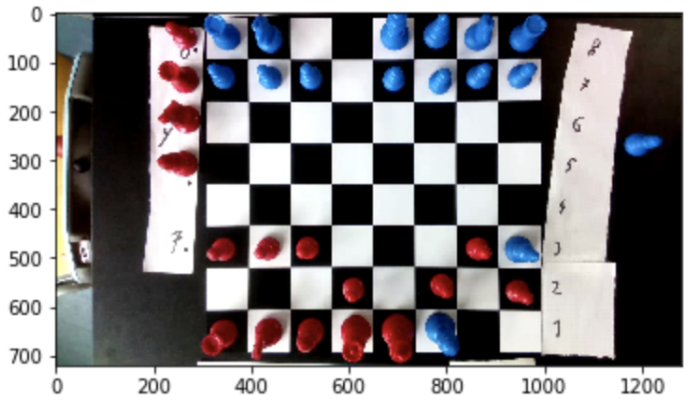
Farberkennung von Schachfiguren

Lukas Drobig

September 29, 2022

Das hier beschreibene Programm ist eine Farberkennung für Schachfiguren. Das Schachfeld wird in 64 einzelne Felder zerlegt. Dann wird jedes Feld in 3 Klassen unterschieden: leer(schwarz/weiß), rot und blau.

```
In [16]: imgplot = plt.imshow(image)
```




```
In [17]: image = image.crop((300, 10, 1020, 690))
         imgplot = plt.imshow(image)
```

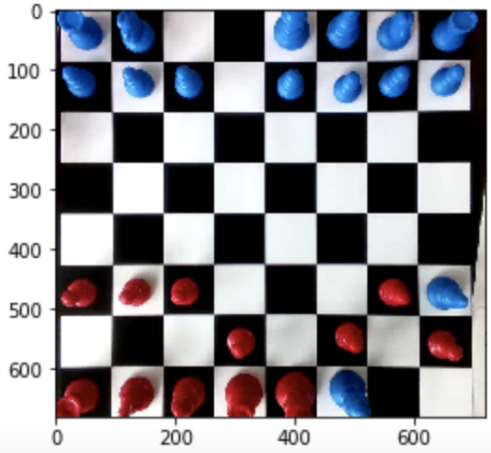


Figure 1: schneide das Schachbrett aus und teile es in 64 Teile

Zuerst wird das Bild von der Lab camera geladen, über den Link "https://lab.bpm.in.tum.de/img/high". Danach wird das Bild an den Rändern geschnitten, sodass nurnoch das Schachfeld darauf ist (Fig 1).

```
imF2 = image.crop((90,425,180,510))
imF2.save('/Users/lukasdrobig/Documents/Master_TUM/schach_bilder/imF2.png', 'PNG')
brightness = calculate_brightness(imF2)
print(brightness)
imF2 = adjust_brightness(brightness,imF2)


imF2plot = plt.imshow(imF2)
red_avg, green_avg, blue_avg = calculate_average_rgb(imF2)
color = calculate_color(red_avg,green_avg,blue_avg)
print(color)
print("avg rgb values are ",red_avg, green_avg, blue_avg)
```

```
0.6660074550653591
red
avg rgb values are  174.3052 69.5208 80.214
```

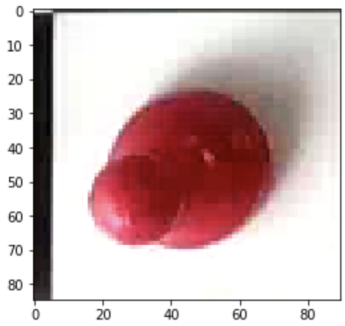


Figure 2: Prozess für ein Feld auf dem Schachbrett

Danach wird das gesamte Schachfeld in 64 einzelne Felder geschnitten. Ein Feld sind ca 85x85 pixel, je nachdem ob das Feld am Rand oder in der Mitte liegt. Nachdem die Felder ausgeschnitten wurden, wird für jedes Teilfeld bestimmt, ob es leer, eine rote Figur oder eine blaue Figur besitzt. Dazu wird für jedes Bild diese Prozedur durchgeführt:
1. Helligkeit anpassen: calculate-brightness (line 3) + adjust-brightness (line 5)
2. Berechnung der durchshnitts RGB Werte: calculate-average-rgb (line 7)
3. Berechnung der Farbe: calculate color (line 8)

```
def calculate_brightness(image):
    greyscale_image = image.convert('L')
    histogram = greyscale_image.histogram()
    pixels = sum(histogram)
    brightness = scale = len(histogram)
    for index in range(0, scale):
        ratio = histogram[index] / pixels
        brightness += ratio * (-scale + index)
    return 1 if brightness == 255 else brightness / scale
```

```
def adjust_brightness(brightness,image):
    factor = 0.5-(brightness)**2
    enhancer = ImageEnhance.Brightness(image)
    image = enhancer.enhance(1+factor)
    return image
```

Figure 3: Helligkeits Filter, um Wetteränderungen auszugleichen

Um die Helligkeit des Bildes zu bestimmen, wird zuerst das Bild in ein Graustufen Bild umgewandelt. Die Anzahl der Pixel im Histogramm wird in ein ratio mit der Länge des Histograms gesetzt. Dadurch wird für brightness ein Wert zwischen 0 und 1 berechnet, wobei 1 ein komplettt weißes Bild und 0 ein komplett schwarzes Bild darstellt.

In der adjust Brightness Function wird ein factor für die Helligkeitsanpassung bestimmt. Wenn das Bild besonders hell ist (brightness¿0,5), dann wird der factor kleiner als 1 und das Bild wird durch enhancer ein dunkler gefiltert.

```
def calculate_average_rgb(image):
    sum_red = 0
    sum_green = 0
    sum_blue = 0
    for i in range(50):
        for j in range(50):
            red,green,blue = image.getpixel((i+20,j+20))
            sum_red = sum_red+red
            sum_green = sum_green+green
            sum_blue = sum_blue+blue
            # 20x20 = 400 pixel rgb values, compute average

    return sum_red/2500,sum_green/2500,sum_blue/2500
```

Figure 4: Berechne die durchschnittswerte für red, green, blue channels

Um die durschnitlichen RGB Werte der Felder zu berechnen, werden nur die inneren 50x50 pixel betrachtet, da die Figuren in der Mitte stehen. Durch die methode image.getpixel(pixel position) werden die Werte für Rot, Grün und Blau zurück gegeben. Die Werte jeder Farbe werden aufaddiert und durch Anzahl der betrachteten Pixel geteilt. Das Ergebnis ist ein Wert zwischen 0 und 255 für jede Farbe.

```python
def calculate_color(r,g,b):
    diff_blue = abs(r-0)+abs(g-0)+abs(b-255)
    #diff_blue = math.sqrt(((r-0)**2)+((g-0)**2)+((b-255)**2))
    diff_red = abs(r-255)+abs(g-0)+abs(b-0)
    diff_yellow = abs(r-255)+abs(g-255)+abs(b-0)
    #diff_red = math.sqrt(((r-255)**2)+((g-0)**2)+((b-0)**2))
    diff_white = abs(r-255)+abs(g-255)+abs(b-255)
    #diff_white = math.sqrt(((r-255)**2)+((g-255)**2)+((b-255)**2))
    diff_black = abs(r-0)+abs(g-0)+abs(b-0)
    #diff_black = math.sqrt(((r-0)**2)+((g-0)**2)+((b-0)**2))
    min_diff = min(diff_blue,diff_red,diff_yellow,diff_white,diff_black)
    if(min_diff == diff_red):
        color = "red"
        return color
    if(min_diff == diff_blue):
        color = "blue"
        return color
    if(min_diff == diff_yellow):
        color = "yellow"
        return color
    if(min_diff == diff_white):
        color = "white"
        return color
    if(min_diff == diff_black):
        color = "black"
        return color
```

Figure 5: Berechne anhand der durchschnittswerte der RGB channels die farbe

In der caluclate color Function wird der kürzeste Abstand zwischen den durschnittlichen RBG Werten des Feldes und den Farben berechnet. Das Ergebnis ist die Farbe, die den kürzesten Abstand zum durschnittlichen RGB Wert hat. Weiß(255,255,255) Schwarz(0,0,0) Rot(255,0,0) Blau(0,0,255)

```python
@app.get("/chess/")
async def root():
    return json.dumps(chessboard)
```

Figure 6: Erstelle ein JSON Array mit 8x8 feldern

Das chessboard array ist ein 8x8 array, welches das Schachbrett beschreibt. Mit der json.dumps Methode wird in ein Json Array umgewandelt.

```
"[[2, 0, 0, 0, 2, 0, 2, 2], [0, 2, 0, 0, 0, 2, 0, 0], [0, 0, 0, 0,
0], [0, 1, 0, 1, 1, 2, 0, 0]]"
```

Figure 7: Das Schachfeld wird als JSON Array ausegebn mit 0 = empty, 1 = red, 2 = blue

Das Ergebnis wenn der Port auf dem lehre Server aufgerufen wird. Das JSON Array chessboard wird zurück gegeben.