PRESENTAZIONE PROGETTO ELABORAZIONE delle IMMAGINI 2017-2018

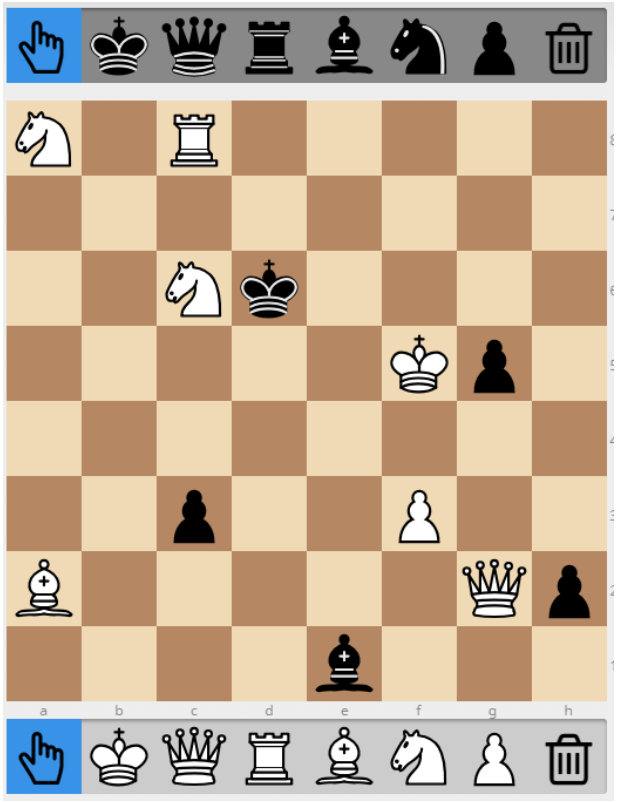
Di Emil Osterhed

OBIETTIVO

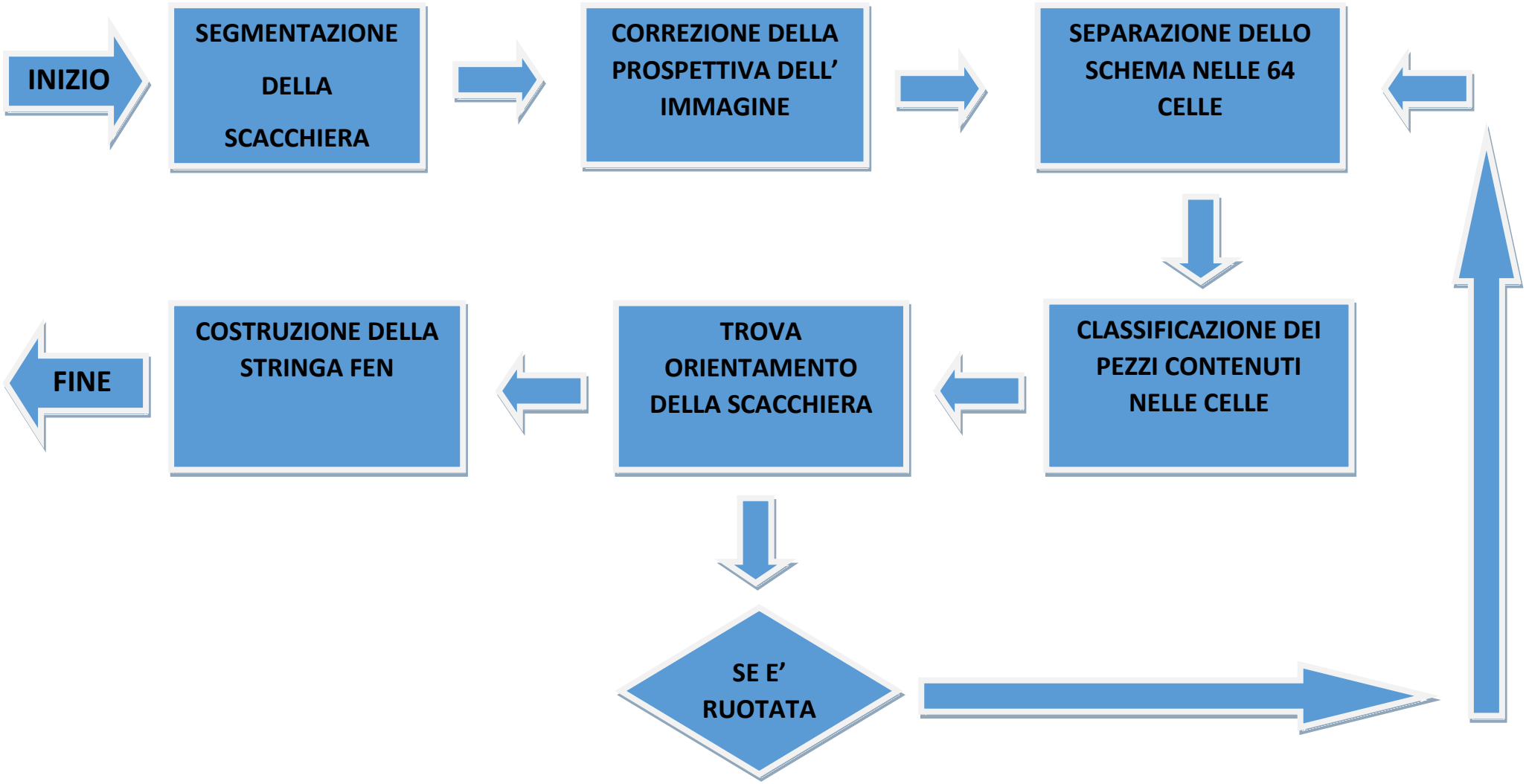
Data un’ immagine, contenente lo schema del gioco degli Scacchi, riconoscere la scacchiera e costruire la stringa di caratteri secondo la codifica FEN usata come standard per descrivere la distribuzione dei pezzi all’ interno dello schema

Stringa FEN

N1R5/8/2Nk4/5Kp1/8/2p2P2/B5Qp/4b3 – 0 1

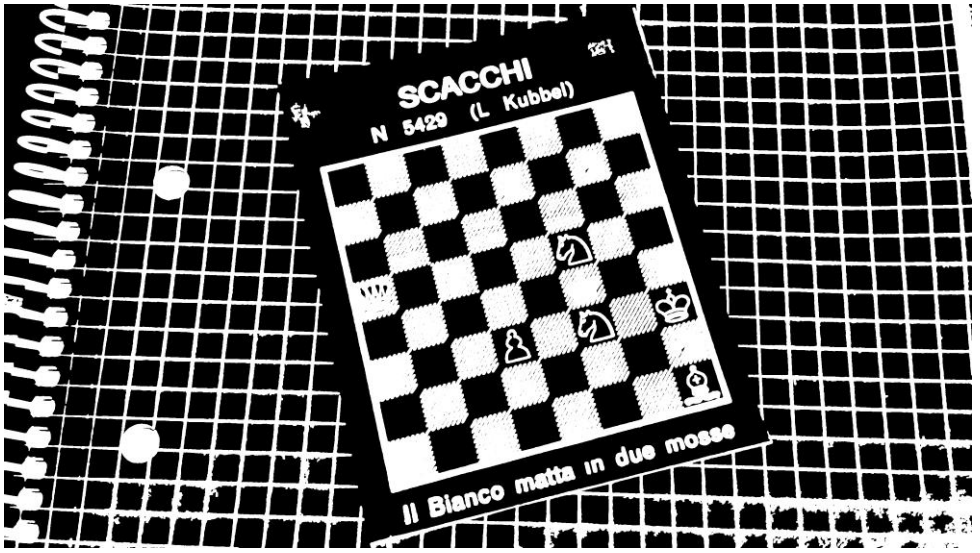


APPROCCIO PER IL PROGETTO



SEGMENTAZIONE DELLA SCACCHIERA

Rimozione rumore e binarizzazione



Blocco di processing:

```
blur = imgaussfilt(Image,2);
T = adaptthresh(blur,0.6);
bin = ~imbinarize(blur,T);
bin2 = bwareaopen(bin,200);
kernel = strel('square',3);
bw = imdilate(bin2,kernel);
```

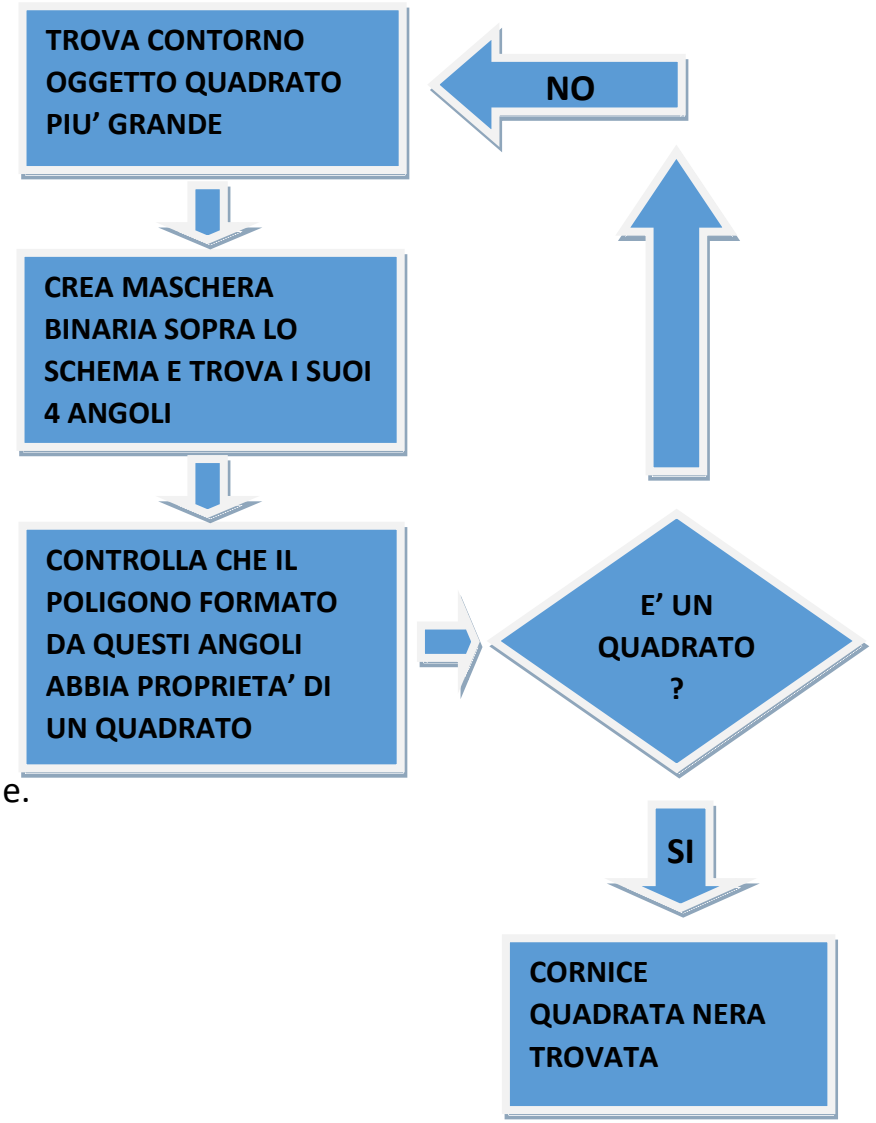
Filtro Gaussiano per rimuovere il rumore e lasciare in risalto le linee.

Binarizzazione con thresholding adattivo per essere meno debole contro le variazioni di luminosità.

Rimozione di piccoli pixel isolati e dilatazione delle linee.

SEGMENTAZIONE DELLA SCACCHIERA

Individuazione della cornice quadrata nera



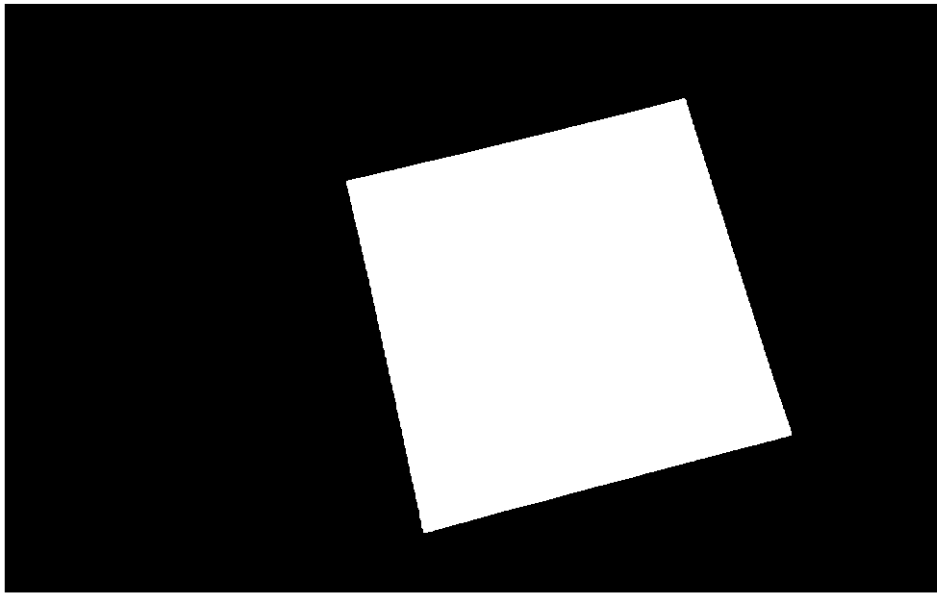
Trovare il contorno dell' oggetto con forma quadrata più grande nell' immagine.

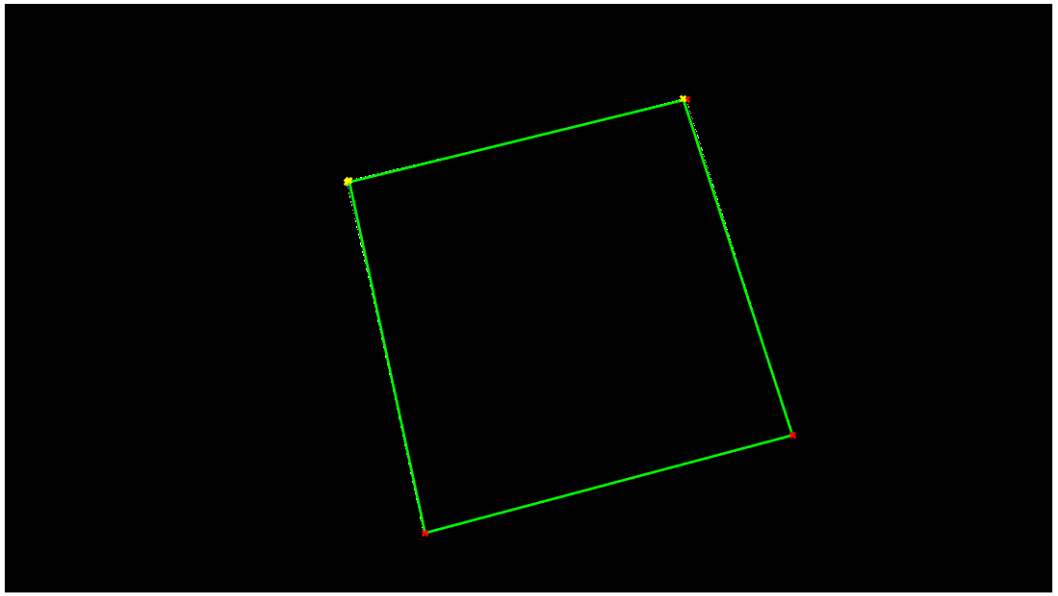
Si presuppone infatti che lo schema sia l' oggetto quadrato con bordo nero

più grande presente.

SEGMENTAZIONE DELLA SCACCHIERA

Crea maschera binaria sopra lo schema e trova i suoi 4 angoli

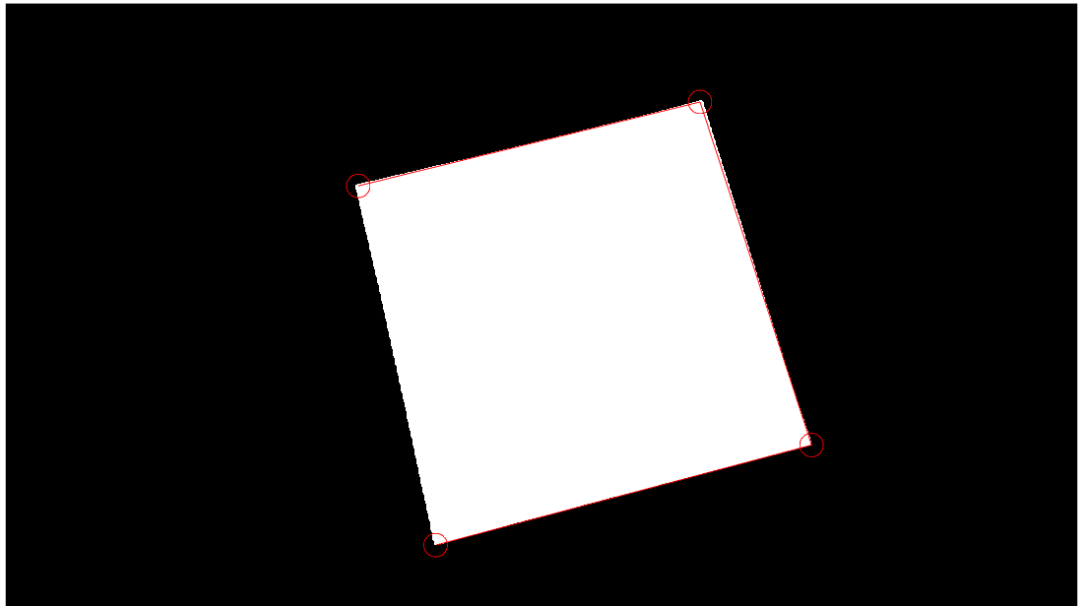




Dalla maschera binaria che sovrappone lo schema ottiene i lati di questo poligono prima trovando gli *edge* dell' immagine e poi tramite la trasformata di Hough; successivamente controlla che le 4 linee siano a due a due parallele e perpendicolari e infine dalle intersezioni di queste linee trova i 4 angoli.

SEGMENTAZIONE DELLA SCACCHIERA

Controlla che il poligono formato da questi angoli abbia le proprietà di un quadrato



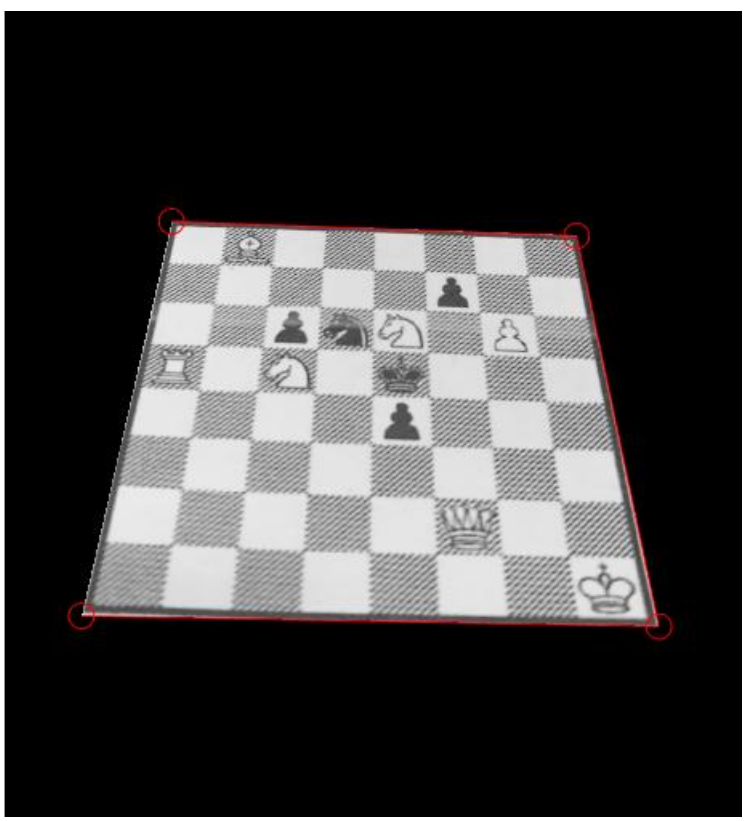
```
T_diagonals = 100;
T_edges = 300;
T_diff = 350;
```

Il poligono deve avere tre requisiti per essere

il quadrato della cornice nera che vogliamo trovare:

- Le diagonali sono circa congruenti, cioè la loro differenza è minore di *T_diagonals* (valutata in base ai quadrati più distorti).
- I lati hanno almeno una lunghezza minima decisa da *T_edges* (scelta osservando gli schemi più piccoli).
- I lati sono circa congruenti, cioè la loro lunghezza meno la loro media è minore di un certo valore *T_diff* (valore alto stabilito per comprendere le scacchiere più in prospettiva).

CORREZIONE DELLA PROSPETTIVA DELL' IMMAGINE



Output Size

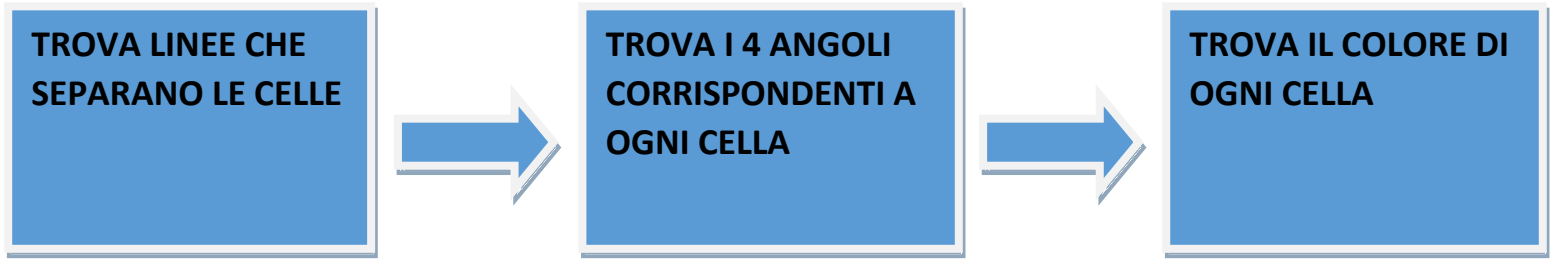
=

1008x1008

Viene stabilita una dimensione di output predefinita (*OutputSize*) con un valore multiplo di 8, cioè il numero di celle in una riga, così da avere le celle di dimensione 125x125 e i restanti 8 pixel per lo spessore della cornice nera. Dunque tramite questo valore e gli angoli precedentemente trovati viene effettuata una trasformazione geometrica in prospettiva, specificata da una matrice 3x3, così da ottenere l' immagine dello schema in una vista piana dall' alto.

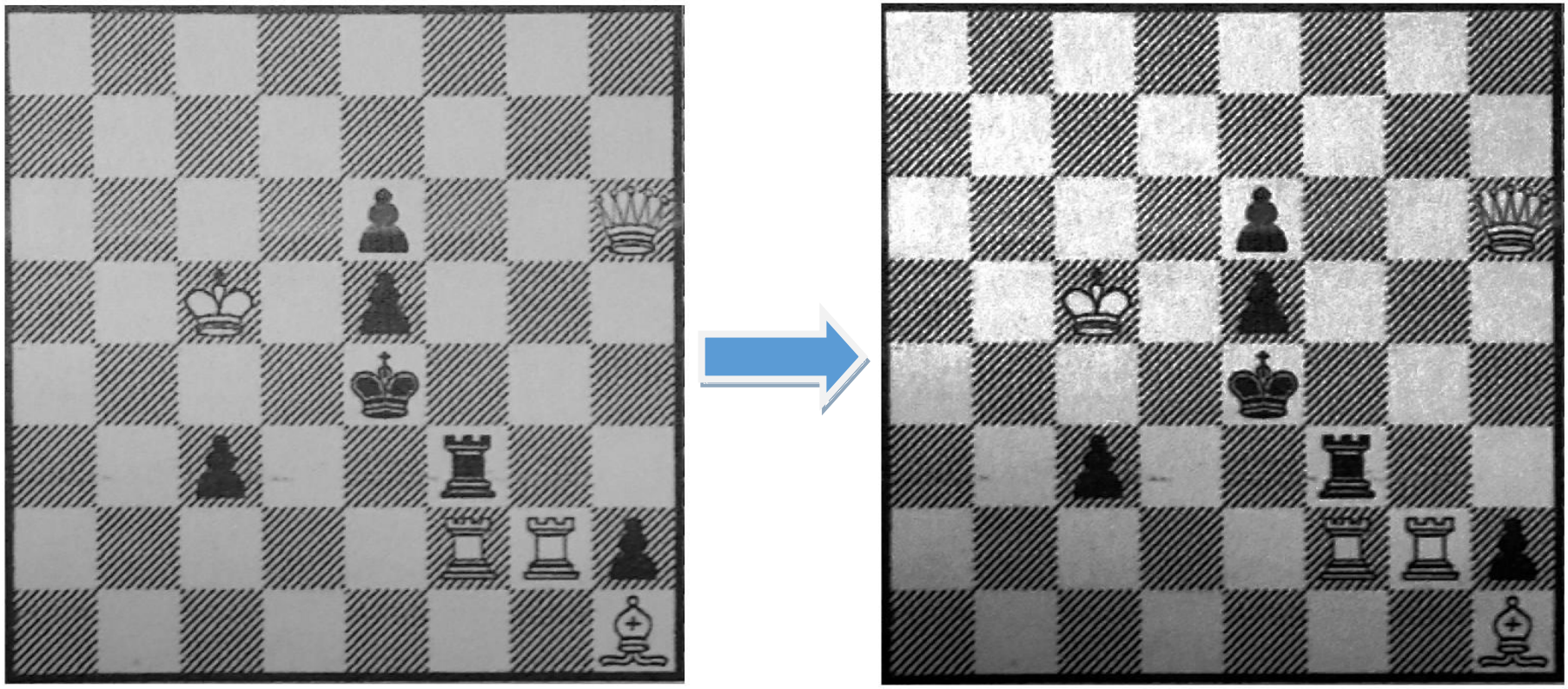
SEPARAZIONE DELLO SCHEMA NELLE 64 CELLE

Questa operazione viene suddivisa in tre blocchi identificati dal seguente diagramma a flussi:



SEPARAZIONE DELLO SCHEMA NELLE 64 CELLE

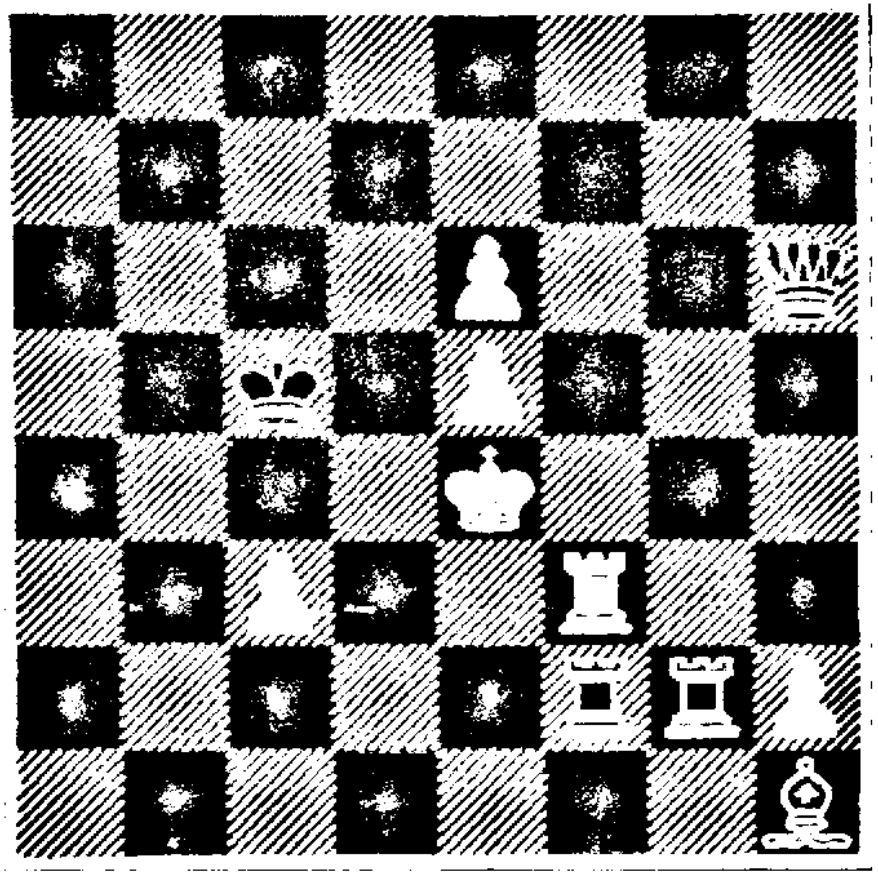
Trova linee che separano le celle



Come primo passo equalizza l’ immagine per aumentare il contrasto e risaltare la cornice nera esterna.

SEPARAZIONE DELLO SCHEMA NELLE 64 CELLE

Trova linee che separano le celle

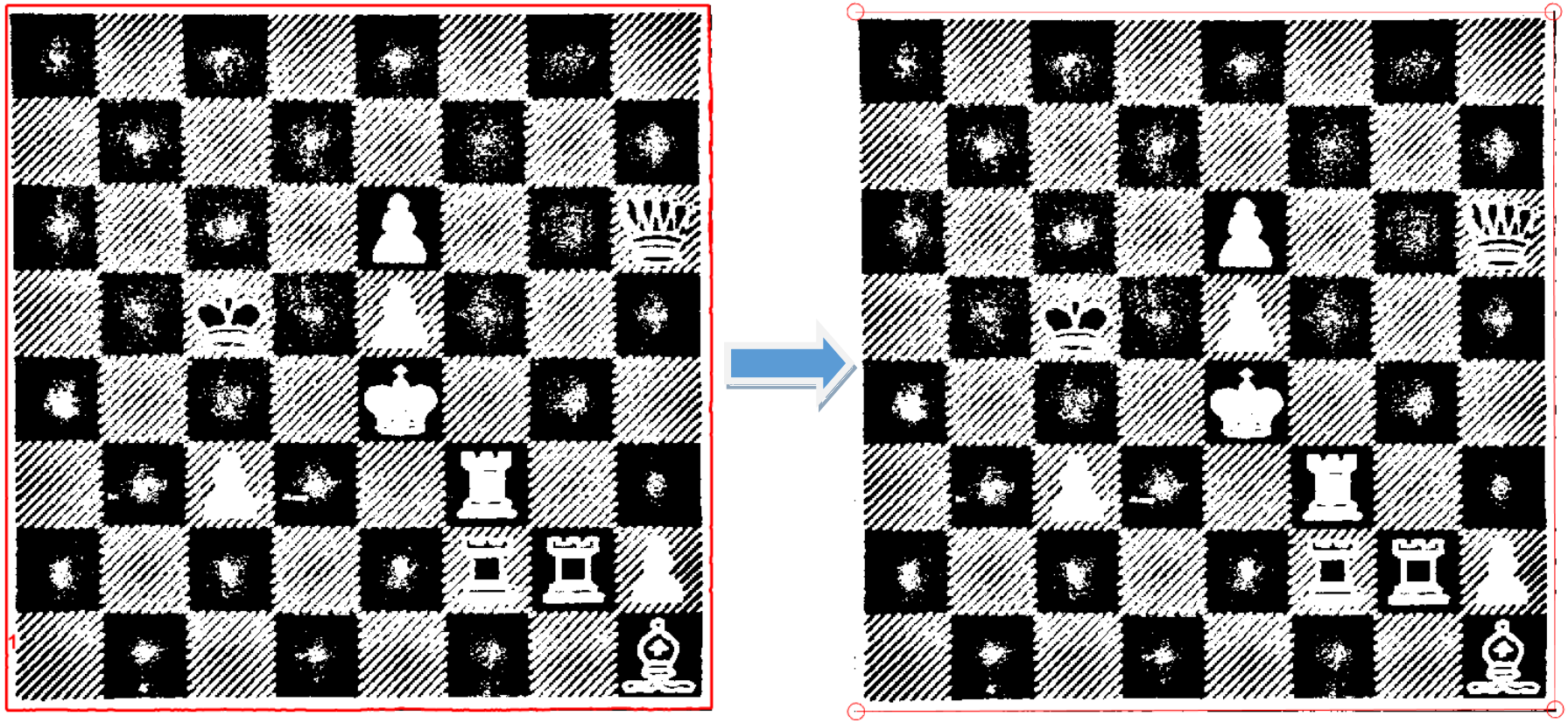


```
T = adaptthresh(h,0.47);
bw = ~imbinarize(h,T);
```

Successivamente binarizza l' immagine, anche qui usando un thresholding adattivo. Questa volta la sensibilità è più bassa perché deve venire considerata di colore nero in binario.

SEPARAZIONE DELLO SCHEMA NELLE 64 CELLE

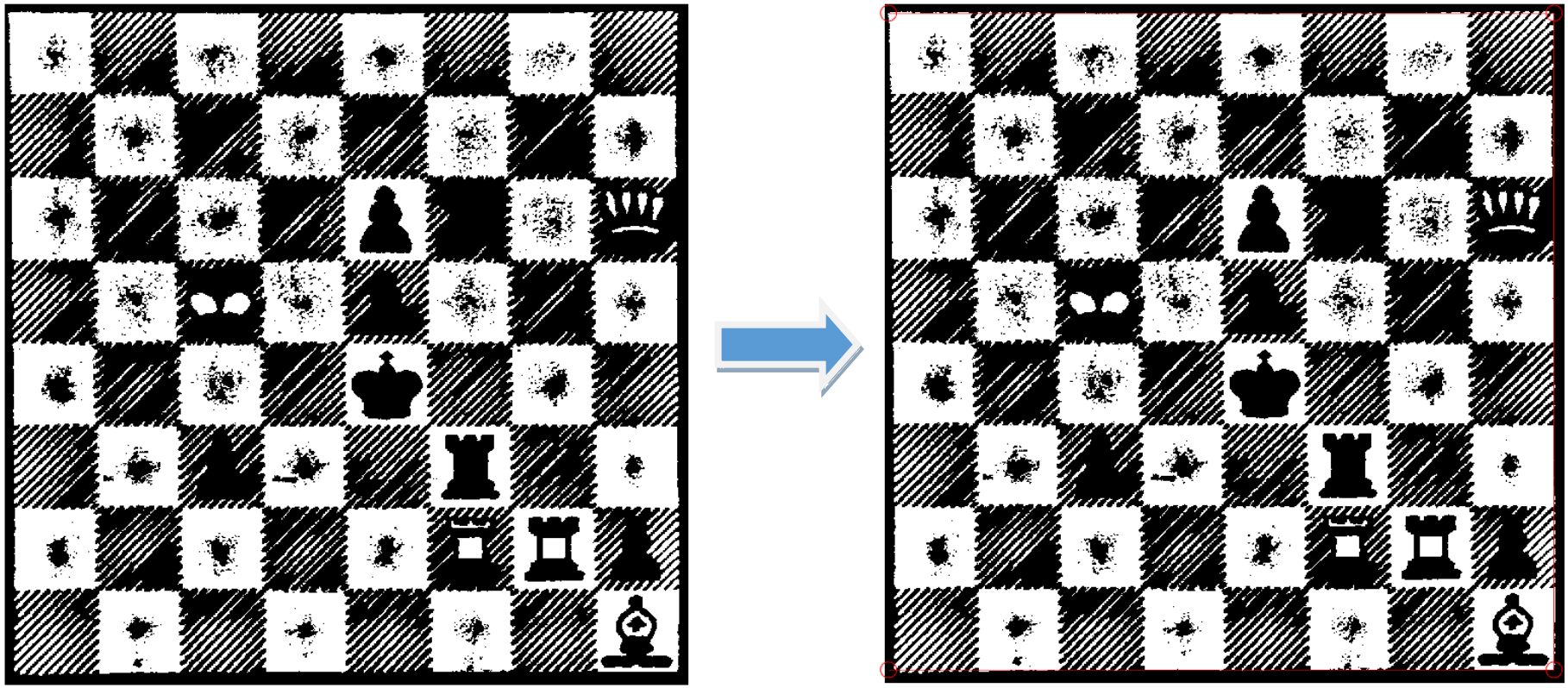
Trova linee che separano le celle



Anche qui trova il contorno esterno per poi trovare gli angoli più esterni della cornice; quest' ultimi li trova cercando i pixel bianchi più vicini ai bordi dell' immagine. Non manca il controllo che questi angoli formano un poligono con le proprietà di un quadrato(come precedentemente fatto) per evitare di prendere un contorno interno allo schema; se così fosse si proverebbe con il prossimo contorno trovato.

SEPARAZIONE DELLO SCHEMA NELLE 64 CELLE

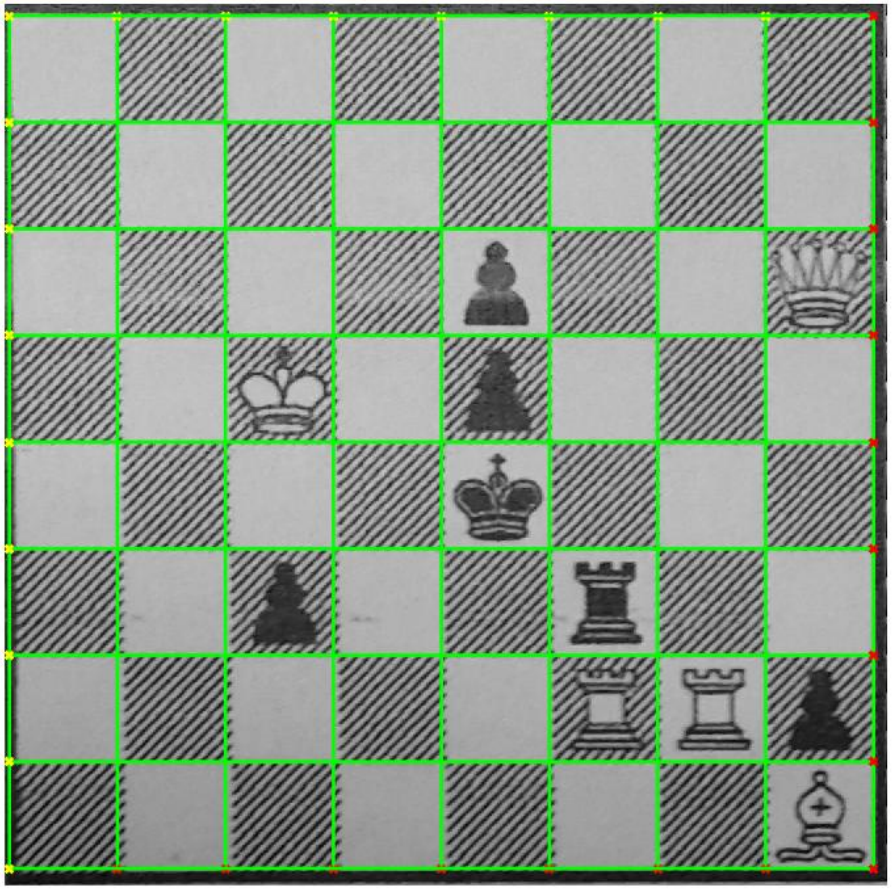
Trova linee che separano le celle



Grazie agli angoli esterni trovati, ottiene la parte di immagine sottile al di fuori di questi(che è nera) che viene sommata all' immagine binaria della slide precedente, e successivamente negata; in questo modo la cornice esterna viene cancellata ed è possibile con maggiore precisione ottenere gli angoli interni proprio attaccate alle celle.

SEPARAZIONE DELLO SCHEMA NELLE 64 CELLE

Trova i 4 angoli corrispondenti a ogni cella



Dagli angoli interni divide i 4 lati per 8, cioè il numero delle celle, e ottiene così le 18 linee che separano tutte le celle. In questo modo la divisione diventa molto più precisa anziché mantenere compresi i pixel della cornice nera.

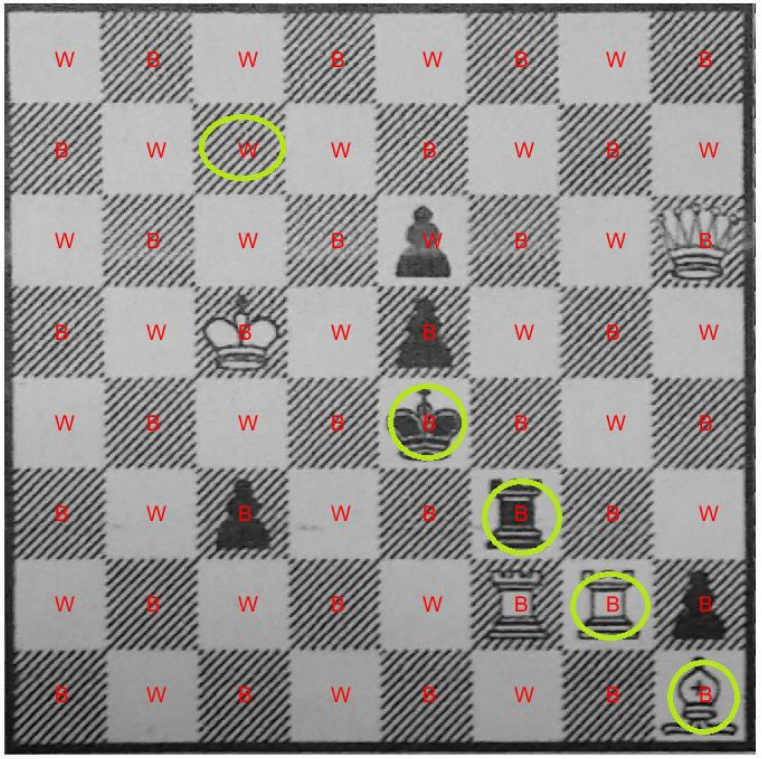
Successivamente crea una *struct* di 64 per memorizzare i 4 angoli di ogni cella ottenuti tramite le intersezioni fra tutte le linee.

Questo perché serviranno per la classificazione delle immagini delle singole celle che verranno ritagliate dalla scacchiera.

Inoltre vengono predisposti in questa *struct* anche i campi per il colore della cella, la classe(Cella vuota, Regina, ecc..) e il colore del pezzo qualora esso fosse presente all' interno della cella.

SEPARAZIONE DELLO SCHEMA NELLE 64 CELLE

Trova colore di ogni cella



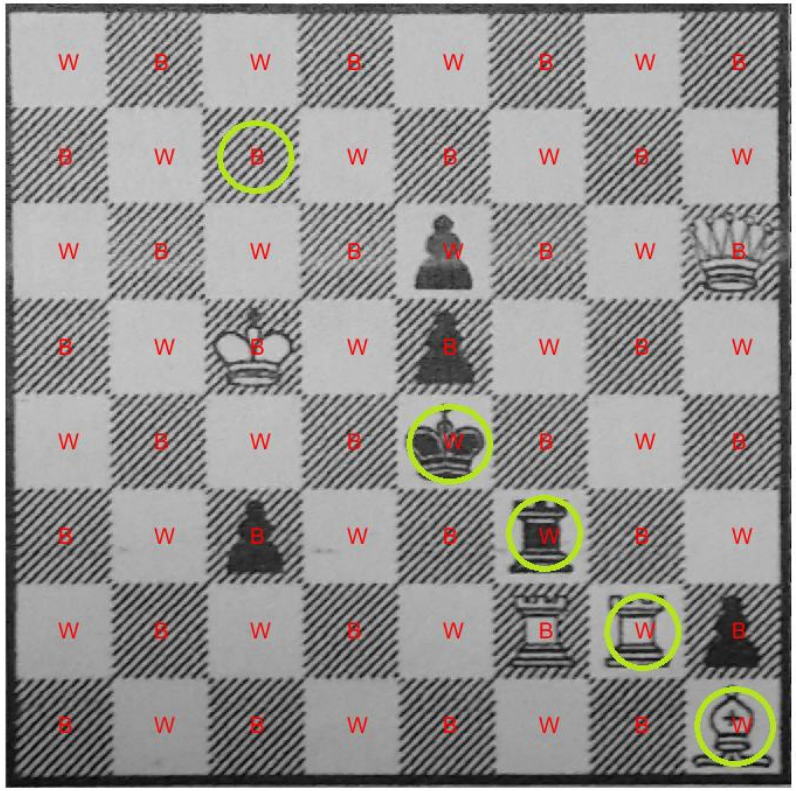
```
TL = Cells(idx).TL;
TR = Cells(idx).TR;
BL = Cells(idx).BL;
cellImage = im(TL(2):BL(2),TL(1):TR(1));
bw = imbinarize(cellImage,'global');
erode = imerode(bw,strel('square',10));

avg = mean2(erode);
if avg > 0.5
    %Colore della cella è Bianca
    myseq(idx) = 1;
    Cells(idx).color = 1;
else
    %Colore della cella è Nera
    myseq(idx) = 0;
    Cells(idx).color = 0;
end
```

Per individuare il colore delle singole celle, di queste la loro immagine viene prima binarizzata tramite Otsu e successivamente erosa con un kernel quadrato 10x10 così che le linee delle celle nere possano far diventare la cella più un quadrato nero. Viene poi guardata la media dei pixel bianchi o neri per stabilire il colore della cella.

SEPARAZIONE DELLO SCHEMA NELLE 64 CELLE

Trova colore di ogni cella



Tuttavia il metodo precedente può portare a degli errori nell' individuazione del colore di alcune celle. Avendo però lo schema degli Scacchi solo due possibili sequenze di colore(bianco e nero, o nero e bianco), la sequenza identificata dal metodo precedente viene confrontata con queste due sequenze di default.

La sequenza di default delle due che disterà di meno da quella trovata sarà quindi la sequenza della scacchiera raffigurata.

Il colore delle celle servirà per capire già parzialmente l' orientamento della scacchiera: infatti se la prima cella in alto a sinistra è bianca, allora lo schema è già dritto oppure solo girato di 180°; viceversa se la prima è nera allora sarà ruotato a sinistra o a destra.

CLASSIFICAZIONE DEI PEZZI CONTENUTI NELLE CELLE

Costruzione del dataset delle immagini delle celle

Label	Count
Bishop	712
EmptySquare	3042
King	912
Knight	960
Pawn	1120
Queen	702
Rook	720

Tramite tutti i passaggi precedenti è stato creato un dataset delle immagini di tutte le celle degli schemi dati; queste vengono tutte salvate in dimensioni 125x125 e si arriva a un numero di immagini pari a 3648; tuttavia tremila erano le celle vuote, dunque il dataset è stato incrementato delle sole immagini contenenti un pezzo con trasformazioni di gamma e applicazioni di filtri gaussiani, mediani, laplaciani e di media. Ottenendo così un numero di immagini pari a 8168, in particolar modo è stato incrementato il numero di regine nere che era davvero di molto inferiore alle altre classi.

Successivamente per ogni immagine del dataset sono state calcolate le features di HOG; è stato scelto questo descrittore per la sua rotation-invariance e perché veloce da calcolare.

Infine è stato addestrato un classificatore KNN in grado di riconoscere le celle vuote o i singoli pezzi.

CLASSIFICAZIONE DEI PEZZI CONTENUTI NELLE CELLE

Riconoscimento del contenuto della cella

```
%Parametri per calcolare HOG su ogni cella per classificarla
ImageCellSize = [125 125];
CellSize = [16 16];
BlockSize = [4 4];

crop = imcrop(rectifiedChessboard,[x y width height]);
resized = imresize(crop, ImageCellSize);
med = medfilt2(resized,[5 5]);
cropCell = imsharpen(med);


%Calcola HOG sull' immagine della cella e individua la
classe di
%appartenenza del pezzo della scacchiera
hog = extractHOGFeatures(cropCell,'CellSize',CellSize,
'BlockSize',BlockSize);

[Label, ~] = predict(PieceClassifier,double(hog));
Cells(c).pieceClass = Label{1};
```

Durante la fase di riconoscimento delle celle effettuo prima delle operazioni di ridimensionamento a 125x125 delle celle segmentate in precedenza, poi applico un filtro mediano e infine eseguo un *unsharp masking* per recuperare i dettagli dei bordi(gli *edge)*. Successivamente estraggo i descrittori HOG dall' immagine della cella per poi andare a classificare il contenuto di essa tramite il classificatore.

CLASSIFICAZIONE DEI PEZZI CONTENUTI NELLE CELLE

Trova colore del pezzo

Data l' immagine della cella, di essa viene prima calcolata la soglia di thresh tramite Otsu e successivamente viene aumentato il contrasto applicando una correzione di gamma su una piccola porzione dell' istogramma dell' immagine per aumentarne il contrasto; questa porzione di intervallo è [T-0.05 T] dove T è la soglia thresh calcolata precedentemente; svolta questa operazione l' immagine viene binarizzata attraverso una nuova soglia ottenuta.

Il colore del pezzo è individuato guardando la finestra(45x45) centrale dell' immagine, dato che se la cella contiene il pezzo esso si trova al centro; dunque viene presa la matrice di questa parte centrale e calcolato il ratio di pixel bianchi e neri; a seconda di quale dei due sia maggiore il pezzo viene considerato bianco o nero.

```
windowSize = 45;
T = graythresh(cellImage);
gamma = imadjust(med,[T-0.05 T],[],0.4);
bw = imbinarize(gamma);

%Trovo la finestra centrale dell' immagine della cella
window = bw(windowTL_Y:(windowTL_Y+windowSize),
windowTL_X:(windowTL_X+windowSize));

%Se il ratio dei pixel neri è maggiore a quelli
bianchi il pezzo allora
%è nero, altrimenti è bianco
blackPixels = sum(sum(window==0));
whitePixels = sum(sum(window==1));
ratio1 = blackPixels / (windowSize*windowSize);
ratio2 = whitePixels / (windowSize*windowSize);
if ratio1 > ratio2
    Color = 0;
else
    Color = 1;
end
```

CLASSIFICAZIONE DEI PEZZI CONTENUTI NELLE CELLE

Trova orientamento del pezzo

Per identificare l' orientamento del pezzo vengono caricati i 12 template e poi tra questi scelto quello corrispondente alla classe e al colore del pezzo individuati con i due step precedenti.

Come detto in precedenza, grazie alla conoscenza del colore della prima cella in alto a sinistra sa che il pezzo nella schema può avere solo 2 orientazioni(dritto e 180° oppure 90° e -90°) e dunque ruota il template nelle rispettive possibili orientazioni.

Applica quindi il template matching fra le due rotazioni e confronto il punto massimo delle due matrici di coefficienti di correlazione.

in base a quale è maggiore indicherà tramite un valore l' orientamento identificato per quel pezzo.

```
%Immagine del template
T = Templates(idx).Template;

%Se la prima cella in alto a sx è bianca allora
la scacchiera può
%essere già dritta oppure ruotata di 180 gradi
if FirstCellColor == 1
    T1 = imrotate(T,-180);
    c = normxcorr2(T, ImageCell);
    c1 = normxcorr2(T1, ImageCell);
    m = max(c(:));
    m1 = max(c1(:));
    if m > m1 %L' immagine è dritta
        Orientation = 1;
    else
        Orientation = 3;
    end
else
    %La scacchiera è ruotata a SX o a DX
```

TROVA ORIENTAMENTO DELLA SCACCHIERA

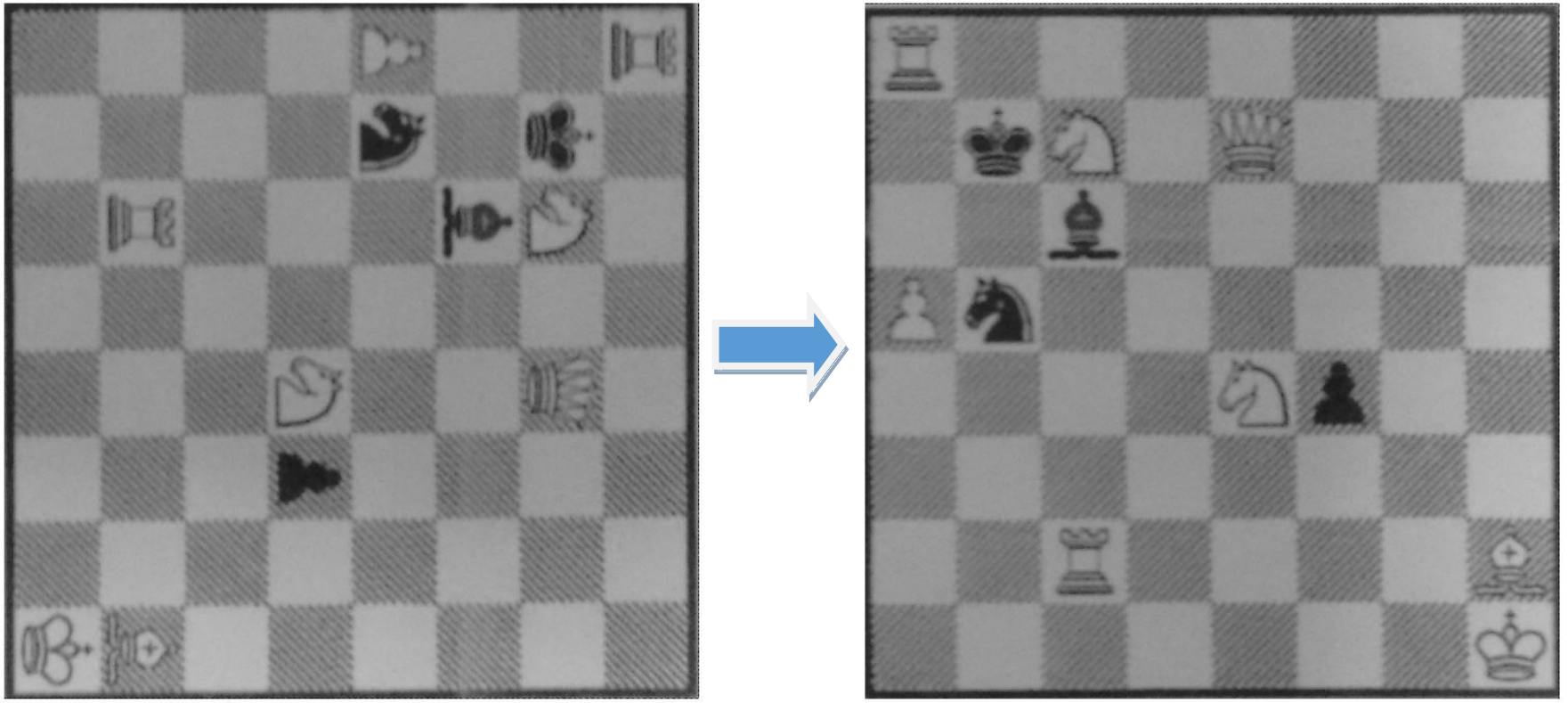
Tramite il vettore *piecesOrientations* ,in cui sono memorizzati tutti i valori di orientamento identificati dallo step precedente, confronta la maggioranza di un valore indicante una posizione verso un altro. Grazie a questo valore verrà poi ruotata la scacchiera di conseguenza.

```
% 1 Dritta
% 2 Ruotata a SX(90 gradi)
% 3 Ruotata a 180 gradi
% 4 Ruoata a DX(-90 gradi)
```

```
%Trova orientamento della scacchiera

%Rimuovo gli zeri
piecesOrientations(piecesOrientations==0) =
[];
isRotated = false;
if firstCellColor == 1
      %Scacchiera è dritta o ruotata di 180°
       or1 = sum(piecesOrientations==1);
       or2 = sum(piecesOrientations==3);
       if or2 > or1
           isRotated = true;
          rectifiedChessboard =
      imrotate(rectifiedChessboard,-180);
       end
else
       %Scacchiera è ruotata a sx o dx
       or1 = sum(piecesOrientations==2);
       or2 = sum(piecesOrientations==4);
       isRotated = true;
       if or1 >= or2 %E' ruotata a sx
           rectifiedChessboard =
imrotate(rectifiedChessboard,-90);
       else %E' ruotata a dx
           rectifiedChessboard =
imrotate(rectifiedChessboard,90);
       end
end
```

TROVA ORIENTAMENTO DELLA SCACCHIERA



Nel caso in cui lo schema sia girato e dunque venga ruotato, l' algoritmo torna allo step di *Separazione dello schema nelle 64 celle*: questo perché il numero della prima cella cambierebbe in ottica della costruzione della stringa FEN e inoltre per effettuare una analisi più precisa del contenuto delle celle.

COSTRUZIONE DELLA STRINGA FEN

STRINGA FEN:

1B6/5p2/2pnN1P1/R1N1k3/4p3/8/5Q2/7K

Tramite la *struct* di cui si parlava precedentemente è ora possibile conoscere il contenuto di ogni cella e dunque costruire la stringa secondo la codifica FEN usata come standard per indicare la distribuzione dei pezzi all' interno del gioco degli Scacchi.

RISULTATI

Precisione del classificatore

Si può analizzare dalla matrice di confusione, riportata a destra sopra, il numero di immagini usate per testare il classificatore e la relativa precisione per ogni classe.

La precisione del classificatore è del 100 %.

La percentuale di *Holdout* per il testing è del 30%,

e ho riportato la matrice ottenuta con questo settaggio poiché quella usata nello script principale *main.m*.

Tuttavia impostando una percentuale di *Holdout*

al 90% otteniamo una percentuale del 99% e si possono osservare i risultati nella tabella a destra sotto.

confMatrix(2).cm_raw

1	2	3	4	5	6	7
213	0	0	0	0	0	0
0	912	0	0	0	0	0
0	0	273	0	0	0	0
0	0	0	288	0	0	0
0	0	0	0	337	0	0
0	0	0	0	0	211	0
0	0	0	0	0	0	216

confMatrix(2).cm_raw

1	2	3	4	5	6	7
624	1	0	0	0	13	2
0	2737	0	0	0	0	0
0	4	812	0	1	3	0
0	9	4	852	0	0	0
4	5	7	4	988	0	1
0	0	4	0	0	619	8
0	1	0	0	0	0	648

RISULTATI

Precisione delle stringhe FEN

Sulle 48 immagini fornite di base dal progetto, attraverso il riscontro con i singoli file contenenti la groundtruth della stringa FEN, questo algoritmo calcola correttamente le stringhe FEN di 43 immagini, ovvero con una precisione del

89,5 %

La dimostrazione di tale risultato è ottenibile tramite l' esecuzione dello script *main.m* creato come demo per visualizzare i risultati dell' approccio dell' algoritmo per ogni singola immagine contenente una scacchiera.

Inoltre ho aggiunto 15 foto, di cui 13 appositamente per verificare che il classificatore dei pezzi riuscisse a riconoscere quelli delle immagini delle celle mai viste prima durante l' addestramento. E infatti il classificatore riesce a identificare correttamente tutti i pezzi di queste 13 immagini aggiunte.

RISULTATI

Precisione orientamento

Per quanto riguarda l' orientamento l' algoritmo riesce a individuare e ruotare correttamente 21 su 22 immagini ruotate di quelle fornite di base dal progetto, con una precisione dunque del

95,5%

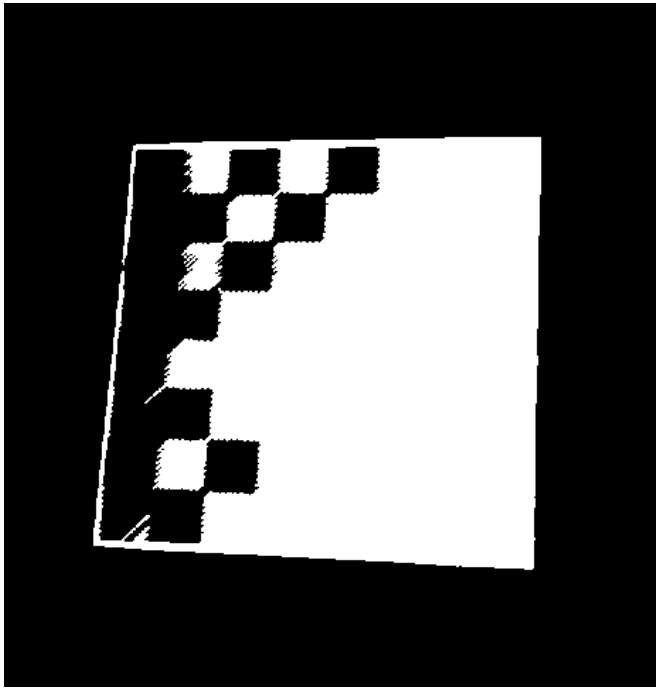
ANALISI DEGLI ERRORI

Le immagini che riportano una stringa FEN errata rispetto alla groundtruth sono le seguenti:

- **004: errore di segmentazione della scacchiera**
  GT: R7/1kN1Q3/2b5/Pn6/4Np2/8/2R4B/7K - 0 1 FEN: 6PP/P1P1P1P1/8/4p3/1Q2N3/1P1P1P1P/8/2q3R1 - 0 1
- **011: errore di individuazione del colore di un singolo pezzo dello schema**
  GT: 2bb1K2/4p3/5k1n/3QNP2/5R1P/8/8/8 - 0 1 FEN: 2bb1K2/4p3/5k1n/3qNP2/5R1P/8/8/8 - 0 1
- **020: errore di classificazione di un pezzo considerato come cella vuota**
  GT: 4rQB1/6n1/3p4/5b2/N2R4/1P6/2k5/KR6 - 0 1 FEN: 4rQB1/6p1/3p4/5b2/3R4/1P6/2k5/KR6 - 0 1
- **032: errore di orientamento non individuato correttamente**
  GT: 8/8/8/5R2/8/2KPkB1b/8/5R2 - 0 1 FEN: 2R5/8/b1BkPK2/8/2R5/8/8/8 - 0 1
- **040: errore di classificazione di un pezzo considerato come cella vuota**
  GT: 1B6/5p2/2pnN1P1/R1N1k3/4p3/8/5Q2/7K - 0 1 FEN: 1B6/5p2/2pnN1P1/2N1k3/4p3/8/5Q2/7K - 0 1

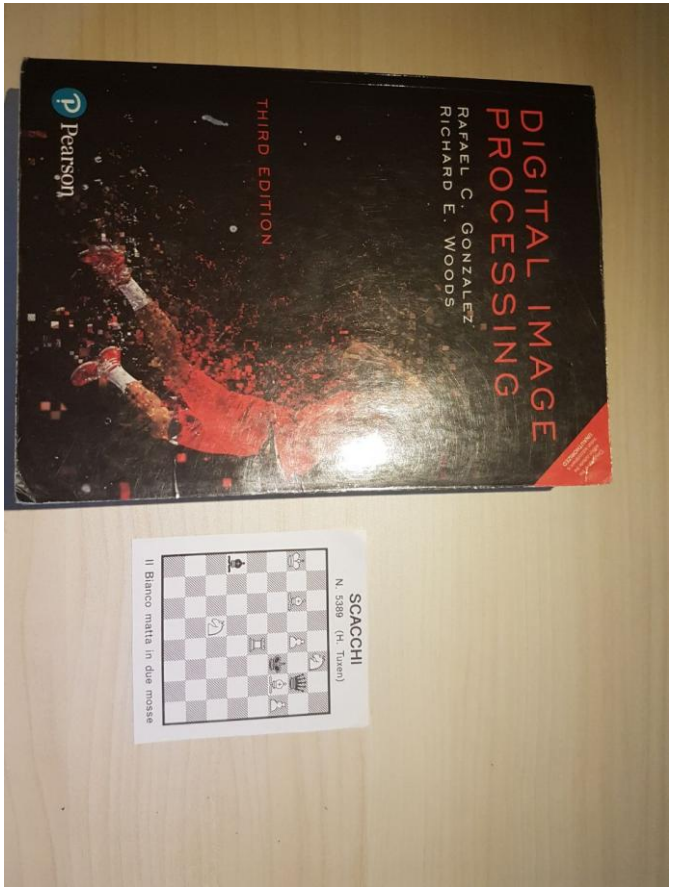
Delle immagini che ho aggiunto io due foto, che consapevolmente sapevo il mio algoritmo avrebbe sbagliato, non riesce a individuare la stringa FEN corretta per un errore di segmentazione dello schema.

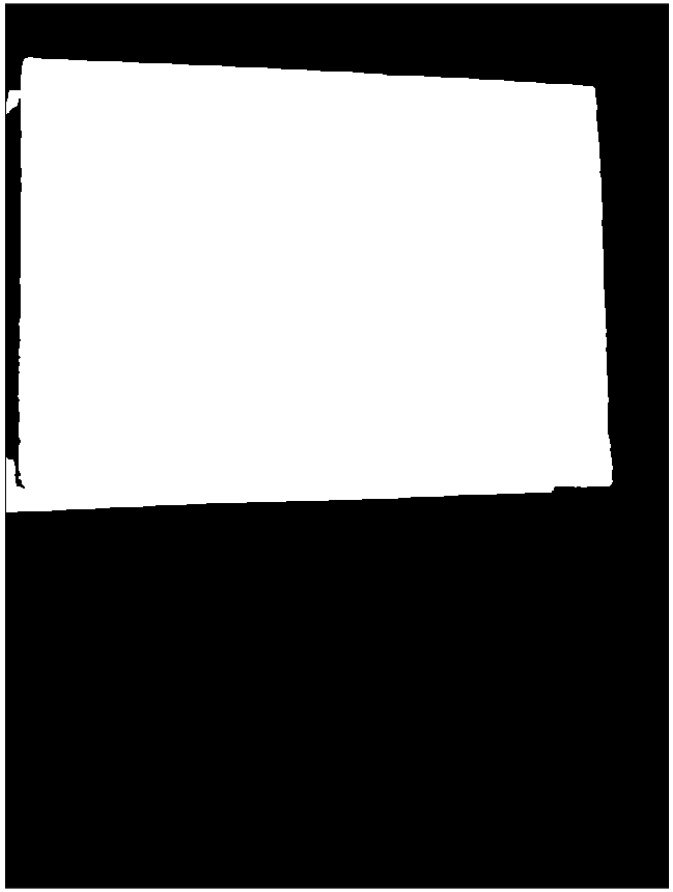
ERRORE 004



Questa immagine non viene segmentata correttamente per via della cornice nera non perfettamente chiusa e dunque viene individuato come lato a sinistra dello schema il bordo delle celle bianche più a sinistra, di fatto tagliando una colonna di celle.

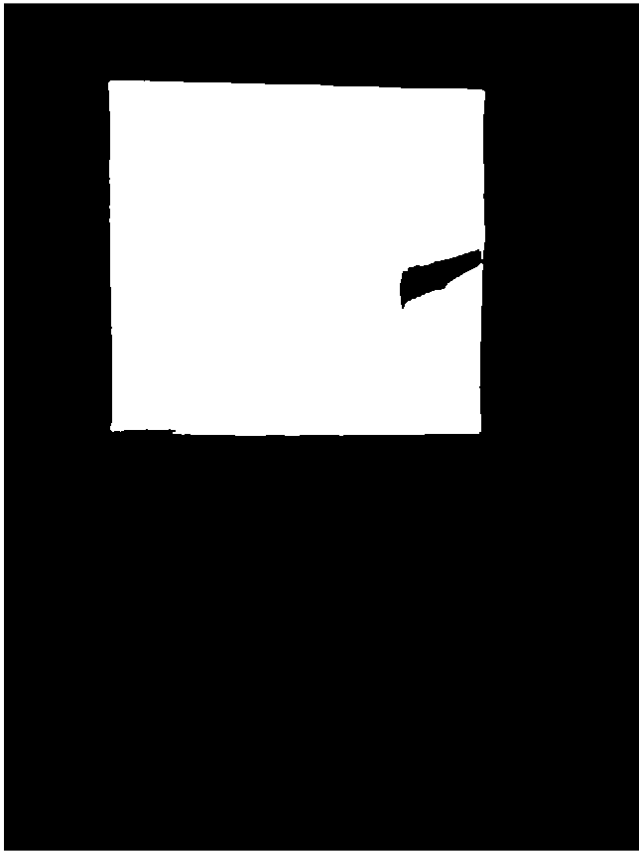
ERRORE 062





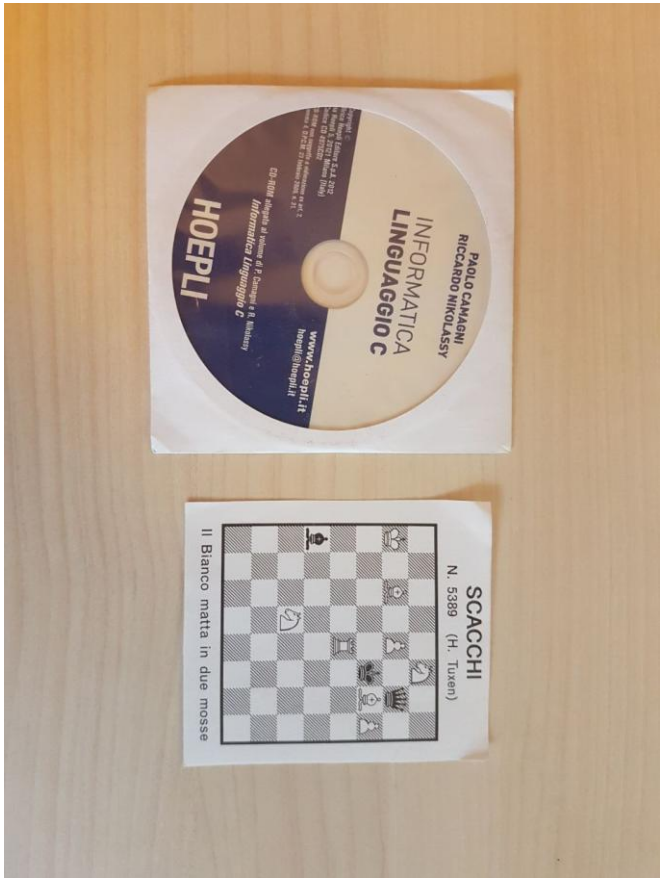
Non individua la scacchiera per via del vincolo che lo schema deve essere l’ oggetto circa-quadrato nero più grande nella foto.

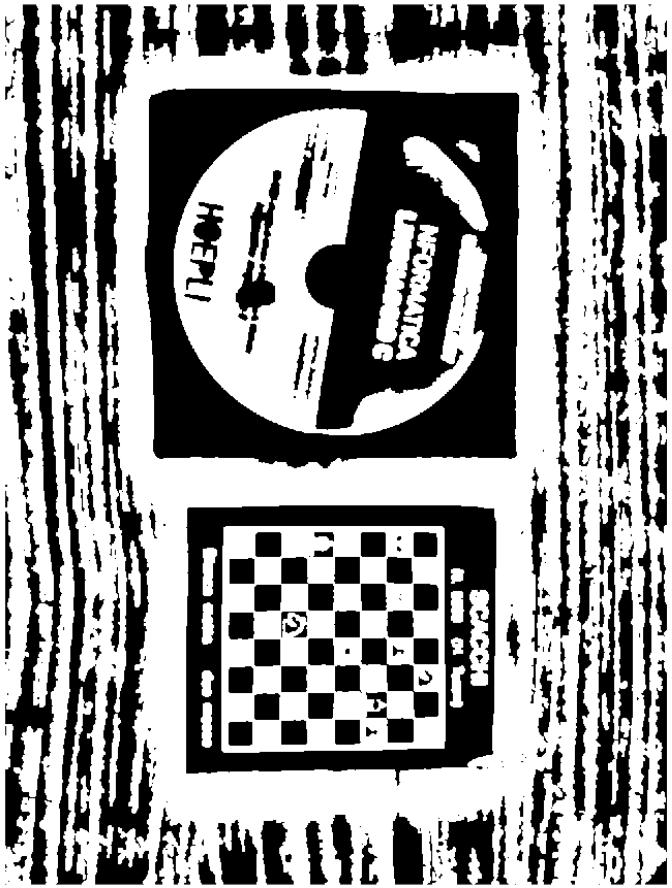
ERRORE 063



Non individua la scacchiera per via del vincolo che lo schema deve essere l’ oggetto circa-quadrato nero più grande nella foto.

IMMAGINE 064





Questa immagine invece riesce a segmentarla correttamente poiché la custodia del CD è bianca, e non nera. Dunque il vincolo è rispettato, ovvero che lo schema sia il quadrato nero più grande nell' immagine.

COSA MIGLIORARE?

Attraverso l' analisi critica dei risultati possiamo indicare quali sono i punti su cui si può migliorare l' algoritmo:

- L' algoritmo è robusto per quanto riguarda le immagini buie, in prospettiva o con variazioni di luminosità ma è leggermente debole per quanto riguarda il rumore/la sfocatura; infatti un colore di un pezzo e la classe di due pezzi non viene identificata correttamente per via di questo motivo
- Aumentare la velocità di esecuzione, che ora impiega circa 5 secondi a immagine
- Rendere più precisa la segmentazione dello schema anche nei casi in cui la cornice nera non è perfettamente intera
- Trovare un altro metodo più veloce e preciso rispetto al *template matching* per individuare l' orientamento della scacchiera
- Incrementare il dataset, in particolare le classi dei pezzi e non di *EmptySquare*, dato che infatti due pezzi sono stati classificati come celle vuote.