MINISTERUL EDUCAŢIEI NAŢIONALE

UNIVERSITATEA „1 DECEMBRIE 1918” DIN ALBA IULIA

MASTER: PROGRAMARE AVANSATĂ ŞI BAZE DE DATE

LUCRARE DE DISERTAŢIE

COORDONATOR ȘTIINȚIFIC, **ABSOLVENT,**

CONF. UNIV. DR. KADAR MANUELLA OLARIU BURBĂU LUCIAN NICOALE

ALBA IULIA

2 0 1 9

MINISTERUL EDUCAŢIEI NAŢIONALE

UNIVERSITATEA „1 DECEMBRIE 1918” DIN ALBA IULIA

MASTER: PROGRAMARE AVANSATĂ ŞI BAZE DE DATE

IMPLEMENTAREA UNUI SISTEM SOFTWARE DE EXTRAGERE A TEXTULUI DIN IMAGINI

COORDONATOR ȘTIINȚIFIC, **ABSOLVENT,**

CONF. UNIV. DR. KADAR MANUELLA OLARIU BURBĂU LUCIAN NICOALE

ALBA IULIA

2 0 1 9

Mulțumiri

În realizarea acestei lucrări, am fost susținut, la tot pasul, de către coordonatorul meu științific, conferențiar universitar doctor Kadar Emanuella. Doresc să îi transmit sincere mulțumiri pentru profesionalismul, integritatea și excelența didactică de care a dat dovadă, fară de care acestă lucrare nu ar fi fost posbilă.

Doresc să mulțumesc și colectivului de profesori al universitații 1 Decembrie 1918 Alba Iulia pentru suportul acordat de-a lungul anilor în cardrul cursurilor de masterat. Efortul lor de a împărți cunoștiinte este titanic, iar rezultatul se vede în stundenții universității, aceștia fiind pregătiți la un nivel înalt, capabili să concureze cu stundenții universităților de calibru înalt, atât din Romania cât și din afara tării. Eu, realizatorul acestei lucrări, mă consider norocos că am făcut parte din colectivul acestei universități.

Nu în ultimul rând, adresez sincere și calde mulțumiri familiei mele pentru neîncetatul suport acordat de-a lungul studenției mele, fară de care, din nou, acestă lucrare nu ar fi fost posibilă.

Abstract

From the end of the First Industrial Revolution approximately 200 hundred years ago, humans have always sought to improve the systems and tools used in the manufacturing process. In our day and age, with the advent of computer assisted engineering and automation processes, these systems have become almost autonomous, requiring very little human interference. Now with the coming of artificial intelligence technologies based in cloud computer infrastructures, it has become increasingly easy to implement software that leverages their unique capabilities.

The goal of this paper is to present the implementation of one such software system dealing with optical character recognition from digital images. The main purpose of this software is to preprocess the image by applying different image processing techniques to make the text as visible and clear as possible and then extract the text from the images by making a HTTP request with the image to Google Vision OCR web service.

The preprocessing of images before they are computed by an optical character recognition software is almost unique in every case and it's very hard to be tied down to a certain method of processing, but there are some common techniques that can applied like logarithmic transforms, binarization, histogram equalization, blurring, image subtractions, morphological transformations and edge detection. Each of these techniques can be used in combination with one and other to help preprocess the image with the goal of making text stand out. In the case detailed in this paper, the images are that are processed consist of the backside of ceramic plates that have text on them. Although the images are very high resolution, the contrast between the text and the rest of the image is almost nonexistent making very hard to see let alone understand the meaning of text even by the human brain making these techniques crucial.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

Rezumat

De la sfârșitul primei revoluții industriale, cu aproximativ 200 de ani în urmă, oamenii au căutat întotdeauna să îmbunătățească sistemele și instrumentele folosite în procesul de fabricație. În zilele noastre, odată cu apariția proceselor de inginerie și automatizare asistate de calculator, aceste sisteme au devenit aproape autonome, necesitând o interferență umană minină. Acum, odată cu apariția tehnologiilor de inteligență artificială bazate pe infrastructurile informatice ale cloud-ului, a devenit din ce în ce mai ușor să se implementeze programe care utilizează capabilitățile lor unice.

Scopul acestei lucrări este de a prezenta implementarea unui astfel de sistem software care se ocupă cu recunoașterea optică a caracterelor din imagini digitale. Rolul principal al acestui software este de a procesa imaginea digitală prin aplicarea diferitelor tehnici de prelucrare a imaginii pentru a face textul cât mai vizibil și mai clar posibil și apoi extrage textul din imagini făcând o cerere HTTP cu imaginea procesată către serviciul web Google OCR.

Procesarea imaginilor înainte ca acestea să fie calculate de un software optic de recunoaștere a caracterelor este aproape unică în fiecare caz și este foarte greu să fie legat de o anumită metodă de procesare, dar există câteva tehnici comune care pot fi aplicate ca: transformări logaritmice, binarizare, egalizarea histogramei, estomparea, scăderea imaginii, transformările morfologice și detectarea marginilor. Fiecare dintre aceste tehnici pot fi utilizate în combinație una cu alta pentru a ajuta la preprocesarea imaginii cu scopul de a face ca textul să iasă în evidență. În cazul detaliat în această lucrare, imaginile care sunt prelucrate constau în partea din spate a plăcilor ceramice care au text pe ele. Deși imaginile au o rezoluție foarte mare, contrastul dintre text și restul imaginii este aproape inexistent, făcând foarte greu procesul de recogniție optică a caracterelor.

Cu ajutorul limbajelor de programare precum Python și bibliotecilor pentru procesarea imaginilor, cum ar fi OpenCV, implementarea software-ului este destul de ușor de realizat și ușor de gestionat.

Cuprins

1. Introducere

Implementarea unui sistem de extragere a textului din imagini presupune realizarea unei aplicații software capabilă să proceseze imagini digitale și să extragă cu o acuratețe cât mai mare textul existent în acea imagine. Aplicabilitatea unui astfel de sistem este vastă, dar se folosește în mod special, pentru digitalizarea textului ce provine din mediul analogic. Textul din imagini poate fi imprimat pe diferite suprafețe sau materiale, iar claritatea textului din imagine nu este întotdeauna bună. Sistemul software dezvoltat și descris în acestă lucrare de disertație trebuie să rezolve cerința următoare: trebuie să proceseze și să extragă text din o serie de imagini.

Astfel de sisteme au impact mare asupra procesului de colectare și catalogare a informației, prin automatizarea procesului de introducere a datelor într-un sistem digital. În cazul unui proces industrial, un sistem de acest tip reduce drastic timpul necesar pentru colectarea datelor necesare, și tot odată, ajută diverse procese tehnologice dependente de aceste informații să fie informate mai rapid și mai precis.

Modul în care imaginile digitale sunt procesate diferă de la caz la caz, fiindcă fiecare imagine sau serie de imagini sunt unice și necesită o procesare unică înainte să fie transmise pentru a extrage conținutul textual. O regulă nescrisă a unui sistem de recunoaștere a caracterelor din imagini este că, dacă ochiul uman nu poate să vadă și să distingă caracterele, atunci nici un astfel de sistem nu poate. De aici rezultă necesitatea de procesare a imaginilor. Este un proces important pentru a îmbunătăți acuratețea procesului de extragere a conținutului textual dintr-o imagine digitală. În literatura de specialitate nu există descrisă o metodă sau un procedeu clar pentru a scoate în evidentă conținutul textual din cauza unicității imaginilor, astfel trebuie aplicată una sau mai multe tehnici de procesare. Pe lângă recunoașterea optică a caracterelor, procesarea imaginilor este necesară pentru o multitudine de alte procese cum ar fi teledetecție, extragerea caracteristicilor, detectare fețelor, detectare amprentelor, sortare optică, augmentarea realității, imagistică microscopică, sistem de avertizare a ieșirii de pe bandă, reprezentare non-fotorealistică, procesarea imaginilor medicale și imagistică morfologică.

Pentru realizarea sistemului software s-a folosit limbajul de programare Python. Acest limbaj a ajuns în ultimii ani să fie din ce în ce mai folosit din cauza suportului care îl acord pentru analiza datelor. Ajutat de o multitudine de biblioteci, limbajul

Python reprezintă un mediu perfect pentru procesarea imaginilor digitale. Sistemul software realizat este compus din patru părți: achiziționarea imaginilor, procesarea imaginilor, transmiterea imaginilor în *cloud* pentru a se extrage conținutul text, afișarea conținutului extras.

1.1 Scopul lucrării

Scopul acestei lucrări de disertație îl constituie realizarea unui sistem software de extragere a textului din imagini digitale. În aceste sens se utilizarea un limbaj de programare și diferite biblioteci capabile să manipuleze imaginile din punct de vedere digital cu scopul de a scoate textul în evidentă pentru a facilita o extragere cu o acuratețe mare.

1.2 Obiective

Obiectivele atinse de acestă lucrare de disertație sunt următoarele:

a) Analiza și procesarea imaginilor.
b) Analiza celor mai comune și utilizate funcții și algoritmi de procesare a imaginilor
c) Extragerea conținutului textual din imagini folosind *cloud computing* prin procesul numit *optical character recognition* (OCR).
d) Proiectarea și implementarea aplicației folosind un caz din lumea reală.

2. Noțiuni generale despre procesarea imaginilor digitale

În ziua de azi, nu există nici un domeniu tehnic care să nu fie afectat într-un fel s-au altul de procesarea digitală a imaginilor [1]. Materialul prezentat în această lucrare de disertației este limitat dar, cu siguranță nu v-a lăsa nici o îndoială despre amploarea și importanța prelucrării imaginilor digitale. În acest capitol sunt enumerate câteva domenii de aplicare, în fiecare dintre acestea utilizându-se în mod curent tehnicile de procesare a imaginilor digitale dezvoltate în următoarele capitole. Câteva exemple de imagini din domenii care utilizarea des tehnici și modalități de procesare digitală a imaginilor: [1]

a) Imagini rezultate folosind raze gama (medicină, astronomie)

b) Imagini rezultate folosind raze X (medicină, astronomie, industrie grea)

c) Imagini rezultate folosind lumină captată din spectrul ultraviolet (industrie grea, microscopie, lasere, astronomie)

d) Imagini rezultate folosind lumină captată din spectrul vizibil și infraroșu (medicină, astronomie, electronică, industrie)

e) Imagini rezultate folosind lumină captată din spectrul microundelor (radar, astronomie, telecomunicații)

Prelucrarea imaginilor este orice formă de manipulare a semnalului luat dintr-o imagine, cum ar fi o fotografie sau un cadru video. Rezultatul prelucrării semnalului imaginii poate fi, fie o imagine, fie un set de caracteristici sau parametri aferenți imaginii neprelucrate. Cele mai multe tehnici de prelucrare a imaginii implică tratarea imaginii ca semnal bidimensional și aplicarea tehnicilor standard de procesare a semnalului.

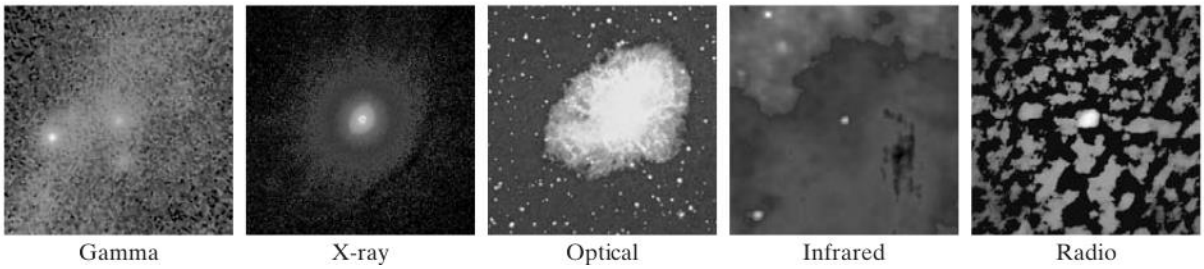


Figura 1 - Exemple imagini rezultate din cele mai comune spectre de lumină

Procesarea imaginilor se referă de obicei la procesarea digitală a imaginilor, dar prelucrarea imaginilor optice și analogice este, de asemenea, posibilă.

Procesarea digitală a imaginilor înseamnă utilizarea de algoritmi de calculator pentru a manipula imagini digitale. Ca sub-domeniu a procesării semnalelor digitale, prelucrarea digitală a imaginilor are multe avantaje față de prelucrarea imaginilor analogice. [2] Aceasta permite o gamă mult mai largă de algoritmi care pot fi aplicați datelor de intrare și pot evita probleme cum ar fi acumularea de zgomot și de denaturare a semnalului în timpul procesării. Deoarece imaginile sunt definite în două dimensiuni (poate mai multe) procesarea digitală a imaginilor poate fi modelată sub formă de sisteme multidimensionale. Un exemplu de procesare a unei imagini digitale se poate observa în Figura 2. Acestă imagine digitală îi este aplicat un filtru de *blur* de tip Gaussina folosit pentru reducerea zgomotului.

Figură 2 - Imagine procesată cu filtru de tip Gaussian (dreapta)

Interesul față de metodele digitale de procesare a imaginii provine din două aplicații principale:

1. îmbunătățirea informațiilor picturale pentru interpretarea umană
2. prelucrarea datelor de imagine pentru stocare, transmitere și reprezentare pentru mașinări.

Procesarea imaginilor pregătește imaginile pentru analiza acestora, fie ea manuală sau automată. Prin procesare digitală, imaginile pot fi analizate chiar și în cazuri în care acest lucru nu a fost posibil anterior din cauza calității sau a achiziției slabe. Hardware-ul de astăzi permite implementarea unei mulțimi de funcții sofisticate de procesare a imaginilor, dar este mult mai important ca imaginea originală să se obțină cât mai clar posibil. Accent trebuie pus pe adaptarea iluminării și optimizarea imaginii

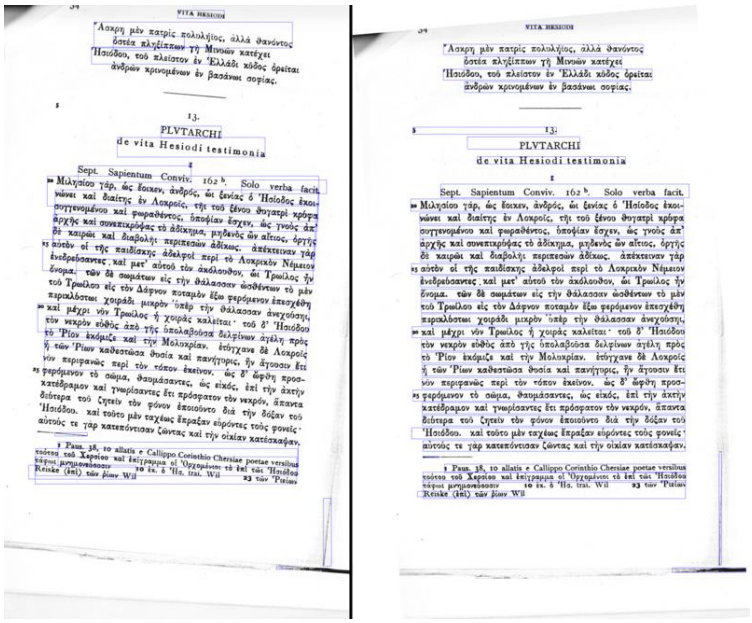
în timpul achiziției iar, dacă sunt necesare reglaje mici, acestea să se facă prin funcții digitale de procesare a imaginilor. [3]

Majoritatea funcțiilor comune de procesare a imaginilor sunt disponibile în sistemele de analiză a imaginii și pot fi clasificate în următoarele trei categorii:

1. Preprocesare
2. Îmbunătățirea imaginii
3. Transformarea imaginii

2.1 Preprocesarea imaginilor

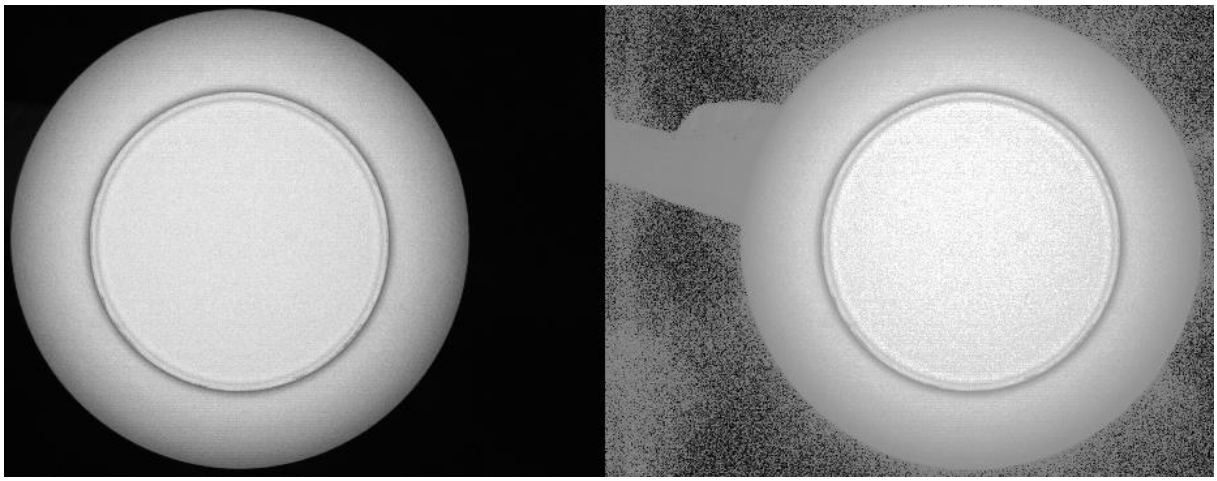
Pre-procesarea este un nume comun pentru operațiile cu imagini la cel mai scăzut nivel de abstractizare. Atât intrarea, cât și ieșirea sunt imagini intensive. Aceste imagini sunt la fel cu datele originale capturate de senzor, cu o imagine de intensitate reprezentată de obicei de o matrice de valori ale funcției imaginii (luminozitate). Scopul pre-procesării este o îmbunătățire a datelor de imagine care suprimă distorsiunile nedorite sau sporește unele caracteristici ale imaginii importante pentru prelucrarea ulterioară, deși transformările geometrice ale imaginilor (de exemplu, rotație, scalare, traducere) sunt clasificate printre metodele de pre-procesare ele sunt doar o mică parte din conceptul de pre-procesare. În Figura 3 se observă o simplă tehnică de preprocesare numită rotire.



Figură 3 - Image rotită folosind metoda Deskwe

2.3 Îmbunătățirea imaginii

Îmbunătățirea imaginii, este doar de a augmenta aspectul vizual al imaginilor pentru a ajuta la interpretarea și analiza vizuală. Exemple de funcții de îmbunătățire includ întinderea contrastului pentru a crește distincția tonală între diferitele caracteristici dintr-o scenă și filtrarea spațială pentru a spori (sau suprima) modelele spațiale specifice dintr-o imagine. În Figura 4 se observă o simplă tehnică de egalizare a histogramei.



Figură 4 - Imagine cu histogramă egalizată

2.4 Transformarea imaginii

O transformare poate fi aplicată unei imagini pentru a o converti de la un domeniu de reprezentare la altul. Vizualizarea unei imagini în domenii precum spațiul de frecvență sau Hough permite identificarea unor caracteristici care nu pot fi detectate la fel de ușor în domeniul spațial. În Figura 5 este prezentată o imagine la care i s-a aplicat o transformare morfologică de eroziune.

Figură 5 - Imagine pe care s-a aplicat o transformare morfologică de eroziune

2.5 Concluzii

Procesarea digitală a imaginilor se ocupă cu manipularea imaginilor digitale printr-un calculator digital. Este un subcâmp de semnale și sisteme, ce se ocupă, în special, cu manipularea imaginilor. Procesarea digitală a imaginilor se concentrează pe dezvoltarea unui sistem informatic care poate efectua procesarea pe o imagine. Intrarea acestui sistem este o imagine digitală și sistemul procesează imaginea folosind algoritmi eficienți și dă o imagine ca ieșire.

3. Noțiuni generale despre imagini digitale

O imagine digitală este o reprezentare a unei imagini reale ca un set de numere care pot fi stocate și manipulate de un calculator. Pentru a traduce imaginea în cifre, ea este împărțită în zone mici numiți pixeli (elemente de imagine). Pentru fiecare pixel, dispozitivul de achiziționare înregistrează un număr sau un mic set de numere care descriu anumite proprietăți ale acestui pixel, cum ar fi luminozitatea (intensitatea luminii) sau culoarea sa. Numerele sunt aranjate într-o serie de rânduri și coloane care corespund pozițiilor verticale și orizontale ale pixelilor din imaginea digitală. [4]

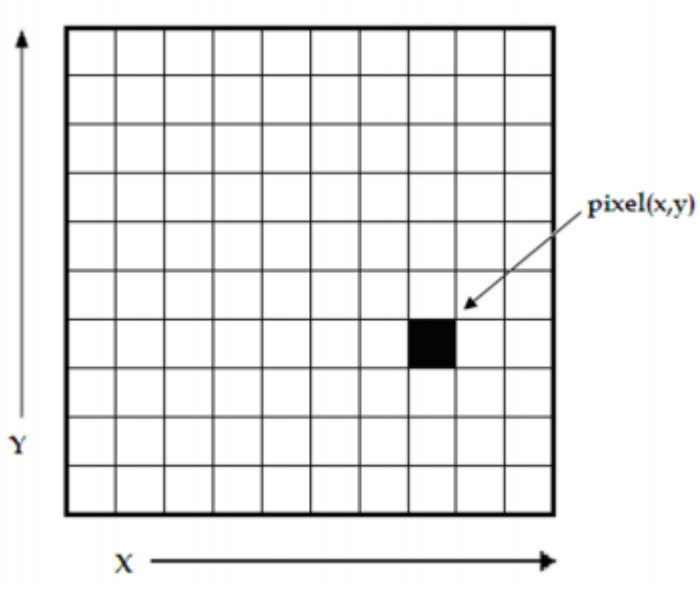
Imaginile digitale au câteva caracteristici de bază. O caracteristcă este tipul imaginii. De exemplu, o imagine alb-negru înregistrează numai intensitatea luminii care se încadrează pe pixeli. O imagine color poate avea trei culori, în mod normal RGB (roșu, verde, albastru) sau patru culori, CMYK (cyan, magenta, galben, black). Imaginile RGB sunt de obicei utilizate în monitoarele de calculator și scanere, în timp ce imaginile CMYK sunt utilizate în imprimante color [5]. Există, de asemenea, imagini non-optice cum ar fi ultrasunete sau raze X în care se înregistrează intensitatea sunetului sau razele X. În imaginile din interval, distanța pixelului de la observator este înregistrată. Rezoluția este exprimată în numărul de pixeli per inch (ppi). O rezoluție mai mare oferă o imagine mai detaliată. Un monitor de calculator are de obicei o rezoluție de 100 ppi, în timp ce o imprimantă are o rezoluție variând de la 300 ppi la mai mult de 1440 ppi. Acesta este motivul pentru care o imagine arată mult mai bine în imprimare decât pe un monitor.

Adâncimea de culoare (a unei imagini color) sau "biți per pixel" reprezintă numărul de biți din caracteristicile care descriu luminozitatea sau culoarea. Mai multe biți fac posibilă înregistrarea mai multor nuanțe de gri sau mai multe culori. De exemplu, o imagine RGB cu 8 biți pe culoare are un total de 24 de biți pe pixel ("culoarea adevărată"). Fiecare bit poate reprezenta două culori posibile, astfel încât să obținem un total de 16.777.216 culori posibile. O imagine GIF tipică pe o pagină web are 8 biți pentru toate culorile combinate pentru un total de 256 de culori. Cu toate acestea, este o imagine mult mai mică decât una de 24 de biți, fiind optimizată pentru transferul print protocolul HTTP. O imagine de fax are doar un bit sau două "culori", alb-negru. Formatul imaginii oferă mai multe detalii despre modul în care numerele sunt aranjate în fișierul imagine, inclusiv tipul de compresie utilizat, dacă este cazul.

Printre cele mai populare dintre cele zece formate disponibile sunt TIFF, GIF, JPEG, PNG și Post-Script. [6]

3.1 Reprezentarea unei imagini în spatiu 2D

Imaginile sunt afișate ca o matrice de elemente de imagine distincte (pixeli) în două dimensiuni (2D) și sunt denumite imagini digitale. Fiecare pixel dintr-o imagine digitală are o valoare a intensității și o adresă de locație (Figura 6). Valoarea pixelului arată numărul de contorizări înregistrate în acesta avantajul fiind că o imagine digitală în comparație cu cea analogică este că datele dintr-o imagine digitală sunt disponibilă pentru procesarea ulterioară cu ajutorul calculatorului.



Figură 6 - Reprezentare 2D a unei imagini si a unui pixel dintr-o imagine digitală

Imaginile digitale sunt caracterizate prin dimensiunea matricei, adâncimea pixelilor și rezoluția. Dimensiunea matricei este determinat de numărul de coloane (m) și numărul de rânduri (n) ale imaginii, astfel rezultă matricea (m × n). În general, dacă dimensiunea matricei este mare, crește rezoluția iar calitatea vizuală se îmbunătățește.

Pixelul sau adâncimea de biți se referă la numărul de biți pe pixel care reprezintă nivelurile de culoare fiecare pixel dintr-o imagine. Fiecare pixel poate lua 2^k diferite valori, unde *k* este adâncimea bitului imagine. Aceasta înseamnă că pentru o imagine pe 8 biți, fiecare pixel poate avea de la 1 la 2^8 (= 256) diferite culori (niveluri de gri).

Termenul de rezoluție a imaginii se referă la numărul de pixeli pe unitatea de lungime a imaginii. În imaginile digitale, rezoluția spațială depinde de dimensiunea pixelilor. Dimensiunea pixelilor este calculată prin câmpul vizual (FoV[1]) împărțit la numărul de pixeli din matrice. Pentru un standard FoV, o creștere a dimensiunii matricei scade dimensiunea pixelilor, iar capacitatea de a vedea detaliile este îmbunătățită.

Biblioteca OpenCV oferă funcții simple care pot citi imagini formate din mai multe tipuri de fișiere și acceptă un numărul mare de spații de culoare. În funcție de tipul de fișier și de spațiul de culoare, matricea întoarsă este, fie o matrice 2D de valori de intensitate (imagini în nuanțe de gri), fie o matrice 3D a valorilor RGB. Imaginile cele mai des utilizate sunt imagini de culoare gri, imagini color (RGB care este roșu, verde și Albastru) sau imagini HSV (hue, saturation, value)

3.2 Pixel

Cuvântul Pixel este o abreviere a expriesiei din engleză *Pixel Element* și reprezintă cel mai mic element constitutiv din o imagine digitală și conține o valoare numerică care este unitatea de bază a informațiilor din cadrul acesteia imaginea la o anumită rezoluție spațială și nivel de cuantificare [7]. În mod obișnuit, pixelii conțin culoare sau intensitate a imaginii ca un eșantion mic de lumină colorată din mediu. Cu toate acestea, nu toate imaginile conțin în mod obligatoriu informații vizuale neapărat. O imagine este pur și simplu un semnal 2-D digitizat ca o rețea de pixeli, ale căror valori se pot referi la alte proprietăți decât intensitatea culorii sau a luminii. Conținutul de informații al pixelilor poate varia considerabil în funcție de tipul de imagine care se procesează [8]:

1. Imagini color / alb-negru - imaginile întâlnite în mod obișnuit conțin informații referitoare la intensitatea culorii sau a griului la un anumit punct al scenei sau imaginii.
2. Infraroșu - spectrul vizual este doar o mică parte a spectrului electromagnetic. IR ne oferă capabilitatea de a realiza scene de imagine bazate fie pe reflexia luminii IR, fie după radiațiile IR pe care le emit. Radiația IR este emisă proporțional cu căldurageneratˑ / reflectat de un obiect și, prin

[1] Termen din engleză ce se referă la câmpul vizual observat (Field of Vision)

urmare, imagistica IR este, de asemenea, cunoscută în mod obișnuit ca imagine termică. Deoarece lumina IR este invizibilă pentru ochiul uman gol, iluminarea IR și sistemele de imagistică oferă o metodă utilă pentru supravegherea ascunsă (ca atare, imagistica IR formează în mod obișnuit baza sistemelor de viziune de noapte).

3. Imagini medicale - Multe imagini medicale conțin valori proporționale cu caracteristicile de absorbție a țesutului în raport cu un semnal proiectat prin corp. Cele mai frecvente tipuri sunt tomografia computerizată (CT) și rezonanța magnetică imagistica (RMN). Imaginile CT, cum ar fi razele X convenționale, reprezintă valori care sunt direct proporțional cu densitatea țesutului prin care semnalul a trecut. În contrast, Imaginile cu rezonanță magnetică prezintă mai multe detalii, dar nu au o relație directă cu o singură proprietate cuantificabilă a țesutului. În CT și RMN, formatul de imagine 2-D este frecvent extins la un volum 3D. Volumul 3D este în esență doar o stivă de imagini 2D.
4. Radar / Sonar - o imagine radar sau sonar reprezintă o secțiune transversală a unei ținte în proporțional cu distanța față de senzor și cu reflexia semnalului asociată. Radarul este utilizat în mod obișnuit pentru navigația aeronavelor, deși este, de asemenea, utilizat pe vehicule rutiere mai nou. Radioul bazat pe satelit pentru monitorizarea vremii este acum obișnuit, la fel ca și utilizarea de sonar pe cele mai moderne nave de mers pe jos. Radar capabil să pătrundă solul este utilizat din ce în ce mai mult pentru investigațiile arheologice și medico-legale.
5. Imagini 3D - utilizând tehnici specifice de detectare 3D, cum ar fi fotografie stereo sau scanarea 3D cu laser se poate capta date din obiectele din lumea din jurul nostru și pot fi reprezentate în sistemele informatice ca imagini 3D. Imaginile 3-D corespund adesea hărților în profunzime în care fiecare locație pixelă conține distanța punctului de imagine de la senzor. În acest caz, avem informații explicite 3D mai degrabă decât doar o proiecție de 3D ca în imaginile convenționale de tip 2-D. În funcție de tehnologia de captare, este posibil să existe doar informații 3D de adâncime sau ambele, adâncime și culoare 3D pentru fiecare locație. Hartă în profunzime poate fi reproiectată pentru a oferi o vedere parțială asupra obiectului capturat 3D.
6. Imagini științifice - Multe ramuri ale științei utilizează un format discret bazat pe 2D (sau 3D) pentru captarea datelor și analiza rezultatelor. Valorile

pixelilor pot, de fapt, să corespundă la densitățile sondelor chimice sau biologice, impedanța acustică, intensitatea sonică etc. În ciuda diferenței dintre conținutul informației, datele sunt reprezentate în același formată, adică o imagine 2D. Tehnicile digitale de procesare a imaginilor pot fi astfel aplicate în multe ramuri diferite ale analizei științifice.

3.2 Spatiul de culoare a unei imagini digitale

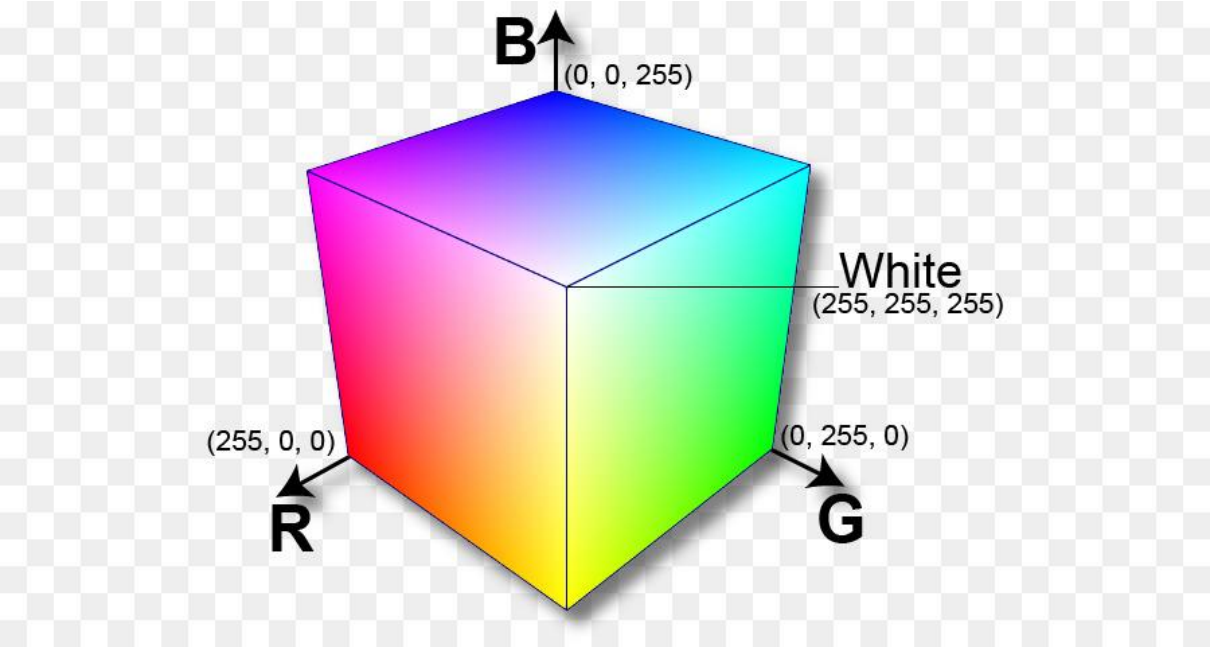
Organizarea culorilor dintr-o imagine într-un format specific se numește spațiu de culoare [9]. Modul în care este reprezentată o culoare se numește un model de culoare [3]. Fiecare imagine utilizează una dintre următoarele spații de culoare pentru o reprezentare eficientă a imaginii:

- RGB (red, green blue) – reprezintă culoare sub forma a trei valori de roșu, verde și albastru
- HSV/HSL (hue, saturation, value / hue, saturation, lightness) – reprezintă culoare sub forma valorilor de tipul culorii, saturația acestia și intensitatea sau luminozitatea acesteia.
- LAB (luminance, green-red, blue-yellow) – reprezintă culoarea sub forma a trei valori diferite; luminozitate, canitate de verder-roșu și canitate de albastru-galben
- LCH (lightness, chroma, hue) – reprezintă culoare sub forma a trei valori similare cu cele ale spațiului de culoare LAB, doar că este reprezentat întru-un spațiu rectangular
- YUV (brightness, chroma or color) – reprezintă culoarea sub forma a două din trei valori posibile.

3.2.1 Spațiul de culoare RGB

Folosind spațiul de culoare RGB, roșu, verde și albastru sunt amestecate în moduri diferite pentru a creea diferite combinații de culori. De ce se folosește RGB? Deoarece ochii noștri au receptori de culoare care pot percepe aceste trei culori și combinațiile lor destul de eficient [7]. Cu doar aceste trei culori se poate forma orice

culoare, teoretic. Valorile fiecărei culori din spațiu sunt definite între 0 și 255. Această gamă se numește adâncime de culoare, după cum se observă în Figura 7[2].

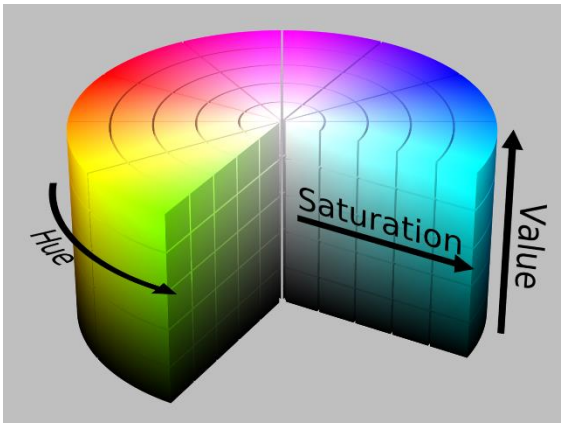


Figură 7 - Reprezentare 3D a spațiului de culoare RGB

3.2.2 Spațiul de culoare HSV/HSL

HSV / HSL reprezintă o reprezentare alternativă a spațiului de culoare RGB. Se compune din următoarele componente: nuanță, saturație, valoare/luminozitate [9]. Nuanța este o proprietate care descrie trei culori: verde, roșu și magenta. Poate fi un amestec de două culori pure: roșu și galben, galben și verde. Saturația măsoară intensitatea unei imagini. Se referă la cât de departe este o culoarea față de gri. O valoare mai mică înseamnă că culoarea se apropie de gri. Luminozitatea se referă la intensitatea culorii în raport cu culoarea albă. Ea ne spune cât de departe este o culoare de culoare albă. Valoarea este o altă măsură de intensitate. Ea ne spune cât de departe este o culoare față de negru. Figura 8[3] prezintă o reprezentare HSV a unei imagini.

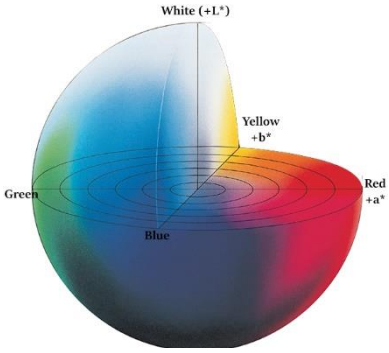
[2] Sursă imagine - https://www.kisspng.com/png-rgb-color-space-rgb-color-model-light-4170880/
[3]Sursă imagine - https://en.wikipedia.org/wiki/HSL_and_HSV#/media/File:HSV_color_solid_cylinder_saturation_gray.png



Figură 8 - Reprezentare 3D a spațiului de culoare HSV

3.2.3 Spațiul de culoare LAB

Culorile pe care pot fi percepute, și cele pe care nu nu pot fi percepute, sunt incluse în acest spațiul de culoare LAB. Oamenii sunt capabili să perceapă un punct, cu coordonatele stabilite și distanța până la acel punct. Împreună un punct și distanța față de el au coordonate cilindrice. Orice ce nu are coordonate cilindrice nu poate fi perceput de către oameni. Cea mai bună parte a spațiului de culoare LAB este că nu este dependent de dispozitiv; acesta poate fi utilizat în imprimare, textile și o serie de alte aplicații. Spațiul de culoare LAB este unul dintre mijloacele cele mai exacte de a reprezenta o culoare. Figura 9[4] prezintă o reprezentare LAB a unei imagini. [3]



Figură 9 - Reprezentare 3D a spațiului de culoare LAB

3.2.4 Spațiul de culoare LCH

LCH este similar spațiului de culoare LAB, dar în loc să utilizeze coordonatele cilindrice, utilizează coordonate dreptunghiulare. Acest lucru face ca coordonatele să fie similare cu modul în care ochiul uma procesează informația, care descrie un punct bazat, nu numai, pe coordonatele sale poziționale, ci și pe distanța de la un punct de

[4] Sursă imagine - https://sensing.konicaminolta.asia/what-is-cie-1976-lab-color-space/

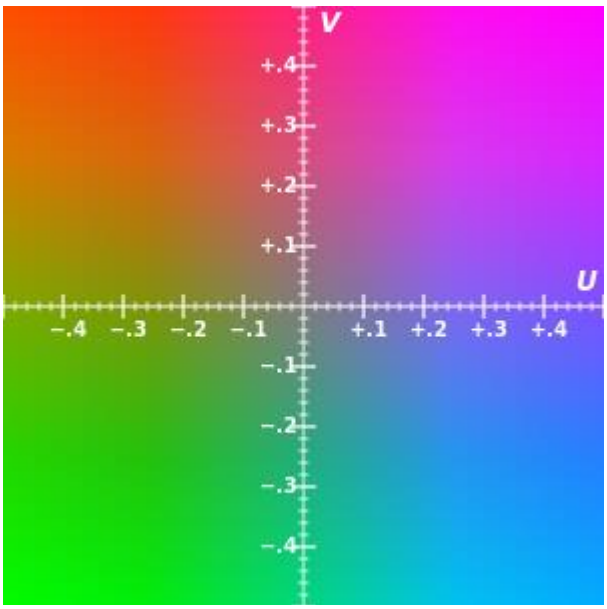
referință. Prin urmare, este ideal pentru percepția ochilor umani, deoarece punctul de referință în acest caz este ochiul nostru [9].

3.2.5 Spațiul de culoare YUV

Spațiul de culoare YUV este oarecum asemănător cu YPbPr, deoarece ambele sunt folosite în electronica video. Diferența este că YUV susține și televiziunea alb-negru. YUV are urmatoarele componente [9]:

- Y - luminozitatea prezentă într-o imagine. Valoarea sa poate varia de la 0 la 255.
- U și V - componenta de cromatică sau cloare. Valoarea sa poate varia de la -128 la +127 (în cazul numerelor întregi semnate) sau de la 0 la 255 (în cazul întregilor nesemnate).

Dacă se scot componentele U și V, se obține o imagine în tonuri de gri (grayscale). U și V sunt matrice de culori (Figura 10[5]).



Figură 10 - Reprezentare 2D a spțiului de culoare YUV

3.3 Concluzii

O image poate fi reprezentată în spațiul 2D ca și o matrice de valori. Aceste valori se numesc pixeli sau PEL, iar totalitate lor se însumează intr-o reprezentare

[5] Sursă imagine - https://upload.wikimedia.org/wikipedia/commons/thumb/f/f9/YUV_UV_plane.svg/300px-YUV_UV_plane.svg.png

digitală a unei secțiuni din lumea reală. Pixeli care alcătuiesc matricea au niște caracteristici, cum ar fi: poziția lor în matrice și culoarea care o reprezintă. Poziția lor este determinată de coordonate *x,y* iar culoare în functie de valoare de numărul si valoriii de culoare. Aranjarea acestori valori de culoare într-un format specific se numește spațiul de culoare al imaginii. Cele mai comune spații utilizare sunt HSV/HSL, RGB, LAB, YUV.

4. Operații cu imagini digitale

Imaginile digitale fiind niște matrici de valori, atunci ele matrici se supun unor reguli de aritmetică, cum ar fi: adunare, scădere, înmulțire, diviziune. De asemnea, pe aceste matrici de valori se pot aplica și oprații logice cum ar fi: AND, OR, XOR, NOT.

Aritmetica imaginii aplică una dintre operațiile aritmetice standard sau un operator logic la două sau mai multe imagini. Operatorii sunt aplicați în mod pixel-cu-pixel, adică valoarea unui pixel în imaginea de ieșire depinde numai de valorile pixelilor corespunzători din imaginile de intrare. Prin urmare, imaginile trebuie să aibă aceeași dimensiune. Deși aritmetica imaginii este cea mai simplă formă de procesare a imaginilor, există o gamă largă de aplicații. Un avantaj principal al operatorilor aritmetici este că procesul este foarte simplu și deci rapid. Operatorii logici sunt deseori folosiți pentru a combina două imagini (mai ales binare). În cazul imaginilor întregi, operatorul logic este în mod normal aplicat într-un mod bitwise [10].

Cea mai de bază operație de procesare a imaginilor este o transformare de puncte care cartografiază valorile la anumite puncte (adică pixeli) din imaginea de intrare la punctele corespunzătoare (pixeli) într-o imagine de ieșire [6]. În sensul matematic, aceasta este o mapare funcțională unu-la-unu intrare la ieșire. Cele mai simple exemple de astfel de transformări de imagine sunt aritmetice sau operații logice pe imagini. Fiecare este efectuată ca o operație între două imagini I_A și I_B sau între o imagine și o valoare constantă C:

$$I_{ieşire} = I_A + I_B \quad (1)$$

$$I_{ieşire} = I_A + C \quad (2)$$

În ambele cazuri, valorile la o locație individuală de pixeli *(i, j)* în imaginea de ieșire sunt mapate după cum urmează:

$$I_{ieșire}(i,j) = I_A(i,j) + I_B(i,j) \qquad (3)$$

$$I_{ieșire}(i,j) = I_A(i,j) + C \qquad (4)$$

Pentru a efectua operațiunea pe o întreagă imagine de dimensiuni C x R, pur și simplu se repetă toți indicii de imagine pentru *(i,j) = {0 ... C1, 0...R-1}.* După cum se observă operații aritmetice de bază pot fi efectuate rapid și ușor pe pixelii din imagine pentru o varietate de efecte și aplicații. Operații aritmetice de bază pot fi efectuate rapid și ușor pe pixelii din imagine pentru o varietate de efecte și aplicații.

4.1 Adunarea și scăderea imaginilor digitale

Adăugarea unei valori pentru fiecare valoare a pixelilor de imagine poate fi utilizată pentru a obține următoarele efecte:

- Reglarea contrastului - Adăugarea unei valori constante pozitive C la fiecare valoare a pixelurilor crește luminozitatea imaginii [10].
- Amestecarea - Adăugarea a două sau mai multe imagini împreună produce o imagine compusă a seriei de imagini de intrare [10]. Aceasta operațiune poate fi utilizată pentru a produce efecte de amestecare utilizând adăugarea ponderată.

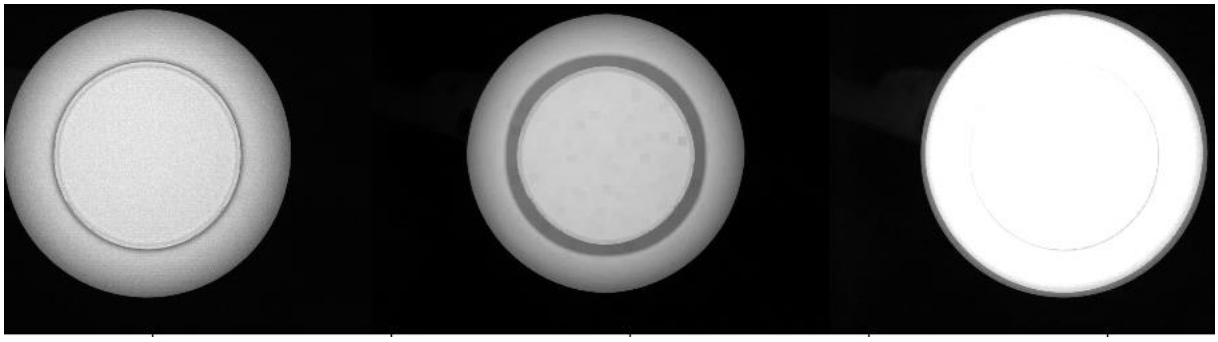


Figura 11 - Exemplu de adunare a două imagini

Scăderea unei valori constante de la fiecare pixel (inversul procedurii de adăugarea) poate fi de asemenea utilizată ca formă de bază pentru ajustarea contrastului [9]. Scăderea unei imagini de la alta ne arată diferența între acestea. Dacă scădem două imagini într-o secvență video, obținem o imagine diferită (presupunând o cameră statică) care arată mișcarea sau schimbările care au apărut între cadrele din

scenă (de exemplu, Figura12). Aceasta poate fi folosită ca formă de bază a detectării schimbării / mișcării în secvențe video.

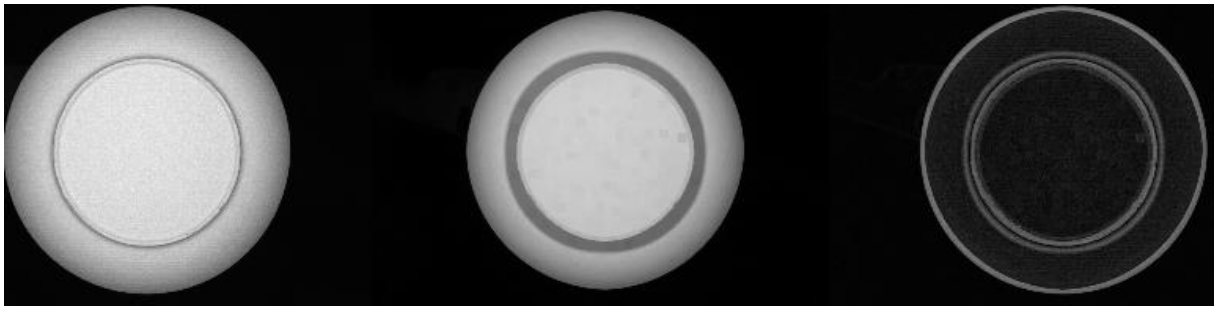


Figura 12 - Exemplu de scădere a două imagini

O variație utilă a scăderii este diferența absolută I_{output} | I_A – I_B | între imagini Acest lucru evită potențiala problemă a numărului întreg pe parcurs atunci când diferența devine negativă [2].

4.2 Înmulțirea și împărțirea imaginilor

Înmulțirea și împărțirea pot fi folosite ca un mijloc simplu de ajustare a contrastului și de extindere la adunare / scădere (de exemplu, reducerea contrastului cu 25% = diviziune cu 4, creșterea contrastului cu 50% = de înmulțire cu un factor până la 1,5) [9]. În mod similar, divizarea poate fi utilizată pentru diferențierea imaginii, împărțind o imagine cu alta, dă un rezultat de 1,0 în care valorile pixelilor de imagine sunt identice și o valoare care nu este egală cu 1,0, în cazul în care apar diferențe. Cu toate acestea, diferența de imagine folosind scăderea este computațional mai eficientă. Pentru toate operațiile aritmetice dintre imagini, trebuie să se asigure că valorile pixelilor care rezultă rămân în intervalul tipului de date / dimensiune disponibilă. De exemplu, o imagine 8-biți (sau o imagine colorată pe trei canale pe 24 de biți) poate reprezenta 256 valori în fiecare pixel. O valoare inițială a pixelilor de 25, înmulțită cu o valoare constantă (sau cu o valoare a pixelului imaginii secundare) de 12, va depăși intervalul de valori 0-255. Depășirea valori maxime va avea loc, iar valoarea va "tinde" în mod obișnuit, la o valoare redusă [3]. Aceasta problemă este cunoscută sub denumirea de saturație în spațiul imaginii: valoarea depășește capacitatea de reprezentare a imaginii. O soluție este de a detecta această depășire și de a o evita prin setarea tuturor acestor valori la valoarea maximă pentru reprezentarea imaginii (de exemplu, trunchierea la 255). Această metodă de manipulare a fluxului este implementată în funcțiile OpenCV imadd, imsubtract, immultiply și imdivide. În mod similar, trebuie acordată atenție, de asemenea, și valorilor negative ale pixelilor

rezultati din scădere și tratarea acestui rezultat în mod corespunzător. De obicei, acestia sunt setați la zero. Pentru imaginile RGB cu trei canale (sau alte imagini cu vectori, ca elemente de pixeli), operația aritmetică este în general efectuată separat pentru fiecare canal de culoare.

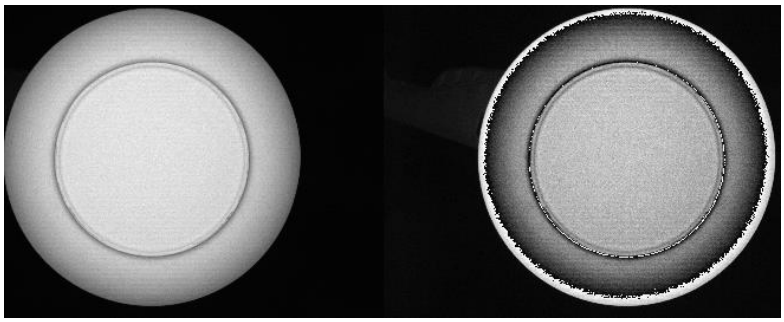


Figura 13 - Exemplu de imagine înmulțită cu un scalar

4.3 Operații logice cu imagini digitale

Tot odată se pot efectua operații logice standard între imagini cum ar fi NOT, OR, XOR și AND. În general, funcționarea logică este efectuată între fiecare bit corespunzător al reprezentării pixelilor de imagine (adică un operator *bit-wise*) [10].

4.3.1 Operația logică NOT

Aceasta operație logică inversează reprezentarea imaginii. În cel mai simplu caz al unei imagini binare, pixelii de fundal (negri) devin (albi) în prim plan și invers. Pentru imaginile în nuanțe de gri și de culoare, procedeul este de a înlocui fiecare valoare a pixelului $I_{ieșire}(i,j)$ după cum urmează:

$$I_{ieșire} = MAX - I_{ieșire}(i,j) \quad (5)$$

unde MAX reprezintă valoare maximă posibilă din reprezentarea imaginii. De exemplu pentru o image cu 8 biți (*grayscale*), MAX are valoare 255 [9].

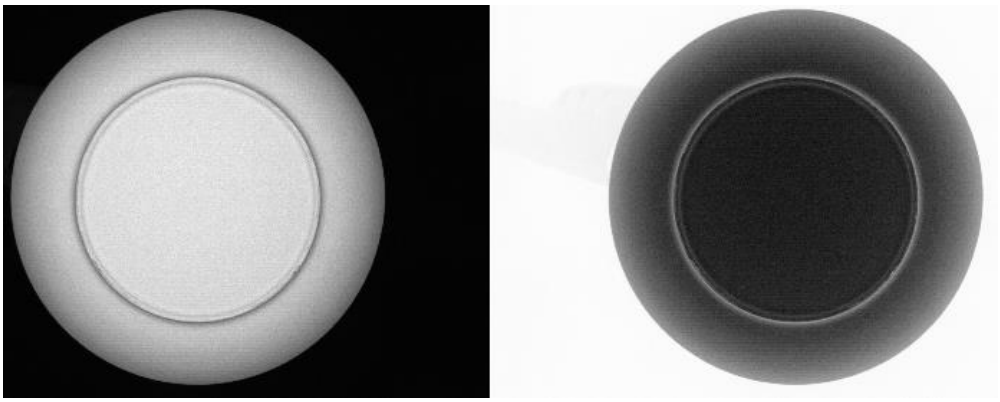


Figura 14 - Imagine procesată cu operatorul logic NOT

4.3.2 Operațiile logice OR/XOR

Oprațiile logice OR (și XOR) sunt utile pentru procesarea imaginilor binare (0 sau 1) pentru a detecta obiectele care s-au mutat între cadre. Obiectele binare sunt produse în mod obișnuit prin aplicarea de praguri până la o imagine de tip gri [9].

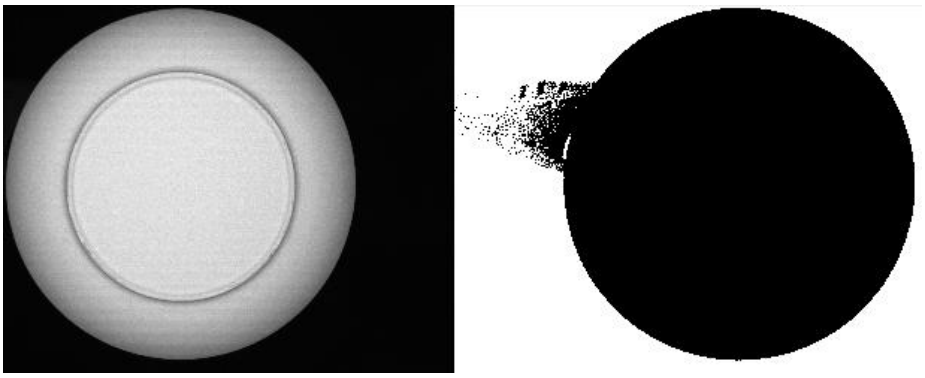


Figura 15 - Exemplu imgine procesată cu operatorul logic OR

4.3.3 Operațiile logice AND

Operatorul logic AND este folosit în mod obișnuit pentru detectarea diferențelor în imagini, evidențierea regiunilor țintă cu o mască binară sau producerea unor planuri de biți printr-o imagine [9].

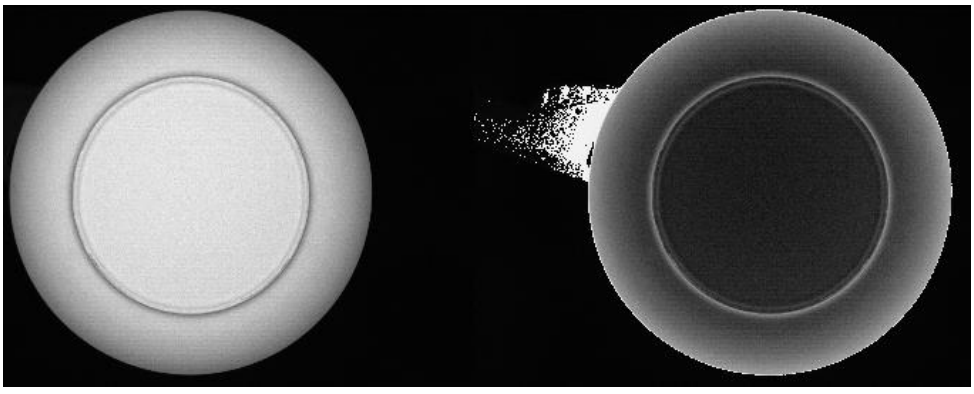


Figura 16 - Exemplu imagine procesată cu operatorul logic AND

Se pot folosi și operatori în combinație, cum ar fi NAND, NOR și NXOR, iar aceștia pot fi de asemenea aplicați într-o manieră similară cu operatorii de procesare a imaginilor [5].

4.3.4 Concluzii

Imginile digitate fiind o matrice de valori ce definesc pixeli, atunci matematica dicteză că asupra acestor matrici putem aplica oprații aritmetice. Imaginile pot fi adunate, scăzute, împărțite și înmulțite în așa fel să fie benefic pentru anumite procedee cum ar fi găsirea diferenței, detectarea obiectelor care sunt în mișcare etc. De asemenea, asupra acestor matrici se pot aplica și operatori logici (AND, OR, NOT, XOR etc.). Acești operatori sunt foarte utili pentru segmentarea imaginii și ajută în toate operațiile de acest gen.

5. Operații avansate de procesare a imaginilor digitale

Deobicei, aceste operații avansate, au de a face, de cele mai multe ori cu îmbunătățirea imaginii. Scopul de bază al îmbunătățirii imaginii este de a procesa imaginea astfel încât să se poată vizualiza și evalua informațiile vizuale pe care le conține cu o claritate mai mare. Îmbunătățirea imaginii este, prin urmare, destul de subiectivă deoarece depinde puternic de informațiile specifice pe care utilizatorul dorește să le extragă din imagine [7]. Condiția primară pentru îmbunătățirea imaginii este aceea că informațiile pe care se doresc a fi extrase, subliniate sau restaurate trebuie să existe în imagine. În principiu, "nu se poate face ceva din nimic" și informațiile dorite nu trebuie să fie în întregime înconjurate de zgomotul. Poate că cea mai precisă și generalizată afirmație care se poate face despre scopul îmbunătățirii imaginii este pur și simplu că, imaginea prelucrată ar trebui să fie mai potrivită decât cea originală pentru sarcina sau scopul dorit. Aceasta face ca evaluarea imbunatatirii imaginii, prin natura sa, sa fie destul de subiectiva si, prin urmare, este dificil să se cuantifice performantele sale in afara domeniului specific de aplicare. În acest capitol, se v-a discuta despre câteva metode, mai avansate de procesare a imaginilor pentru a obține o informație specifică conținută în imagine s-au pur și simplu, îmbunătățirea calității ei vizuale.

Pentru operații avansate de îmbunătățire a calității imaginilor, se cunosc patru metode principale:

1. Segmentarea imaginii
2. Convoluția imaginilor
3. Operații morfologice
4. Manipularea historgramei

Aceste patru metode de manipulare și procesare a imaginilor digitale pot fi folosite în combinație una cu cealaltă. Folosirea lor în combinție produce rezultate destul de bune, iara dacă se mai utilizează și operații simple, rezultatele sunt de-a dreptul uluitoare.

5.1 Segmentarea imaginii (Thresholding)

Unul dintre cei mai simpli și mai frecvent utilizați algoritmi de segmentare este segmentarea pe bazată pe praguri (*Thresholding*). *Thresholding*-ul oferă o imagine binară, care reduce complexitatea datelor și simplifică procesul de recunoaștere și clasificare [5]. Există trei tipuri de abordări de praguri, și anume, Global, Local și Adaptive. O tehnică de *Thresholding*-ul globală este una care face uz de o singură valoare de prag pentru întreaga imagine, în timp ce tehnica locală de prag utilizează valorile pragului unic pentru subimaginile partiționate obținute din întreaga imagine. În *Thresholding*-ul de adaptare, pentru fiecare pixel din imagine, trebuie calculat un prag. Totuși, selectarea automată a unei valori robuste extrem de semnificative este o provocare dificilă. Dacă valoarea pixelului este sub prag, este setată la valoarea de fundal, în caz contrar, aceasta își asumă valoarea primară. Există cinci tipuri de thresholding:

1. Binar - dacă valoarea pixelului este mai mare decât valoarea pixelului de referință (valoarea de prag), atunci pixeli sub acestă valoare sunt convertiți la alb (255), altfel, sunt convertiți la negru (0)
2. Binar inversat - dacă valoarea pixelilor este mai mare decât valoarea pixelului de referință (valoarea de prag), atunci pixeli sub acestă valoare sunt convertiți la negru (0), în caz contrar, sunt convertiți la alb (255). Este opusul thresholding-ului binar simplu
3. Truncat - dacă valoarea pixelului este mai mare decât valoarea pixelului de referință (valoarea pragului), atunci convertiți la valoarea de prag, altfel, nu valoarea rămâne neschimbată
4. Către zero - dacă valoarea pixelului este mai mare decât valoarea pixelului de referință (valoarea pragului), atunci nu se schimbă valoarea, altfel se convertește la negru (0)
5. Către zero inversat - Dacă valoarea pixelilor este mai mare decât valoarea pixelului de referință (valoarea de prag), atunci se convertește la negru (0); altfel, nu se schimba.

Thresholding-ul reprezintă una dintre cele mai bune operații pentru separarea imaginilor primare din fundal.

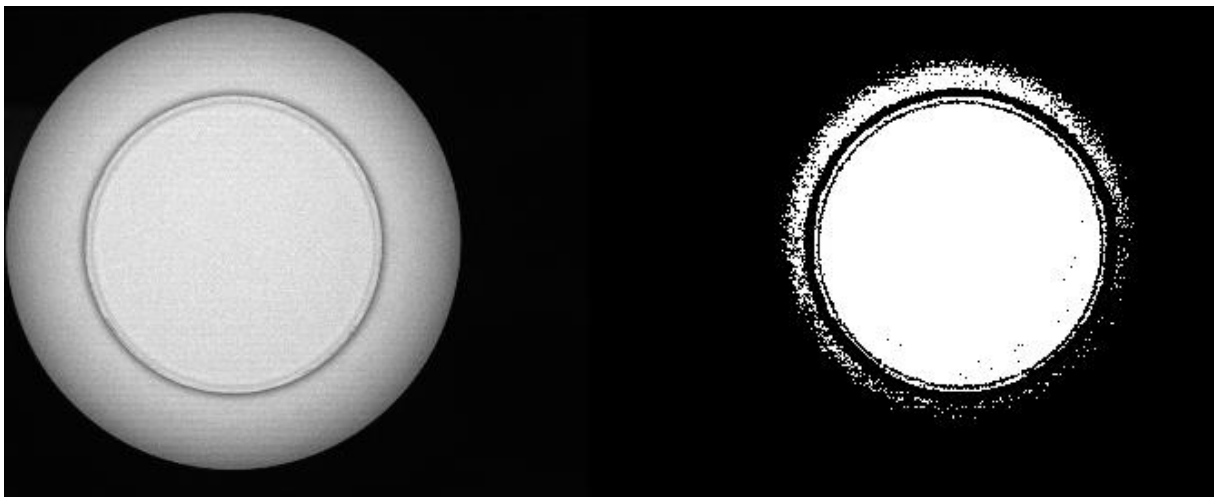


Figura 17 - Exemplu image procesată cu threshold la valoare de prag 200

5.2 Convoluția imaginilor

Convoluția este o operație importantă în procesarea imaginilor digitale. Convoluția funcționează pe două semnale (în 1D) sau două imagini (în 2D): se poate gândi la unul ca semnal de intrare (sau imagine), iar celălalt (numit kernel) ca "filtru" pe imaginea de intrare, producând o imagine de ieșire (astfel încât convoluția să ia două imagini ca intrare și produce o a treia imagine ca ieșire) [11]. Convoluția este un concept incredibil de important în multe domenii ale matematicii și ingineriei (inclusiv viziunea pe calculator).

Se dau doi vectori *f* si g. Convoluția lor este dată de relația următoare:

$$(\mathrm{f} * \mathrm{g})(i) = \sum_{j=1}^{m} g(j) * f(i - j + m/2) \quad (6)$$

O modalitate de a interpreta această operație este că *kernel*-ul trece peste imaginea de intrare. Pentru fiecare poziție a kernelului, se înmulțesc împreună valorile suprapuse ale kernelului cu cele a imaginii și se adună rezultatele. Această sumă de produse va fi valoarea imaginii de ieșire în punctul din imaginea de intrare în care nucleul este centrat [9]. Să presupunem că imaginea de intrare 1D este:

$$f = [10][50][60][10][20][40][30]$$

și kernelul este:

$$g = \left[\frac{1}{3}\right]\left[\frac{1}{3}\right]\left[\frac{1}{3}\right]$$

Imaginea de ieșire se va numi *h.* Care este valoare punctului *h(3)* din imaginea de ieșire? Pentru a calcula acest punct, punctul de centru a kernelului trebuie aliniat cu punctul *f(3)* din imaginea de intrare, astfel rezultă următoare figura:

10	50	60	10	20	30	40
0	1/3	1/3	1/3	0	0	0

Figura 18 - Poziționarea kernelului în relație cu imaginea de intrare

Din Figura 16, imagine de ieșire *h* este dată de înmulțirea fiecărui punct din imaginea de intrare *f* cu fiecare punct din kernel. Deoarece dimensiunea kernelului este mai mică ca imaginea de intrare, punctele care nu corespund au valoare 0, astfel avem următoare relație:

$$\frac{1}{3}50 + \frac{1}{3}60 + \frac{1}{3}10 = \frac{50}{3} + \frac{60}{3} + \frac{10}{3} = 40 \qquad (7)$$

Astfel punctul *h(3)* din imaginea de ieșire are valoare 40. Acest kernel calulează media de fereastră a imaginii de intrare și produce o imagine de ieșire.

De asemenea, se poate aplica convoluția și în spțiul 2D - imaginile și kernelurile sunt acum funcții 2D (sau matrici în OpenCV). Pentru convoluția 2D, la fel ca înainte, glisăm kernelul peste fiecare pixel al imaginii, se înmulțesc intrările corespunzătoare imaginii și a kernelului de intrare și se adună - rezultatul este noua valoare a imaginii. Dimesiunea kernelului (o matrice de 5 x 6 de exemplu) și valorile din acesta poate duce la o multitudine de efecte ce produc imagini de ieșire diferite, ajutând la evidețierea informației s-au obfuscarea ei, în funție de necesități. Dimensiuni și valorile diferite a kernelului se mai numesc comun și filtre.

5.2.1 Filtrarea aritmetică (Mean filtering)

Filtrul mediu este probabil cel mai simplu filtru liniar și funcționează prin acordarea unei greutăți egale W_k tuturor pixelilor din vecinătate. O greutate de W_k = *1/*(N x M) este folosită pentru un bloc N x M și are efectul de a netezi imaginea, înlocuind fiecare pixel în imaginea de ieșire cu valoarea medie din vecinătatea lui N x M [7]. Această schemă de ponderare garantează că greutățile din kernel se calculează la una peste orice dimensiune a blocului dat. Filtrele medii pot fi folosite ca metodă de suprimare a zgomotului într-o imagine. O altă utilizare comună este ca o etapă preliminară de procesare pentru a netezi imaginea pentru ca unele operații de

prelucrare ulterioare să fie mai eficiente [11]. În Figura 19[6], se observă un kernel simplu folosit pentru filtrarea aritmetică.

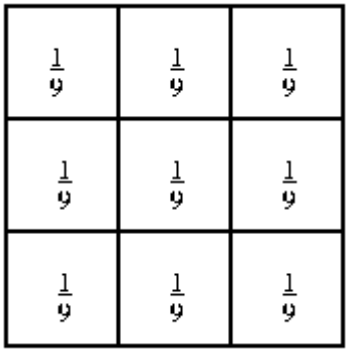


Figura 19 - Exemplu kernel de 3 x 3 folosit pentru filtrarea aritmetică

5.2.2 Filtrare mediană

Un alt filtru frecvent utilizat este filtrul median. Filtrarea mediană depășește principalele limitări ale filtrului mediu, deși în detrimentul costurilor computaționale mai mari [6]. Pe măsură ce fiecare pixel este adresat, acesta este înlocuit de mediana statistică a vecinătății lui N x M mai degrabă decât media. Filtrul median este superior filtrului mediu în sensul că este mai bun în a păstra un detalile de înaltă frecvență (adică marginile), eliminând totodată zgomotul, în special zgomotele cum ar fi zgomotul "sare și piper". În Figura 20[7] se poate observa un kernel folosit pentru filtrarea mediană.

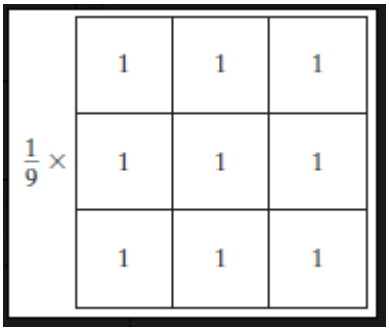


Figura 20 -Exemplu kernel de 3 x 3 folosit pentru filtrarea mediană

Media *m* a unui set de numere este acel număr pentru care jumătate din numere sunt mai mici de *m* iar jumătate sunt mai mari. Reprezintă punctul central al distribuției

[6] Sursă imagine - https://www.markschulze.net/java/meanmed.html
[7] Sursă imagine - https://imagelpcmatlab.blogspot.com/2014/06/how-to-apply-average-filter-weighted.html

sortate a valorilor. Deoarece valoarea mediană este o valoare a pixelilor extrași din blocului pixelului, este mai robustă la valori exagerate și nu creează o nouă valoare a pixelilor nerealiste. Acest lucru ajută la prevenirea estompării marginilor și la pierderea detaliilor imaginii.

5.2.3 Filtrare Gaussian

Ideea de filtrare Gaussian este de a folosi această distribuție 2-D ca o funcție de "răspândire a punctului"(clopot), și acest lucru este realizat prin convoluție [6]. Deoarece imaginea este stocată ca o colecție de pixeli discreți, trebuie să se producă o aproximare discretă a funcției Gaussian înainte de a putea efectua convoluția [6]. În teorie, distribuția Gaussiană este non-zero pretutindeni, ceea ce ar necesita un kernel de convoluție infinit de mare, dar în practică este efectiv zero mai mult de aproximativ trei abateri standard față de medie și astfel putem trunchia kernel-ul. Figura 21[8] prezintă un kernel de convoluție adecvat cu valoare totală, care se apropie de un Gaussian cu o valoare de 1,0. Nu este evident cum să se aleagă valorile măștii pentru a aproxima un Gaussian. S-ar putea folosi valoarea gaussianului în centrul unui pixel din mască, dar acest lucru nu este corect deoarece valoarea Gaussianului variază neliniar pe pixel.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figura 21 - Exemplu kernel pentru filtrare de tip Gaussian

Efectul de filtrare Gaussian este de a netezi o imagine, într-un mod similar cu filtrul mediu. Gradul de netezire este determinat de abaterea standard Gaussian [3].

[8] Sursă imagine - https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

Abaterea standard mai mare a Gaussilor, desigur, necesită nuclee de convoluție mai mari pentru a fi reprezentate cu precizie.

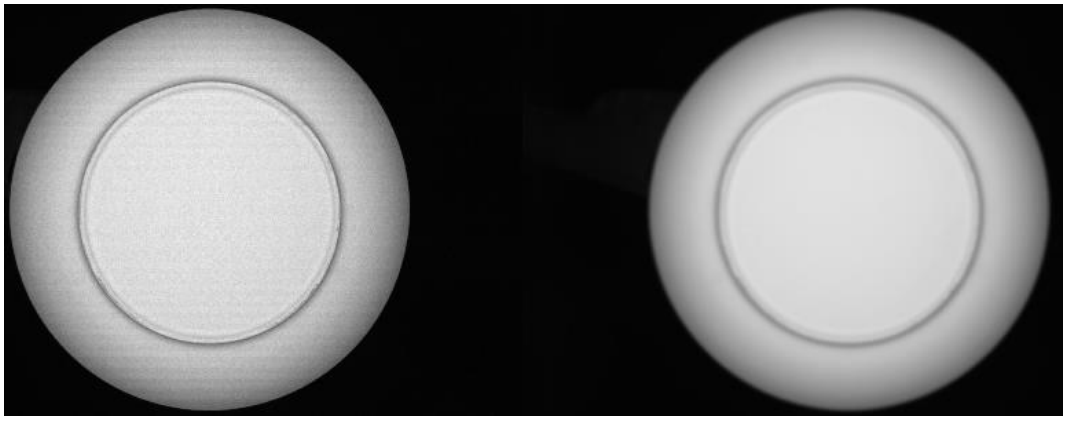


Figura 22 - Imagine procesată cu un filtru Gaussian

5.3 Hisograma unei imagini digitale

O histogramă a imaginii este un tip de histogramă care acționează ca o reprezentare grafică a distribuției tonale într-o imagine digitală [8]. Acesta descrie numărul de pixeli pentru fiecare valoare tonală. Privind histograma unei imagine, un observator poate analiza întreaga distribuție tonală dintr-o privire. Histogramele de imagini sunt prezente pe multe camere digitale moderne. Fotografii le pot folosi ca ajutor pentru a arăta distribuția tonurilor capturate și dacă detaliile imaginilor au fost pierdute pentru evidențierea sau umbrele întunecate. Acest lucru este mai puțin util atunci când se utilizează un format de imagine brut, deoarece intervalul dinamic al imaginii afișate poate fi doar o aproximare cu cea din fișierul brut [5]. Axa orizontală a graficului reprezintă variațiile tonale, în timp ce axa verticală reprezintă numărul de pixeli din tonul respectiv. Partea stângă a axei orizontale reprezintă zonele negre și întunecate, mijlocul reprezentând un mediu de culoare gri și partea dreaptă reprezintă zone luminoase și de culoare albă pură. Axa verticală reprezintă dimensiunea zonei capturate în fiecare dintre aceste zone. Astfel, histograma pentru o imagine foarte întunecată va avea majoritatea punctelor sale de date pe partea stângă și în centrul graficului. Dimpotrivă, histograma pentru o imagine foarte luminată, cu câteva zone întunecate și / sau umbre, va avea majoritatea punctelor sale de date pe partea dreaptă și în centrul graficului.

Într-un context de procesare a imaginii, histograma unei imagini se referă, în mod normal, la o histogramă a valorilor intensității pixelilor [6]. Această histogramă este un grafic care arată numărul de pixeli dintr-o imagine la fiecare valoare a intensității diferite găsită în acea imagine. Pentru o imagine în nuanțe de gri (*grayscale*)

de 8 biți există 256 de intensități diferite, astfel încât histograma va afișa grafic 256 de numere care arată distribuția de pixeli între valorile de tonuri de gri. De asemenea, histogramele pot fi luate din imagini color - pot fi realizate fie histograme individuale ale canalelor roșii, verzi și albastre, fie o histogramă 3-D, cele trei axe reprezentând canalele roșii, albastre și verzi și luminozitatea la fiecare punct reprezentând numărul pixelilor. Reprezentare ei depinde de implementare - poate fi pur și simplu o imagine a histogramei necesare într-un format de imagine adecvat sau poate fi un fișier de date de un fel reprezentând statisticile histogramă.

Histogramele au multe utilizări. Una dintre cele mai frecvente este să se afle ce valoare a să se folosească la conversia unei imagini în tonuri de gri într-o imagine binară prin *threshholding* [8]. Dacă imaginea este potrivită pentru prag, atunci histograma va fi bi-modală - adică intensitățile pixelilor vor fi grupate în jurul a două valori bine separate. Un prag adecvat pentru separarea acestor două grupuri se găsește undeva între cele două vârfuri ale histogramei. Dacă distribuția nu este așa, atunci este puțin probabil ca o segmentare bună să poată fi produsă prin pragul de prag.

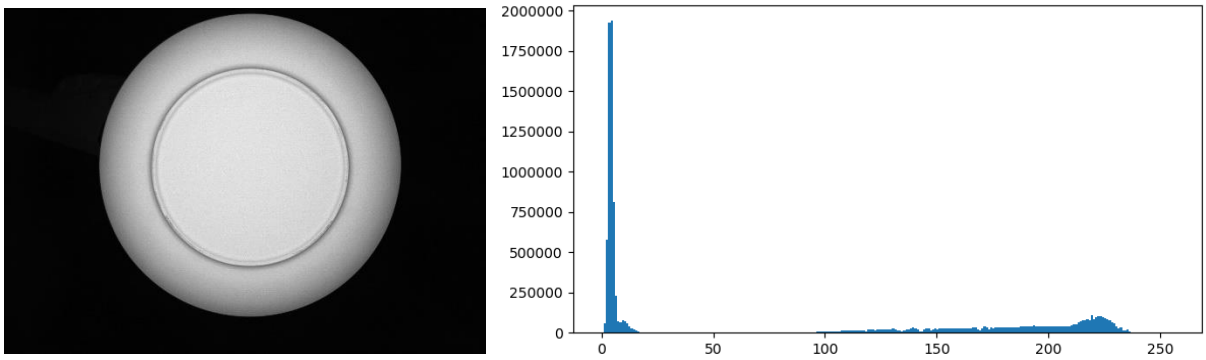


Figura 23 - Exemplu hisogramă pentru o image alb-negru

5.3.1 Egalizarea hisogramei

Tehnicile de modelare a histogramelor (de exemplu, egalizarea histogramei) oferă o metodă sofisticată de modificare a intervalului dinamic și a contrastului unei imagini prin modificarea imaginii astfel încât histograma de intensitate să aibă forma dorită [7]. Spre deosebire de întinderea contrastului, operatorii de modelare a histogramei pot utiliza funcții de transfer non-lineare și non-monotonice pentru a cartografia între valorile intensității pixelilor în imaginile de intrare și de ieșire. Egalarea histogramei utilizează o mapare monotonică, neliniară care re-atribuie valorile de intensitate ale pixelilor în imaginea de intrare astfel încât imaginea de ieșire să conțină

o distribuție uniformă a intensităților (adică o histogramă plată) [8]. Această tehnică este folosită în procesele de comparare a imaginii (deoarece este eficientă în îmbunătățirea detaliilor) și în corectarea efectelor neliniare introduse de către, de exemplu, un sistem de digitizare sau de afișare.

Pe scurt, egalizarea histogramei este o tehnică de procesare a imaginilor pe calculator, utilizată pentru a îmbunătăți contrastul în imagini. Aceasta metodă realizează acest lucru prin răspândirea efectivă a celor mai frecvente valori ale intensității, adică întinderea domeniului intensității imaginii. Această metodă crește, de obicei, contrastul global al imaginilor atunci când datele utilizabile sunt reprezentate de valori de contrast apropiate. Acest lucru permite ca zonele cu un contrast local mai scăzut să obțină un contrast mai ridicat.

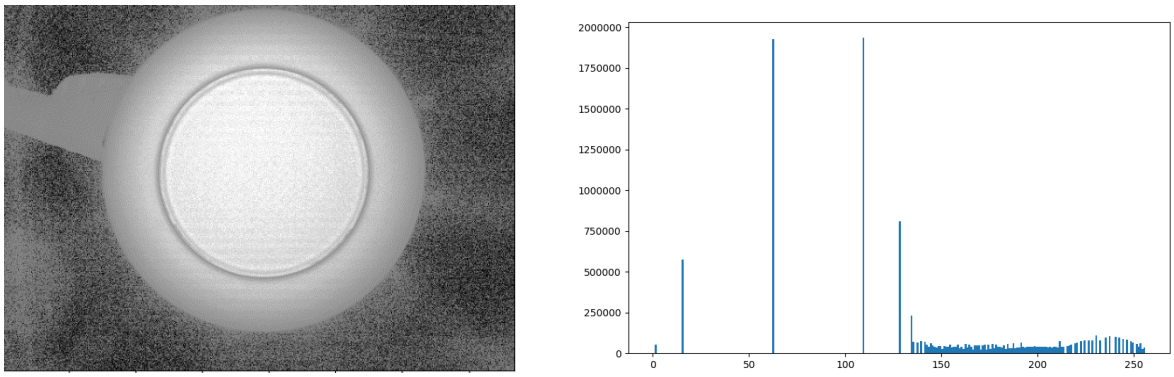


Figura 24 – Imagine cu histogramă egalizată adaptiv

În cazul imaginilor color egalarea histogramelor nu poate fi aplicată separat componentelor roșii, verzi și albastre ale imaginii, deoarece duce la modificări dramatice ale balanței culorilor imaginii. Cu toate acestea, dacă imaginea este mai întâi convertită într-un alt spațiu de culoare, cum ar fi spațiul de culoare HSL / HSV, atunci algoritmul poate fi aplicat canalului de luminanță sau de valoare, fără a duce la modificarea nuanței și saturației imaginii.

5.3.1 Egalizarea histogramei adaptate cu limită de contrast (Contrast Limited Adaptive Histogram Equalization)

Uneori, histograma globală a unei imagini poate avea o distribuție largă, în timp ce histograma regiunilor locale este foarte înclinată spre un capăt al spectrului gri. În astfel de cazuri, este adesea de dorit să se sporească contrastul acestor regiuni locale, dar egalizarea histogramei globale este ineficientă. Acest lucru poate fi realizat prin

egalizarea histogramei adaptive. Termenul adaptiv implică faptul că diferite regiuni ale imaginii sunt prelucrate în mod diferit (adică se aplică diferite tabele de căutare) în funcție de proprietățile locale [6].

Există mai multe variante privind egalizarea histogramei adaptabile, dar cea mai simplă și cea mai obișnuită este așa-numita fereastră de alunecare (sau țiglă). În această metodă, imaginea de bază este împărțită în regiunile relativ mici de țigle sau regiuni locale vecine N x M (de exemplu, 16 x 16 pixeli). Fiecare țiglă sau fereastră interioară este înconjurată de o fereastră exterioară mai mare, care este utilizată pentru a calcula tabelul de căutare pentru egalizarea histogramei adecvate pentru fereastra interioară. Acest lucru este în general eficient în creșterea contrastului local, dar pot apărea "artefacte bloc" ca rezultat al procesării fiecărei astfel de regiuni în izolare, iar artefactele la limitele dintre ferestrele interioare pot avea tendința de a produce impresia unei imagini care constă dintr-un număr ușor blocuri incongruente. Artefactele pot fi în general reduse prin mărirea dimensiunii ferestrei exterioare față de fereastra interioară.

O metodă alternativă pentru egalizarea histogramei adaptive a fost aplicată cu succes, pașii cărora sunt după cum urmează:

1. O rețea regulată de puncte este suprapusă peste imagine. Spațierea punctelor de rețea este o variabilă în această abordare, dar este, în general, este de câțiva zeci de pixeli.
2. Pentru fiecare punct al grilei, se determină o fereastră dreptunghiulară cu dublul distanței dintre grilă. O fereastră dată are astfel o suprapunere de 50% cu vecinii săi din apropriere.
3. Se calculează o tabelă de căutare echilibrată pentru fiecare fereastră. Datorită suprapunerii de 50% dintre ferestre, fiecare pixel din imagine se află în patru ferestre adiacente, dreptunghiulare sau patru cartiere
4. Valoarea transformată a unui pixel de imagine dat este calculată ca o combinație ponderată a valorilor de ieșire din cele patru tabele de căutare în vecinătate utilizând următoarea formulă bilineară:

   $$I = (1-a)(1-b)I_1 + a(1-b)I_2 + (1-a)bI_3 + abI_4 \quad (8)$$

   pentru greutățile la distanță 1 și valorile tabelului de căutări echilibrate histograme I1, I2, I3 și I4 din fiecare din cele patru blocuri adiacente. De reținut modul în care această formulă acordă o importanță adecvată blocurilor de care aparțin cel mai mult; de exemplu. un pixel situat exact pe

un punct al rețelei își ia valoarea numai sa egală din vecinătatea sa înconjurătoare, în timp ce un punct care este echidistant față de cele patru puncte cele mai apropiate ale rețelei este o combinație echilibrată a tuturor celor patru valori din jur.

O extindere finală la această metodă este abordarea de egalizare a histogramei adaptive limitate de contrast. În termeni generali, calculul histogramei regiunii locale și egalizarea valorilor intensității în acele regiuni are tendința de a crește prea mult contrastul și de a amplifica zgomotul. Unele regiuni dintr-o imagine sunt în mod inerent netede, cu un contrast real scăzut și o egalizare orbită a histogramei de intensitate din regiunea dată pot avea rezultate nedorite [5].

Ideea principală din spatele utilizării limitelor de contrast este de a plasa o limită l,0 < l < 1, pe creșterea generală (normalizată) a contrastului aplicat oricărei regiuni. Această metodă extensivă de egalizare a histogramei adaptive limitată de contrast este disponibilă în OpenCV. Adaptabilitatea egalizării histogramei poate uneori să producă îmbunătățiri semnificative ale contrastului imaginii locale. Cu toate acestea, datorită naturii fundamental diferite a unei imagini de la următoarea, capacitatea de a varia numărul de plăci în care este descompusă imaginea și forma specifică a probabilității țintă distribuție, există puțină teorie generală pentru ghidare. Obținerea unei imagini cu caracteristicile dorite de contrast este astfel, într-o măsură considerabilă, o artă în care acești parametri sunt variați pe o bază experimentală.

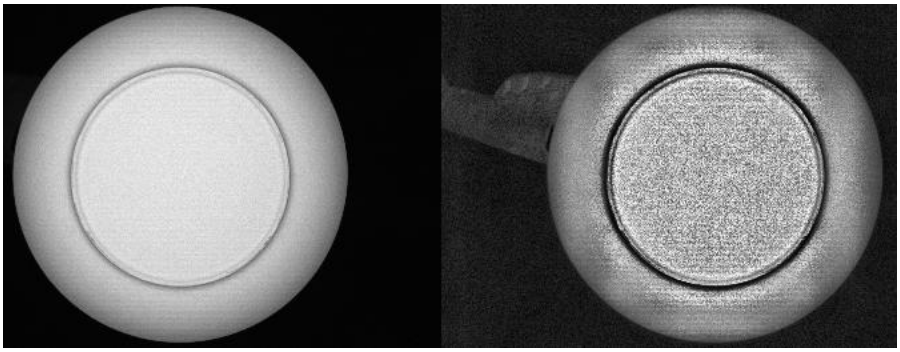


Figura 25 - Image procesată utilizând CLAHE

5.4 Operații morfologice

Morfologia este un set de operațiuni de procesare a imaginilor care procesează imagini bazate pe elemente de structurare predefinite, cunoscute și ca kerneluri. Valoarea fiecărui pixel din imaginea de ieșire se bazează pe o comparație a pixelului corespunzător în imaginea de intrare cu vecinii săi. Prin alegerea mărimii și formei

kernelului, puteți construi o operație morfologică care este sensibilă la anumite forme existente în imaginea de intrare [3].

Două dintre cele mai elementare operații morfologice sunt dilatarea și eroziunea. Toate celelalte operații morfologice pot fi definite în termeni de acești operatori primitivi. Dilarea adaugă pixeli la limitele obiectului dintr-o imagine, în timp ce eroziunea face exact opusul. Cantitatea de pixeli adăugați sau eliminați, respectiv depinde de dimensiunea și forma elementului de structurare utilizat pentru procesarea imaginii. În general, regulile care rezultă din aceste două operațiuni sunt următoarele:

5.4.1 Dilatare

Valoarea pixelului de ieșire este valoarea maximă a tuturor pixelilor care se încadrează în dimensiunea și forma elementului de structurare. De exemplu, într-o imagine binară, dacă oricare dintre pixelii imaginii de intrare care se încadrează în domeniul kernel-ului este setat la valoarea 1, pixelul corespunzător al imaginii de ieșire va fi setat la 1, de asemenea. Acesta din urmă se aplică oricărui tip de imagine (de ex., Tonuri de gri, bgr, etc) [8].

Figura 26 - Exemplu de imagine procesată cu operația morfologică de dilatare

5.4.2 Eroziune

Operațiunea de eroziune reprezintă inversul operației de dilatare. Valoarea pixelului de ieșire este valoarea minimă a tuturor pixelilor care se încadrează în dimensiunea și forma elementului de structură. Cele mai multe implementări ale acestui operator au nevoie ca imaginea de intrare să fie binară, de obicei cu pixelii din prim plan la valoarea intensității 255 și pixelii de fundal la valoarea intensității 0. O astfel de imagine poate fi adesea produsă dintr-o imagine în tonuri de gri folosind thresholding [6].

Figura 27 - Exemplu de imagine procesată cu operația morfologică de eroziune

5.5 Concluzii

Operațile avanasate de procesare a imaginilor digitale permit evidențierea și obfuscarea datelor vizuale prezentate de imagine. Metodele descrise pe scurt în acest capitol, sunt cel mai des utilizare în procesarea imaginilor digitate în vederea pregătiri lor pentru analiză.

- Segmentarea transformă imaginea într-o imagine binară care scoate în evidentă anumite artefacte necesare.
- Prin convoluția imaginii, se obțin anumite filtre utilizate în reducerea zgomotului și detectarea muchiilor.
- Analizarea histogramei duce la folosire mai eficientă a acestor metode, iar egalizarea ei, fie o egalizare globală sau regio [1]nlă folosind algoritmul CLAHE ajută la ajustarea contrastului pentru a scoate în evidentă anumite elemente.
- Operațiile morfologice ajută la evidențierea sau diminuarea elementelor.

Există si alte metode avasante cum ar fi transformarea logaritmică sau exponențilă, gradiente, matrițe și segmentare avansată, dar majoritatea acestor metode se bazează pe metodele enumerate mai sus.

6. Implementarea aplicției

6.1 Prezentare generală

Implementarea unei aplicații ce procesează imaginile și le pregătește de analiză presupune utilizarea unui limbaj de programare capabil să manipuleze imaginile. Totodată este necesar identificarea elementului ce trebuie scos in evidență din imaginile neprocesate. Acest aspect face ca procedeul de procesarea a imaginilor digitale în vederea pregătirii lor pentru analiză să aibe un caracter unic. Din păcate nu se poate generaliza din cauza varietății imaginilor. Totuși, există cazuri cand, un set de imagini pot fi generate într-un mediu controlat, astfel procedeul dezvoltat de procesare a imaginilor poate fi aplica tuturor acestor imagini, adică automatizat. În lucrarea acesta, imaginile trebuie procesate în vedere pregătirii lor pentru extragerea textului, adică analiza lor, în acest caz, presupune recogniția optică a caracterelor.

Imaginile reprezintă o serie de farfuri ceramice ce au ștanțat pe spate o serie de caractere. Aceste caractere trebuie extrase ca și date ce pot fi manipulate digital (*strings*). Pentru procesul de recogniție optică a caracterelor, se folosește serviciul pus la dispoziție de Google, adică Google Vision OCR prin expunerea unui API, prin care se poate transmite imagine pentru a fi analizată.

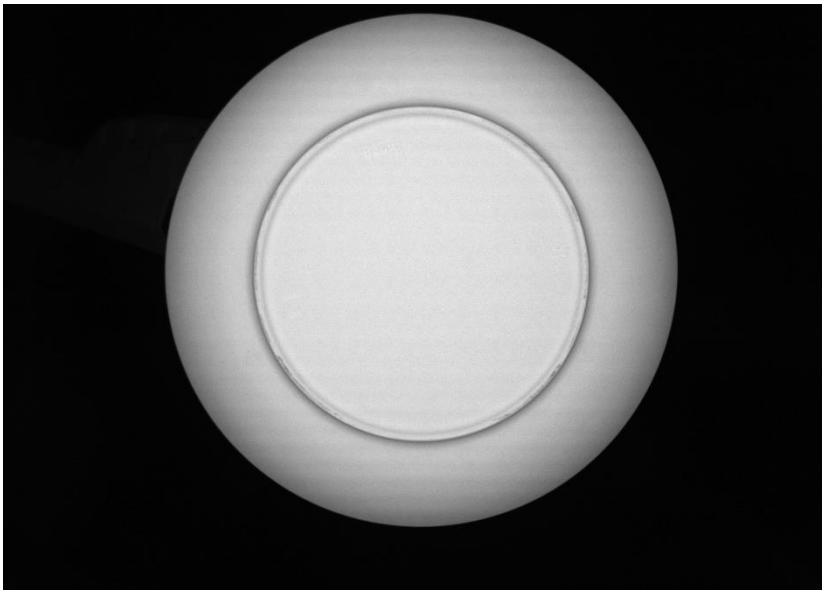


Figura 28 - Exemplu image de intrare neprocesată

După cum se observă din Figura 25, caracterele ce trebuie analizate și extrase nici nu sunt vizibile din cauza constrastului apropiat. Acest lucru, sugerează faptul că

există o problemă de contrast, iar acestă imagine este un candidat perfect pentru aplicarea unei metode de adjustarea contrastului.

Pentru a implementa aplicația, deobicei se urmează principiul “unealta potrivită pentru slujba potrivită”, astfel limbajul de programare Python reprezintă o uneală foarte bună pentru implementarea aplicației. Acesta, alături de o pletoră de biblioteci permite dezvoltarea cu usurință a sistemului. Biblioteca OpenCV este folosită pentru manipularea imaginilor cu ajutorul algoritmilor și proceselor gata construite, ceea face procesarea mult mai usoră, mai rapidă și cu sansă mică de eroare. Bineințeles, exsită numeroase alternative, cum ar fi: limbajul de programre C# sau Matlab. Motivul pentru care Python a fost ales este că, Python este compatiblil cu toate sistemele de operare existente, iar instalarea lui și al bibleotecilor este relativ ușor de realizat. Diferența între C# și Python în utilizarea bibleotecii OpenCV este o mică penalitate de performață, dar pentru a realiza aceste sistem, este neseminifcativă.

Pentru detectarea zonelor din imagine în care există text (caractere), se folosește un model, gata antrenat, pentru detectarea lor. Acest model se numește PyEAST, iar în combinație cu bibleoteca Keras ajută la extragerea acestor zone. Pentru detectarea acestor zone, imaginea trebuie să fie aproape la fel de bine procesată ca și pentru OCR, astfel ajută la reducerea erorilor.

Pentru a testa paramaterii aplicați proceselor și algoritmilor, se implemetează o interfață grafică cu ajutorul bibliotecii OpenCV. Acestă interfață grafică oferă un *feedback* vizual la ce se întamplă imaginii. Prin implementarea unor elemente de interfață grafică, respectiv *sliders* cu care se pot adjusta parametrii proceselor. După adjustarea dorită se poate testa funcționalitatea de OCR trimițând imaginea la serviciul web Google Vision OCR pentru a vedea vizual calitatea conținutului extras.

6.2 Reguli generale pentru procesare înainte de a efectua OCR

Pentru a efectua cu succes analiza imaginii folosid OCR, există niște reguli generale pentru procesarea imaginii înainte de a efectua procedeul cum ar fi:

- Reducerea zgomotului - este unul dintre cele mai importante procese. Datorită aceastuia, calitate imaginii va crește, iar procesul de recunoaștere optică a caracterelor va avea rezultate mai bune. Există multe metode pentru eliminarea zgomotului de imagine, cum ar fi filtrul mediu, filtrul min-

max, filtru Gaussian etc. În această aplicaţtie se foloșeste un filtru Gaussian ce are un *kernel* de 17 x 17.

- Normalizarea – presupune standardizarea caracterelor și a imaginilor în sine. În contextul aplicației descrise, normalizarea presupune un procedeu de rotație a imaginii, astfel încât textul să fie “în picioare”, adică poziționarea lui pentru a fi ușor de citit și de om.
- Schimbarea spațiului de culoare - prin schimbarea spațiului de culoare în nuanțe de gri (*grayscale*), imagine poate mai usor procesată aplicând tehnici cum ar fi tăierea de prag. Aplicația descrisă în această lucrare aplică acestă metodă.
- Adjustarea contrastului – acestă metodă de îmbunătățire oferă posibilitatea ridicării nivelului de contrast dintre elementele neimportante și caracteristica pe care se vrea a o scoate în evidentă. Aplicația aplică un asemenea algoritm, numit CLAHE (*Contrast Limited Adaptive Histogram Equalization*), fiind chiar unul dintre principalele metode care dă rezultate.

6.3 Structura aplicației

Sistemul de procesare a imaginilor și extragere a textului este împărțit în două părți. O parte a sistemul reprezintă modulul de *feedback* vizual, iar cealaltă parte este modulul de aplicația de linie de comadă. Cursul datelor (imaginilor) este determinat de niște directoare pe sistemul de fișieri, astfel există director pentru imaginile de intrare și pentru imaginile de ieșire. Aceste directoare sunt utilizate în cod ca locații de unde imaginile pot fi luate sau stocate.

Sistemul depinde și de o serie de bibleoteci pentru a efectua procesul. Pe langă bibleoteca OpenCV, folosită pentru aplicarea metodelor de procesare, se folosește biblioteca *NumPy*, utilizată pentru manipularea matricilor de pixeli. O altă bibleotcă utilizată este SDK[9] dezvoltat de Google. Acest SDK faciliteză comunicarea usoară cu serviciile web oferite Google, respectiv Google Vision OCR.

[9] SDK – Software development kit

O altă unealtă folosită în implementarea sistemului este *virtualenv*[10]. Acestă unealtă se folosește în creare unui mediu virtual unde poate fi instalată aplicația cu usurintă.

6.3.1 Modulul de feedback vizual

Modulul de *feedback* vizual este altcătuit dintr-o fereastră și o serie de *sliders* prin care se modifică parametri aplicați proceselor (CLAHE, Gaussian Blur). Prin modificarea parametrilor, se observă modificarea vizuală a imaginii, fapt ce facilitează o metodă usoară de testare. După adjustarea parametrilor, se poate testa extragerea conținutului textual din imagine, iar dacă rezultatele sunt satisfăcătoare, valorile parametrilor sunt salvate într-un fișiser de tip *json* pentru a fi folosite de modulul de linie de comandă.

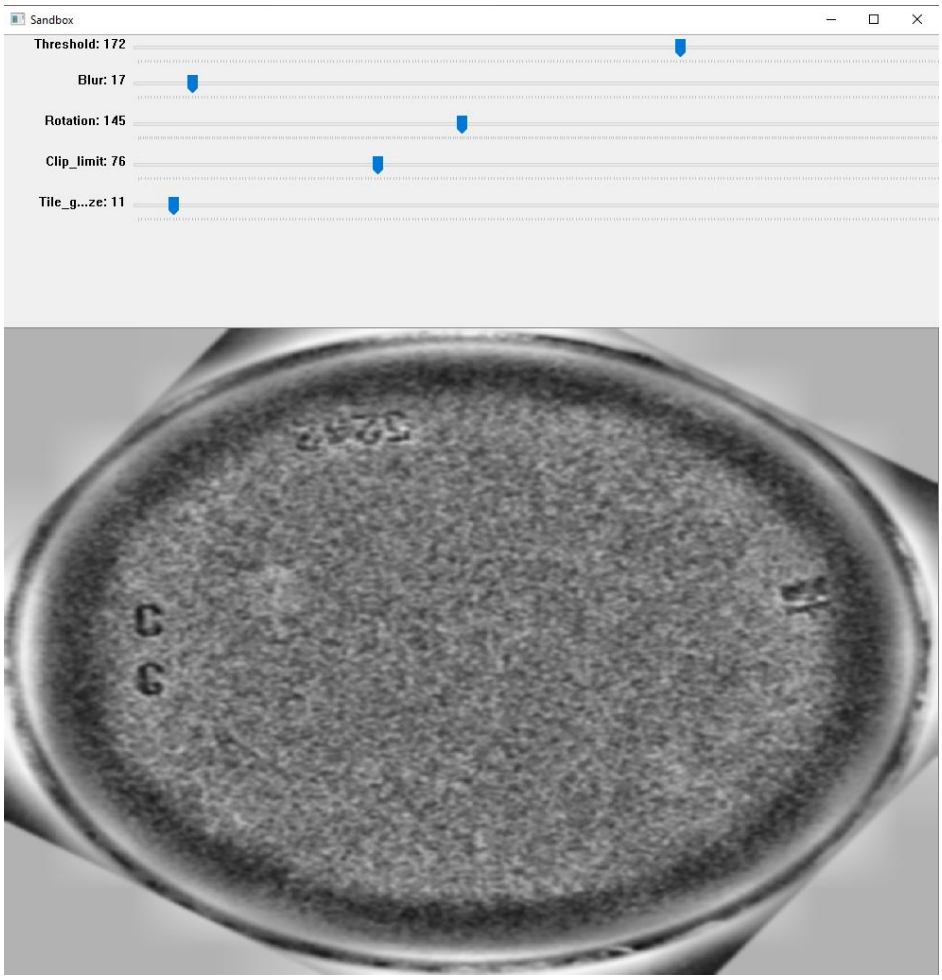


Figura 29 - Captură de ecran a modulului de *feedback* vizual

[10] Virtualenv – sistem software utilizat în virtualizare

După cum se observa în Figura 29, modulul de *feedback* vizual este compus din două părți:

Prima parte este alcătuită din funcționalitate de *sliders*. Acestea modifică parametrii utilizați în metodele de procesare și se reflectă în partea a doua a modulului vizual, respectiv partea vizuală. Parametrii care pot fi modificați sunt:

- *Threshold* – valoare pragului aplicat segmentării prin taierea de prag. Acesta valoare este 127 implicit.
- *Blur* – reprezintă dimensiunea *kernelului* calculat ca și N x N. Valoare implictă este 17 adică dimensiunea *kernelului* este de 17 x 17.
- *Rotation* – reprezintă unghiul de rotație a imaginii. Implicit are valoarea 145 grade.
- *Clip Limit* – reprezintă factorul de tăiere a contrastului la aplicarea alogritmului CLAHE. Implicit are valoare 7.6
- *Tile Grid Size* – reprezintă mărimea blocului sau regiunii în care este împărțită imaginea. Implicit are valoare 11, astfel se formează un kernel de 11 x 11.

Modulul de *feedback* vizual ajută utilizatorul să determine dacă parametrii selectați facilitează analiza imaginii, adică extragerea conținutului textual folosind recogniția optică a caracterelor. Pentru a testa recogniția optică a caracterelor, se poate apasă tasta *m*. Aplicația trimite poza serviciului de recogniție optică a caracterelor Googe Vision OCR, iar acesta trimite înapoi un șir de tip *string*, cu caracterele extrase. Dacă serviciul nu a reușit să extragă caractere, atunci, serviciul trimite înapoi un *string* gol.

Odată setat, la închidere sau la extragere, programul salvează valorile parametriilor în fișierul numit *config.json*. După cum sugerează fișierul, forma datelor sunt de tip *json*[11], iar scopul său este de fi citit de modulul de linie de comandă si de a prelua parametrii setați.

[11] JSON – Javscript object notation. Un mod de a structura date.

6.3.2 Modulul de linie de comandă

Modulul de linie de comandă este implementat pentru procesarea și analiza conținutului textual în masă. Acesta citește fișierul de configurare *config.json*, preia parametrii scriși în el de modului de *feedback* vizual și aplică acești parametrii la metodele de procesare a imaginilor digitale, dar și comunică cu serviciul Google Vision OCR pentru a extrage conținutul textual. Odată executat, acest modul urmărește activ directorul de fișiere neprocesate, numit *unprocessed* pentru imagini noi. Cand o imagine este găsită este preluată, procesată și analizată. După acest proces, imaginea este ștearsă din directorul *unprocessed* și este scrisă în directorul *processed* cu conținutul textual scris în ea.

```
Found 3 new images in input folder...
Processing image img_2018-12-05 14.07.08.571.jpg
Extracting region of interest...
Rotating image to config parameter value
Applying Contrast Limited Adaptive Histogram Equalization
(11, 11)
Applying Gausian Blurr
Thresholding
Proceeding to text extraction, plrease stand by...
Texts:

"5280
E E
IC:
```

Figura 30 – Exemplu executare modul de linie de comandă

Acest modul este implement pentru utilizarea lui ca un proces de sistem de operare pe Windows, Linux, MacOS fiind usor de utilizat în combinație cu alte unelte de managementul proceselor.

6.4 Metodele de procesare implementate și analiza codului

Analiza codului și a metodelor de procesare se face doar pe modulul de *feedback* vizual, deoarece modulul de linie de comandă este aproape indentic cu acesta când vine vorba de metodele aplicate. Diferența dintre acestea, o reprzintă doar funcționalitate modului de linie de comandă de “veghea” directorul ce stochează imaginile de intrare.

Ca și orice sistem software ce are ca scop procesare datelor, se poate începe prin a descrie sistemul de citire a lor. Datele sunt structurare pe sistemul de fișiere în directoare specific create pentru a ține imaginile procesate și neprocesate. Pyhton pune la dispoziție funcții destinate traversării pomului de directoare și fișiere astfel putem construi variabile cu calea către aceste directoare după cum se vede în Figura 31.

```
cwd = os.path.dirname(os.path.abspath(__file__))
input_dir_path = cwd + '/data/test_input'
output_dir_path = cwd + '/data/test_output'
input_image_names = os.listdir(input_dir_path)
```

Figura 31 – Variabile ce țin locația pe pomul de fișiere

Variabila *cwd* reprezintă cărarea către rădacina aplicației. Această este format dinamic în funcție de sistemul de operare și de locația fișierelor în pomul de fișiere folosind funcția *os.path.dirname()* și funcția *os.path.abspath(__file__)*. Acestă variabilă este folosită în continuare pentru a forma cărări către directoarele de unde se citesc sau scriu date, astfel se creează variabila *input_dir_path* variabile ce reprezintă un *string* format din variabila cwd concatenată cu *string-ul `/data/test_input`*. Aceași operație se face și pentru variabilă *output_dir_path*. Odată create aceste variable, ele se folosesc în combinție cu funcția *os.listdir()*. Această funcție itereză cărarea ce o primește ca și argument și întoarce o listă cu toate numele fișierelelor găsite în acel director, astfel se crează o variabilă să stocheze această listă numită *input_images_name* prin care se pot accesa imaginile.

Pentru a reda vizual bările de control al valorilor și imaginea, se folosesc funcții speciale implementate în bibleoteca OpenCv după cum se observă în Figura 32.

```
wnd = 'Sandbox'
cv2.namedWindow(wnd, cv2.WINDOW_NORMAL)

cv2.createTrackbar("Threshold", wnd, 172, 255, nothing)
cv2.createTrackbar("Blur", wnd, 17, 255, nothing)
cv2.createTrackbar("Rotation", wnd, 145, 360, nothing)
cv2.createTrackbar("Clip_limit", wnd, 76, 255, nothing)
cv2.createTrackbar("Tile_grid_size", wnd, 11, 255, nothing)
```

Figura 32 – Creare ferestrei de vizualizare și a bărilor de valori

Se definește o variabilă numită *wnd* care stochează numele ferestrei, în cazul de față numele este *Sandbox*. Cu ajutorul funcției *cv2.namedWindow()* se crează o fereastră, iar funcția primește ca parametrii numele și tipul de fereastră. În acest caz

numele este variabila *wnd* definită mai sus, iar tipul ferestrei este o constantă definită în bibleoteca OpenCV numită *WINDOW_NORMAL* ce se referă la o fereastră de dimensiunea ecranului pe care este redată.

Cu ajutorul funcției *cv2.createTrackbar()* se crează o bară (slider) de valori cu care utilizatorul poate interacționa cu ajutorul *mouse*-ului, modificând valoarea. Funcția primește ca și argumente următoarele: numele bării, fereastra pe care să fie afișată bara, valoare implictă, valoare maximă, funcție *callback*[12]. Ca și exemplu, prima bară definită se numește *Threshold*, este atasată ferestrei cu numele *wnd*, are valoare implictă 172, valoare maximă 255, iar funcția de *callback* este nulă. Utilizând această funcție se definesc cinci bări, respectiv, bară pentru valoare de *threshold, bară* pentru valoare *kernelului* folosit într-un filtru Gaussian de reducere a zgomotului, bară pentru valoarea de rotație a imaginii, bară pentru valoare limitei de tăieri a contrastului utilizat în CLAHE, bară pentru valoare dimensiunii blocului utilizat în CLAHE. Creare acestor băr este necesară pentru modului de *feedback* vizual deoarece permite utilizatorului să modifice în timp real aceste valorile metodelor de procesare.

După definirea acestor elemente necesare pentru redare vizuală, se trece la citirea propriu zisă a imaginii și transformarea ei într-o matrice. Pentru această operație se folosește funcția *cv2.imread().*

```
img = cv2.imread(input_dir_path + '/' + input_image_names[3])
```

Figura 33 – Citire și transformare imagine în matrice prin funcția cv2.imread()

Se inițializează o variabilă *img* în care se stochează rezultatul executării funcției *cv2.imread().* Această funcție primește ca și parametru cărarea din sistemul de fișiere unde se află imaginea. În cazul de fată parametrul este un *string* format din valoare variabilei *input_dir_path* și concatenarea *string-ului* din lista de imagini ce se află la indexul 3. Se obține o imagine cu este afișată în Figura 28.

Având imaginea stocată într-o variabilă, respectiv variabila *img*, se constată că nu este nevoie de procesare ei în totalitate. Regiune din imagine ce conține textul este în centru, iar asta sugerează că trebuie să se aplice o operație de transformare (decupare).

[12] (funcție) callback – reprezintă referința către o funcție definită de utilizator ce este executată de fiecare dată când valoare bării este modificată

Pentru a determina coordonatele de decupare, se observă că în imagine există un cerc, respectiv, o bordură pe spatele farfuriei ceramice. Tot ce este în interiorul acelui cerc, reprezintă regiunea de interes, deoarece acolo este conținutul textual ce trebuie separat de restul imaginii înainte de a fi dat spre serviciu de recogniție optică a caracterelor. OpenCV are la dispoziție o metodă prin care se pot determina cercurile existente într-o imagine, astfel s-a implementat urmatoarea secvență de cod redată în Figura 34.

```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

clahe = cv2.createCLAHE(clipLimit=7.6, tileGridSize=(11, 11))
cll = clahe.apply(gray_img)

blurr = cv2.GaussianBlur(cll, (17, 17), 0)

circles = cv2.HoughCircles(blurr, cv2.HOUGH_GRADIENT, 1.2, 170, 70, 80)

radius = 0
rect_height = 0
rect_width = 0

if circles is not None:
    # convert the (x, y) coordinates and radius of the circles to integers
    circles = np.round(circles[0, :]).astype("int")

    # loop over the (x, y) coordinates and radius of the circles
    for (x, y, r) in circles:
        # draw the circle in the test_output image, then draw a rectangle
        # corresponding to the center of the circle
        cv2.circle(img, (x, y), r, (0, 255, 0), 4)

        cv2.rectangle(img, (x-r, y-r), (x+r, y+r), (0, 128, 255), 2)
        rect_height = y
        rect_width = x
        radius = r

top = rect_width - radius
left = rect_height - radius
bottom = rect_width + radius
right = rect_height + radius
roi = gray_img[left:right, top:bottom]
```

Figura 34 – Procedeul de extragere a regiunii de interes

Pentru a începe extragerea regiunii de interes, prima dată trebuie să se scoată în evidentă cercul, respectiv bordura. Primul pas se efectează utilizând funcția *cv2.cvtColor()*, ce transformă imagine într-o imagine alb-negru (grayscale) ce este stocată în variabila *gray_image.* Transformând imaginea prin acest procedeu, acesta

poate fi procesată în continuare cu CLAHE, deoarece în stadiul actual cercul (brodura nu iasă în evidentă).

Pentru a aplica metoda CLAHE, prima dată, trebuie creat un obiect CLAHE prin funcția *cv2.createCLAHE()*. Argumentele ce le primește funcția sunt *clipLimit* și *tileGridSize* cu parametrii impliciți. Odată create acest obiect, este stocat în variabila *clahe*. Obiectul *clahe* are implement o metodă ce se numește *apply()*, iar această metodă primește ca și argument imaginea pe se vrea a fi procesată cu acest algorimt. În cazul de fată imaginea procesată este cea care este stoctă în variabila *gray_image.* Invocarea funcției *apply()* din obiectul *clahe* produce o imagine procesată cu acest algorim (Exemplu Figura 22) și are ca efect obținerea unui contrast mai bun, dar în același timp și scoaterea în evidență a cercului (bordurii) de pe farfurie.

Având imaginea cu cercul scos în evidentă și stocat în variabila *cll*, se constată că exită mult zgomot, astfel imaginea reprezintă un candidat perfect pentru o metodă de reducere a zgomotului. Metoda alesă este o filtrare de tip Gaussian, deoarece acestă metodă de filtrare este perfectă pentru a reduce zgomotul, dar în același timp prezervă marginile (cercul). Metoda este implementată prin funcția *cv2.GaussianBlur()*, funcție ce primește ca parametrii imaginea pe care se dorește să sa aplice filtrarea, în acest caz, imaginea stocată în variabila *cll* și dimensiunea *kernelului* Gaussian de 17 x 17 pasat sub formă de *tuple*[13]*.* Rezultatul filtrării este stocat în variabila *blurr.*

Obținând o imagine cu zgomot redus, se poate trece la determinarea cercurilor existente în ea. Pentru a realiza acest lucru, se foloește metoda *cv2.HoughCircles()*. Această funcție primește ca și argumente următoarele: variabila cu imaginea pe care să se aplice funcția, în acest caz fiind variabila *blurr,* constanta de gradient *cv2.HOUGH_GRADIENT*, rația inversă a rezoluției implicit fiind 1.2, valoare de prag ce este pastă detectorului intern de margini Canny implicit fiind 170, distanța minimă dintre centrele cercurilor găsite, distanța maximă dintre centrele cercurilor găsite. Rezultatul metodei *cv2.HoughCircles()* este stocat în variabila *circles*, aceasta fiind o listă cu coordonatele cercurilor găsite în imagine precum si centrul acestora. Dacă în imagine au fostă găsite cercuri, următorul pas este iterarea lor și trasarea lor pe imaginea originală. Cum s-a menționat mai sus, scopul pentru care trebuie derminate

[13] Tuple – structură de date specifică limbajului Python.

cercurile din imagine este de calcula regiune de interes, regiune cu conținutul texual, aceasta fiind centrul farfuriei. Pentru a extrage regiune de interes, imagine trebuie decupată, iar coordonatele de decupare se bazeză pe determinarea unui poligon derivat din raza cercului. Raza cercurilor existente în imagine se obține prin iterarea listei stocate de crecuri stocate în variabila *circles.* Formula de determinare a poligonului ce include un cerc este urmatoare:

$$P_{lățime/lungime} = 2r$$

Pentru a evidenția cercul și poligonul, se folosec funcțiile *cv2.circle()* și cv2.rectangle(). Aceste funcții permit desenarea pe o imagine a cercurilor și a poligoanelor. Funcția *cv2.circle()* primește ca și argumente imaginea pe care va fi trasat cercul, coordonatele de centru a cercului, raza, culoarea liniei care va fi trasată, iar ultimul argument este grosimea liniei. În Figura 35 se redă acest lucru.

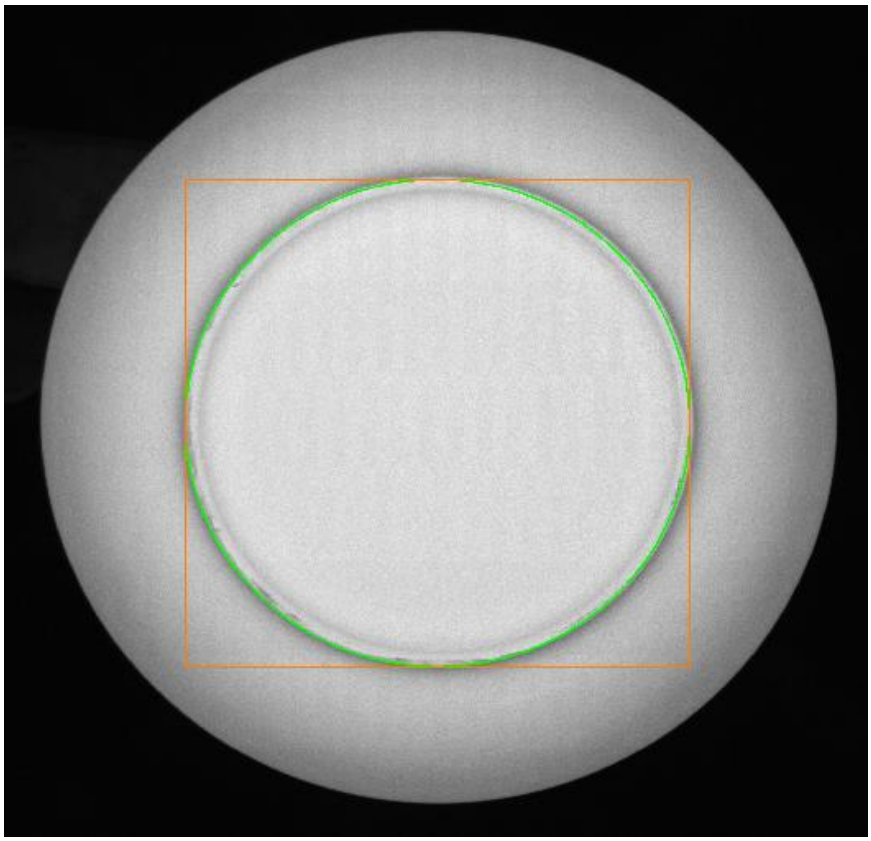


Figura 35 – Cerc și poligon derivat trasat

Pentru a decupa regiune de interes, cunoscând coordonate poligonului se folosește capabilitatea librăriei OpenCV și NumPy de a taia matrici, astfel pentru a decupa este nevoie de coordonatele a două puncte din matrice. Punctul ce definește colțul stânga-sus a poligonului și punctul ce definește colțul stânga jos. Pentru a stoca acestă regiune se inițializeză o variabilă numită *roi* iar această este echivalent cu

operețiunea de stragere scrisă sub forma *roi = gray_img[left:right, top:bottom]*. După extragerea regiunii de interes se obține imagine relatată în Figura 36.

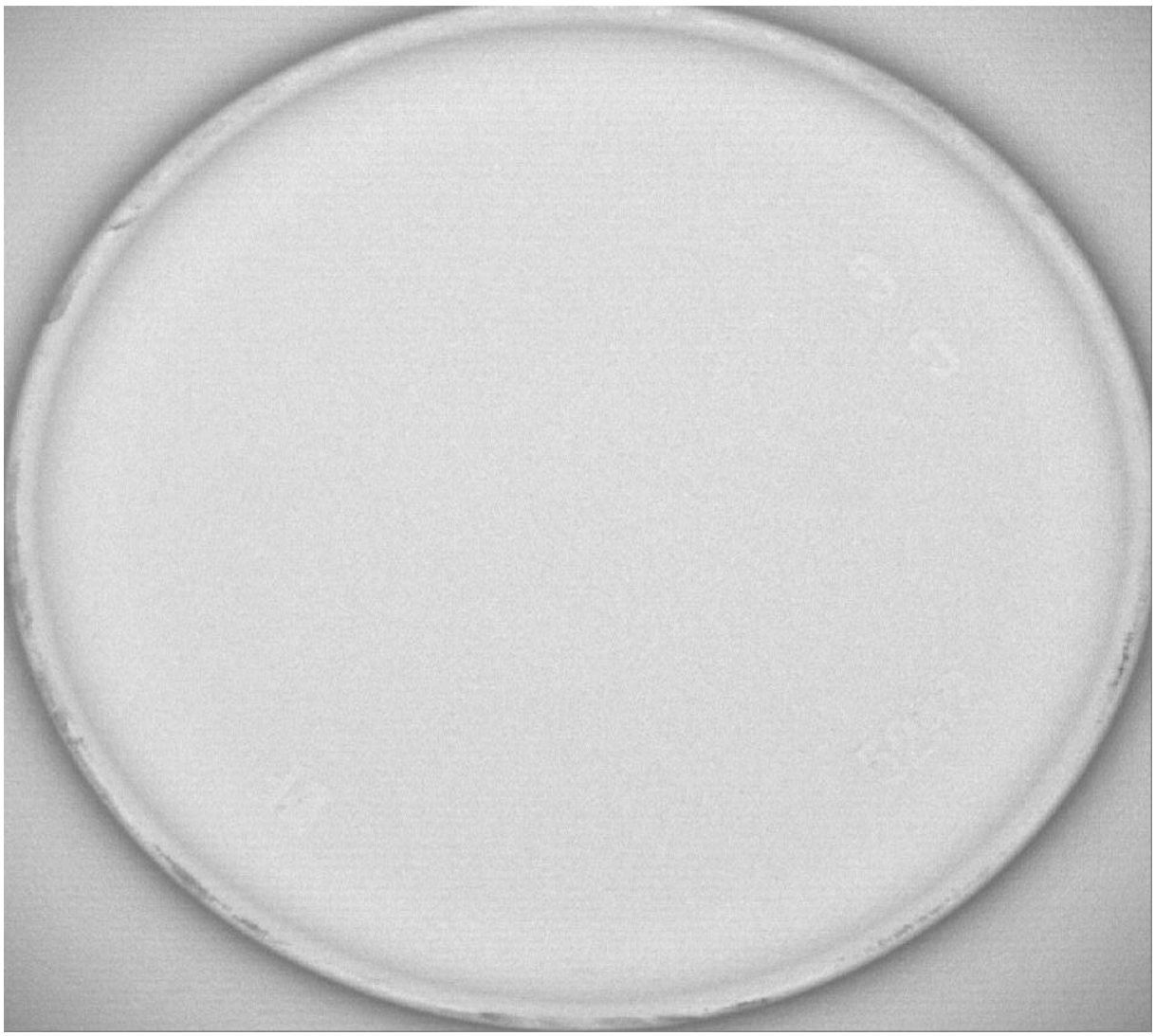


Figura 36 – Regiune de interes decupată

Acestă regiune formează noua imagine neprocesată ce va fi trecută prin metode de îmbunătățire în vederea analizării, adică extragerea conținutului textual în vederea extragerii textului prin recogniția optică a caracterlor cu ajutorul serviciului web Google Vision OCR.

Modulul de *feedback* vizual preia această regiune și aplică metode de procesare ce au parametrii determinați de bările descrise la capitolul 6.3.1. Pentru a reda încontinu procesarea imagini toate metodele sunt puse într-o buclă de tip *while.* Acest lucru ajută la preluarea noilor valori din bările de valori, de fiecare dată când se

întâmplă o iterație completă. La intrare în buclă, prima dată sunt inițializate variable ce stochează valorile date de bările definite mai sus.

```
thresh_bin_val = cv2.getTrackbarPos("Threshold", wnd)
rotation = cv2.getTrackbarPos("Rotation", wnd)
blur_kernel_value = (makeValueOdd(cv2.getTrackbarPos("Blur", wnd)), makeValueOdd(cv2.getTrackbarPos("Blur", wnd)))
clip_limit = cv2.getTrackbarPos("Clip_limit", wnd)
tile_grid_size = (cv2.getTrackbarPos("Tile_grid_size", wnd), cv2.getTrackbarPos("Tile_grid_size", wnd))
```

Figura 37 – Preluare valorilor din bările de valori

După cum se observă în Figura 37, variabile inițiate sunt: *thresh_bin_value, rotation, blur_kernel_value, clip_limit, tile_grid_size*. Aceste variable stochează rezultatul funcției *cv2.getTrackbarPos*() ce primște ca argumente numele bării definite și variabila ce stochează numele ferestrei la care bara este atașată. Aceste valori sunt preluate la fiecare iterație a buclei *while,* si stocate în variabilele lor respective pentru a fi date mai departe metodelor de procesare a imagini digitale.

Prima operația care se află în buclă, este cea de rotire a imagini. Rotirea imaginii digitale se poate face în OpenCV folosind două funcți speciale pentru asa ceva. Prima funcție este *cv2.getRotationMatrix2D()* ce primește ca si argumente coordonate centrului imaginii și valoare de rotire. Coordonatele centrului imaginii se pot determina prin împărțirea la 2 a numărului de coloane și numărului de rânduri a matricii imaginii. Al doi-lea argument ce îl primește funcția este valoarea de rotire, ce este dată de variabila *rotation.* Rezultatul primei funcții este stocat în variabila *M* și se numește matricea de rotire. A doua funcție utilizată în metoda de rotire a imaginii este funcția *cv2.warpAffine()*. Ca și argumente această funcție primește imaginea pe care să se facă rotirea, matricea de rotație și dimensiunea matricii imaginii definită prin rânduri și coloane.

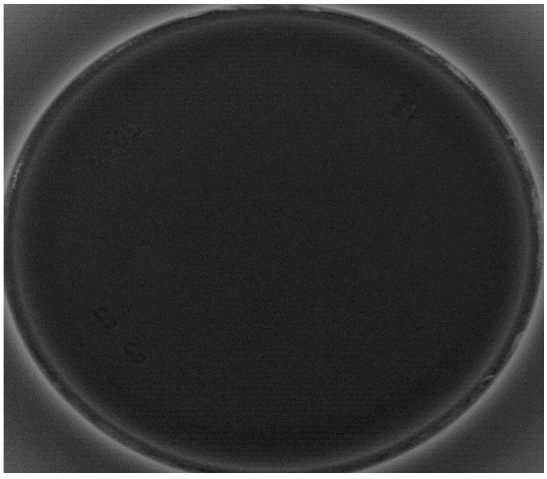


Figura 38 – Imagine rotiă

Imaginea rotită este preluată și stocată în variabila *output*. Acestă variabila se folosește la fiecare metodă pentru a stoca rezultatul ei, adică se supra-scrie. Următorul pas în buclă este aplicare metodie de egalizare a histogramei cu limită de contrast (CLAHE).

```
clahe = cv2.createCLAHE(clipLimit=clip_limit, tileGridSize=tile_grid_size)
output = clahe.apply(output)
```

Figura 39 – Aplicare CLAHE

După cum se observă în Figura 39, pentru a aplica algoritmul CLAHE, prima dată trebuie create un obiect de tip CLAHE utilizând funcția *cv2.createCLAHE().* Funcția primește ca și argumente valoarea *cliptLimit* stocată în variabila *clip_limit* și *tileGridSize* stocată în variabila *tile_grid_size*, amandouă valorile find preluate din bările lor de valori respective. Obiectul creat este stocat în variabila *clahe*. Aceast obiect dispune de o metodă numită *apply()* cu care se aplică algoritmul pe imaginea dorită. Reszultatul aplicării algoritmului este stocat în variabila *output* definită mai sus. Imaginea care rezultă după aplicarea algorimului se poate vede în Figura 40 de mai jos.

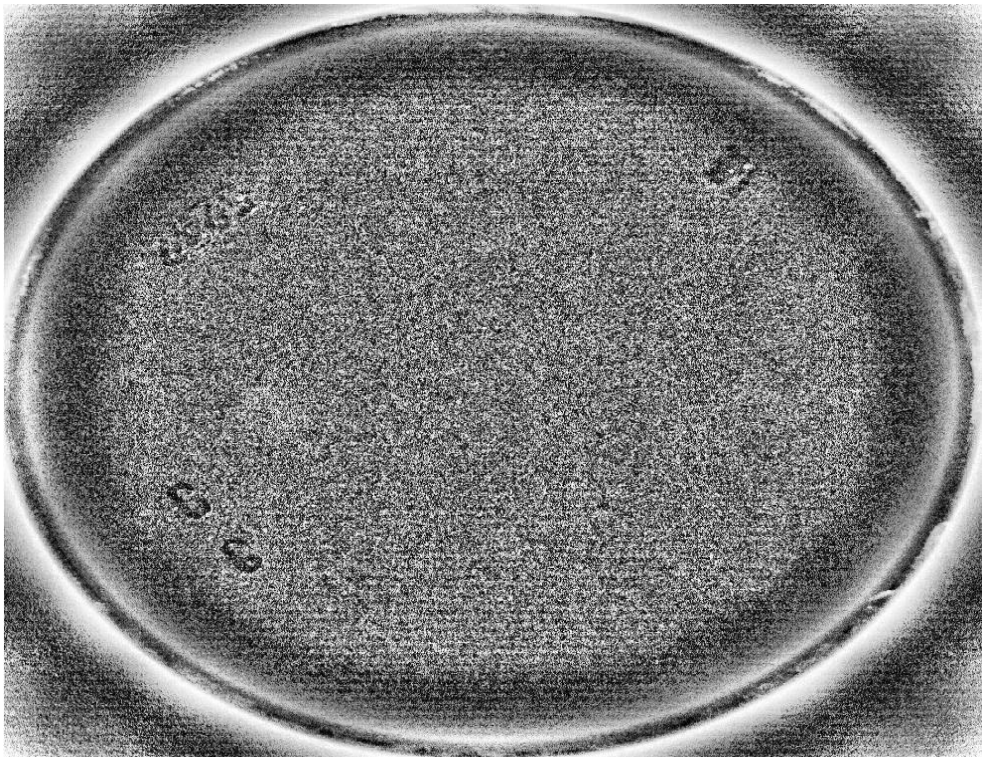


Figura 40 – Imagine procesată cu CLAHE

După cum se observă în Figura 40, conținutul textual a ieșit în evidentă, dar imagine în sine prezintă mult zgomot, ceea ce o face un candidat perfect pentru un filtru de reducere a zgomotului de tip Gaussian.

Aplicarea unui filtru de tip Gaussian se face prin funcția *cv2.GaussianBlur().* Această funcție primește ca și parametrii, imaginea pe care să se aplice filtru, în cazul de fată imagine este stocată în variabila *output,* iar cel de al doilea argument este dimensiune kernelului Gaussian în cazul de față este valoare acestuia este dat de valoarea variabilei *blur_kernel_value,* având o valoare implicită de 17 x 17. Rezultatul este stocat în variabila *output*, iar imagine redată se poate vedea în Figura 41 de mai jos.

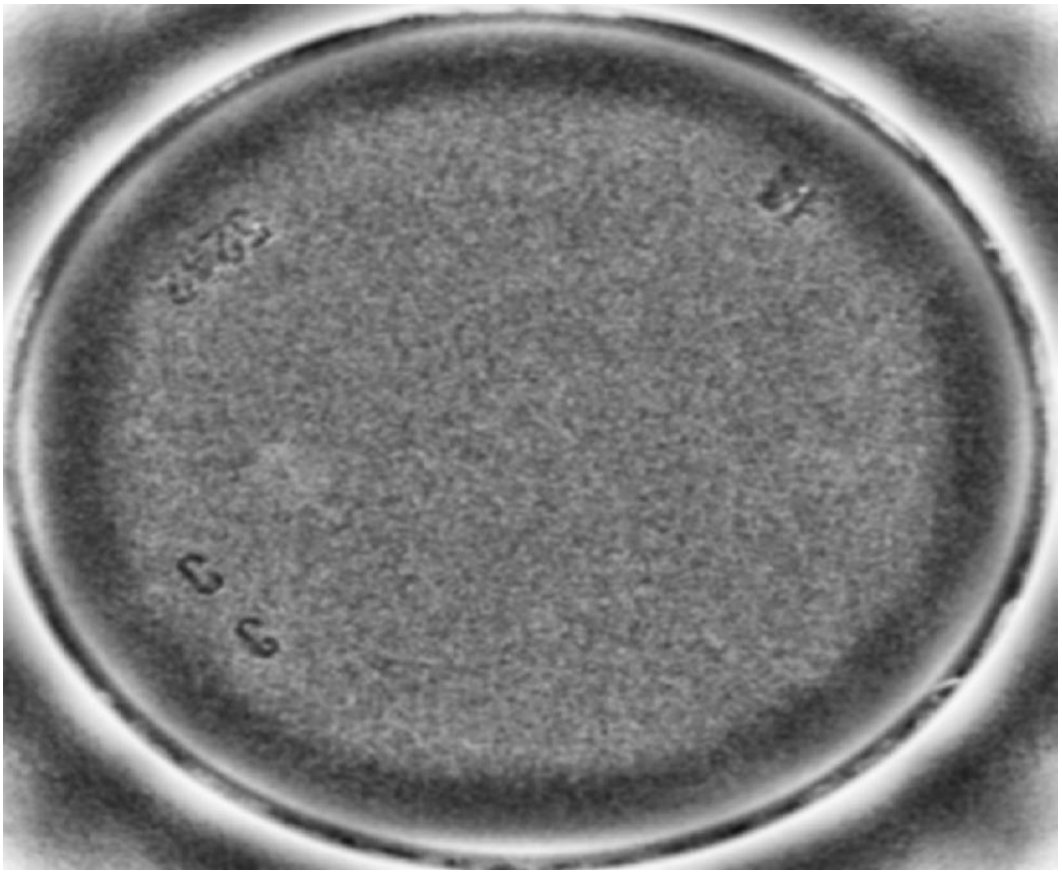


Figura 41 – Imagine filtrată cu un filtru de tip Gaussian

Deși imaginea din Figura 35 pare a fi mai clară, din punct de vedere a procesului de recogniție optică a caracterelor o imagine cu mult zgomot reduce mult rata de succes a recunoașteri lor, astfel imaginea din Figura 36 este un candidat mult mai bun pentru acest proces.

O altă metodă ce se poate aplica este un proces de *thresholding*. Acestă metodă *binarizează* imaginea expunând mai mult textul, dar poate și tăia din text, astfel se pot pierde date din conținutul textual. Procesul de *thresholding* se poate realiza în OpenCV prin funcția *cv2.threshold()*. Parametrii necesari pentru acestă funcție sunt:

imaginea pe care să se execute, valoare de tăiere, valoare la care se modifică pixelul ce are valoare mai mare decât pragul de tăiere, tipul de *threshold*.

```
ret, output = cv2.threshold(output, thresh_bin_val, 255, cv2.THRESH_BINARY)
```

Figura 42 – Aplicarea funcției de *threshold*

În dezvoltarea sistemului s-a constatat că rata de succes a extrageri de text este mai mare dacă imagine este ***binarizată*** rezultatul acestei funcții este stocat în variabila *output*. Rezultatul unei imagini pe care s-a aplicat funcția *cv2.threshold()* se poate vedea în Figura 43.

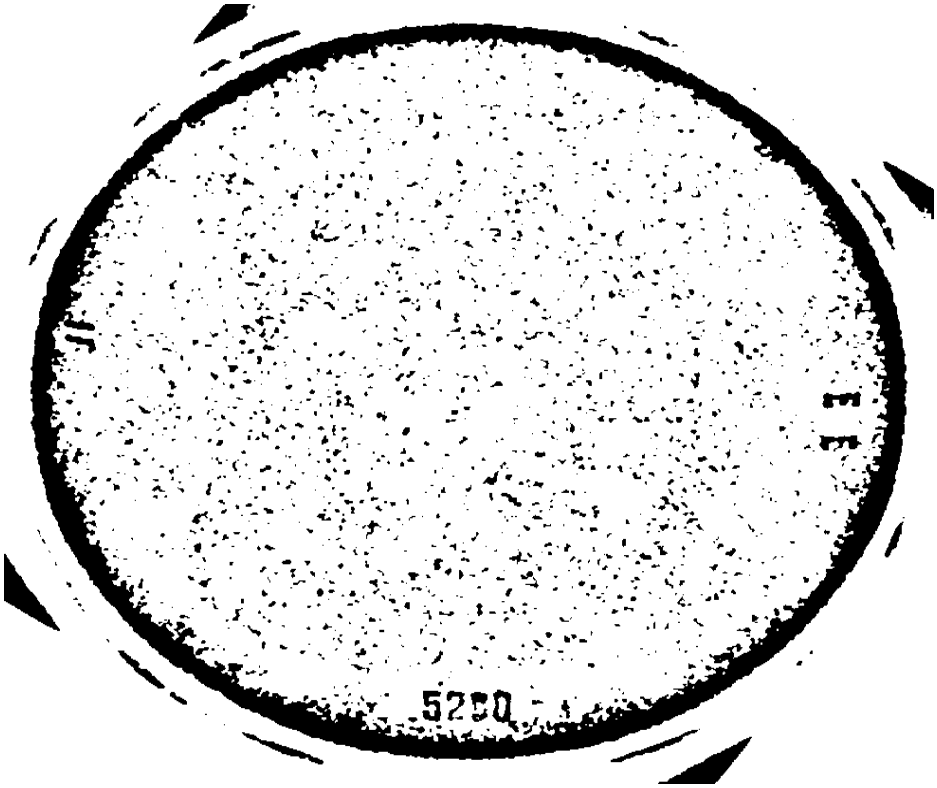


Figura 43 – Rezultat imagine segmentată folosind funcția *cv2.threshold()*

Pentru a afișa fereastra, se folosește funcția *cv2.imshow()*. Acestă funcție acceptă ca și parametrii numele ferestrei, ce este dat de variabila *wnd* și matricea imaginii ce trebuie redată, acesta fiind stocată în variabila *output*. Rezultatul este prezentat în Figura 29 din subcapitolul 6.2.1.

6.4.1 Preluarea imagini pentru extragerea textului și salvarea ei

Modulul de *feedback* vizual implementează și o funcționalitate de testare a extragerii textului. Imaginea este preluată si pasată SDK-ului dat de Google Vision OCR. Figura 39 relatează tot procesul implementat în cod.

```
k = cv2.waitKey(1) & 0xFF
if k == ord('m'):
    print("Extracting...")
    text_list = detect_text(output)
    if len(text_list) is not 0:
        text_list.pop(0)
        text_string = ''.join(text_list)
        output = cv2.cvtColor(output, cv2.COLOR_GRAY2BGR)
        cv2.putText(output, text_string, (10, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 255, 0), 2, cv2.LINE_AA)
        cv2.imwrite(output_dir_path + '/test_2.png', ~output)

        config = {
            "Threshold_value": thresh_bin_val,
            "Rotation": rotation,
            "Blur_kernel_value": blur_kernel_value,
            "Clip_limit": clip_limit,
            "Tile_grid_size": tile_grid_size
        }

        with open(cwd + '/config.json', 'w') as file:
            file.write(json.dumps(config))

    else:
        text_string = 'Could not extract text'
        cv2.putText(output, text_string, (10, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 255, 0), 2, cv2.LINE_AA)
        output = cv2.cvtColor(output, cv2.COLOR_GRAY2BGR)
        cv2.imwrite(output_dir_path + '/test_2.png', ~output)

        config = {
            "Threshold_value": thresh_bin_val,
            "Rotation": rotation,
            "Blur_kernel_value": blur_kernel_value,
            "Clip_limit": clip_limit,
            "Tile_grid_size": tile_grid_size
        }

        with open(cwd + '/config.json', 'w') as file:
            file.write(json.dumps(config))

elif k == 27:
    break
```

Figura 44 – Codul sursă de extragere și salvare a textului extras

După cum se observă în Figura 44, la apasărea tastei *m* sistemul începe procesul de extragere a textului și salvare a imaginii în directorul destinat imaginilor procesate de test. Extragerea efectivă este o cerere HTTP ce o face SDK-ul Google Vision OCR trimițând imaginea serviciului web. Acestă extragere se face prin funcția *detect_text()* ce primește ca parametru matricea imaginii procesate. Logica implementată în acestă funcție este preluată din documentația[14] serviciului web. Dacă serviciul înapoiază text, acesta putând fi conținutul textual în totalitate sau parțial se

[14] Link documentație - https://cloud.google.com/vision/docs/ocr

trece la prelucrare lui. Prelucrare presupune concatenarea tuturor *string-urilor* înapoiate și stocarea lor într-o variabilă. Textul este ca și un *string* și este pasat funcției *cv2.putText()*. Acestă funcție este folosită pentru a scrie textul extras, pe imaginea procesată. Bineînțeles, este doar un simplu exemplu de a face ceva cu el. Odată obținut ca un *string* textul poate fi scris în fisier, într-o bază de date sau chiar trimis prin HTTP unui serviciu web. Imaginea procesată și inscripționată cu textul extras este salvată în fișierul destinat salvării imaginilor de test numit *test_output.* În Figura 45, se relatează o imagine procesată și inscriptionată cu text.

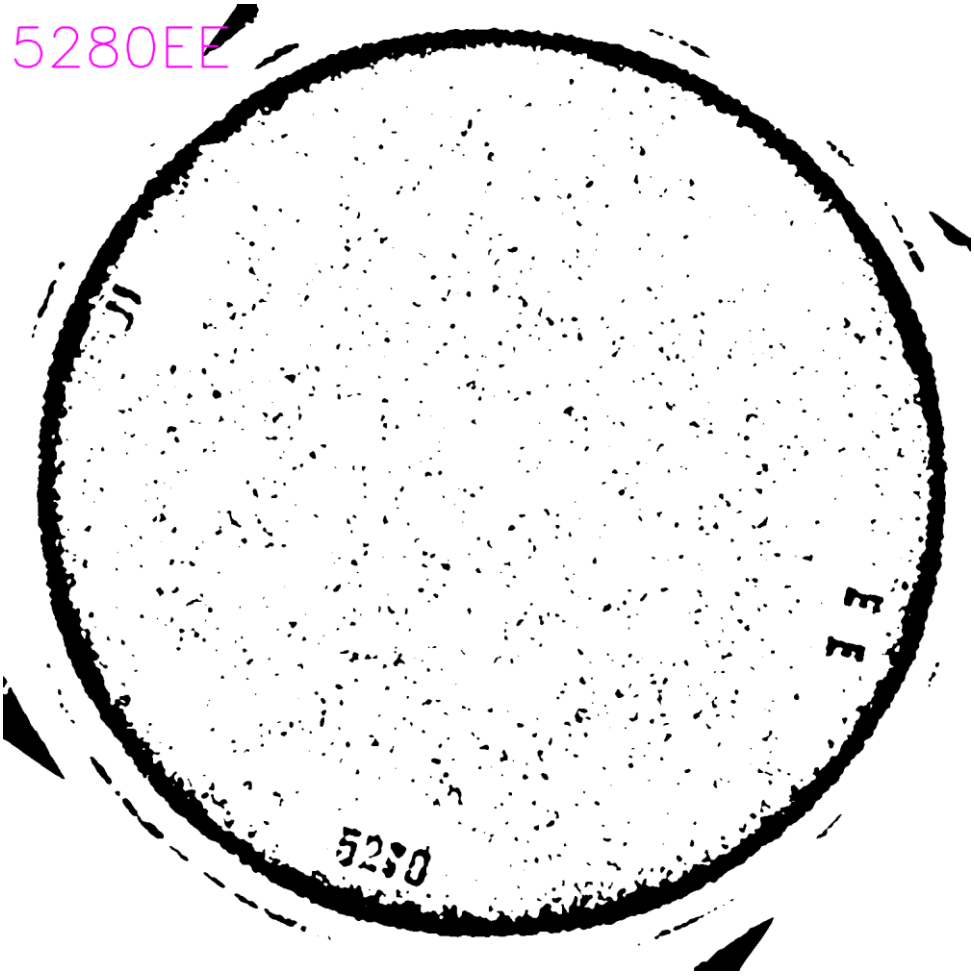


Figure 45 – Imagine procesată cu conținut textual extras

Tot aici se întamplă și salvarea parametrilor configurați cu ajutorul modului de *feedback* vizual. Aceștia sunt preluați din variabile lor respective și structurați sub forma unui dicționar Python. Acesta este serializat în format de tip JSON și este scris în fișierul *config.json*. Modulul de linie de comadă citește acest fișiser și aplică acesti parametri metodelor de procesare a imginilor digitale.

Concluzii

Procesarea imaginilor digitale face pare dintr-un spectru mai larg ce ține de digitalizarea imaginilor. Metodele de procesare sau procesarea se situează între achiziție și analiză imaginilor. Fiecare dintre aceste aspecte se bazează una pe cealaltă pentru o rată de succes mare. Procesarea are ca scop evidențierea caracteristicilor din imagine pentru a ușura analiza acesteia. Totuși, procesarea lor nu poate fi automatizată, cel puțin nu pe imagini ce relatază lucruri total diferite. Pentru o serie de imagini ce relateză același aspect, automatizarea procesării în vederea analizei devine mai ușor de realizat. Caracterul unic al procesării denotă un aspect subiectiv, procesarea putând fi abordată din mai multe puncte de privire în funcție de seria de imagini și de caracteristica care se vrea a fi analizată. Calitatea procesării imaginilor digitale este direct proprțională cu calitatea achiziției.

Pentru procesarea imaginilor digitale, vin în ajutor limbaje de programare și biblioteci capabile să manipuleze pixeli imaginilor. Limbajul folosit pentru a implementa sistemul descris în această lucrare de disertație este Python, deoarece este simplu de utilizat, ușor de instalat și dispune de o gamă largă de biblioteci ce îi vin în ajutor, cum ar fi biblioteca OpenCV, o bibliotecă dedicată procesării imaginilor digitale. O concluzie ce poate fi afirmată este că orice sistem de acest gen este format din minim trei părți, acestea fiind:

- partea de citire a imaginilor de intrare
- partea de procesare
- partea de scriere a imaginilor

Ca și o concluzie determinată pe partea de procesare efectivă a imaginilor, deși subiectivă, totuși se supune unor reguli de minime sau mai bine zis unor ghiduri. Prima regulă este determinarea regiunii de interes, adică regiunea în care se află conținutul textual sau caracteristica de interes. A doua regulă este reducerea zgomotului prin aplicarea unor filtre în asa fel în cât să păstreze caracteristica de interes intactă. O a treia regulă, este, mărirea contrastului între caracteristica de interes și fundal, acestă putând fi realizată prin aplicarea algoritmilor de îmbunătățire a contrastului sau chiar binarizarea imaginii.

În final, o studiere în detaliu a setului de imagini și aplicarea metodelor și algoritmilor corespunzători, duce la o procesare eficientă și rapidă cu o rată de succes mare a analizei.

Bibliografie

[1] M. Lyra, A. Ploussi and A. Georgantzoglou, "**MATLAB – A Ubiquitous Tool for the Practical Engineer," in *MATLAB – A Ubiquitous Tool for the Practical Engineer***, Athens, University of Athens Greece , 2015, p. 489.

[2] R. E. Woods and R. C. Gonzlez, ***Digital Image Processing***, London: Pearson, 2007.

[3] W. Burger, ***Principles of Digital Image Processing***: Springer, 2009.

[4] Himanshu, ***Practical Machine Learning and Image Processing***, Uttar Pradesh: Apress, 2019.

[5] K. Dawson-Howe, ***A Practical Introduction to Computer Vision with OpenCV (Wiley-IS&T Series in Imaging Science and Technology)***: Wiley, 2014.

[6] R. C. Gonzalez and R. C. Woods, ***Digital Image Processing***, London: Pearson, 2017.

[7] M. Nixon and A. Aguado, ***Feature Extraction & Image Processing for Computer Vision***: Academic Press, 2012.

[8] J. C. Russ and B. F. Neal, ***The Image Processing Handbook***: CRC Press, 2015.

[9] C. Solomon and T. Breckon, ***Fundamentals of Digital Image Processing***, London: Wiley-Blackwell, 2010.

[10] OpenCV, "OpenCV Documentation," [Online]. Valabil la: https://docs.opencv.org/3.0-beta/doc/tutorials/tutorials.html. [Accesat 02 06 2019].

[11] Wikipedia, "Kernel (image processing)," [Online]. Valabil la: https://en.wikipedia.org/wiki/Kernel_(image_processing). [Accesat 15 05 2019].

Lista figurilor

Lista formulelor

ANEXA A – Codul sură modul de *feedback* vizual

```
import os
import io
import numpy as np

from PIL import Image

try:
    from cv2 import cv2
except ImportError:
    pass

import json

cwd = os.path.dirname(os.path.abspath(__file__))
input_dir_path = cwd + '/data/test_input'
output_dir_path = cwd + '/data/test_output'
input_image_names = os.listdir(input_dir_path)



def nothing(x):
    pass

def makeValueOdd(value):
    if (value % 2) is not 0:
        return value
    return value + 1

def detect_text(imageArray):
    """Detects text in the file."""
    from google.cloud import vision
    client = vision.ImageAnnotatorClient()

    content = Image.fromarray(imageArray)

    imgByteArr = io.BytesIO()
    content.save(imgByteArr, format='PNG')
```

```
    imgByteArr = imgByteArr.getvalue()

    image = vision.types.Image(content=imgByteArr)

    response = client.text_detection(image=image)
    texts = response.text_annotations
    print('Texts:')
    text_list = []
    for text in texts:
        print('\n"{}"'.format(text.description))
        text_list.append(text.description)
        # vertices = (['({},{})'.format(vertex.x, vertex.y) for vertex in text.bounding_poly.vertices])
        #
        # print('bounds: {}'.format(','.join(vertices)))

    return text_list


wnd = 'Sandbox'
cv2.namedWindow(wnd, cv2.WINDOW_NORMAL)

cv2.createTrackbar("Threshold", wnd, 172, 255, nothing)
cv2.createTrackbar("Blur", wnd, 17, 255, nothing)
cv2.createTrackbar("Rotation", wnd, 145, 360, nothing)
cv2.createTrackbar("Clip_limit", wnd, 76, 255, nothing)
cv2.createTrackbar("Tile_grid_size", wnd, 11, 255, nothing)


img = cv2.imread(input_dir_path + '/' + input_image_names[1])


gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

clahe = cv2.createCLAHE(clipLimit=7.6, tileGridSize=(11, 11))
cl1 = clahe.apply(gray_img)

blurr = cv2.GaussianBlur(cl1, (17, 17), 0)
```

```
circles = cv2.HoughCircles(blurr, cv2.HOUGH_GRADIENT, 1.2, 170, 70, 80)


radius = 0
rect_height = 0
rect_width = 0


if circles is not None:
    # convert the (x, y) coordinates and radius of the circles to integers
    circles = np.round(circles[0, :]).astype("int")


    # loop over the (x, y) coordinates and radius of the circles
    for (x, y, r) in circles:
        # draw the circle in the test_output image, then draw a rectangle
        # corresponding to the center of the circle
        cv2.circle(img, (x, y), r, (0, 255, 0), 4)


        cv2.rectangle(img, (x-r, y-r), (x+r, y+r), (0, 128, 255), 2)
        rect_height = y
        rect_width = x
        radius = r


top = rect_width - radius
left = rect_height - radius
bottom = rect_width + radius
right = rect_height + radius
roi = gray_img[left:right, top:bottom]


while (1):
    thresh_bin_val = cv2.getTrackbarPos("Threshold", wnd)
    rotation = cv2.getTrackbarPos("Rotation", wnd)
    blur_kernel_value = (makeValueOdd(cv2.getTrackbarPos("Blur", wnd)),
makeValueOdd(cv2.getTrackbarPos("Blur", wnd)))
    clip_limit = cv2.getTrackbarPos("Clip_limit", wnd)
    tile_grid_size = (cv2.getTrackbarPos("Tile_grid_size", wnd), cv2.getTrackbarPos("Tile_grid_size",
wnd))


    rows = roi.shape[0]
```

```
    cols = roi.shape[1]
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), rotation, 1)
    output = cv2.warpAffine(roi, M, (cols, rows))

    clahe = cv2.createCLAHE(clipLimit=clip_limit, tileGridSize=tile_grid_size)
    output = clahe.apply(output)

    output = cv2.GaussianBlur(output, blur_kernel_value, 0)



    ret, output = cv2.threshold(output, thresh_bin_val, 255, cv2.THRESH_BINARY)



    cv2.imshow(wnd, ~output)

    k = cv2.waitKey(1) & 0xFF
    if k == ord('m'):
        print("Extracting...")
        text_list = detect_text(output)
        if len(text_list) is not 0:
            text_list.pop(0)
            text_string = ''.join(text_list)
            output = cv2.cvtColor(output, cv2.COLOR_GRAY2BGR)
            cv2.putText(output, text_string, (10, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 255, 0), 2,
cv2.LINE_AA)
            cv2.imwrite(output_dir_path + '/test_2.png', ~output)


            config = {
                "Threshold_value": thresh_bin_val,
                "Rotation": rotation,
                "Blur_kernel_value": blur_kernel_value,
                "Clip_limit": clip_limit,
                "Tile_grid_size": tile_grid_size
            }


            with open(cwd + '/config.json', 'w') as file:
                file.write(json.dumps(config))
```

```
        else:
            text_string = 'Could not extract text'
            cv2.putText(output, text_string, (10, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 255, 0), 2,
cv2.LINE_AA)
            output = cv2.cvtColor(output, cv2.COLOR_GRAY2BGR)
            cv2.imwrite(output_dir_path + '/test_2.png', ~output)

            config = {
                "Threshold_value": thresh_bin_val,
                "Rotation": rotation,
                "Blur_kernel_value": blur_kernel_value,
                "Clip_limit": clip_limit,
                "Tile_grid_size": tile_grid_size
            }

            with open(cwd + '/config.json', 'w') as file:
                file.write(json.dumps(config))

    elif k == 27:
        break

cv2.destroyAllWindows()
```

ANEXA B – Cod sursă modul linie de comandă

```
import os
import io
import numpy as np
try:
    from cv2 import cv2
except ImportError:
    pass

import json
from PIL import Image
import time

cwd = os.path.dirname(os.path.abspath(__file__))
input_dir_path = cwd + '/data/unprocessed'
output_dir_path = cwd + '/data/processed'

os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = cwd +
'/data/credentials/vision/credentials.json'

with open(cwd + '/config.json', 'r') as file:
    config = file.read()

config = json.loads(config)

def nothing(x):
    pass



def makeValueOdd(value):
    if (value % 2) is not 0:
        return value
    return value + 1

def detect_text(imageArray):
    """Detects text in the file."""
    from google.cloud import vision
```

```
    client = vision.ImageAnnotatorClient()

    content = Image.fromarray(imageArray)

    imgByteArr = io.BytesIO()
    content.save(imgByteArr, format='PNG')
    imgByteArr = imgByteArr.getvalue()

    image = vision.types.Image(content=imgByteArr)

    response = client.text_detection(image=image)
    texts = response.text_annotations
    print('Texts:')
    text_list = []
    for text in texts:
        print('\n"{}"'.format(text.description))
        text_list.append(text.description)
        # vertices = (['({},{})'.format(vertex.x, vertex.y) for vertex in text.bounding_poly.vertices])
        #
        # print('bounds: {}'.format(','.join(vertices)))

    return text_list


thresh_bin_val = config['Threshold_value']
rotation = config["Rotation"]
blur_kernel_value = config["Blur_kernel_value"]
clip_limit = config["Clip_limit"]
tile_grid_size = config["Tile_grid_size"]

while(1):
    image_names = os.listdir(input_dir_path)
    if len(image_names) > 0:
        print("Found {} new images in input folder...".format(len(image_names)))

        for image_name in image_names:
            img = cv2.imread(input_dir_path + '/' + image_name)
```

```
print('Processing image {}'.format(image_name))
print('Extracting region of interest...')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
clahe = cv2.createCLAHE(clipLimit=7.6, tileGridSize=(11, 11))
cl1 = clahe.apply(gray_img)
blurr = cv2.GaussianBlur(cl1, (17, 17), 0)

circles = cv2.HoughCircles(blurr, cv2.HOUGH_GRADIENT, 1.2, 170, 70, 80)

radius = 0
rect_height = 0
rect_width = 0

if circles is not None:
    circles = np.round(circles[0, :]).astype("int")

    for (x, y, r) in circles:
        rect_height = y
        rect_width = x
        radius = r

top = rect_width - radius
left = rect_height - radius
bottom = rect_width + radius
right = rect_height + radius
roi = gray_img[left:right, top:bottom]

print('Rotating image to config parameter value')
rows = roi.shape[0]
cols = roi.shape[1]
M = cv2.getRotationMatrix2D((cols / 2, rows / 2), rotation, 1)
output = cv2.warpAffine(roi, M, (cols, rows))

print("Applying Contrast Limited Adaptive Histogram Equalization")
print(tuple(tile_grid_size))
clahe = cv2.createCLAHE(clipLimit=clip_limit, tileGridSize=tuple(tile for tile in tile_grid_size))
output = clahe.apply(output)
```

```
            print("Applying Gausian Blurr")
            output = cv2.GaussianBlur(output, tuple(blur_kernel_value), 0)

            print("Thresholding")
            ret, output = cv2.threshold(output, thresh_bin_val, 255, cv2.THRESH_BINARY)

            print("Proceeding to text extraction, plrease stand by...")

            text_list = detect_text(~output)
            text_string = "No text found"
            if len(text_list) is not 0:
                text_list.pop(0)
                text_string = ''.join(text_list)
                output = cv2.cvtColor(output, cv2.COLOR_GRAY2BGR)
                cv2.putText(output, text_string, (10, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 255, 0),
2, cv2.LINE_AA)
                cv2.imwrite(output_dir_path + '/' + str(time.time()) + '_' + image_name , ~output)
            else:
                cv2.putText(output, text_string, (10, 100), cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 255, 0),
2, cv2.LINE_AA)
                output = cv2.cvtColor(output, cv2.COLOR_GRAY2BGR)
                cv2.imwrite(output_dir_path + '/' + str(time.time()) + '_' + image_name , ~output)

            print("{} was processed. Removing...".format(image_name))
            os.remove(input_dir_path + '/' + image_name)
        else:
            print("No new images found. Awaiting images...")

        time.sleep(1)
```