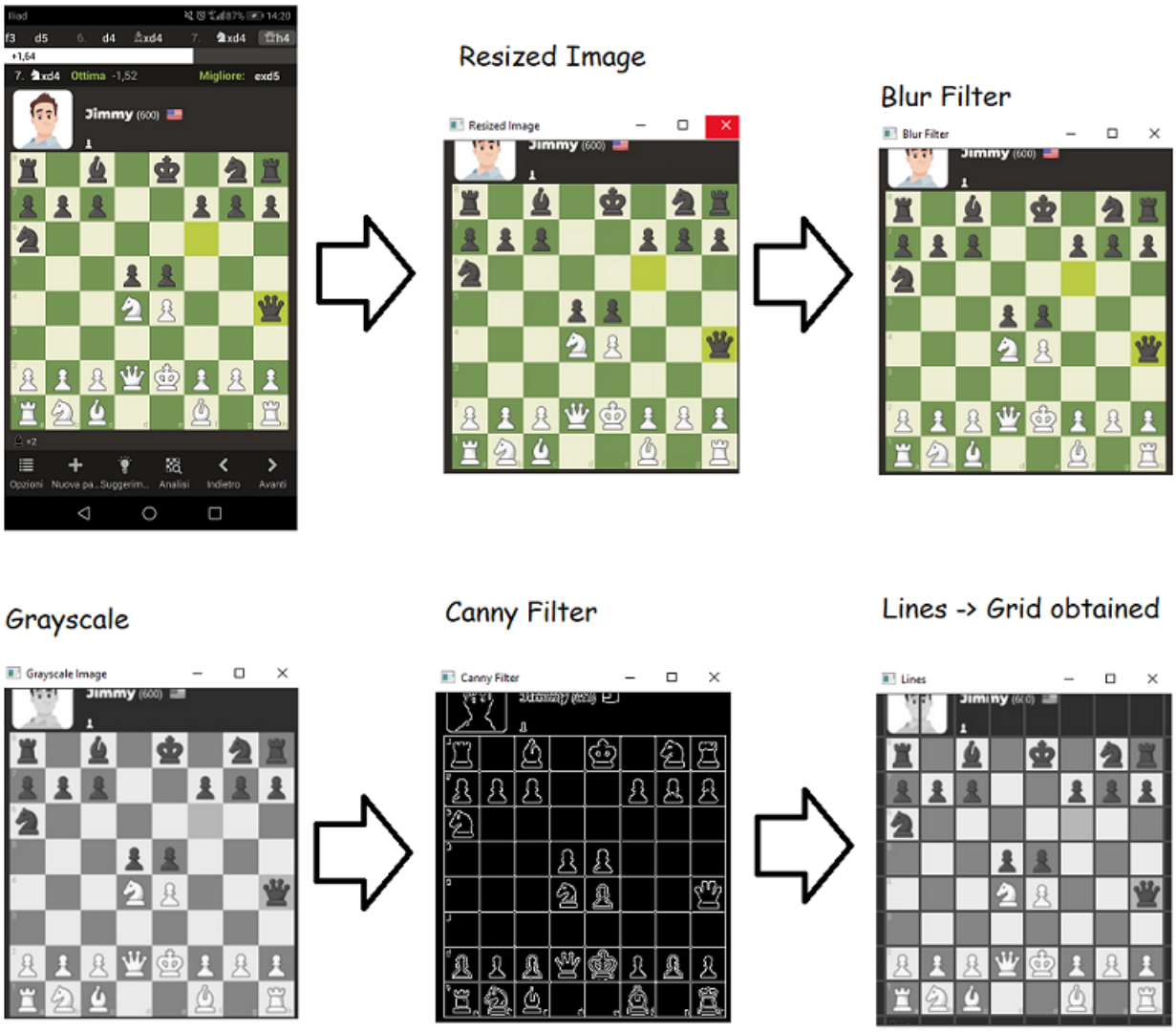


Figura 5 - Operazioni sulla board per il suo riconscimento.

3.2 Riconoscimento dei pezzi

Il secondo step consiste nel riconoscere una ad una le 64 immagini in cui la scacchiera è stata suddivisa. Per fare ciò abbiamo bisogno di un modello che avendo in input un'immagine sia in grado di predire il pezzo della scacchiera che rappresenta (o eventualmente una casella vuota).
Tale modello è stato ottenuto mediante rete neurale allenata su immagini da noi generate utilizzando il modulo Python precedentemente presentato.

L'output atteso di questo modulo Python è una stringa contenente la situazione della scacchiera mediante notazione FEN. La Notazione Forsyth-Edwards è una notazione scacchistica utilizzata per descrivere una particolare posizione sulla scacchiera nel corso di una partita di scacchi. Lo scopo della notazione FEN è quello di fornire tutte le informazioni necessarie a consentire la prosecuzione di una partita a partire da una posizione data [1].

Notazione fen per la posizione iniziale:
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

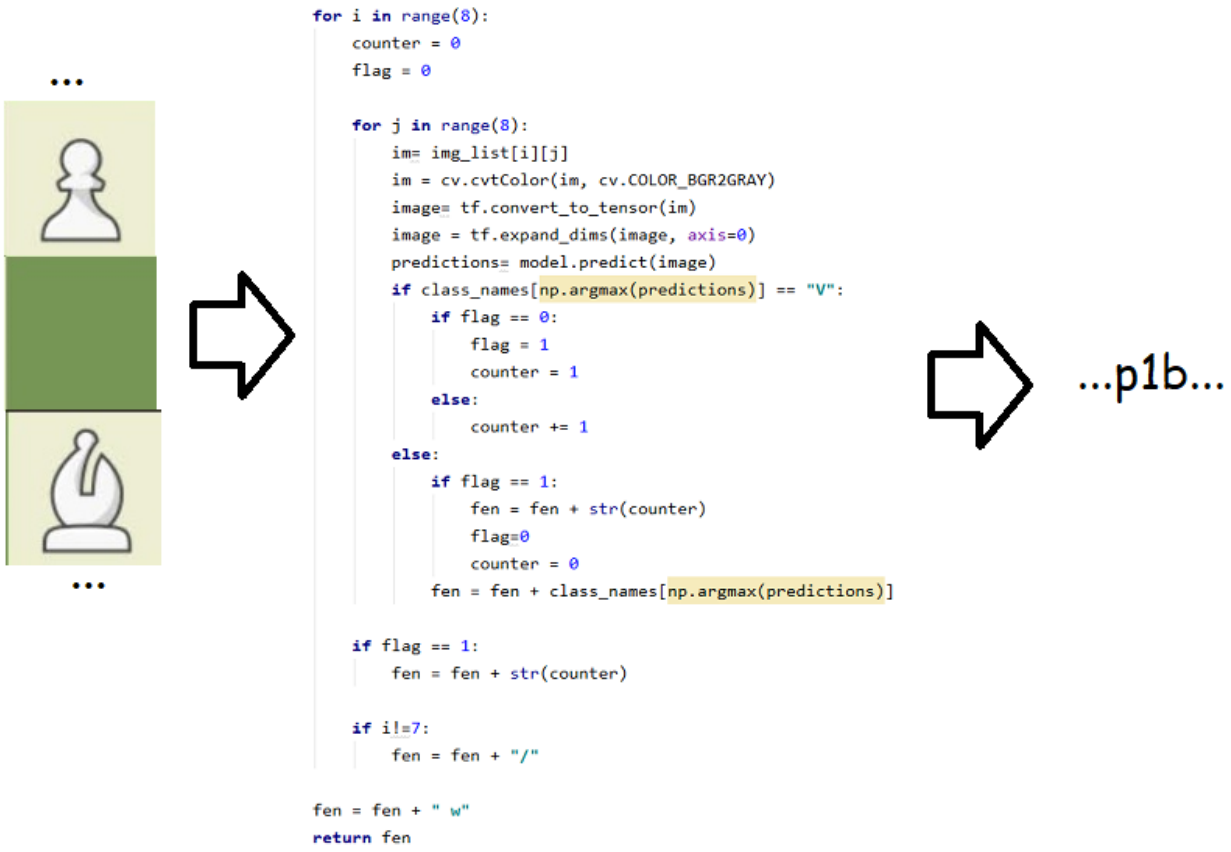


Figura 6 - Da un array di immagini alla sua notazione FEN
p = pedone-bianco , 1 = un vuoto , b = alfiere-bianco

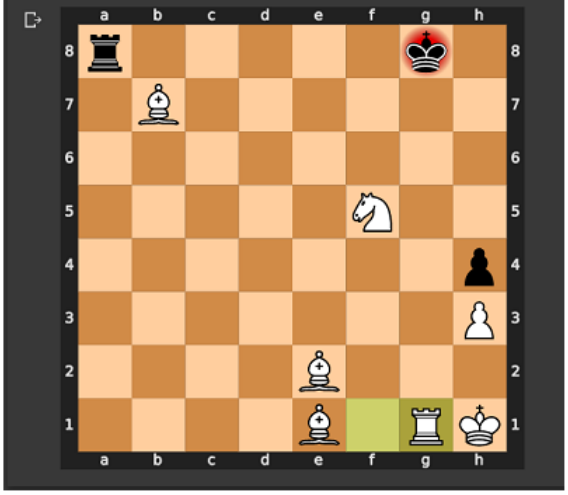
3.3 Predizione della mossa migliore

Il terzo step consiste nel predire la mossa migliore che il giocatore potrebbe effettuare nella situazione in cui si trova. La situazione di partenza è rappresentata dalla notazione FEN ottenuta dal precedente step.

Mediante libreria chess-python è possibile enumerare tutte le possibili mosse legali a partire dalla situazione attuale e quindi generare tutte le possibili situazioni della scacchiera in base alla mossa effettuata. Focus di questo step è capire quale tra queste mosse sia la migliore. Per fare ciò è necessario valutare le possibili situazioni alla ricerca di quella con valore più alto.

Genero una board randomica

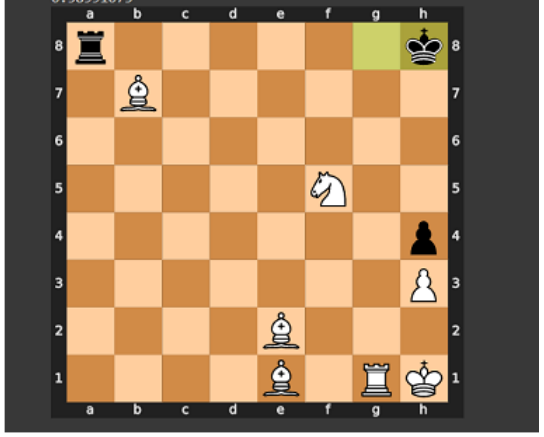
```
board = random_board()
board
```



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
```

Figura 7 - Utilizzo di chess-python

La valutazione di queste situazioni viene fatta mediante un modello ottenuto da una rete neurale allenata su un dataset contente un gran numero di scacchiere a cui ognuna è stato associato un valore. Le scacchiere presenti in questo dataset sono rappresentate da array di 14 matrici 8x8 di zeri e uni:

- 6 matrici per rappresentare le posizioni dei pezzi bianchi (una per i pedoni, una per i cavalli, una per gli alfieri, una per le torri, una per la regina, una per il re).
- 6 matrici per rappresentare le posizioni dei pezzi neri.
- 2 matrici per rappresentare i pezzi sotto attacco (una per i bianchi, una per i neri).

Per questo motivo ogni possibile situazione futura della scacchiera attuale viene rappresentata mediante questo array di matrici attraverso una funzione e valutata dal modello. Quella che otterrà il migliore valore verrà stampata a schermo insieme alla mossa da effettuare.

3.4 Applicazione

Vediamo innanzitutto come le varie parti del codice lavorano insieme. Tutto viene gestito da un'unica classe: MainActivity. Abbiamo implementato un file chooser che permette la scelta dell'immagine, di cui viene ritornato il path.
Il path del file viene passato al primo script Python (incapsulato all'interno di un PyObject grazie al framework Chaquopy), chiamato board-detection, che lavora sull'immagine al fine di riconoscere la scacchiera.
Un secondo script Python, chiamato piece-classification, si occuperà di classificare ognuna delle 64 immagini in cui è stata suddivisa quella di partenza, mediante modello di rete neurale, e generare la notazione FEN della board che verrà riportata a schermo.
Un terzo e ultimo script Python, chiamato best-move, si occuperà di generare tutte le possibili mosse legali a partire da quella board (in base al colore dei pezzi del giocatore) e valutare ogni singola board ottenuta con un modello CNN. Sarà riportata a schermo la mossa da compiere (come posizione iniziale - posizione finale) per giungere alla board che ha ottenuto il valore migliore.

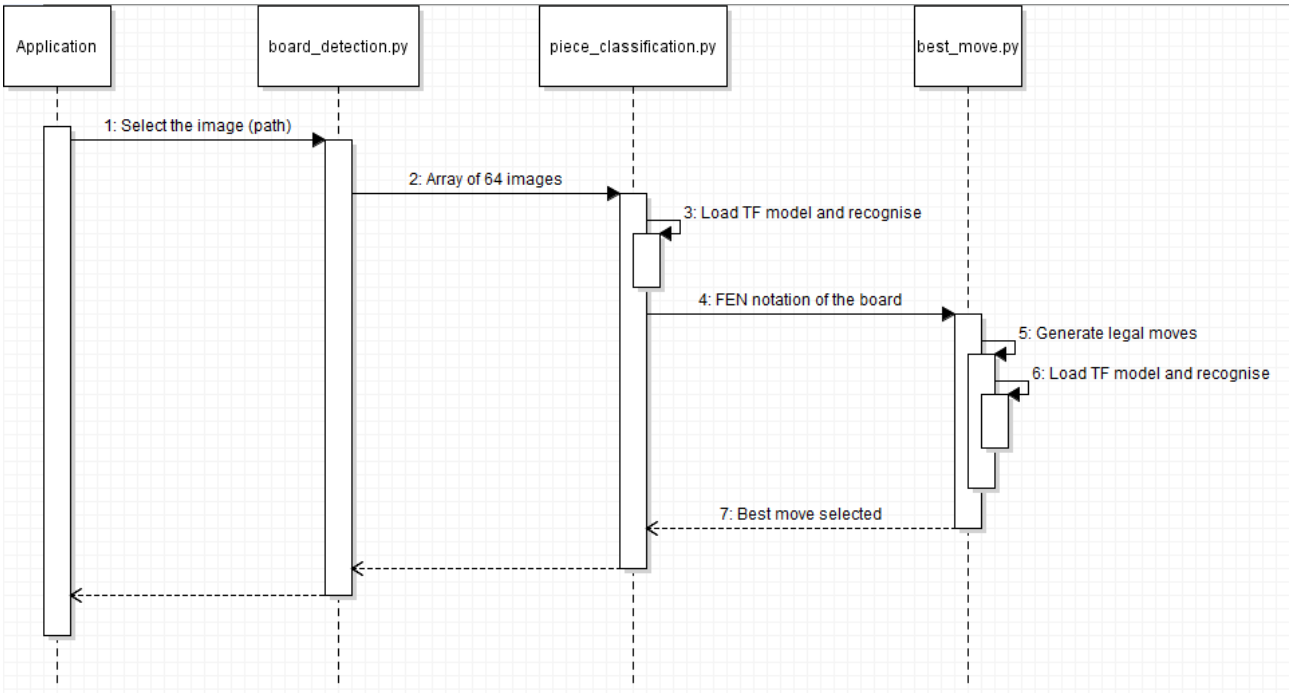


Figura 8 - Sequence Diagram

3.4.1 Funzionamento dell'applicazione

Vediamo ora in breve gli step concreti e alcune immagini esplicative del funzionamento dell'applicazione.

- Button in evidenza in alto per la selezione dell'immagine da file system del dispositivo.
- Selezione dell'immagine.
- Compare un nuovo Button per il calcolo della FEN. Prima di procedere è necessario, nel caso in cui le proprie pedine siano quelle nere, selezionare l'opportuno checkbox.
- Visualizzazione della notazione FEN corrispondente.
- Compare un ultimo Button dedicato al calcolo della predizione.
- Viene stampata la mossa da eseguire come posizione iniziale - posizione finale.

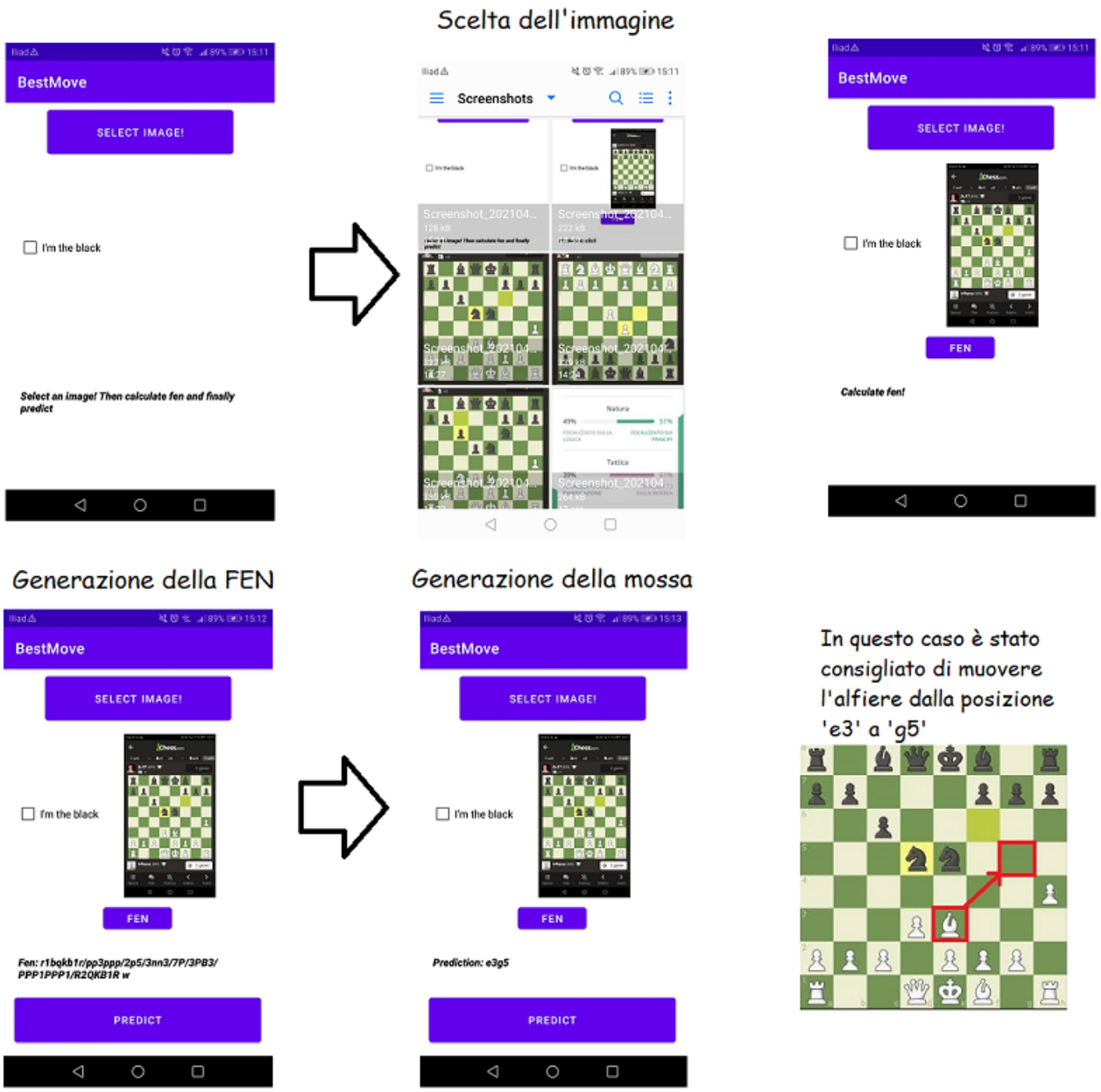


Figura 9 - Funzionamento dell'applicazione, step per step.

Chapter 4

Modelli Tensorflow

Come già anticipato, l'applicazione fa uso di due modelli di rete neurale: il primo, per la classificazione delle immagini utilizza una semplice Neural Network, mentre il secondo per la predizione della mossa migliore è costituito da una Convolutional Neural Network.

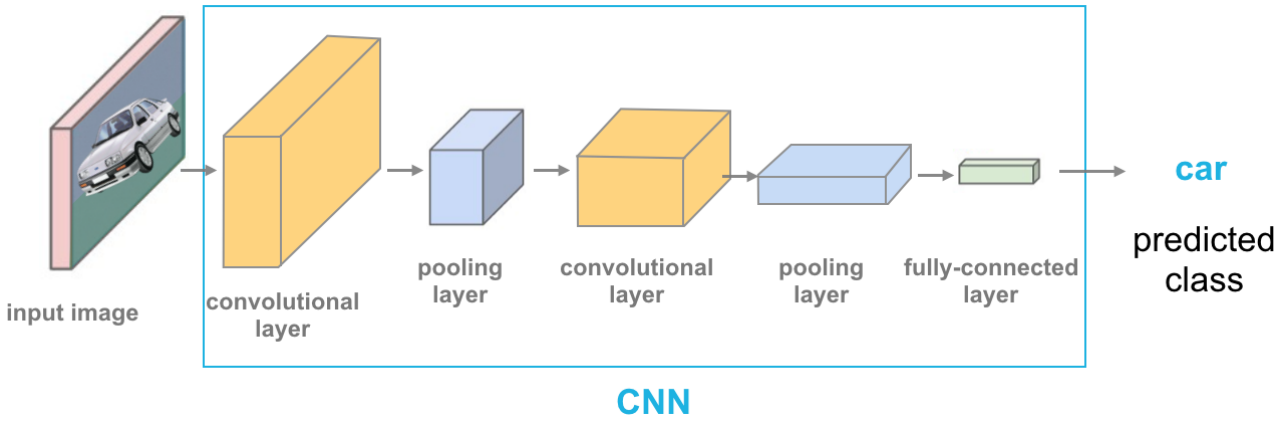


Figura 10 - esempio di CNN.

4.1 Piece Classification Model

Modello trainato su dataset contenente poco più di 13'000 immagini, divise in 13 classi, rappresentanti i 6 pezzi bianchi e 6 neri (Bishop,Knight,Pawn,King,Queen,Rook)e gli spazi vuoti(Void).

Per lo scopo di questa rete, si è verificata più efficace una rete semplice composta da soli quattro layer:

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

L'input è composto dalle immagini che sono state riadattate a una dimensione di 100x100 e con solo un valore di colore della scala di grigi.

Altre informazioni sul modello:

- funzione di ottimizzazione: Adamax.
- batch size: 1200
- numero di epochs: 20
- parametro di riferimento: loss = $'SparseCategoricalCrossentropy'$

I valori raggiunti dopo il training della rete relativi al train loss e al validation loss sono:

TRAIN LOSS = 0.1005 VALIDATION LOSS = 0.1592

4.2 Board Classification Model

Modello trainato su un dataset contenente board rappresentate con array di 14 matrici 8x8 di zeri e uni. Ad ogni board è associato un valore in base al quesito: "questa situazione è favorevole al giocatore?".
Il modello si compone di:

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

Altre informazioni sul modello:

- funzione di ottimizzazione: adam.
- batch size: 2048
- numero di epochs: 37
- parametro di riferimento: loss = 'mean squared error'
- presenza di funzioni di callbacks che terminano il training del modello quando non vi sono più miglioramenti nei parametri controllati.

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

Chapter 5

Problemi e Ottimizzazioni

Il focus di questo capitolo è mostrare le ottimizzazioni apportate ai vari step del progetto. Alcuni tentativi di ottimizzazione, tuttavia, non sono riusciti e ne illustriamo i problemi.

Per prima cosa illustriamo alcuni valori sul funzionamento dell'applicazione sul cellulare (test eseguito su Huawei P10 Lite).

- Occupazione di memoria : 1.17GB
- Picco utilizzo di CPU : 13%
- Picco utilizzo di memoria : 379MB

Per quanto riguarda la velocità di esecuzione su mobile richiede tra i 20 e i 30 secondi per l'intero processo, dal riconoscimento della board alla predizione della mossa. L'applicazione su computer, invece, è notevolmente più veloce: a partire da uno screenshot impiega tra i 5 e i 10 secondi per completare il tutto.

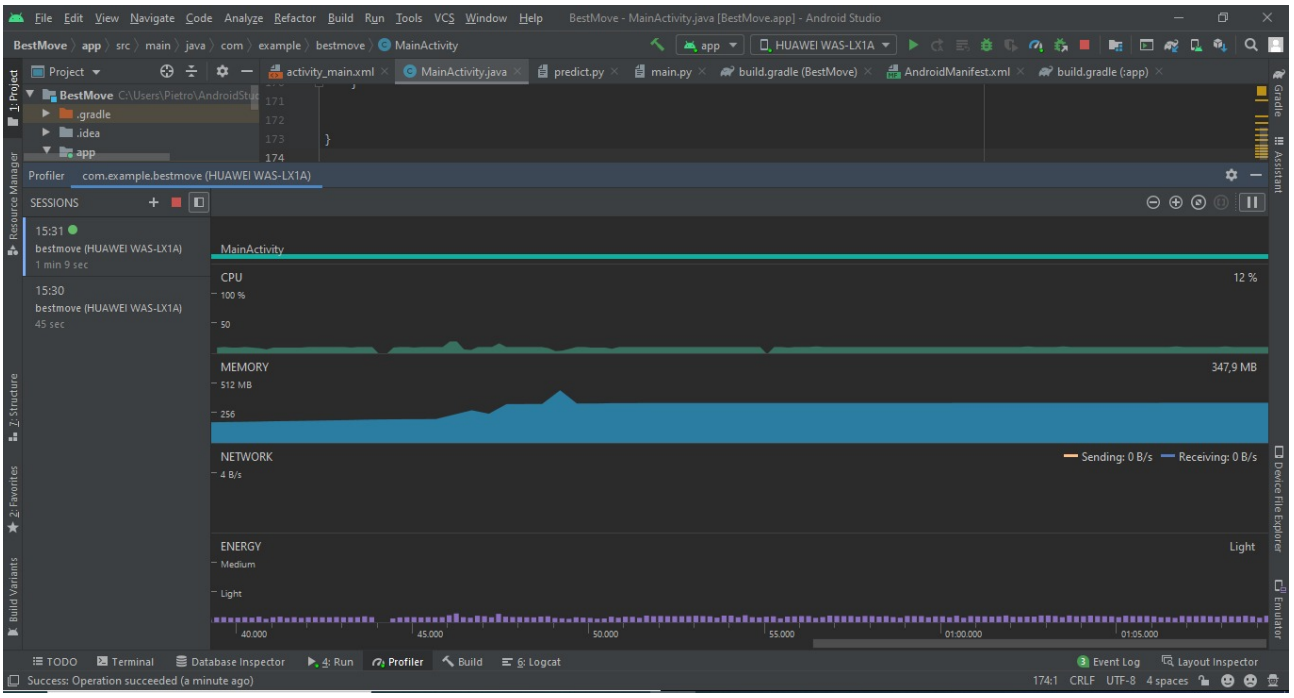


Figura 11 - Dettagli del test su dispositivo mobile, da Android Studio.

5.1 Ottimizzazione: TensorBoard

Per quanto riguarda il modello relativo alla classificazione dei pezzi di scacchi, si è seguito un miglioramento sperimentale. Dopo numerose prove con diverse reti neurali e convoluzionali, a partire ad esempio dalla rete fornita da Keras VGG16 per la classificazione di immagini fino a reti più semplici, si è potuto constatare che, per il tipo di immagini molto semplici e per la ridondanza di queste, una semplice rete con pochi layer Dense fosse sufficiente e migliore per riconoscere con sufficiente accuratezza i pezzi della scacchiera.

Dopo una graduale semplificazione del modello, fino alla scelta dei layer nel paragrafo precedente, l'aumento della batch size ha costituito un fattore rilevante per l'incremento dell'accuratezza. Nonostante l'accuratezza di diversi modelli si portasse a valori alti e il loss a numeri molto bassi, all'applicazione pratica del modello spesso emergevano errori nella classificazione dell'immagine, in particolare nei casi di pedoni bianchi su sfondo bianco.

La scelta del numero di immagini del dataset e i sopraelencati fattori hanno condotto a una rete sufficientemente accurata.

Per quanto riguarda il modello relativo all'attribuzione di un valore alla board in base al vantaggio offerto al giocatore, l'ottimizzazione è stata fatta mediante TensorBoard. Abbiamo provato varie tipologie di reti neurali:

- aumentando il numero di convolutional layer: da 1 a 4.
- aumentando il numero di nodi per C.L: da 32 a 128.
- aumentando il numero di Dense Layer: da 0 a 2.

Il parametro che abbiamo studiato durante queste prove è sempre stato 'loss'. In generale abbiamo sempre osservato un miglioramento con l'aggiunta di layer e nodi. In particolare il valore di validation loss per un modello con 4 C.L. e 32 nodi (quello scelto come finale) è molto simile ad un modello con 3 C.L. e 128 nodi.

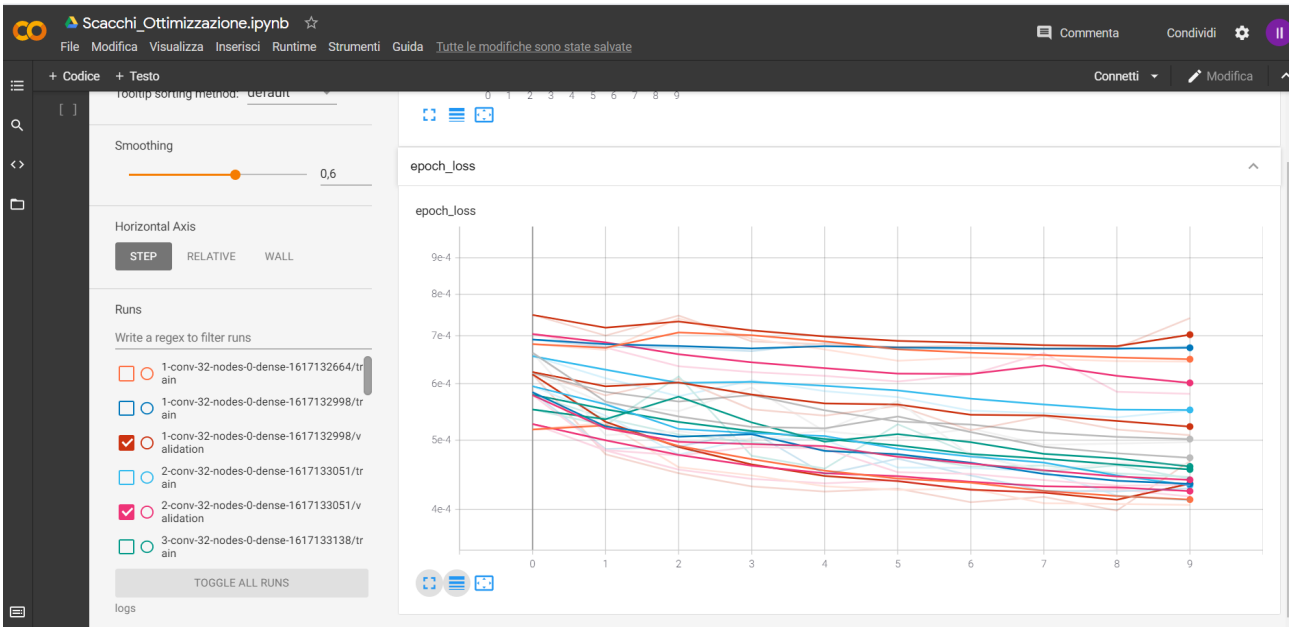


Figura 12 - Esempio di valutazione fatta con TensorBoard.

5.2 Problema: immagine della scacchiera di partenza

Il funzionamento dell'applicazione richiede uno screenshot preso dal dispositivo e questo porta ad un paio problematiche che influiscono notevolmente sul funzionamento del modulo che riconosce la scacchiera.

- L'applicazione di scacchi da cui viene scattato lo screenshot impatta notevolmente sul risultato. Il nostro modulo funziona bene su immagini prese dall'applicazione Chess.com, molto meno su altre.
- La dimensione e la risoluzione dello schermo del dispositivo che scatta lo screenshot. Abbiamo notato che questo modifica il posizionamento e la dimensione della scacchiera. Abbiamo risolto parzialmente inserendo modifiche nei valori dei parametri che si occupano del riconoscimento della scacchiera in base alle dimensioni dello schermo. Tuttavia andrebbero fatti molti più test su diversi dispositivi. Rimane uno dei problemi aperti per futuri sviluppi.

5.3 Problema: TensorFlow Lite

TensorFlow Lite è un ToolKit il cui scopo è quello di ottimizzare i modelli di partenza al fine di renderli più leggeri e con maggiore velocità di inferenza (ovvero di generare predizioni), in modo da renderli portabili anche su dispositivi con limitate capacità computazionali (come un buon numero di smartphone). Abbiamo convertito entrambi i modelli, ma il risultato non è stato soddisfacente. La perdita in accuratezza di entrambi i modelli è stata, a nostro avviso, troppo grande per poterla preferire al modello non ottimizzato. In particolare il modello di classificazione dei pezzi riconosceva in modo corretto solo le caselle vuote e i pedoni. Essendo i modelli di partenza comunque leggeri in termini di occupazione di memoria (70MB il primo, 3MB il secondo) abbiamo deciso di testarli con Android Studio mediante Chaquopy. Entrambi sono risultati sufficientemente veloci nelle predizioni, in particolare ragionando sullo scopo dell'applicazione che non richiede risposte in real time o in tempi particolarmente brevi, pertanto abbiamo deciso di non impiegare i modelli Lite. Seguendo questa scelta implementativa, l'attesa per la generazione della notazione FEN della scacchiera impiega diversi secondi, mentre quella per la predizione è abbastanza immediata.

5.4 Ottimizzazione: algoritmo minimax

Si è presentato un altro problema: la mossa consigliata risultava talvolta poco sensata. Abbiamo notato che questo era dovuto al fatto che i valori delle predizioni generate dal modello di valutazione della board erano molto simili tra loro. Era chiaro, dunque, che in alcuni casi occorreva un maggior discernimento della scelta. Abbiamo dunque implementato un algoritmo di minimax nella ricerca della scelta migliore, andando ad analizzare anche le possibili mosse successive dell'avversario. Questo migliora notevolmente il risultato finale scartando gran parte delle mosse poco sensate, tuttavia richiede un tempo di esecuzione notevolmente maggiore. Su PC impiega circa 30 secondi, mentre su mobile arriva fino a 6 minuti, un tempo decisamente troppo alto.

```
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
```

Figura 13 - Algoritmo Minimax

Chapter 6

Conclusioni e sviluppi futuri

In questa relazione abbiamo visto una panoramica dell'applicazione implementata e vari dettagli degli step del processo.
L'applicazione è passibile di notevoli miglioramenti. Possibili sviluppi futuri prevedono una scelta dinamica di parametri per gestire il riconoscimento della scacchiera a partire da uno screenshot, la generazione di nuovi modelli più accurati per TF-Lite per rendere l'esecuzione più veloce su dispositivi mobile.

Chapter 7

Bibliografia

[1] Rete neurale convoluzionale, su it.wikipedia.org

[2] Notazione Forsyth-Edwards, su it.wikipedia.org

[3] Documentazione Chaquopy su https://chaquo.com/chaquopy/doc/

[4] Documentazione tensorflow su https://www.tensorflow.org/