MINISTERUL EDUCAȚIEI
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Detecția semafoarelor din traficul urban folosind tehnici de clasificare bazate pe trăsături extrase din spațiile de culoare și din histograma gradienților

LUCRARE DE LICENȚĂ

Absolvent: **Marius-Gabriel MUREA**
Coordonator științific: **ȘL. dr. ing. Robert VARGA**

Cuprins

Capitolul 1. Introducere – Contextul proiectului

Această aplicație software (sistem software) are titlul "**Detecția semafoarelor din traficul urban folosind tehnici de clasificare bazate pe trăsături extrase din spațiile de culoare și din histograma gradienților**". La capitolul curent se va descrie conceptul de conducere autonomă, acesta fiind contextual principal în care s-a creat aplicația, dar și nivelele de conducere autonomă și un set de avantaje dar și dezavantaje pentru acest domeniu vast și rezultatele obținute la ora actuală din el. În continuare se va prezenta contextul în care s-a realizat acest proiect.

1.1. Contextul proiectului – conducerea autonomă

Acest document reprezintă descrierea detaliată a funcționării unei aplicații de inteligență artificială cu ajutorul căreia se poate face condusul autonom al autovehiculelor mai sigur și mai eficient. Proiectul de față se dorește a fi o aplicație pentru detecția și recunoașterea semafoarelor din trafic, precum și a culorilor lor. Această aplicație servește, după cum s-a menționat în primele rânduri, pentru conducerea autonomă din mediul urban.

Conceptul de conducere autonomă (în engleză, **autonomous driving**) este o ramură a tot ce ține de inteligența artificială. După denumire, vine în mod normal de la vehiculele care se conduc singure, sau de la sistemele de transport care se pot deplasa din punctul A în punctul B, fără intervenția factorului uman. Ca și continuare la începutul de răspuns la întrebarea "Ce este o mașină autonomă?" se va descrie în următorul paragraf această entitate.

O mașină autonomă (în engleză, **self-driving car**) este un vehicul care poate, fără intervenția șoferului uman, să se deplaseze între două sau mai multe locații fără a pune în pericol siguranța traficului din jurul ei. Deci, cu alte cuvinte, este un vehicul care se conduce singur. Aceste mașini autonome au în componența lor o varietate largă de senzori și sisteme software, precum: radare, sonare, camera video, sisteme GPS, odometre etc cu ajutorul cărora ele se pot deplasa în siguranță în mediul urban, chiar și rural. Într-un astfel de vehicul, pasagerii nu sunt obligați să preia controlul mașinii, chiar se va ajunge în viitorul apropiat la a nu fi nevoiți să se afle în mașină în timpul deplasării. Deci, un vehicul autonom, se poate deplasa oriunde s-ar putea deplasa și un vehicul obișnuit (tradițional) și poate face cam tot ceea ce poate face un șofer cu ani de experiență.

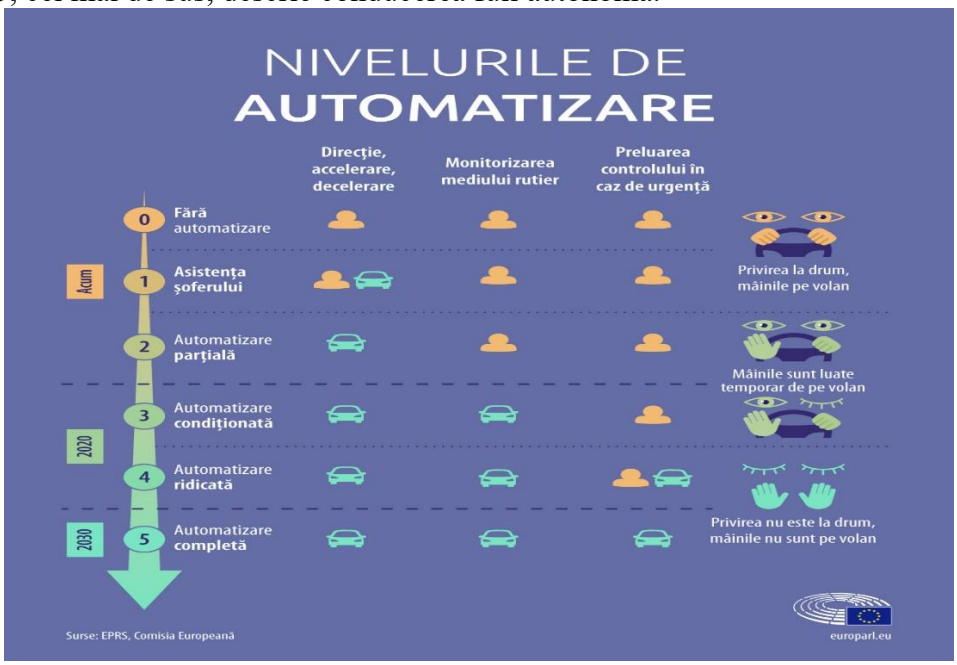
Mașinile autonome au în componența lor sisteme software și senzori care pot identifica obstacole, semafoare, semne de circulație, senzori care pot face ca autovehiculele să schimbe singure direcția sau chiar sisteme care le fac să se deplaseze pe distanțe lungi, chiar de ordinal sutelor de kilometri.

Ca un mic istoric, s-au efectuat experimente pe sisteme automate încă din anii '20, dar cu precădere din anii '50, după sfârșitul celui de-Al Doilea Război Mondial. În 1977 a fost prima încercare de producere a unei mașini semi-autonome, în laboratoarele din Japonia. Pe această mașină erau amplasate două camere și un computer analog și aveau ca misiune să interpreteze câteva străzi marcate. Cu anii, numeroase companii producătoare de mașini au început demersurile pentru implementarea unor astfel de vehicule, iar acest domeniu a adâncit și mai mult războiul tehnologic între state, marile puteri dorindu-și fiecare să fie pe prima poziție pe această ramură. Astăzi, cel mai mare producător de mașini autonome este Tesla, Inc. dar și alți producători de mașini, cum ar fi BMW, Audi, Ford etc sunt fruntașe pe acest domeniu. Din păcate, trebuie adăugat, încă nu există vreo țară care să aibă o legislație full permisivă pentru conducerea autonomă și este de înțeles pentru că trebuie să se facă demersuri în continuare pentru a nu exista pericole în trafic cu acest tip de mașini. [1]

În cele ce urmează se vor descrie puțin nivelurile de conducere autonomă existente și definite de SAE (**Society of Automotive Engineers**).

1.2. Nivele de conducere autonomă

Conducerea autonomă a fost definite de SAE (**Society of Automotive Engineers**) pe 6 nivele. Cel mai de jos nivel, nivelul 0 caracterizează conducerea clasică, folosind șoferul uman, iar nivelul 5, cel mai de sus, descrie conducerea full autonomă.



[Figura 1.1. Cele 6 nivele de conducere autonomă]

În cele ce urmează se vor detalia cele 6 nivele de conducere autonomă:

0. Nivelul 0 (fără automatizare)

Suntem în secolul 21, iar în continuare, cele mai multe vehicule întâlnite în trafic sunt de nivelul 0. Ele nu beneficiază de sisteme inteligente care pot face să pară mașini autonome (ca de exemplu, sisteme care permit menținerea benzii de circulație). Aceste vehicule, chiar dacă au sisteme care să-i vină în ajutor șoferului (ABS, ESP, etc), ele nu pot fi încadrate ca mașini autonome. Cu alte cuvinte, mașinile întâlnite la nivelul 0 sunt cele controlate manual, de către un șofer.

1. Nivelul 1 (Asistența șoferului)

Nivelul 1 este cel mai scăzut nivel de automatizare. Cu anii ce au trecut, tehnologia în domeniul de conducere autonomă a evoluat și marii producători de mașini au creat sisteme automate pentru asistența șoferului. Un exemplu aici ar fi controlul direcției și al accelerației (sistemul de control al vitezei). Chiar dacă acesta este un sistem automat și se situează la nivelul 1 al piramidei de conducere autonomă, șoferul tot va trebui să monitorizeze cu atenție și alte aspect ale conducerii, cum ar fi direcția și frânarea mașinii.[2]

După cum s-a dat exemplu mai sus, cel mai folosit sistem software ce este implementat pe mașinile autonome ce se regăsesc la nivelul 1 este controlul vitezei. Acesta ajută șoferul să mențI-

nă o viteză constantă, doar setând această viteză dorită cu ajutorul unei manete sau al unui buton. Acest sistem duce la un condus mai relaxat pe autostrăzi, eventual pe drumuri lungi și drepte, în care traficul este normal, nefiind nevoie de schimbări dese ale vitezei de deplasare. Mașinile cu sisteme mai inteligente dispun de Controlul Vitezei Adaptiv (în engleză, adaptive cruise control), prin care se poate și redresa viteza de deplasare a vehiculelor.

2. **Nivelul 2 (Automatizare parțială)**

La nivelul 2 de conducere autonomă avem deja sisteme avansate care vin în ajutorul conducătorului auto. Prin aceste sisteme, vehiculul poate controla atât direcția, cât și accelerarea/decelerarea. În cele ce urmează se va descrie în câteva rânduri sistemul de menținere a direcției care face parte din nivelul 2 de conducere autonomă.

În primul rând, trebuie spus că șoferii tind să nu mențină direcția de mers pe o anumită bandă din diferite motive: oboseală, neatenție, incapacitatea de a vedea benzile, sau chiar plictiseală. Pe autostrăzi, de exemplu, se întâmplă din păcate, extrem de multe accidente în care sunt implicate autovehicule din aceste motive. Sistemele de menținere a direcției de mers (a benzii de circulație) au în componența lor alerte, audio sau vizuale, prin care șoferul este notificat că mașina trece pe o altă bandă fără semnalizare și poate fi pus el, implicit și ceilalți participanți la trafic în pericol. Există sisteme mai avansate de asistență la păstrarea benzii care fac ca mașina să-și corecteze direcția de mers, împingând-o înapoi spre centrul drumului. Asistența pentru păstrarea benzii de circulație are la bază o cameră orientată spre direcția de deplasare a mașinii, aceasta putând fi încorporată în oglinda retrovizoare sau în alte locuri care să-i ofere o rază cât mai mare de acoperire. Această cameră scanează în permanență marcajele longitudinale de pe drum și caută liniile albe sau galbene de pe suprafața întunecată (neagră) al asfaltului. Dacă autovehiculul se apropie prea mult de o linie despărțitoare de bandă, sistemul va trimite un semnal acustic și chiar și vizual șoferului, notificându-l că trebuie să mențină banda. Unele sisteme trimit chiar și un feedback către volan (se va resimți de către șofer ca o vibrație). Dacă nu există răspuns din partea șoferului, mașina, prin acest sistem, va prelua controlul direcției și va mișca ușor volanul în direcția opusă pentru a îndepărta vehiculul de linia benzii.

3. **Nivelul 3 (Automatizare condiționată)**

La nivelul 3 de conducere autonomă, vehiculele au capacitatea de a înțelege mediul în care se află și pot lua decizii în funcție de posibilele interacțiuni cu celelalte elemente participante la trafic. Acest nivel nu reprezintă un cumul de sisteme care să facă mașina să fie full autonomă și din această cauză, șoferul trebuie să rămână alert și trebuie oricând să fie gata să preia controlul mașinii, dacă sistemul nu este capabil să execute sarcina curentă. Un exemplu de sistem inteligent, ar fi modul prin care mașinile Tesla, de la celebrul producător de mașini autonome, pot să recunoască anumite evenimente din trafic (trecerea pietonilor prin fața autovehiculului, schimbarea benzii și asigurarea că se poate executa această sarcină în condiții de siguranță).

Studiile arată că acest nivel constă în aproximativ 75% automatizare totală a mașinii. Cu alte cuvinte, sumarizând ce s-a descris mai sus, mașina la acest nivel, îndeplinește funcții critice pentru deplasare, cum ar fi: accelerația, decelerarea și direcția. Chiar dacă șoferul uman va fi doar un copilot care va putea prelua controlul mașinii imediat, scopul este ca el să se poată implica și în alte activități (urmărirea împrejurimilor sau trimiterea de mesaje text, ca de exemplu). Mașina autonomă va face față situațiilor care necesită un răspuns imediat. Ca un exemplu crucial, dacă un alt vehicul apare brusc în fața mașinii, sistemele încorporate în mașina autonomă vor face ca ea să efectueze frânarea de urgență. În cele ce urmează se vor enumera și descrie cele mai importante

componente sau caracteristici care fac ca o mașină să fie inclusă la nivelul 3 de conducere autonomă:

- **Sistemele computerizate –** O mașină încadrată la nivelul 3 de automatizare are nevoie de o putere semnificativă de calcul. Cele mai multe mașini de acest tip au senzori și enorm de multe camere care culeg informații din trafic, iar imaginile extrase trebuie să fie prelucrate (procesare de imagini). Dacă aceste vehicule au capacitatea de a lua decizii pe baza imaginilor culese înseamnă că acolo, în sistemele încorporate, sunt implementați diferiți algoritmi de Învățare Automată sau se lucrează cu rețele neurale, iar din acest motiv este nevoie de o imensă putere de calcul, prelucrându-se în timp real sute, chiar mii de imagini.
- **Camere –** Într-o mașină autonomă, camerele sunt utilizate pentru a surprinde traficul și implicit șoseaua pe care se deplasează mașina. Imaginile extrase cu ajutorul acestor camere ajută vehiculul autonom să decidă în funcție de situația în care se află la momentul curent de timp. De exemplu, dacă camerele amplasate în fața mașinii detectează un obstacol, ele vor transmite sistemelor computerizate ale ei, să acționeze frâna de urgență. Cele mai noi mașini autonome sunt echipate cu camere cu infraroșu pentru a îmbunătăți și mai mult vizibilitatea. Un alt exemplu, pot exista camere care să detecteze semafoarele din trafic, dintr-o intersecție și implicit culorile lor. Dacă semaforul detectat are culoarea roșie, camerele vor trimite un semnal către sistemele mașinii, iar aceasta va opri. Dacă culoarea semaforului este verde se va începe deplasarea mașinii.
- **Radar –** Un sistem radar implementat pe o mașină autonomă poate face ca ea să facă distincție între diferitele tipuri de vreme (între o zi însorită și o zi ploioasă, ca de exemplu). În funcție de vremea și de mediul în care se deplasează autovehiculul, implicit de situația în care se află carosabilul (uscat sau umed), sistemele mașinii pot adapta viteza de deplasare și distanța între vehiculul din față.
- **Senzori –** Într-o mașină autonomă de nivel 3, senzorii sunt folosiți pentru a detecta situații și pentru a lua decizii pe baza acestora. Ca și exemplu de situație, un senzor poate detecta un pieton care traversează drumul, ceea ce va determina ca mașina să decidă să acționeze frâna și implicit să oprească deplasarea. Cele mai noi mașini autonome au în componența lor, o multitudine de senzori LIDAR, amplasați în partea de sus a mașinilor, aceștia furnizând date 3D despre mediul din jurul autovehiculelor. Senzorii LIDAR folosesc unde de lumina pentru a furniza informații despre obiectele din apropierea mașinii.

4. Nivelul 4 (Automatizare ridicată)

La nivelul 4 găsim vehicule care pot funcționa în modul de conducere autonomă în mod sigur, dar constrânse de legislația aferentă acestui domeniu și de infrastructură. Aceste vehicule pot circula doar în zone limitate (de exemplu, în mediul urban, unde viteza maximă este de 50-60 km/h). Aceste mașini încadrate la nivelul 4 de automatizare nu necesită interacțiunea, controlul unui om, deoarece ele sunt programate să se oprească singure când un sistem din componența lor pică. Din acest motiv, se poate întâmpla ca autovehiculele întâlnite la acest nivel să nu necesite volan pentru schimbarea direcției sau pedale de control a vitezei. Există țări și orașe în lume care au implementat în transportul lor public un sistem și o flotă de astfel de autovehicule. De exemplu, s-au fabricat taxiuri fără șofer sau servicii de transport public autonome. Astfel de vehicule sunt programate să circule din punctul A în punctul B, în diferite intervale orare, dar pot fi limitate sau serviciile lor pot fi anulate în funcție de vremea și de condițiile de trafic.

5. **Nivelul 5 (Automatizare completă)**

Ultimul nivel de automatizare, nivelul 5, descrie posibilitatea ca un vehicul să se poată deplasa oriunde, în toate condițiile de trafic și de vreme, fără nicio interacțiune umană. Singura implicare umană va fi stabilirea unei destinații, deoarece aceste mașini vor reprezenta o alternativă eficientă și confortabilă de transport pentru ființe umane, fără necesitatea unui șofer. Aceste vehicule autonome sunt supuse testelor în diferite zone dezvoltate ale lumii, dar niciuna nu este încă disponibilă publicului larg.

1.3. Avantajele și dezavantajele conducerii autonome

Autovehiculele cu conducere autonomă (sau autovehiculele autonome, în engleză **self-driving cars**) au o mare șansă de a deveni în viitorul apropiat principala sursă de transport pentru omenire. Ca orice sistem sau tehnologie nou apărută există un set de avantaje dar și dezavantaje pentru aceste mașini autonome. În cele ce urmează, mai exact în următoarele două sub-capitole se vor prezenta pe rând avantajele și dezavantajele mașinilor viitorului.

1.3.1. Avantaje

Avantajele conducerii autonome sunt nenumărate, atât pentru calitatea vieții umane, cât și pentru mediul înconjurător. Marii antreprenori și marile companii de inginerie și producătoare de mașini spun că adevărata promisiune a mașinilor autonome este potențialul de a reduce dramatic emisiile de CO2. Alte avantaje ar putea fi: reducerea congestiei traficului (cu 30% mai puține vehicule pe șosea, spun unele studii), reducerea costurilor de transport cu 40% (în ceea ce privește vehiculele, combustibilul și infrastructura), îmbunătățirea capacității de mers și de locuit, eliberarea parcărilor pentru alte utilizări etc.

Un avantaj enorm de mare îl reprezintă faptul că se reduce extrem de mult numărul de accidente provocate de oboseala șoferului. O mașină autonomă neavând nevoie de șofer, "ea nu va obosi". Tot pe această zonă al riscurilor provocate de factorul uman îl reprezintă și reducerea la 0 al numărului de accidente provocate de șoferi care se află sub influența alcoolului sau al substanțelor interzise.

Mașinile autonome prin sistemele încorporate în ele își mențin constant atenția asupra traficului din jur și asupra a tot ceea ce se întâmplă în afara autoturismului. În contrast cu omul, care de exemplu poate fi distras de telefonul mobil, un robot sau o mașină autonomă în cazul de față, va fi permanent atentă și nu va fi distrasă de nimic. În plus, ele vor respecta regulile de circulație încorporate în bazele de date ale sistemelor software implementate de le dețin.

Alte avantaje extrem de importante este economia de combustibil și îmbunătățirea calității aerului. Începând de câțiva ani încoace, marii producători auto au dat startul fabricării de mașini electrice. Chiar dacă acestea au acum doar câteva sute de kilometri autonomie și cele economice (la preț accesibil) nu sunt la fel de puternice precum cele tradiționale, ele cu timpul vor avansa și vor revoluția lumea. Societatea de azi tinde să treacă pe combustibili alternativi, deoarece emisiile de carbon generate de mașinile tradiționale dăunează planetei și de asta producătorii auto se întrec în oferte cât mai accesibile, iar statele oferă potențialilor cumpărători de mașini electrice vouchere sau discount-uri la achiziționarea uneia.

1.3.2. Dezavantaje

Chiar dacă mașinile autonome prezintă numeroase avantaje, ele oferă din păcate și dezavantaje. În primul rând, nu va mai fi nevoie de atât de mulți șoferi de autobuze sau taxi-uri. Cum s-a precizat la sub-capitolul anterior, la nivelul 4 de conducere autonomă, vor exista autobuze și taxiuri fără

șoferi, capabile să transporte pasageri din punctul A în punctul B fără intervenția unui șofer. Deci, cu alte cuvinte, vor exista din ce în ce mai puține job-uri de acest fel.

Un alt dezavantaj îl constă faptul că oamenii vor înceta să mai conducă manual, deci vor fi din ce în ce mai puțini oameni care se vor apuca de școala de șoferi sau care vor avea dorința să învețe să conducă.

Mașinile autonome sau care au sisteme inteligente în componența lor, au un preț destul de ridicat, iar mulți oameni nu își vor permite achiziționarea uneia din ele. Acest fapt prezintă un dezavantaj, dar cu timpul, o dată cu evoluția tehnologiei și prin tranziția treptată de la mașinile cu combustibil tradițional (benzină sau motorină) la una electrică, prețurile vor începe să fie din ce în ce mai accesibile. Se vor oferi vouchere din partea țărilor, pentru a promova combustibil ecologic (fiind vorba aici de mașinile electrice) în detrimentul celor pe combustibil tradițional sau diferite oferte de la marii producători de autovehicule. Chiar și așa, mentenanța unor astfel de mașini va fi una destul de complicată pentru că vehiculele vor consta mai mult din componente electrice și sisteme computerizate, senzori, radare și nu numai și nu din componente mecanice.

Pentru mai multe avantaje și dezavantaje ale conducerii autonome se poate urmări figura de mai jos.

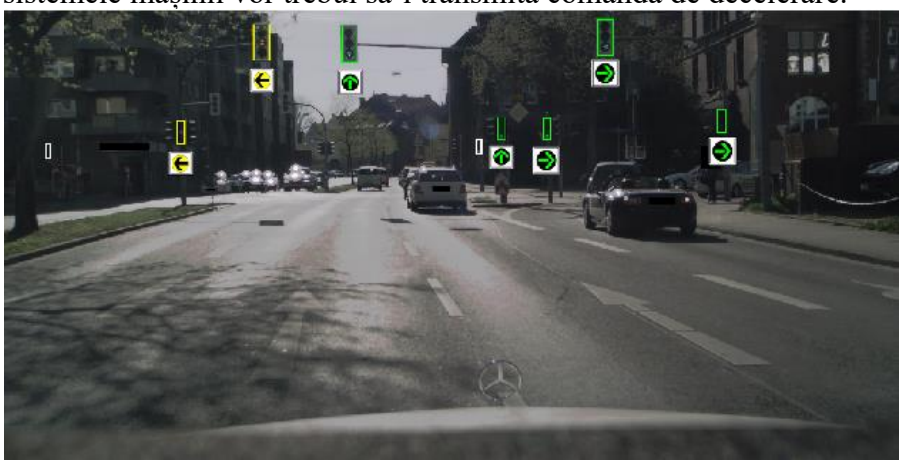
Capitolul 2. Obiectivele proiectului

Recunoașterea semaforului este o tehnologie elementară, indispensabilă pentru conducerea autonomă atât în zonele urbane cât și în cele rurale. Marii antreprenori și producători de autovehicule vin cu diferite soluții care mai de care mai eficiente și unicate prin care se acoperă și această ramură a conducerii autonome.

Mașinile autonome trebuie să fie capabile să recunoască semafoarele din trafic, dar și starea lor actuală (culoarea roșie, verde, galbenă sau neiluminat – semafor nefuncțional). Astfel, ele vor fi capabile să treacă prin intersecții semaforizate și să împartă traficul cu ceilalți șoferi umani (sau celelalte mașini autonome). Din punct de vedere al ființei umane, de cele mai multe ori, un șofer poate identifica cu ușurință semafoarele din trafic (asta în cazul în care acesta nu suferă de boli care îl fac să nu poată distinge culorile relevante). Pentru a rezolva această problemă în cazul mașinilor autonome, cele mai multe soluții integrează în sistemele mașinii diferite hărți și pattern-uri ale semafoarelor. Soluții mai complexe ar consta în tehnici și algoritmi de învățare a semafoarelor și a pozițiilor unde se pot regăsi cel mai probabil.

2.1. Cerința și obiectivele proiectului

Se dă următorul scenariu. Avem o mașină autonomă, care, fiind în deplasare prin intersecții semaforizate va trebui să fie în stare să recunoască semafoarele întâlnite, dar și stadiul lor actual (adică culoarea afișată). Nefiind implementat un astfel de sistem pe această mașină, el va trebui să facă detecție pe aceste obiecte și în cazul în care, de exemplu, un semafor detectat, are culoarea roșie, sistemul va trebui să transmită mașinii comanda de oprire. Dacă semaforul are culoarea verde și mașina se află în deplasare, ea va trebui să-și continue mersul. În cazul în care semaforul era pe culoarea roșie și mașina era oprită, ea va trebui să înceapă să accelereze. La culoarea galbenă a semaforului, sistemele mașinii vor trebui să-i transmită comanda de decelerare.



[Figura 2.1. Sistem care detectează semafoare, implementat pe o mașină]

Un astfel de sistem inteligent implementat pe o mașină ar avea ca obiective următoarele:

- Atenția asupra persoanelor care suferă de **daltonism** sau care au **deficiențe de vedere**. Legislația în vigoare este destul de restrictivă când vine vorba de astfel de persoane. O mașină inteligentă care ar beneficia de un astfel de sistem de recunoaștere a semafoarelor din trafic

precum și a culorilor lor va putea permite oamenilor aflați în această situație să se simtă șoferi și să prindă încredere în ei. Astfel, vom vedea în trafic din ce în ce mai multe persoane care suferă de aceste probleme fără a pune traficul sau viețile celorlalți participanți la el în pericol.

- **Condițiile nefavorabile de vreme/trafic.** De multe ori, șoferii pot întâmpina dificultăți în a distinge culorile de la semafoare din cauza următoarelor motive: becuri care funcționează la capacitate maximă (intensitatea luminii este redusă), sticla de apără becul semaforului este acoperită de praf, sau este zgâriată, ori din cauza vremii nefavorabile luminile generate de semafor (roșu, galben, verde) nu se pot distinge corect. Un astfel de sistem de recunoaștere a semafoarelor va trebui să rezolve astfel de impedimente și să detecteze cu ușurință culorile specifice semafoarelor.
- **Trecerile de pietoni.** Unii șoferi, din neatenție, lipsa experienței sau din alte motive ilegale (consumul de alcool sau de substanțe interzise) nu opresc la trecerile de pietoni semaforizate și persoanele aflate în trecere sunt accidente, poate chiar omorâte. Mașinile autonome ce dețin sisteme inteligente de recunoaștere a semafoarelor, în combinație cu sisteme de recunoaștere a trecerilor de pietoni și al obstacolelor vor putea preveni accidentele de acest tip.
- **Barierele de cale ferată.** Aplicațiile software implementate pe astfel de mașini autonome ar trebui să detecteze și barierele de la nivelul unei căi ferate, mai exact sistemele semaforizate ale lor. Dacă culoarea unui astfel de semafor este roșie și bariera este lăsată, mașina în cauză va trebui să se oprească.
- **Raza de detecție a semafoarelor.** Ochiul uman poate observa un semafor chiar și de la câteva sute de metri depărtare. Scopul unor astfel de sisteme de detecție a semafoarelor este și de a recunoaște forma unui semafor de la o distanță considerabilă, astfel încât, mașina autonomă să poată opri în condiții de siguranță.
- **Distincție între semafoare.** Avem următorul scenariu: o intersecție semaforizată, unde, în fața autovehiculului se află două semafoare, primul pentru direcția înainte, iar al doilea pentru schimbarea sensului de mers spre stânga. Obiectivul principal al unui astfel de sistem inteligent este ca, în astfel de situații, mașina să poată distinge între semafoarele care se din fața ei în funcție de banda pe care se află aceasta. Dacă de exemplu, în cazul prezentat, mașina se află pe banda cea mai din stânga, sistemul să urmărească și să facă ca autovehiculul să ia decizii în funcție de culoarea semaforului aferent benzii pe care se află (aici, banda din stânga – semaforul pentru schimbarea direcției de mers spre stânga). Dacă, mașina se află pe banda ce îi permite să se deplaseze doar înainte, să ia decizii doar de pe urma semaforului aferent acestei benzi.

Pentru aplicația descrisă în acest document există câteva alternative, diferite companii producătoare de mașini având ingineri specializați, lucrează de ani de zile la soluții pentru crearea unor astfel de aplicații software. Acest proiect își propune să ofere o soluție pentru recunoașterea semafoarelor din trafic în timp real. La nivelul la care va fi implementată, această soluție nu va putea fi instalată pe o mașină. Pentru acest lucru, sunt necesare componente hardware sau senzori care să permită funcționarea unor astfel de aplicații software. În plus, ea nu va putea executa o bună parte din acțiunile prezentate mai sus, ea fiind o alternativă făcută la nivel de facultate. Acest proiect face parte dintr-o serie de aplicații sau sisteme software care pot fi încorporate pe mașinile autonome, cum ar fi: detecția marcajelor longitudinale, navigația automată între diferite locații

date, schimbarea direcției de mers, parcarea mașinii în regim autonom etc. Proiectul, la nivelul la care va fi făcut, va putea să facă următoarele:

- Dintr-un set de imagini să prezică locațiile în care se găsesc semafoarele (dacă există). Aceste preziceri vor fi evidențiate pe imaginile rezultate prin încadrările semafoarelor.
- Să prezică statusul la care se găsesc semafoarele detectate la punctul anterior (să precizeze și culoarea lor: roșu, galben sau verde).

Tema proiectului	Detecția semafoarelor și a culorii lor în timp real.
Pe cine afectează	Afectează autovehiculele care nu au sisteme de detecție a semafoarelor și modul lor de a se conduce.
Impactul unui astfel de sistem	Impactul este că șoferii pot sta "mai liniștiți" în traficul urban, autovehiculele în care se află putând să recunoască semafoarele din trafic, chiar și din intersecțiile cu un nivel de dificultate ridicat.
O soluție de succes care ar putea rezolva această problemă	Soluția acestei probleme este realizarea unei aplicații, bazate pe învățare automată și folosind algoritmi eficienți care să permită recunoașterea semafoarelor din traficul urban.

[Tabel 2.1 – Sumar al temei alese și obiectivelor ei]

Tabelul de mai sus rezumă principalele teme de discuție ale acestui capitol. Se prezintă tema proiectului, pe cine afectează un astfel de sistem inteligent, impactul asupra societății și asupra domeniului de conducere autonomă și pe scurt ce are la bază soluția propusă.

Capitolul 3. Studiu bibliografic

În acest capitol se va documenta studiul individual făcut care a dus la realizarea acestui proiect, precum și legăturile făcute între anumite materii studiate în cadrul Facultății de Automatică și Calculatoare și domeniul de față. Referințele bibliografice vor fi înșirate toate la capitolul aferent lor, aici doar făcându-se referire la ele.

3.1. Studiu individual asupra domeniului conducerii autonome

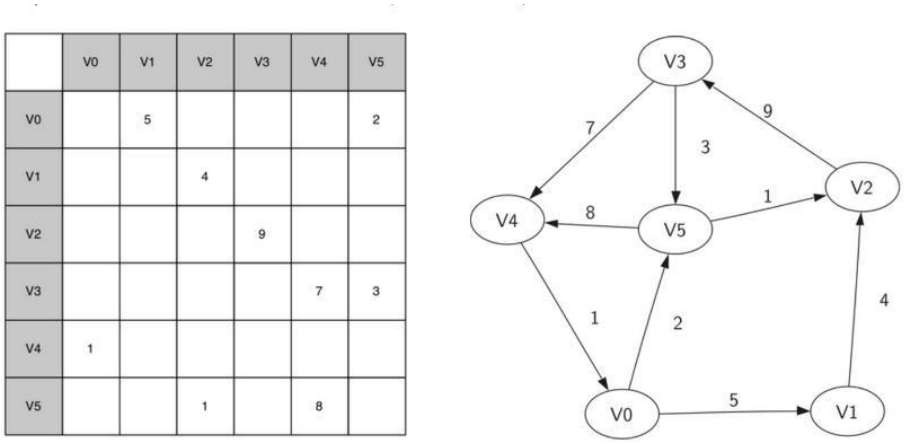
Cel mai știut exemplu de inteligență artificială este dat de mașina Turing. În [8] se descrie cum a fost inventat și ce presupune un astfel de sistem inteligent. Această mașină, pe baza unor reguli bine definite interpreta simboluri. Un sistem inteligent asemănător, [9], care la rândul lui folosește modele matematice abstracte este reprezentat de mașina Enigma, mașină folosită în al-Doilea Război Mondial de către Germania Nazistă pentru transmiterea de mesaje codificate. Aceste două mașini, sisteme inteligente, au pus bazele a tot ce înseamnă astăzi: Inteligența Artificială, Învățarea Automată și conducerea autonomă, cele trei mari teme abordate de această aplicație prezentată în acest document și implementată de autor.

În cadrul studiilor de licență la Universitatea Tehnică din Cluj-Napoca, la Facultatea de Automatică și Calculatoare, secția de Calculatoare și Tehnologia Informației s-au prezentat în diferite materii bazele și principiile acestor teme de interes. La materia Inteligență Artificială, studiată în anul 3 de facultate, autorul cursului a descris pentru studenții înrolați la acesta diferiți algoritmi care ajută la crearea unor sisteme inteligente, precum și modul în care funcționează aceste aplicații. Conform cursului și al bibliografiei specializate, [10]-[11], inteligența artificială este considerată o tehnologie a viitorului în care mașinile au capacitatea de a imita funcții umane, cum ar fi: învățarea, raționamentul, planificarea și creativitatea. Aceste mașini au sisteme ce percep mediul în care funcționează, pot prelucra datele colectate din jurul lor și să rezolve probleme pentru a atinge un anumit obiectiv setat și definit de către inventatorul lor. Mașinile/aplicațiile ce folosesc IA (inteligența artificială) sunt capabile să își adapteze comportamentul, în funcție de acțiunile anterioare, deci funcționează autonom. Legat de domeniul de interes, conducerea autonomă, se poate lua aici ca și exemplu o mașină de taxi autonomă. [10] Utilizatorii acestei mașini vor fi interesați de următoarele:

1. **Performanța** – aici se pot aminti următoarele metrici: siguranța, destinația, profitul, legalitatea și confortul.
2. **Mediul de utilizare** – în funcție de factorii de risc și de funcționare a acestei mașini avem: mediul urban, autostrada, interacțiunea cu alte mașini/pietoni, impactul vremii asupra funcționării unui astfel de autovehicul.
3. **Efectori** – aceștia vor reprezenta elementele cu care pasagerii vor lua contact sau prin care mașina autonomă va funcționa (elementele mecanice). Aici, se pot menționa: volanul, accelerația, frâna, claxonul, vorbitorul/afișajul ecranului care îl va notifica pe pasager dacă a ajuns sau nu la destinație.
4. **Senzori** – reprezintă elementele, sistemele prin care o astfel de mașină funcționează. Aici, avem următoarele: video, accelerometru, senzorii de la motor, senzorul de combustibil, GPS, senzorii de mișcare, etc.

Acest curs prezentat în facultate descrie și anumite modalități prin care astfel de sisteme inteligente funcționează și importanța structurilor de date abstracte și a algoritmilor aplicați pe ele.

Aici, se pot aminti grafurile și metodele de parcurgere ale lor. Această abstractizare în forma structurilor de date amintite a luat naștere de la modul în care este privită lumea prin ochii unui sistem inteligent, și anume printr-un set de stări abstracte care succed o serie de acțiuni din care vor rezulta o mulțime de soluții posibile.



	V0	V1	V2	V3	V4	V5
V0		5				2
V1			4			
V2				9		
V3					7	3
V4	1					
V5			1		8	

[Figura 3.1 – Un graf orientat care are pe muchii și un set de costuri]

În figura de mai sus, este prezentat un graf orientat și o listă de costuri. Echivalentul acestei structuri de date abstracte în viața reală este un traseu format din mai multe orașe conectate între ele prin drumuri. Fiecare drum are costul lui (adică numărul de kilometri și costul rezultat din punct de vedere al combustibilului) și de exemplu, un șofer, va trebui să aleagă drumul cu costul minim pentru a ajunge din orașul X în orașul Y, pe traseele disponibile și știute de el. Sistemele GPS implementate pe mașini și dezvoltate în ultimii ani de inginerii software dețin diferiți algoritmi și structuri de date abstracte cum ar fi arbori sau grafuri propriu-zise. În cadrul cursului de Inteligență Artificială, s-au prezentat diferite moduri de parcurgere a grafurilor, cum ar fi: parcurgerea în adâncime, parcurgerea în lățime, căutarea pe arbori, căutarea în funcție de anumite costuri, algoritmul A* etc.

Un alt punct sensibil ce trebuie atins și prin care multe sisteme de conducere autonomă sunt formate este domeniul rețelelor neurale (“neural networks” în engleză). Aceste rețele neurale reprezintă o ramură din domeniul Inteligenței Artificiale și caracterizează ansambluri de elemente de procesare interconectate prin care se interacționează cu mediul înconjurător, având sisteme de funcționare asemănătoare cu cele ale creierului uman, deoarece prezintă capacitatea de a învăța modele. Pentru această parte, s-au făcut cercetări și documentări pe baza articolului [12]. În cursul [13], de Sisteme Inteligente, prezentat în cadrul facultății s-au descris diferitele modalități de calcul al probabilităților ce ajută în formarea rețelelor neurale. În cadrul acestui proiect s-au folosit rețele neurale și pentru că țin de sub-domeniul de Învățare Automată se va aminti pe scurt în această documentație și despre rețelele neurale Bayesiane. Din [13] și [14] extragem ideea că aceste rețele reprezintă un model grafic probabilistic pentru reprezentarea unei baze de cunoștințe dintr-un domeniu incert. Cum rețelele neurale sunt reprezentate prin noduri, aici, aceste noduri corespund fiecare unei variabile aleatorii și muchiile din graful rețelei reprezintă probabilitatea condiționată pentru acele variabile aleatorii. Pentru mai multe detalii despre modul de funcționare și structura acestor rețele neurale se recomandă studiul materialelor, articolelor și cărților de specialitate.

Recunoașterea modelelor (pattern recognition) este o recunoaștere automată a modelelor și a regularităților în datele de se procesează de sistemele inteligente și computerizate. Acest capitol, cu referire la [15], este foarte important în domeniul inteligenței artificiale și al conducerii autonome și are o multitudine de aplicații în viața reală. Aici, amintim: analizarea datelor, procesarea de semnale/imagini, grafica computerizată, învățarea automată (învățare automată), etc. De cele mai multe ori, și se va vedea acest lucru și în capitolele următoare când se va intra mai în detaliu la modul de implementare al proiectului, sistemele de recunoaștere a modelelor (formelor) necesită date de antrenare, iar din modelul obținut în urma antrenării se vor putea prezice anumite rezultate pe seturi de date neștiute de sisteme. Se va intra în detaliu pe acest subiect în capitolele următoare. Din cursul [16], prezentat în cadrul studiilor de licență, se poate deduce că studiul recunoașterii modelelor reprezintă pentru mașinile și sistemele inteligente următoarele puncte:

- Modul prin care ele observă mediul înconjurător.
- Modalitatea și capacitatea de a învăța diferite modele în urma vizualizării unor zone/puncte de interes dintr-un set de date.
- Capacitatea de a lua decizii pe baza cunoștințelor acumulate în urma algoritmilor de învățare.

În domeniul conducerii autonome, se va auzi de multe ori de termenul de clasificare. Acesta reprezintă procedeul prin obiectele sunt împărțite pe categorii, după ce acestea au fost recunoscute. În cadrul cursului [16] sunt prezentate diverse metode de clasificare, în funcție de performanță și de tipul de obiecte pe care pot fi folosite. Acești algoritmi sunt: AdaBoost, K-NN, Naive Bayes, etc. Fiecare dintre aceste metode va fi detaliată în capitolul următor.

3.2. Aplicații similare dezvoltate anterior

În continuare se vor prezenta două aplicații similare dezvoltate de doi mari producători de autovehicule.

Pentru început se poate aminti despre sistemul de recunoaștere a semafoarelor realizat de grupul BMW și descris în [16]. Sistemul de la firma producătoare de mașini recunoaște semafoarele din traficul din oraș când acesta este pornit și ei promit că va fi disponibil pe cât mai multe modele de mașini. Acest sistem se bazează pe legătura dintre banda pe care este încadrat autovehiculul și semafoarele aferente. În cazul acesta, semafoarele sunt înregistrate/memorate automat de sistemul de control al vitezei și al distanței. Dacă semaforul este roșu, sistemul decelerează vehiculul până la oprire. O lumină verde la semafor va da semnal sistemelor inteligente încorporate în autovehicul pentru a continua călătoria ori pentru a accelera. Într-o situație de intersecție mai complexă în care, de exemplu, diferite semnale luminoase se aplică pe benzi diferite, un simbol semafor este afișat în ecranul bordului, iar șoferul va trebui să ia o decizie. Acesta din urmă poate apoi să confirme semnalul luminos afișat și să continue să utilizeze controlul automat al vitezei și distanței sau să ia el controlul mașinii complet. Pentru un control suplimentar, sistemul dispune și de o funcție de amintire.

Un alt sistem inteligent asemănător este prezentat în [17] și a fost dezvoltat de producătorul de mașini autonome Tesla. Aici, aceste vehicule dețin un sistem care folosește camere orientate spre înainte, sisteme GPS și hărți de date pentru a detecta semafoarele și semnalele de oprire. Pe măsură ce mașina se apropie de o intersecție cu pilotul automat activat, șoferul este informat de existența unui semafor (în caz că există) și indiferent dacă culoarea este verde, galbenă sau roșie,

mașina va încetini până la oprirea completă. O linie roșie afișată pe ecranul tactil, arată unde mașina (de fapt pilotul automat) a stabilit că trebuie să se oprească (se oprește numai dacă se afișează o linie roșie). Dacă lumina care se apropie este verde și șoferul crede că poate continua deplasarea, el poate anula decelerarea apăsând în jos pedala de accelerație sau pe maneta de viteză. Dacă lumina devine verde după o oprire completă (înseamnă că înainte a fost roșie), șoferul poate să facă același lucru pentru a da mașinii permisiunea de a continua. La figura 3.2, de mai jos, se poate observa cum recunoaște o mașină Tesla semafoarele din trafic.

[Figura 3.2 – Sistemul de recunoaștere a semafoarelor de la Tesla]

În timp ce în forma sa actuală, detectarea intersecțiilor sau a semafoarelor poate părea puțin ciudată, este concepută o idee ceva mai utilă pe termen scurt: învățare automată, sau învățarea automată. Pentru a antrena rețeaua neuronală, Tesla are nevoie de un număr mare de exemple din lumea reală și de feedback despre modul în care oamenii au reacționat la ele.

Capitolul 4. Analiză și fundamentare teoretică

În capitolul ce urmează se vor descrie încă o dată, pe scurt, care sunt principalele obiective și cerințe ale acestui proiect. În plus, se vor detalia toate noțiunile și protocoalele utilizate pentru ca cititorii acestui articol să poată înțelege mai bine funcționalitatea acestei aplicații.

4.1. Scopul și cerințele proiectului

Scopul proiectului a fost prezentat și la capitolul 2) din această documentație și el este realizarea unei aplicații ce va fi capabilă să identifice semafoarele din trafic precum și starea lor actuală (culoarea roșie, galbenă sau verde). S-a ajuns la conceperea unui astfel de proiect deoarece în domeniul conducerii autonome, este necesar ca autovehiculele să poată recunoaște semafoarele din trafic. Alte proiecte cu tematică asemănătoare ce se aplică tot mașinilor autonome sunt: detecția marcajelor longitudinale, detecția pietonilor, recunoașterea semnelor de circulație, păstrarea benzii de circulație autonom, etc.

Un proiect cu această temă, dar cu implementare la nivel de industrie (adică rezultatul software să poată fi pus pe autovehicule) va avea ca cerințe următoarele puncte importante:

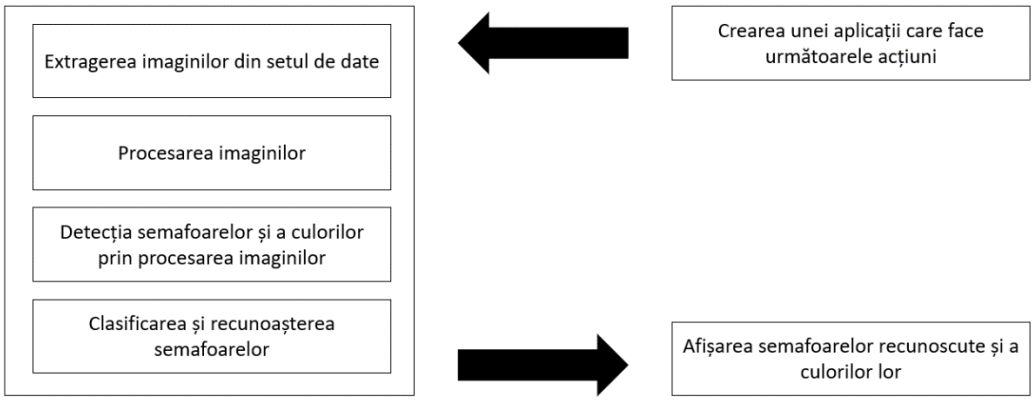
- Recunoașterea semafoarelor de pe banda aferentă pe care se află autovehiculul și respectarea cu strictețe a culorii afișate de acestea:
    - Dacă se detectează culoarea roșie se va opri în siguranță mașina
    - Dacă se detectează culoarea galbenă se va reduce viteza în cazul în care culoarea de dinainte era verde sau se va aștepta culoarea verde în cazul în care era înainte culoarea roșie.
    - Dacă se detectează culoarea verde se va începe în siguranță deplasarea autovehiculului.
- Detecția semafoarelor în condiții nefavorabile de vreme.
- Atenție sporită la culorile afișate de acestea (roșu, galben și verde) pentru a putea fi detectate cu precizie cât mai mare (între 99%-100%). S-a luat în calcul aici, dificultatea persoanelor cu deficiență de vedere sau care suferă de daltonism de a recunoaște culorile semafoarelor.
- Detecția semafoarelor de la barierele de cale ferată.
- Recunoașterea semafoarelor de la o distanță cât mai mare.
- Distincția dintre semafoarele din trafic.

Proiectul de față își propune să recunoască semafoarele din trafic, mai exact din imagini extrase din traficul urban. Se va urmări doar recunoașterea lor și a culorii afișate de ele.

4.2. Algoritmul propus pentru realizarea proiectului

Întreaga aplicație este un sistem software bazat pe Învățare Automată (se va detalia într-un sub-capitol următor ce reprezintă acesta). Realizată în limbajul de programare Python, ea va putea fi testate prin apăsarea unui fișier executabil. Pentru realizarea acestei aplicații inteligente, s-a folosit un set de date destul de mare (secvențe din trafic și locații exacte ale semafoarelor din aceste secvențe). După prelucrarea imaginilor din setul de date s-au folosit diferiți algoritmi de clasificare din care au rezultat predicții ale semafoarelor, iar cu ajutorul acestora din urmă s-au detectat anumite semafoare pe un set de imagini de test. Rezultatele finale sunt reprezentate de imagini,

în care se află semafoarele cu culorile afișate de ele, încadrate în cercuri. Pentru o mai bună urmărire a funcționalității aplicației, se poate urmări figura de mai jos:



[Figura 4.1 – Analiza aplicației]

Diagrama de mai sus conține următoarele:

- **Extragerea imaginilor din setul de date**: pentru a putea face o aplicație software care să facă detecție pe semafoarele din trafic trebuie mai întâi să avem un set de date de imagini de antrenare, bine definit, pentru partea de Învățare Automată, deoarece algoritmul principal se va baza pe învățarea unor rețele neuronale sau a unor algoritmi de clasificare.
- **Procesarea imaginilor**: va trebui să se proceseze imaginile pentru a se putea extrage o serie de trăsături. Aceste trăsături vor fi relevante pentru învățarea algoritmilor de clasificare. Se vor detalia în următoarele sub-capitole trăsăturile alese
- **Detecția semafoarelor și a culorilor prin procesarea imaginilor**: Algoritmul propus va primi poze și secvențe din trafic, deci va trebui să se implementeze ceva algoritmi ajutători prin care să se poată extrage semafoarele din aceste imagini, ori dacă nu, să se evidențieze semafoarele, astfel încât să se poată pregăti partea de clasificare și antrenare. Ca și la pasul anterior, se vor extrage trăsături din imagini.
- **Clasificarea și recunoașterea semafoarelor**: După extragerea tuturor trăsăturilor (ale semafoarelor și ale culorilor aferente lor) se vor putea pregăti și antrena diferiți algoritmi de clasificare. La fel, se vor detalia acești pași în sub-capitolele, eventual în capitolul următor în care se va descrie implementarea propriu-zisă.
- **Afișarea semafoarelor recunoscute și a culorilor lor**: Aplicația software implementată va furniza la final date ale predicțiile obținute, precum și imagini rezultate în urma algoritmilor utilizați, iar semafoarele cu culorile lor aferente vor fi încadrate în cercuri sau chenare dreptunghice.

Pentru a putea testa aplicația, un utilizator va trebui să-și descarce mediul de dezvoltare software PyCharm, pentru a putea compila programul și a crea eventual un executabil. El va trebui să-i dea ca și input la aplicație un set de imagini, iar aceasta din urmă va recunoaște obiectele de interes din ele. Pentru a vedea rezultatele aplicației, utilizatorul va trebui totuși să aleagă imagini care să conțină semafoare și să fie cât de cât vizibile.

4.3. Setul de imagini de antrenare și de test

Un set de date (în engleză, dataset) este în principiu o colecție de date. Aceste seturi de date se folosesc în diferite domenii, dintre cele mai importante amintim: analiza datelor și inteligență artificială. Dacă ne gândim de exemplu la datele descrise sub formă de tabele, un set de date corespunde uneia sau mai multor tabele din baza de date țintită. Aici, fiecare coloană a unui tabel reprezintă o anumită variabilă, iar fiecare rând corespunde unui cumul (sau înregistrare) de date a setului de date curent. De la problema la problemă, aceste colecții de date descriu valori pentru fiecare dintre variabile. De exemplu, putem avea un set de date care să reprezinte analiza unei mulțimi de oameni, din punct de vedere al înălțimii, greutății sau al vârstei. Aici, fiecare coloană va conține valori pentru aceste variabile, iar fiecare rând va reprezenta o înregistrare pentru un anumit individ, sau un set de caracteristici pentru individul în cauză.

Seturile de date, în funcție de problemă, pot fi reprezentate și dintr-o colecție de documente sau fișiere. În acest proiect se vor folosi seturi de date sub formă de fișiere tabelare, cu extensia .csv. Valorile din astfel de colecții de date sunt în principiu numere, cum ar fi numere reale sau numere întregi (se poate da exemplu aici, înălțimea sau greutatea unei persoane reprezentate în centimetri, respectiv în kilograme, dar aici la ultima cu valoare flotantă). Pot fi totuși și date nominale (adică, valori non numerice), aici putând fi dat ca exemplu etnia unei persoane. Deci, cu alte cuvinte, valorile pot fi de oricare dintre tipurile descrise ca un nivel de măsurare. Totuși, pentru fiecare variabilă în parte, adică pentru fiecare coloană în cazul în care se alege colecția de date sub formă de tabel, valorile sunt toate de același tip. [18]

Pentru domeniul de Învățare Automată, care va fi descris mai târziu într-un alt sub-capitol, colecțiile de date se vor împărți în două: **date de antrenare** și **date de test**. De ce se întâmplă acest lucru? În principiu, când auzim de Învățare Automată ne gândim la procesul de învățare automată al mașinilor sau algoritmi prin care sistemele computerizate ajung să poată să reproducă anumite modele automat. Pentru acest lucru, la acești algoritmi se vor da colecții de date de antrenare, adică foarte multe date cu ajutorul cărora ei vor putea să învețe să reproducă date. Să dăm un exemplu: vrem să construim un sistem prin care să se poată face distincție între două animale: un câine și o pisică. Pentru început, i se vor da aplicației foarte multe date de antrenare (în principiu aceste date vor fi reprezentate de poze) care să reprezinte pe rând câinele și după pisica, iar prin învățare ea va crea anumite modele, adică va ști prin ce se diferențiază un câine de o pisică. Astfel se va ajunge la o altă noțiune ce va fi detaliată într-un sub-capitol următor și anume clasificarea. După această clasificare sistemul computerizat va avea creat un model, cu ajutorul datelor primite, iar acest model va fi folosit în ultima parte și anume testarea. Aici, pe un set de date de test (și aici se presupune că va fi constituit tot din imagini), se va testa modelul creat de aplicație, iar fără a se specifica exact ce animal constituie acel set de date de test, aplicația va trebui să prezică animalul în cauză, dacă este câine sau pisică. Acesta a fost doar un exemplu din miile existente care pot fi date pentru a descrie datele de antrenare și datele de test. Rezumând pe scurt cam ce s-a vrut prin acest exemplu dat, datele de antrenare sunt cele care se vor da sistemului computerizat pentru a forma un model cu ajutorul căruia să se clasifice obiectele în cauză, iar datele de test sunt cele pe care se aplică modelul creat anterior și se verifică dacă rezultatele sunt cele așteptate.

#	Sex	Age	Weight (kg)	Height (m)	Smoker	Religion	Social Class
1	F	32	94.87	1.72	Y	Christian	C
2	F	34	99.39	1.63	N	Christian	D
3	F	33	124.15	1.66	N	Hindu	C
4	M	52	49.77	1.71	Y	Christian	E
5	F	57	65.13	1.80	N	Hindu	C
6	F	39	58.71	1.74	N	Buddhist	E
7	F	39	67.41	1.56	N	Christian	C
8	F	47	67.19	1.79	Y	Christian	B
9	M	58	42.95	1.48	N	Christian	A
10	M	17	109.52	1.62	N	Christian	C
11	F	42	91.12	1.76	N	Buddhist	D
12	F	48	58.07	1.50	N	Islamist	D
13	M	43	46.69	1.61	N	Hindu	B
14	M	55	85.38	1.54	N	Islamist	C
15	M	34	39.77	1.70	N	Christian	B
16	M	34	83.90	1.74	N	Islamist	D
17	M	51	55.72	1.93	Y	Islamist	B
18	F	47	57.10	1.51	N	Christian	C
19	M	38	54.01	1.85	Y	Islamist	C
20	M	45	73.10	1.59	N	Islamist	C

[Figura 4.2 – Exemplu de dataset]

În figura de mai sus, avem un set de date care cel mai probabil a fost folosit pentru antrenarea algoritmilor de Învățare Automată. Din ce se poate observa, avem 20 de rânduri, adică înregistrări diferite pentru o serie de 20 de indivizi. Acești indivizi au fost caracterizați fiecare prin următoarele coloane: sex, vârstă, greutate, înălțime, fumător/nefumător, religie și clasă socială.

Pentru proiectul de față s-au folosit mai multe seturi de date de antrenare, încercându-se numeroase variante pentru a avea rezultate cât mai bune. Se vor detalia aceste colecții de date în capitolul următor.

4.4. Procesarea imaginilor

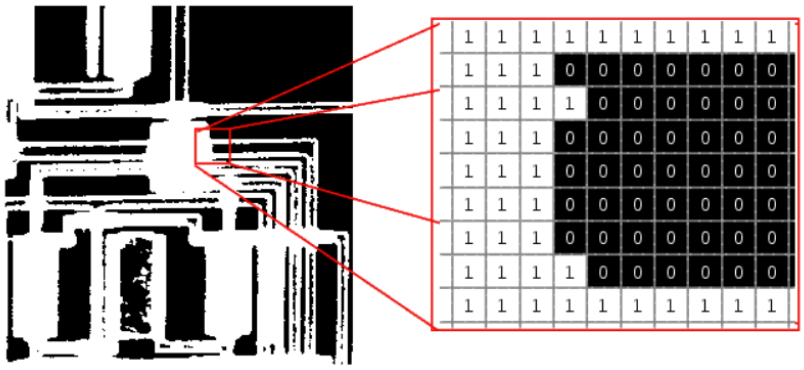
Procesarea imaginii (în engleză, image processing) este modalitatea prin care se efectuează anumite operații pe o imagine. Scopul acestor prelucrări (procesări) este de a obține o imagine îmbunătățită sau pentru a extrage unele informații utile din aceasta. Astăzi, procesarea imaginilor, se numără printre domeniile din sectorul IT care beneficiază de o creștere rapidă, constantă, devenind din ce în ce mai populară.

După cum am precizat mai sus, procesarea imaginilor se ocupă cu studiul proprietăților imaginilor și cu transformarea acestora. La un capitol anterior, amintisem de viziunea artificială, o altă ramură importantă al domeniului de Inteligență Articială. Majoritatea algoritmilor de viziune artificială necesită folosirea unor algoritmi de procesare a imaginilor mai întâi. Mai jos, se vor specifica câteva metode prin care procesarea de imagini ajută în diferite situații:

- **Îmbunătățirea calității imaginilor** – prin transformarea imaginilor se vor pune în evidență anumite detalii ascunse/obscure, sau trăsături de interes pentru cel ce va viziona imaginile.
- **Compresia imaginilor** – reprezintă descrierea compactă a imaginilor/secvențelor de imagini (aici, mă refer la secvențele dintr-un video, de exemplu) pentru transmisia/stocarea către, respectiv pe anumite servere/dispozitive de stocare etc.
- **Restaurarea imaginilor** – prin restaurare se înțelege eliminarea elementelor de degradare cunoscute/modelabile.

- **Extragerea de trăsături din imagini** – ce se va folosi și în acest proiect, reprezintă localizarea anumitor puncte de interes și extragerea detaliilor (muchii, colțuri, structuri complexe, etc) care pot ajuta în proiecte de Învățare Automată sau Viziune Artificială, ca de exemplu.

În acest document se vor face referiri de multe ori la imagini, iar pentru început se consideră ca fiind important, a se ști cum va fi privită o imagine de către procesor. Aceasta va fi, după cum urmează, reprezentată ca o matrice:



[Figura 4.3 – Reprezentarea în formă matriceală a unei imagini binare]

După cum se poate observa în figura de mai sus, în stânga avem o imagine alb-negru (în termeni tehnici se va numi imagine binară – se va reveni cu detalii despre acest tip de imagine într-un sub-capitol următor), iar în dreapta avem reprezentarea sub formă matriceală a imaginii. O imagine binară va avea în matricea aferentă ei doar două valori: 0 pentru negru, iar 1 pentru alb.

4.4.1. RGB și HSV

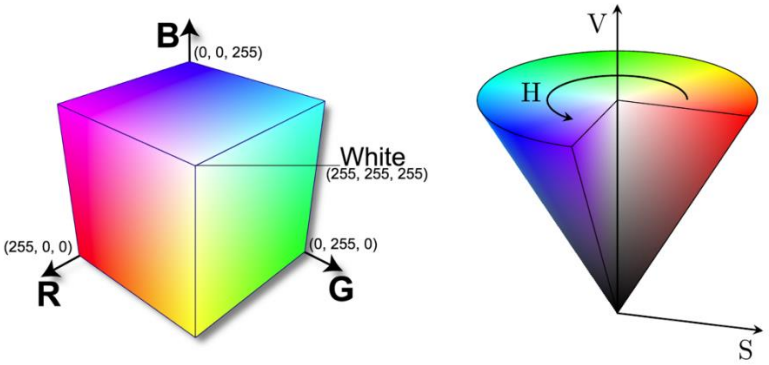
Modelul de culoare RGB (Red-Green-Blue, în traducere, Roșu-Verde-Albastru) este un model de culoare în care culorile primare roșu, verde și albastru sunt combinate în diferite moduri pentru a reproduce alte culori. După cum probabil se poate deduce, numele modelului provine de la inițialele celor trei culori primare anterior. Acest model de culoare are ca scop principal detectarea, reprezentarea și afișarea imaginilor din sistemele electronice moderne (exemplu, televizoare, computere, monitoare, telefoane, etc), chiar dacă acest model a fost și este în continuare folosit și în fotografia convențională. Modelul de culoare RGB, are ca și background (bază de unde a apărut acest concept) percepția umană a culorilor. Dispozitivele electronice moderne folosesc modele de culoare RGB dependente. Acest lucru înseamnă că dispozitivele detectează și reproduc anumite palete de culori, din modelul RGB, în funcție de viziunea fiecărui producător; asta înseamnă că nivelurile individuale de roșu, verde și albastru variază de la producător la producător sau de la dispozitiv la dispozitiv, în funcție de cum au fost programate ele să perceapă culorile.

Pentru a forma o culoare cu modelul RGB, trebuie să suprapunem cele trei fascicule de lumină (roșu, verde și albastru). Fiecare dintre cele trei componente ale acestui model poate avea o valoare a intensității arbitrară, de la complet stins până la complet aprins (se va vedea imediat ce înseamnă acest lucru). Intensitatea zero pentru fiecare componentă în parte dă cea mai închisă

culoare (se va considera culoarea neagră), iar intensitatea maximă a fiecărui canal al modelului va da un alb.

De menționat aici este că pentru culoarea albă, calitatea acesteia depinde foarte mult de natura surselor primare de lumină, dar dacă acestea sunt echilibrate corespunzător, rezultatul va fi un alb neutru, adică albul perfect. Când intensitățile pentru cele trei canale sunt la fel rezultatul este o nuanță de gri, mai închisă sau mai deschisă în funcție de intensitate. Dacă valorile intensităților diferă, rezultatul va fi o nuanță colorată. Această nuanță va fi mai mult sau mai puțin saturată, în funcție de diferența dintre cea mai puternică intensitate și cea mai slabă, ale celor trei culori primare utilizate din modelul RGB.

Deci, un pixel dintr-o imagine va fi descris prin câte o valoare pentru fiecare componentă din modelul RGB. Culoarea pixelului va reprezenta un punct în spațiul 3D al modelului, acest fapt fiind descris în figura 4.4, prin cubul general al spațiului de culori RGB. În acest cub se disting următoarele: originea axelor R, G și B – corespunde culorii negre (0, 0, 0), vârful opus originii care corespunde culorii albe (255, 255, 255), cele trei axe principale care vor reprezenta fiecare câte o culoare primară (roșu, verde și albastru) și încă trei vârfuri care vor reprezenta culorile turcoaz, mov și galben. Ca și observație, dacă schimbăm originea sistemului de coordonate în punctul alb (255, 255, 255) vom obține spațiul de culoare CMY (Turcoaz, Magenta și Galben) de la cele trei vârfuri menționate anterior.



[Figura 4.4 – Reprezentarea sub formă de cub al modelului RGB și sub formă de con al modelului HSV]

Cel mai comun model de codificare al culorilor este modelul RGB. Fiind vorba tot de codificări digitale este clar că și culorile pot fi reprezentate în funcție de numărul de biți. Intervalul de codificare, este de obicei de la **0** la $\mathbf{2^{n-1}}$, pentru a putea încadra culorile în grupări de biți. În mod normal, cele mai multe codificări sunt pe 1, 2, 4, 5, 8 și 16 biți de culoare, numărul total de biți utilizați pentru o culoare în modelul RGB numindu-se adâncimea culorii. Ca și observație, spațiul de culoare poate fi reprezentat complet pe o codificare pe 24 sau chiar 32 de biți. Modul de stocare și codificare al imaginilor, este acum mult mai puțin costisitor decât pe vremuri, de asta încercându-se idealul de a se reduce cât mai mult din dimensiunea imaginii fără să se piardă din detalii. Utilizând o combinație adecvată de intensități pentru canalele roșu, verde și albastru, pot fi afișate foarte multe culori. Mai jos, în următorul tabel, se pot observa adâncimea și tipul imaginii pentru diferite tipuri de codificări.

Adâncimea culorii	Numărul de culori
1 bit	2
4 biți	16
8 biți	256
16 biți	65536
24 biți	16777216
32 biți	16777216

[Tabelul 4.1 – Tabel pentru diferite codificări ale imaginilor]

Tot în figura 4.4 este reprezentat și modelul de culoare HSV. Acesta este un model care remapează culorile primare din modelul primar RGB, descris anterior, în dimensiuni care pot fi mai ușor înțeles nouă, oamenilor. Cu alte cuvinte, este un model asemănător cu modul prin care percepem noi oamenii culorile. În acest model de culoare avem trei mari componente:

1. **Culoarea (în engleză, hue)** – reprezintă unghiul făcut de culoarea curentă pe cercul de culoare RGB. Aici se pot da câteva exemple: dacă avem un unghi de 0 grade obținem culoarea roșie, pentru 120 de grade obținem verde, iar pentru 240 de grade avem ca rezultat albastru. Astfel obținem chiar culorile principale din modelul de culoare RGB. Legat de figura 9, canalul H reprezintă unghiul făcut de culoarea curentă cu raza corespunzătoare culorii roșii.
2. **Saturația (în engleză, saturația)** – prin saturație se controlează volumul de culoare utilizat. De exemplu, o culoare cu saturație de 100% va putea fi numită culoare pură, în schimb una cu saturație de 0% poate produce o nuanță de gri. Din punct de vedere matematic și în legătură și cu piramida/conul din figura 9, saturația reprezintă distanța culorii față de centrul figurii geometrice.

- **Valoarea (în engleză, value)** – valoarea sau intensitatea descrie controlul luminozității culorii. De exemplu, o culoarea cu 0% luminozitatea reprezintă negrul pur, iar una cu 100% luminozitate nu va avea deloc culoarea neagră în amestecul culorii curente. La fel ca la celelalte două canale, matematic vorbind, canalul V reprezintă înălțimea culorii curente în piramida/conul ce este descris în figura 9.

Desigur, este foarte important a se ști cum se poate converti dintr-un model de culoare în alt model de culoare. Au fost folosite în această aplicație software doar două modele de culoare, iar acestea două sunt cele menționate mai sus (RGB – roșu, verde, albastru și HSV – culoarea, saturația, valoarea). Conversia din modelul RGB în HSV, se realizează folosind formulele de mai jos, ele fiind descrise mai detaliat în capitolul următor:

- $\mathbf{R' = R/255}$ – unde R este valoarea canalului Red (roșu) din modelul RGB
- $\mathbf{G' = G/255}$ – unde G este valoarea canalului Green (verde) din modelul RGB
- $\mathbf{B' = B/255}$ – unde B este valoarea canalului Blue (albastru) din modelul RGB
- $\mathbf{C_{max} = max(R', G', B')}$ – aici, $\mathbf{C_{max}}$ va reprezenta maximul dintre valorile aflate în cele trei relații de mai sus
- $\mathbf{C_{min} = min(R', G', B')}$ – aici, $\mathbf{C_{min}}$ va reprezenta minimul dintre valorile aflate primele trei relații de mai sus.
- $\mathbf{\Delta = C_{max} - C_{min}}$ – aici, avem diferența dintre $\mathbf{C_{max}}$ și $\mathbf{C_{min}}$.

- $H = \begin{cases} 0°, & \Delta = 0 \\ 60° \times \left(\frac{G' - B'}{\Delta}\right) \bmod 6, & C_{max} = R' \\ 60^o \times \left(\frac{B' - R'}{\Delta} + 2\right), & C_{max} = G' \\ 60^o \times \left(\frac{R' - G'}{\Delta} + 4\right), & C_{max} = B' \end{cases}$

- $S = \begin{cases} 0, & C_{max} = 0 \\ \frac{\Delta}{C_{max}}, & C_{max} \neq 0 \end{cases}$

- $V = C_{max}$

4.4.2. Imagini binare

Imaginile binare sunt imagini în care pixelii au doar două valori posibile (alb sau negru). Numeric, aceste valori sunt 0 pentru negru și 1 sau 255 pentru culoarea alb. Fiind doar două culori reprezentate pe imagine, dintr-o paletă foarte largă posibilă (se știe că modelul RGB oferă peste 16 milioane de culori în total), este foarte ușor să se separe obiectele prin algoritmi de prelucrare a imaginilor. După cum sugerează și numele, o imagine binară va avea doar două valori, deci se va putea stoca în memorie ca o hartă de biți. Asta înseamnă în plus și capacitate de stocare redusă, majoritatea imaginilor binare comprimându-se bine, cu scheme simple de biți comprimate pe lungimea lor.

De multe ori, în domeniul procesării imaginilor se va discuta de conceptul de binarizare. Deci, ce este o binarizare? Binarizarea este procedeul prin care datele se transformă în date binare. În procesarea imaginilor, mai exact în lucrul cu imagini, când ne referim la binarizare spunem de fapt că se vor transforma (converti) imaginile din formatul curent dat, în formatul binar, adică în imagini binare. La cursul specializat de Procesare de Imagini (Image Processing) făcut în anul 3 de facultate, s-au descris numeroase aplicații și operații pe aceste tipuri de imagini binare. Se vor aminti mai jos, câteva dintre ele.

1. **Subțierea** (în engleză, thinning) este operația morfologică prin care se elimină pixelii selectați din prim-plan, din imaginile binare.
2. **Eroziunea** este tot o operație morfologică ce are a ca efect erodarea limitelor regiunilor de pixeli din prim-plan.
3. **Dilatarea** este operatorul morfologic care se ocupă cu mărirea treptată a limitelor regiunilor de pixeli din prim-planul imaginilor binare (ne vom referi aici, la pixelii cu valoarea 255 – adică pixelii albi).

4. **Deschiderea și închiderea** sunt alți doi operatori morfologici aplicați pe imaginile binare. De exemplu, deschiderea (asemănătoare cu eroziunea) îndepărtează o parte din pixelii din prim-planul imaginilor.

O imagine cu tonuri de gri este o imagine în care valoarea fiecărui pixel este reprezentată de către un singur eșantion, care la rândul lui este descris de o cantitate fixă de lumina. Asta înseamnă că fiecare eșantion deține informații despre intensitate. Imaginile în tonuri de gri (grayscale, în engleză), un fel de alb-negru sau gri monocrom, sunt compuse exclusive din nuanțe de gri. Contrastul variază de la negru (la cea mai slabă intensitate) la alb (la cea mai puternică intensitate, valoare). Aceste imagini, adică cele în tonuri de gri au multe nuanțe de gri printre pixeli, asta înseamnă că sunt diferite de imaginile binare (adică alb-negru). O nuanță de gri este acea culoare în care componentele RGB (roșu, verde și albastru) au intensitate egală. Deci, este necesar să se specifice doar o valoare unică pentru fiecare pixel în parte, spre deosebire de imaginile color în care fiecare pixel este reprezentat în model RGB prin intensități diferite pentru toate din cele trei canale disponibile. În general, intensitatea pe tonuri de gri, este stocată ca un număr întreg de 8 biți, oferind 256 de nuanțe diferite posibile de gri, pe întreg intervalul negru-alb.

În proiectul descris prin documentația de față, s-au folosit imaginile binare. Dar pentru a se ajunge la imagini binare, mai întâi au fost necesare anumite conversii. În cele ce urmează, se vor prezenta, pe rând, conversia unei imagini color într-o imagine grayscale și conversia unei imagini grayscale în imagine binară (adică alb-negru).

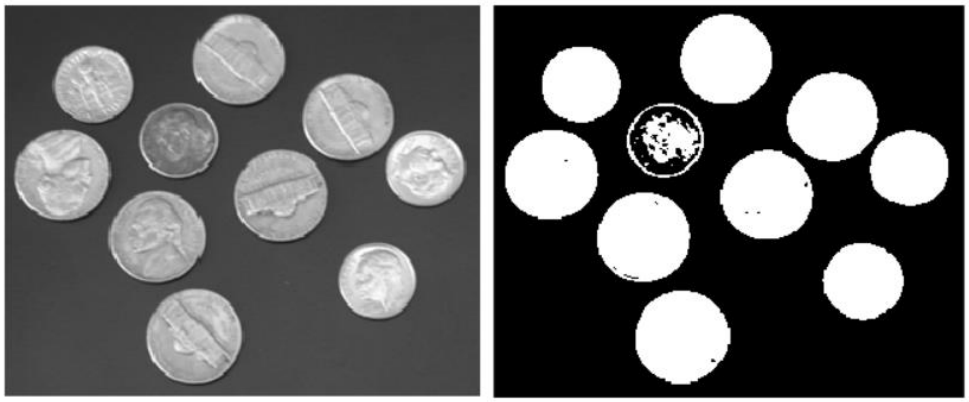
Conversia unei imagini color într-o imagine grayscale

Sunt numeroase metode prin care se poate face conversia unei imagini color într-o imagine grayscale. Două dintre ele sunt: metoda mediei și metoda ponderilor. Prin metoda mediei valoarea grayscale a unui pixel va fi egală cu media valorilor canalelor roșu, verde și albastru, din modelul RGB. Formula, după cum a fost descrisă anterior este aceasta: **Grayscale = (R + G + B) / 3**. Prin metoda ponderilor, adesea numită și metoda intensităților de lumină, se va calcula valoarea grayscale a unui pixel cu ajutorul ponderilor aferente fiecărui canal în parte din spațiul RBG. Formula este următoare: **Grayscale = 0.299R + 0.587G + 0.114B.**

Conversia unei imagini grayscale într-o imagine binară

După cum s-a specificat și mai sus, binarizarea reprezintă transformarea (convertirea) unei imagini din tonuri de gri (grayscale) în imagine binară (alb-negru). Această transformare este utilă în detectarea petelor – a se studia teme pe acest subiect, de exemplu Detecția petelor (în engleză, Blob Detection), și a reduce și mai mult complexitatea de calcul al operațiilor și al aplicațiilor de procesare de imagini. Obiectul principal în această conversie este de a găsi un prag adecvat după care să se poată face binarizarea. Sunt două metode principale prin care se poate calcula acest prag:

1. **Calcularea pragului local** – aici, se va calcula pragul parcurgând imaginea pixel cu pixel.
2. **Calcularea pragului global** – aici, se va calcula un prag global, deodată pentru toți pixelii din imagine.



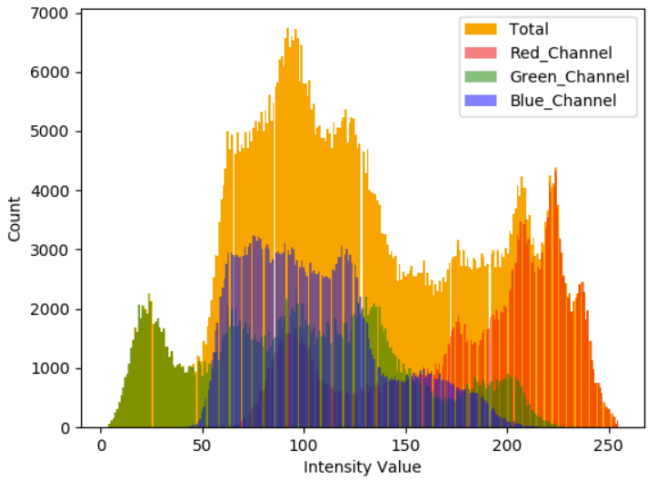
[Figura 4.5 – Procesul de binarizare]

În figura 4.5, de mai sus, avem reprezentate două imagini: cea din stânga este o imagine grayscale, iar cea din dreapta o imagine binară. Cu ajutorul uneia dintre cele două metode de binarizare s-a putut face conversia din imagine grayscale în imagine binară. A se observa tonurile de gri de la imaginea grayscale și cele două culori binare din imaginea din dreapta (alb și negru).

În proiectul descris în această documentație s-au folosit atât imagini grayscale cât și imagini binare, dar metodele de conversie între ele au fost făcute prin funcții predefinite din librăriile limbajului Python.

4.4.3. Histograme și histograma gradienților

O histogramă este un grafic care oferă o reprezentare vizuală a datelor numerice. Aceasta are aplicații largi în statistici, procesare de imagini sau chiar învățare automată. Histogramele dau ca și rezultate numărul de puncte de date care se află într-un interval de valori. Frecvența datelor care se încadrează în fiecare clasă din grafic este descrisă prin utilizarea unei linii orizontale (a se observa figura 4.6). În procesarea de imagini, histogramele sunt folosite pentru a stoca frecvențele pixelilor din imagine. Deci, o histogramă a unei imagini, este un tip de histogramă, un grafic, prin care se reprezintă grafic distribuția fiecărui pixel dintr-o imagine digitală.



[Figura 4.6 – Histograma unei imagini color pe toate dintre cele trei canale RGB]

La un sub-capitol anterior s-a descris spațiul de culoare RGB și cele trei culori primare reprezentate de el. O histogramă a unei imagini color, ia fiecare canal din modelul RGB și îi calculează frecvența. Se poate observa acest lucru în figura de mai sus. Histogramele imaginilor, în mare parte, sunt asemănătoare ca și descriere cu cea anterioară. Axa orizontală a graficului reprezintă variațiile de culori, sau variațiile tonale, iar axa verticală reprezentând numărul total de pixeli din acel ton de culoare particular. În procesarea de imagini, dacă se folosesc histograme, se va putea observa că de fiecare dată partea stângă a axei orizontale a unui astfel de grafic este reprezentată de zonele mai întunecate (culorile închise), iar partea dreapta fiind descrisă de culorile deschise. Deci, cu alte cuvinte, într-o imagine mai întunecată și putem lua ca exemplu aici o imagine binară, în care mare parte a pixelilor sunt negri, majoritatea punctelor sale de date din histograma ei vor în partea stângă al graficului. În schimb, la o imagine mai deschisă, cu multe culori vii, cu intensități ridicate, partea dreaptă a axei orizontale din histogramă va fi mai populată.

În domeniul viziunii artificiale, histogramele sunt foarte folositoare pentru a descoperi praguri. Aceste praguri pot fi folosite pentru detectarea muchiilor, sau pentru segmentarea imaginilor. Ca o referință la o noțiune învățată în liceu, histogramele pot fi asimilate cu vectorii de frecvență, doar că aici, elementele contorizate sunt pixelii și valorile lor.

Histograma gradienților

Pe parcursul acestei documentații s-a amintit și se va aminti în continuare despre Învățarea Automată, adică despre învățarea automată. Acest domeniu presupune lucrul cu date, cu date cât mai precise și bogate în informații pentru a-i putea fi ușor sistemului automat să învețe. Histograma gradienților orientați (HOG, abreviere ce va fi folosită de acum în toată documentația) este unul dintre algoritmii mult folosiți în Învățarea Automată. Acest tip de histogramă este utilizată în principal pentru detectarea feței, sau în mare al obiectelor din imagine. Ajută foarte mult la clasificarea obiectelor, clasificarea fiind o noțiune ce va fi detaliată în sub-capitolul 4.6.

În domeniul temei de față este foarte important a se ști ce înseamnă **descriptorii de trăsături** (în engleză, feature description). Chiar dacă se vor detalia mai mult în sub-capitolul 4.6. este necesar a fi puțin descriși și aici, pentru că histograma gradienților ține foarte mult de aceste noțiuni. Descriptorii de trăsături înseamnă reprezentarea unei imagini în care sunt extrase toate informațiile utile și se ignore cele inutile. Ca și un exemplu, referitor la tema proiectului, la detecția semafoarelor din trafic, gândiți-vă că dintr-o imagine va trebui să se extragă doar box-urile (cutiile) semafoarelor pentru că ele vor reprezenta singurele obiecte de interes din imagini. Descriptorii de caracteristici HOG vor reprezenta imaginile pe care se lucrează convertite în vectori caracteristici de lungime **n**, alese de utilizator. Acești descriptori sunt foarte buni pentru algoritmi de clasificare deoarece produc rezultate excepționale. De ce sunt atât de buni descriptorii HOG în clasificarea obiectelor? Ei bine, gradienții sunt extrem de importanți și se folosesc pentru verificarea marginilor și a colțurilor unei imagini (prin regiuni de schimbări de intensitate). Față de regiunile plane, ei vor împacheta adesea mai multe informații. În cele ce urmează, se va explica cum se calculează acești gradienți.

Gradienții reprezintă mici schimbări în direcțiile X și Y (aceste direcții reprezintă de fapt axele de coordonate în reperul cartezian XoY). Se va vedea și în capitolul următor că principalul obiectiv pentru început este de a împărți imaginea în sub-matrici (adică în chenare de diferite dimensiuni, matematic vorbind, în matrici de NxN). Fiecare celulă din aceste sub-matrici va

valoarea pixelului curent corespunzător acelei celule. În figura 4.7, se poate observa cum sunt împărțite imaginile în sub-matrici.

A(0, 0)	A(0, 1)	A(0, 2)	A(0, 3)	A(0, 4)	A(0, 5)	A(0, 6)	A(0, 7)	A(0, 8)	A(0, 9)	A(0, 10)	A(0, 11)	A(0, 12)	A(0, 13)	A(0, 14)
A(1, 0)	A(1, 1)	A(1, 2)	A(1, 3)	A(1, 4)	A(1, 5)	A(1, 6)	A(1, 7)	A(1, 8)	A(1, 9)	A(1, 10)	A(1, 11)	A(1, 12)	A(1, 13)	A(1, 14)
A(2, 0)	A(2, 1)	A(2, 2)	A(2, 3)	A(2, 4)	A(2, 5)	A(2, 6)	A(2, 7)	A(2, 8)	A(2, 9)	A(2, 10)	A(2, 11)	A(2, 12)	A(2, 13)	A(2, 14)
A(3, 0)	A(3, 1)	A(3, 2)	A(3, 3)	A(3, 4)	A(3, 5)	A(3, 6)	A(3, 7)	A(3, 8)	A(3, 9)	A(3, 10)	A(3, 11)	A(3, 12)	A(3, 13)	A(3, 14)
A(4, 0)	A(4, 1)	A(4, 2)	A(4, 3)	A(4, 4)	A(4, 5)	A(4, 6)	A(4, 7)	A(4, 8)	A(4, 9)	A(4, 10)	A(4, 11)	A(4, 12)	A(4, 13)	A(4, 14)
A(5, 0)	A(5, 1)	A(5, 2)	A(5, 3)	A(5, 4)	A(5, 5)	A(5, 6)	A(5, 7)	A(5, 8)	A(5, 9)	A(5, 10)	A(5, 11)	A(5, 12)	A(5, 13)	A(5, 14)
A(6, 0)	A(6, 1)	A(6, 2)	A(6, 3)	A(6, 4)	A(6, 5)	A(6, 6)	A(6, 7)	A(6, 8)	A(6, 9)	A(6, 10)	A(6, 11)	A(6, 12)	A(6, 13)	A(6, 14)
A(7, 0)	A(7, 1)	A(7, 2)	A(7, 3)	A(7, 4)	A(7, 5)	A(7, 6)	A(7, 7)	A(7, 8)	A(7, 9)	A(7, 10)	A(7, 11)	A(7, 12)	A(7, 13)	A(7, 14)
A(8, 0)	A(8, 1)	A(8, 2)	A(8, 3)	A(8, 4)	A(8, 5)	A(8, 6)	A(8, 7)	A(8, 8)	A(8, 9)	A(8, 10)	A(8, 11)	A(8, 12)	A(8, 13)	A(8, 14)
A(9, 0)	A(9, 1)	A(9, 2)	A(9, 3)	A(9, 4)	A(9, 5)	A(9, 6)	A(9, 7)	A(9, 8)	A(9, 9)	A(9, 10)	A(9, 11)	A(9, 12)	A(9, 13)	A(9, 14)

[Figura 4.7 – Împărțirea unei imagini sub formă de matrice în sub-matrici]

În figura de mai sus avem reprezentată o matrice. La capitolul 4.4. s-a descris puțin cum sunt reprezentate imaginile în domeniul de procesarea imaginilor. Ele în memorie sunt doar niște șiruri de valori, care arată și sunt aranjate sub formă matriceală. În domeniul temei discutate în această documentație și se va vedea asta și la capitolul 5, unde se va descrie pas cu pas cum s-a implementat aplicația software de față, este foarte folositor să se împartă aceste imagini în sub-matrici. De fapt, se vor împărți matricele lor aferente și asta se poate observa și în figura 13. Aici, ne putem imagina că o imagine este reprezentată de o matrice de dimensiunea de 10x15 (înălțimea fiind 10, iar lățimea va fi egală cu 15). Pentru o mai bună extragere a informațiilor vitale pentru clasificarea unor obiecte din această imagine, s-a ales împărțirea matricei în sub-matrici (patch-uri) de dimensiunea de 5x5.

Primul pas în calcularea histogramei gradienților este de a determina gradientul pentru fiecare pixel în parte, atât pe direcția X, cât și pe direcția Y. Să luăm de exemplu un patch din figura de mai sus. Pentru calculele de față s-a decis luarea patch-ului ce începe de la A(0, 0) până în colțul opus, la A(4, 4). Conceptual, A(0, 0), A(0, 1) ... A(9, 14) reprezintă doar valori la nivel de matrice, în format matematic. Dar în procesarea de imagini, aceste valori vor fi de fapt valorile pixelilor (în model RGB, HSV etc). Să presupunem că vrem din acest patch să calculăm gradientul în direcția X, iar apoi în direcția Y pentru elementul A(2, 2). Pentru a determina gradientul acestuia, în direcția X, trebuie să scădem valoarea din stânga din valoarea din dreapta. În mod similar, pentru a calcula gradientul acestui element A(2, 2), în direcția Y, vom scădea valoarea de jos din valoarea de deasupra. Prin urmare, dacă ar fi să scriem aceste operații ele vor fi cele ce urmează mai jos:

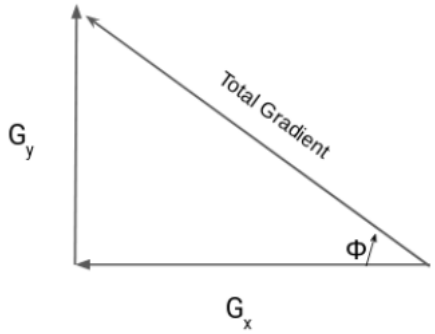
$$(G_x) = A(2, 3) - A(2, 1)$$
$$(G_y) = A(1, 2) - A(3, 2)$$

Făcând aceste calcule pentru fiecare element în parte, se vor stoca (forma) încă două matrici: una pentru stocarea gradienților pe direcția X, iar cealaltă pentru stocarea gradienților pe direcția Y. Următorul pas în calcularea histogramei gradienților ar fi calcularea magnitudinii și direcției pentru fiecare valoare a pixelilor în parte. Pentru acest pas, se va folosi teorema lui Pitagora. Gradienții pe direcția X și pe Y sunt practic cele două catete ale triunghiului dreptunghic reprezentat de figura 4.8. Aplicând această teoremă ajungem la relațiile de mai jos:

Total Gradient Magnitude = $\sqrt{(G_x)^2 + (G_y)^2}$

După calcularea magnitudinii mai trebuie calculată orientarea. Pentru aceasta, va trebui să aflăm unghiul **Φ** din triunghiul de mai jos. Se vor folosi următoarele formule:

- **tan(Φ) = Gy / Gx**
- **Φ = atan(Gy / Gx)**



[Figura 4.8 – Determinarea magnitudinii]

Dacă presupunem că aceste calcule efectuate mai sus au fost pentru pixelii unei imagini, atunci înseamnă că deocamdată avem aflate gradientul total (adică magnitudinea) și orientarea (direcția). Trebuie mai departe să generăm histograma folosind aceste valori. Histograma se va calcula precum se calculează un vector de frecvență învățat în ciclul liceal. Va trebui pentru fiecare pixel în parte să găsim orientarea lui și să-i actualizăm frecvența în tabelul aferent frecvențelor orientărilor. După ce este complet acest tabel, se va putea genera histograma orientărilor cu valorile din tabelul frecvențelor (acolo vom aveam pe prima linie frecvențele, iar pe a doua orientările). Histograma generată va fi o matrice de 1x9 (va avea o singură linie și 9 coloane).

4.5. Învățarea automată

Învățarea automată (în engleză, machine learning), este studiul algoritmilor de inteligență artificială care se pot îmbunătăți automat prin exemple și prin utilizarea datelor. Algoritmii de învățarea automată construiesc un model bazat pe un set de date de antrenare ce va fi folosit ulterior pentru a face predicții sau pentru a lua decizii fără a fi programat manual pentru a face acele lucruri. Domeniul de Învățare Automată are o grămadă de aplicații, cum ar fi: recunoașterea vorbirii, a fețelor, detecția obiectelor, viziunea computerizată etc. S-a ales a se vorbi despre acest domeniu pentru că aplicația descrisă aici folosește învățarea automată – se va vedea la capitolul 5 cum anume a fost folosită. Multe probleme de învățare automată sunt bazate pe statisticile de calcul, care se concentrează pe realizarea predicțiilor folosind calculatoare, dar nu toate sunt așa. Acest domeniu a fost dezvoltat foarte mult și multe probleme de învățare automată se bazează pe teoreme matematice și metode complexe de automatizare. Unele implementări ale machine learning-ului folosesc date și rețele neuronale într-un mod care imită funcționarea unui creier uman (pentru că un creier funcționează învățând după exemple).

Învățarea automată presupune ca sistemele computerizate să beneficieze de algoritmi prin care să descopere modul în care se pot îndeplini sarcini date de utilizatori fără a fi programate manual pentru a face acele lucruri. Se vor transmite date, iar din acestea calculatoarele vor trebui să învețe și să-și dea seama de anumite modele (pattern-uri) din care să poată lua decizii. Pentru

sarcinile simple, e foarte ușor să se programeze un calculator să le execute și să le ducă la bun sfârșit. Prin cod, fiind descriși pașii necesari pentru îndeplinirea sarcinilor, sistemele inteligente vor putea lua decizii automat. Dar pentru sarcinile mai avansate, vor fi necesare date prin care calculatoarele să poată învăța când și cum să ia deciziile.

Învățarea automată și extragerea datelor utilizează adesea aceleași metode și de multe ori „merg mână-n mână”, dar în timp ce învățarea automată se concentrează pe predicție pe baza a ce există deja în modelul antrenat, extragerea datelor se ocupă cu descoperirea trăsăturilor cele mai bune, pentru ca algoritmii de învățare automată să poată da rezultate cât mai bune.

Abordările prin care se pot face algoritmi de machine learning sunt împărțite în trei mari categorii, în funcție de natura feedback-ului disponibil sistemului inteligent de învățare:

- **Învățarea supervizată (în engleză, supervised learning):** sistemul inteligent deține exemple de intrări și rezultatele de pe urma lor. Aceste exemple sunt date de programator, iar scopul lor este de a ajuta sistemul de a învăța o regulă generală de mapare a intrărilor cu ieșirile.
- **Învățarea nesupervizată (în engleză, unsupervised learning):** aici, nu sunt date etichete algoritmului de învățare, adică, setul de date nu va conține exact etichete pentru datele obiectelor de interes. Asta înseamnă că algoritmul va fi pe cont propriu și va trebui să găsească singur modele în aceste seturi de date.
- **Învățarea cu ajutoare (în engleză, reinforcement learning):** sistemul inteligent interacționează cu un mediu dinamic pentru a putea îndeplini un scop bine stabilit. Pe măsură ce navighează prin acest mediu, sistemul primește feedback de pe urma căruia el va învăța să ia decizii.

Se vor detalia în următoarele două capitole (4.5.1 și 4.5.2) primele două categorii pentru că ele au stat la baza implementării acestui proiect.

4.5.1. Învățarea supervizată

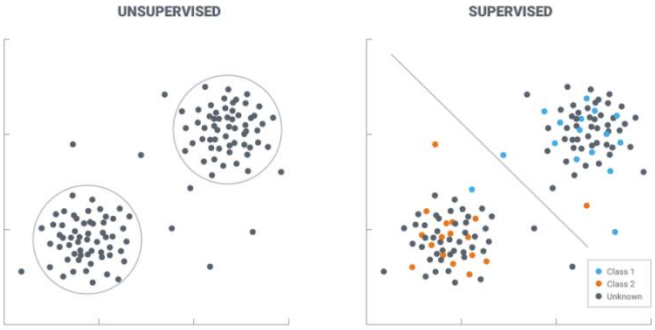
Algoritmii de învățare supervizată construiesc un model matematic pe baza unui set de date în care sunt incluse atât intrările, cât și ieșirile (rezultatele dorite). Datele aici sunt cunoscute sub numele de set de antrenament. Fiecare exemplu de antrenament are una sau mai multe intrări (de cele mai multe ori aceste intrări vor fi reprezentate de trăsături), iar ieșirea mai este cunoscută și sub numele de semnal de supraveghere (semnal supervizat). În modelul matematic, fiecare exemplu de antrenament este reprezentat de o matrice sau un vector, adesea numit și vector caracteristic, iar datele în sine de antrenament sunt reprezentate de o matrice. Acești algoritmi de învățare supravegheată vor genera o funcție optimă, sau un model, prin care îi permit sistemului inteligent să determine corect ieșirea pentru intrările din care au făcut parte datele de antrenament. Un algoritm care își îmbunătățește acuratețea rezultatelor sau predicțiilor în timp, se spune că a învățat să îndeplinească acea sarcină. Învățarea pe exemple similare este un domeniu al machine learning-ului strâns legat de regresie și clasificare (despre clasificare se va povesti în sub-capitolul 4.6.). Există o grămadă de aplicații în care se folosesc aceste tipuri de învățare supravegheată, cum ar fi: detectarea fețelor, a simbolurilor, etc. În acest proiect prezentat de documentația de față s-a folosit învățarea automată supervizată, algoritmii de clasificare descriși în sub-capitolul următor

primind seturi de date de antrenare pe fiecare tip de semafor în parte, pe urmă generând modele cu ajutorul cărora putând face predicții pe imagini de test.

4.5.2. Învățarea nesupervizată

Algoritmii de învățare nesupervizați posedă seturi de date care conțin doar intrări. Acolo ei găsesc structuri de date complexe și grupări de date, de pe urma cărora vor învăța să ia decizii. Prin urmare, algoritmii aceștia de învățare vor încerca să ia decizii doar cu ajutorul seturilor de date de testare pe care le dețin, ele nefiind etichetate sau clasificate într-un fel. Se vor identifica elementele comune din fiecare grup nou de date primit, iar pe baza a ceea ce știau până în acel punct, vor încerca să facă legături între date. La acest tip de învățare automată vom auzi de denumirea de cluster (grup). Analiza cluster-ului (analiza grupului) este atribuirea observațiilor în subgrupuri (acestea având numele de clustere), astfel încât aceste observații din cadrul aceluiași subgrup să fie similare, conform unor criterii bine stabilite.

Cele două tipuri de învățări automate (supervizată și nesupervizată) sunt frecvent discutate împreună în domeniul acesta de Învățare Automată. După cum s-a văzut, principala diferență între cele două este că la învățarea automată nesupervizată obiectele, datele, nu sunt etichetate.



[Figura 4.9 – Învățarea Nesupervizată vs Învățarea Supervizată]

În figura 4.9, se pot observa diferențele între cele două tipuri de învățare automată. Dacă la învățarea automată nesupervizată, obiectele nu au etichete și doar sunt grupate în clustere (grupuri), în funcție de trăsăturile comune, la învățarea automată supervizată fiecare obiect (dată) este atribuit unei clase specifice, iar sistemul inteligent prin învățare va putea face distincție între ele.

4.6. Clasificarea și algoritmi de clasificare

Clasificarea este procesul prin care obiectele pot fi grupate în diferite clase, pe urma unor trăsături. În domeniul învățării automate, clasificarea unui set de date în clase, poate fi realizată atât pe date (grupuri de date) structurate, cât și pe date nestructurate. Procesul acesta de clasificare începe cu încercarea de a se prezice o clasă dată. Clasele mai poartă denumirea și de ținte, etichete sau categorii. După cum s-a mai discutat și în capitolele anterioare, pentru început, ca un sistem inteligent să fie capabil să prezică anumite rezultate el va trebui să fie antrenat mai întâi, sau să i se dea un model deja antrenat. Pe baza acelui model, în cazul în care i se va da un nou set de date necunoscute, el va putea identifica (acesta va fi scopul) în ce clasă/categorie vor intra noile date primite.

Să încercăm acum să realizăm un exemplu de problemă de clasificare. Din punct de vedere medical, detectarea bolilor de inimă ar putea fi o problemă destul de importantă de clasificare. Aici, în cazul în care vrem să folosim învățarea automată supervizată, vom avea în principiu doar două clase: da sau nu (ori, bolnav de inimă sau sănătos). În funcție de datele de care dispune un sistem inteligent acesta ar trebui să poată face distincție între un om care poate suferi de boli de inimă și un om sănătos. Aceste date ar putea fi ca de exemplu: dacă suferă de diabet, dacă este fumător sau nu, vârsta etc.

În capitolul următor se vor folosi anumiți termeni care pot fi destul de noi pentru un cititor obișnuit. În cele ce urmează voi încerca să fac o scurtă prezentare a terminologiei ce ține de clasificare și de acest domeniu:

- **Clasificatorul** – este algoritmul care va fi folosit pentru a grupa intrările din setul de date cu scopul de a prezice o anumită categorie.
- **Modelul de clasificare** – acesta este un model antrenat care va fi folosit de algoritm (adică de către clasificator). Modelul a fost antrenat pe un set de date, de antrenare din care s-au putut distinge toate trăsăturile și clasele posibile.
- **Trăsătură** – un feature (trăsătură) este acea observație ce se poate face de pe urma datelor. Ca și exemplu aici, un feature (trăsătură) pentru un om este înălțimea.
- **Clasificarea binară** – clasificarea binară este acea clasificare în care obiectele sunt împărțite doar în două categorii.
- **Clasificarea pe mai multe clase** – aceasta este clasificarea în care obiectele sunt împărțite în mai mult de două categorii.
- **Tipul Clasificatorului** – reprezintă tipul clasificatorului, unde datele sunt asignate unui set de ținte.
- **Inițializarea** – înseamnă inițializarea clasificatorului pentru a putea fi folosit pentru prima oară.
- **Antrenarea clasificatorului** – fiecare clasificator trebuie antrenat, iar pentru asta i se va da un set de date de antrenament de pe care să învețe cum să ia deciziile.
- **Predicția țintei** – după ce a fost antrenat, clasificatorul va fi testat pe un set de date de test. Lui i se va da sarcina de a prezice clasele din acel set fără a i se pune la dispoziție aceste etichete. El va deține în acel punct doar trăsăturile din noul set de date, iar pe el va trebui să prezică și să ia decizii.
- **Evaluarea** – reprezintă evaluarea clasificatorului, adică cât de bune sunt rezultatele obținute de el.

Clasificatorii se pot împărți în două mari categorii, în funcție de modul în care aleg ei să funcționeze și să ia următoarele decizii:

1. **Clasificatori leneși –** aceștia sunt clasificatorii care rețin datele de antrenare până când întâlnesc date de test. Sunt renumiți pentru viteza lor destul de scăzută (de aici, și denumirea), iar câteva exemple pentru fi: K-Nearest Neighbor, Adaboost etc.
2. **Clasificatori dornici să învețe –** clasificatorii din această categorie construiesc un model pe baza datelor de antrenare fără a fi nevoiți să aibă informații din predicțiile obținute până în acel punct (adică fără a avea nevoie de date de test). Aceștia au nevoie de o perioadă de

timp mai mare pentru partea de antrenare, dar la predicția pe seturi de test sunt destul de rapizi. Câteva exemple de astfel de clasificatori: Decision Tree, Naïve Bayes, Rețele Neuronale etc.

În următoarele sub-capitole (4.6.1. – 4.6.4.) se vor prezenta patru algoritmi de clasificare, ei fiind folosiți și la realizarea proiectului de față.

4.6.1. AdaBoost

Primul clasificator ce va fi descris în această lucrare este AdaBoost. Denumirea lui vine de la Adaptive Boosting și este o tehnică, o modalitate de clasificare prin care se combină mai mulți clasificatori slabi pentru a obține, a construi, un clasificator puternic. Acest algoritm a fost compus de Yoav Freund și Robert Schapire ce au și câștigat Godel Prize-ul din anul 2003, un premiu oferit în fiecare an cercetătorilor care contribuie la universul computer-science (știința calculatoarelor). Există posibilitatea ca un singur clasificator slab să nu fie capabil să prezică clasa unui obiect cu exactitate, iar Adaboost-ul vine cu soluția de a grupa cât mai mulți clasificatori slabi pentru a se ajuta și a da rezultate cât mai bune, fiecare dintre ei învățând progresiv din greșelile celorlalți (din obiectele clasificate greșit). Tot menționând de clasificatorii slabi s-a omis definirea lor. Un clasificator este considerat slab dacă el, are într-adevăr performanțe mai bune decât ghicitul aleatoriu (în engleză, random), dar când vine vorba de prezisul claselor nu se descurcă suficient de bine.

Matematic vorbind, ideea generală a algoritmului AdaBoost este de a construi un clasificator puternic $\mathbf{H_T(x)}$ cu ajutorul a multor clasificatori slabi $\mathbf{h_t}$. Formula generală care ar reprezenta această combinație liniară de clasificatori slabi este următoarea:

$$H_T(\boldsymbol{x}) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x})\right)$$

Să presupunem că avem setul de date de antrenare reprezentat de $\{(\mathbf{x_i}, \mathbf{y_i})\}_{\mathbf{i=1}}^{\mathbf{N}}$, unde $\mathbf{x_i} \in \mathbf{R^K}$ și $\mathbf{y_i} \in \{-\mathbf{1}, \mathbf{1}\}$. Și să presupunem că deținem un număr destul de important de clasificatori slabi, adică $\mathbf{f_m(x)} \in \{-\mathbf{1}, \mathbf{1}\}$, și o funcție de pierdere între **0-1**, **I**, definită după cum urmează:

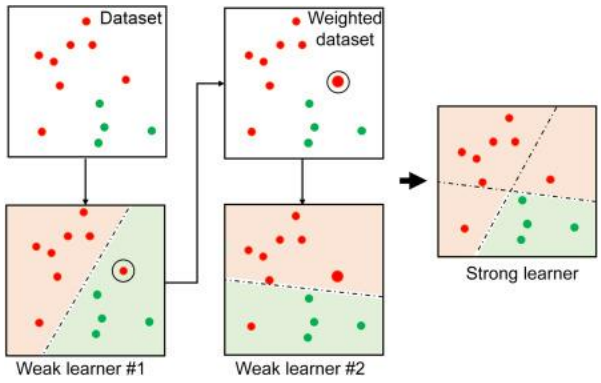
$$I(f_m(\mathbf{x}), y) = \begin{cases} 0 \text{ if} f_m(\mathbf{x}_i) = y_i \\ 1 \text{ if} f_m(\mathbf{x}_i) \neq y_i \end{cases}$$

Pașii pentru realizarea unui astfel de clasificator puternic sunt următorii:

1. Pentru primul pas se vor lua în considerare următorii parametrii (de fapt, clasificatorul va beneficia de ei): M, adică numărul total de iterații – deci numărul total de arbori de decizie și un set de date de antrenare ce conține N linii, sau grupuri de date.
2. Al doilea pas este cel în care se vor egala greutățile (acele ponderi), fiecare al N-lea se va egala cu greutatea 1/N.
3. Pentru pasul 3, se ia primul arbore de decizie G1(x) ce minimizează o funcție care generează o eroare a greutății dată de formula:

$$err_1 = \frac{\sum_{i=1}^{N}(w_i \ \ I(y_i \neq G_1(x_i)))}{\sum_{i=1}^{N} w_i}$$

4. Pentru pasul 4 se va calcula un parametru de actualizare **α1** pentru primul arbore, folosind $\log\left[\frac{1-err1}{err1}\right]$, iar pentru un calcul cât mai general, pentru fiecare **m = 1, 2, 3, ..., M αm** se va folosi formula $\log\left[\frac{1-err.m}{err.m}\right]$.
5. **α1** va actualiza greutățile, procesul continuându-se în buclă executându-se pașii 3, 4 și 5.



[Figura 4.10 – Funcționarea clasificatorului AdaBoost]

În figura 4.10, este prezentată funcționarea algoritmului de clasificare AdaBoost. Se pot observa în această imagine 4 secțiuni importante: un set de date general care conține toate datele și informațiile necesare clasificării, un set de date al greutăților (ni-l putem imagina ca un set de date de clase), doi clasificatori slabi și la final un clasificator puternic. Cei doi clasificatori slabi nu produc rezultate excepționale, dar din greșelile lor va rezulta un clasificator puternic care va fi folosit la final pentru luarea deciziilor. Acest clasificator de la final se obține însumând rezultatele date de cei doi clasificatori slabi.

Algoritmul de clasificare AdaBoost a fost folosit în cadrul proiectului de față, dar pentru că nu a generat rezultate destul de bune, pentru varianta finală s-a ales să se continue cu un alt algoritm de clasificare. Se vor detalia la capitolul următor toate încercările de implementare al proiectului.

4.6.2. K-NN (K-Nearest Neighbors)

Algoritmul de clasificare K-NN (abreviere de la K-Nearest Neighbors) este un algoritm de învățare automată supravegheată (despre supervised learning s-a discutat în sub-capitolul 4.5.1), simplu și ușor de implementat. Acesta a fost implementat pentru prima dată de către Evelyn Fix și Joseph Hodges, în anul 1951 și extins ulterior de către Thomas Cover. El poate fi utilizat atât pentru problemele de clasificare, cât și pentru cele de regresie. Acest algoritm presupune că obiectele, trăsăturile asemănătoare, dacă există, sunt apropiate unele de celelalte. În clasificarea cu K-NN, rezultatul oferit este un membru al unei clase existe din dataset. Un obiect, din setul de antrenare sau de test, este clasificat cu ajutorul votului vecinilor săi, obiectul fiind atribuit clasei cea mai comună care se întâlnește printre cei K vecini ai lui (de aici vine și numele algoritmului). Dacă de exemplu K=1, atunci obiectul curent este pur și simplu atribuit clasei acelui vecin care se

află cel mai aproape de el. Deoarece acest algoritm se bazează pe distanțe pentru a putea clasifica corect, dacă trăsăturile reprezintă unități fizice diferite sau vin în scări foarte diferite, atunci normalizarea datelor (acelor date din setul de antrenament) poate îmbunătăți considerabil acuratețea acestuia.

Pentru că acest algoritm de clasificare se bazează foarte mult pe distanțe, atât pentru clasificare cât și pentru regresie, o tehnică utilă pentru implementarea lui poate fi atribuirea de ponderi vecinilor (în funcție de contributivitate) – adică, dacă obiectul mai îndepărtat de cel curent să primească o pondere mai mare decât cel mai îndepărtat. Un astfel de exemplu de pondere este dacă se atribuie fiecărui obiect vecin ponderea **1/d**, unde **d** este distanța până la acel vecin.

În prima parte a clasificării se vor pregăti seturile de antrenament. Aceste seturi, exemple date algoritmului, vor fi niște vectori într-un spațiu de caracteristici multidimensional (de obicei în 2 dimensiuni), fiecare obiect având o etichetă clară, bine definită. Faza de antrenament constă în stocarea vectorilor de trăsături (vectorii caracteristici) și a etichetelor de clasă pentru a putea fi pregătite pentru clasificare. La partea de clasificare, **K** este o constantă definită de către utilizator, iar un vector neetichetat (un set de testare) este clasificat prin atribuirea etichetei care este cel mai frecvent printre cele **k** probe de antrenament mai apropiate de punctul acela de se testează. Pentru calculul distanței se poate folosi distanța **euclidiană** sau distanța **Hamming**.

Pentru că de cele mai multe ori se va folosi distanța euclidiană, se va reaminti pe scurt aceasta. Într-un plan Euclidian, să presupunem că avem două puncte: **P(x1, y1)** și **Q(x2, y2)**. Distanța dintre punctele **P** și **Q** se va calcula cu formula: $\sqrt{(\mathbf{Q_x} - \mathbf{P_x}) + (\mathbf{Q_y} - \mathbf{P_y})}$.

Pașii pentru implementarea algoritmului de clasificare K-NN sunt următorii:

1. Se va încărca setul de date. Acesta va conține un set de trăsături pentru obiecte bine stabilite de către utilizator.
2. Se va inițializa parametrul K care va reprezenta numărul de vecini după care să clasifice algoritmul K-NN. Pentru proiectul de față se va descrie în capitolul următorul o tehnică de optimizare prin care s-a putut alege eficient valoarea acestui parametru.
3. Pentru fiecare exemplu de obiect din setul de date, se vor calcula distanțele dintre el și cei k vecini ai săi. Distanța cea mai mică se va adăuga ca și pondere pentru obiectul curent, deci, se va reține pentru acesta atât distanța minimă cât și indicele lui, adică poziția la care se află în dataset.
4. Se vor forma două colecții, două liste: cu distanțe optime și cu indici. Aceste colecții (de fapt, se va implementa eventual o variantă în care să fie o singură colecție ce să conțină atât distanțe cât și indici) se vor sorta crescător în funcție de distanțe.
5. Se vor selecta primele K elemente din lista sortată la punctul 3.
6. Pentru elementele selectate la punctul 5 se vor lua din setul de date acele trăsături corespunzătoare lor.
7. Dacă algoritmul se va folosi pentru regresie, se va returna media celor k elemente.
8. Dacă algoritmul se va folosi pentru clasificare, se va returna eticheta (clasa) celor k elemente selectate.

Pentru proiectul de față K-Nearest Neighbors a fost ales algoritmul de clasificare cel mai potrivit pentru a se face predicțiile pe semafoarele din trafic. Acesta a rezultat cele mai bune

rezultate, atât din punct de vedere al timpului de execuție al clasificării, dar și din punct de vedere al acurateței. Pentru mai multe detalii, la capitolul 5 se va detalia exact cum a fost folosit acesta.

4.6.1. Naïve Bayes

Clasificatorii Naïve Bayes sunt o colecție de algoritmi bazați pe teorema lui Bayes. S-a menționat cuvântul colecție pentru că nu este doar un singur algoritm, este o grupare întreagă de algoritmi de clasificare ce au un principiu comun, și anume că perechile de trăsături clasificate sunt independente una de cealaltă. Nu se va insista foarte mult pe acest tip de clasificatori pentru că inițial nu s-a dorit utilizarea lor în aplicația de față. Doar s-a încercat o variantă de clasificare cu acești algoritmi, dar în final s-a rămas doar cu rezultatele obținute de pe urma clasificării de atunci. Există trei tipuri de clasificatori Naïve Bayes:

1. **Multinomial Naïve Bayes** – aceste tipuri de clasificatori sunt utilizați în mare parte în probleme pe clasificarea documentelor, de exemplu, avem mai multe categorii de documente: sport, politică, tehnologie etc. și se cere prezicerea categoriei unui document dat ca și test. Acești algoritmi se folosesc de frecvența cuvintelor prezente în documente.
2. **Bernoulli Naïve Bayes** – similari cu algoritmii tipului de mai sus, clasificatorii de aici au în schimb predictori cu valori booleene. Parametrii care vor fi folosiți pentru a prezice variabila de clasă iau doar valori din mulțimea **M = {“Da”, “Nu”}**, de exemplu, dacă un cuvânt apare sau nu într-un text.
3. **Gaussian Naïve Bayes** – pentru acest tip de clasificatori, presupunem că predictorii iau valori continue și care nu sunt discrete. Asta înseamnă că aceste valori sunt eșantionate dintr-o distribuție gaussiană.

4.6.2. Random Forest

Random Forest (în traducere, pădure aleatorie) este un algoritm de învățare automata supravegheată care poate fi construit din algoritmi ce stau la baza arborilor de decizie. Acest algoritm de clasificare este aplicat în diverse industrii, cum ar fi: serviciile bancare, comerțul electronic etc. Cu acest tip de algoritm se pot rezolva probleme atât de regresie, cât și de clasificare. Este asemănător cu algoritmul prezentat la sub-capitolul 4.6.1, adică AdaBoost, deoarece combină numeroși clasificatori pentru a oferi soluțiile cele mai bune, la cele mai complexe probleme. Caracteristicile unui algoritm de clasificare cu ajutorul metodei Random Forest:

- Este mult mai precis decât algoritmii bazați pe arborii de decizie.
- Poate asigura metode bune pentru a acoperi datele care se pierd în urma clasificării.
- Rezolvă problema încadrării excesive (în engleză, overfitting) care se întâlnea la arborii de decizie.
- Sunt selectate seturi de trăsături random pentru fiecare “pădure de arbori”, atunci când nodurile arborilor se împart în două.
- Acest tip de algoritmi poate produce rezultate bune de predicție fără metode de optimizare.

Ca și algoritmul Naïve Bayes de la sub-capitolul 4.6.3. acesta nu a fost folosit foarte mult în implementarea proiectului de față, încercându-se doar o variantă pentru a se vedea rezultatele obținute.

Capitolul 5. Proiectare de detaliu și implementare

Dacă până la acest capitol s-a prezentat partea teoretică studiată de mine pentru a putea duce la bun sfârșit proiectul propus, acum, la capitolul 5, se va descrie cu detalii, implementarea aplicației precum și principalele module, proceduri care au ajutat la realizarea ei. Se vor descrie schemele principale ale proiectului, precum și modulele aplicației (ea fiind împărțită pentru a putea fi explicată cât mai corect și concis).

5.1. Mediul de dezvoltare utilizat și limbajul de programare folosit

Fiind un programator, înainte de începerea implementării aplicației, m-am intersectat cu problema de a nu ști exact cu ce limbaj de programare să fac acest software sau în ce mediu de dezvoltare integrat să lucrez. Având în vedere că în cadrul studiilor de licență am studiat materia SRF (Sisteme de Recunoaștere a Formelor), iar limbajul de programare folosit, precum și mediul de dezvoltare fiind C++, respectiv Visual Studio 2019, inițial am vrut tot cu ajutorul lor să realizez acest proiect. Dar, lucrând într-o firmă în care mare parte din dezvoltatorii de software sunt Python Dezvoltatori (adică programatori în limbajul de programare Python), iar eu neștiind atât de bine acest limbaj, doar având o dorință mare de a-l învăța, mi s-a părut o oportunitate foarte bună de a încerca în limbajul de programare Python.

Pentru început să definim noțiunea de mediu de dezvoltare integrat. Acesta (în engleză "Integrated Development Environment" sau IDE) reprezintă o aplicație prin care o persoană care posedă cunoștințe de programare poate dezvolta alte aplicații la rândul lui. Aceste medii de dezvoltare posedă o grămadă de instrumente pentru a-i face munca unui programator mai ușoară, dar trebuie menționate următoarele: editorul de text, compilatorul și depanatorul codului.

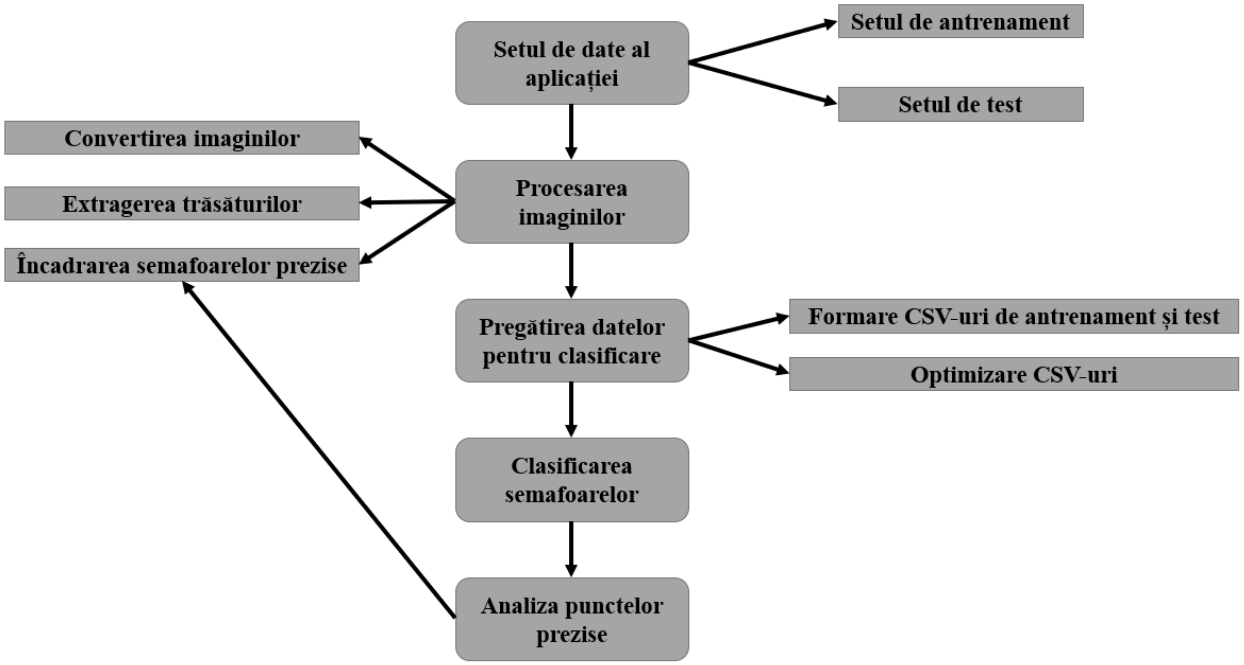
Pentru ușurarea muncii, în cadrul acestui proiect de Machine Learning, s-au folosit următoarele două medii de dezvoltare integrate: PyCharm, aplicație prin care se pot dezvolta aplicații în limbajul de programare Python și Visual Studio, aplicație prin care se poate scrie cod în limbajele C/C++, Assembly, C#, etc. Aceste medii de dezvoltare au fost create de către companiile JetBrains, respectiv Microsoft și sunt printre cele mai populare, ele având o multitudine largă de instrumente, dintre care se pot reaminti: posibilitatea de a face debugging, refactorizarea codului, etc.

În cele ce urmează se va discuta puțin și despre limbajul de programare Python, cel în care a fost realizată aplicația de față. Limbajul Python, este un limbaj de programare dinamic, multi-paradigmă, cu el putând fi create aplicații atât cu ajutorul principiilor de programare orientată pe obiecte, precum și cu programarea de tip funcțională, procedurală etc. Acesta a fost creat în anul 1989 de către programatorul olandez Guido van Rossum. Python, are un vocabular și un număr de construcții și cuvinte cheie care sunt cunoscute de către orice programator, deci, indiferent de limbajul folosit în mare parte a timpului, acest limbaj de programare nu este greu de învățat. Python, este un limbaj de programare ce este în mare parte folosit în următoarele domenii de activitate: programare web, Machine Learning, analiza datelor, dezvoltarea de aplicații Desktop, etc.

După cum s-a menționat și mai sus, s-a ales folosirea limbajului de programare Python fiind o ocazie bună și pentru a mă dezvolta din punct de vedere profesional, fiind un limbaj modern și folosit în mare dintre proiectele care se dezvoltă în companiile IT.

5.2. Schema generală a aplicației

Schema generală a aplicației este cea prezentată mai jos, în figura 5.1. La acest sub-capitol se vor descrie pe scurt modulele aflate în aceasta, fiind ulterior detaliate în sub-capitolele următoare.



[Figura 5.1 – Schema generală a aplicației]

În cele ce urmează se vor detalia, pe scurt, principalele module care se pot regăsi în figura de mai sus, ele fiind descrise mai în detaliu și din punct de vedere al implementării, abia în sub-capitolele următoare:

1. **Setul de date al aplicației** – fiind o aplicație care va implementa algoritmi de Machine Learning, adică de învățare automată, este necesar a se deține un set de date bine definit, recomandabil voluminos, care să conțină suficiente date astfel încât rezultatele oferite de aplicație să fie cât mai bune. Acest set de date va fi descris separat, într-un sub-capitol special, el fiind împărțit în două, atât pentru partea de antrenare a sistemului, cât și pentru testarea lui.
2. **Procesarea de imagini** – la acest modul, s-au realizat proceduri prin care să se poată manipula imaginile și pixelii din ele. S-au realizat diferite conversii între imagini, cum ar fi conversia dintr-o imagine color (în model de culoare RGB), într-o imagine grayscale (adică cu nuanțe de gri), sau o conversie dintr-o imagine grayscale într-o imagine binară. Fiind vorba despre un proiect bazat pe învățarea automată, vor fi necesare anumite trăsături prin care să se facă seturi de antrenare sau de test, care vor fi date clasificatorului. Aceste trăsături, fie că vor fi nuanțe de pixeli, valori medii ale lor, valori în spațiile de culoare RGB sau HSV, sau chiar histograme, toate aceste detalii vor fi extrase cu ajutorul unor proceduri făcute de mine. Pe lângă aceste puncte, la final, după ce vor fi prezise punctele

care vor reprezenta coordonatele la care se găsesc semafoare în imagini, vor fi nevoie de anumite metode care să interpreteze aceste puncte și să încadreze semafoarele în cercuri sau dreptunghiuri, cu ajutorul lor.

3. **Pregătirea datelor pentru clasificare** – Clasificarea, noțiune ce a fost detaliată în capitolul precedent, are la bază manipularea datelor deținute de către algoritmul de clasificare. Aceste date, de cele mai multe ori vor fi împărțite în astfel de probleme în două categorii: date de antrenament și date de test. Se va reveni cu detalii pentru fiecare tip de date în sub-capitolul 5.5.
4. **Clasificarea semafoarelor** – După ce au fost pregătite fișierele care vor conține datele reprezentative pentru semafoare, ele vor fi date unor algoritmi de învățare automată. Aceștia au fost toți în sub-capitolul 4.6, iar acum nu se vor mai detalia din punct de vedere teoretic, ci cum anume au fost folosiți pentru a genera rezultate în cazul proiectului de față.
5. **Analiza punctelor prezise** – Algoritmii de clasificare de la modul anterior vor genera câteva rezultate, mai exact, fișiere cu puncte prezise de ei. Aceste rezultate vor trebui să fie interpretate, iar cu ajutorul unor proceduri de procesare de imagini, punctele rezultate vor da naștere unor cercuri sau dreptunghiuri, după dorințele utilizatorului, care acestea la rândul lor vor fi încadrările semafoarelor din imaginile de test.

La sub-capitolele următoare se vor detalia, pas cu pas realizarea fiecărui modul în parte descris în această secțiune.

5.3. Setul de date disponibil

Pentru implementarea proiectului de față, s-au căutat diferite seturi de date pe Internet, care să conțină imagini din traficul urban și nu numai. O prioritate o aveau acele seturi de date care dispuneau de imagini ce conțineau cât mai multe semafoare.

S-a ales într-un final setul de imagini (setul de date) disponibil de la cei de la Bosch, un set de date precis, ce poate fi folosit nu doar pentru proiectul de față cu detectarea semafoarelor din traficul urban, ci și în alte proiecte cu teme asemănătoare, precum: detecția semnelor de circulație, a pietonilor, sau chiar a marcajelor longitudinale. Setul de date prezentat în continuare este ideal pentru realizarea unor astfel de probleme deoarece el conține următoarele caracteristici:

- Scene din orașe cu străzi dens populate de mașini.
- Drumuri cu mai multe benzi pe sens, suburbane, cu o densitate de trafic ce variază de la imagine la imagine.
- Scene din trafic în care mașinile circulă “bară-la-bară”.
- Schimbări de lumină datorate scenelor imortalizate sau a luminii soarelui.
- Scene în care sunt imortalizate secvențe din trafic pe timp de ploaie.
- Multe secvențe în care părți din imagini se pot confunda cu semafoarele din trafic (luminile de la stop-urile mașinilor).
- O multitudine de semafoare imortalizate.

Acest set de date conține aproximativ 13000 de imagini de rezoluție 1280x720 de pixeli și conține un număr de aproximativ 24000 de semafoare. În plus, împreună cu aceste imagini în setul

de date se găsește și un fișier care conține pozițiile exacte ale semafoarelor din fiecare imagine în parte, asta dacă ele există. Mai jos, în figura 5.2, se poate observa o parte din fișierul ce conține pozițiile semafoarelor.

```
- boxes: []
  path: ./rgb/train/2017-02-03-11-44-56_los_altos_mountain_view_traffic_lights_bag/207384.png
- boxes:
  - {label: Yellow, occluded: true, x_max: 615.75, x_min: 610.625, y_max: 358.625,
    y_min: 351.5}
  - {label: Yellow, occluded: false, x_max: 638.125, x_min: 633.875, y_max: 351.0,
    y_min: 342.25}
  - {label: Yellow, occluded: false, x_max: 655.0, x_min: 649.5, y_max: 360.75, y_min: 350.375}
  path: ./rgb/train/2017-02-03-11-44-56_los_altos_mountain_view_traffic_lights_bag/207386.png
```

[Figura 5.2 – Exemplu de poziții ale semafoarelor dintr-o imagine]

În figura 5.2, avem structurată o parte foarte mică a fișierului din setul de date în care sunt reprezentate pozițiile semafoarelor. Acesta, este un fișier de tip **.yaml** și are următoarea structură:

- **boxes** – reprezintă cumulul de semafoare dintr-o imagine. Dacă imaginea curentă, descrisă de path-ul de la câmpul "**path**" nu are semafoare, câmpul boxes va avea un vector (în engleză, array) gol.
- **path** – reprezintă calea (în engleză, path-ul) la care se găsește imaginea curentă.
- **label** – reprezintă tipul semaforului: pentru galben avem șirul de caractere "**Yellow**", pentru verde avem "**Green**", iar pentru roșu avem cuvântul "**Red**". Se poate întâmpla să avem situații în care semaforul să fie oprit, atunci, valoarea câmpului label fiind "**Off**".
- **occluded** – vrea să însemne dacă semaforul din imaginea curentă este pornit sau nu. Dacă nu, valoarea câmpului va fi setată pe **false** (fals), dacă da, va fi setată pe **true** (adevărat).
- **x_max** – reprezintă în sistem de coordonate XoY, valoarea X-ului maxim al cutiei semaforului.
- **x_min** – reprezintă în sistem de coordonate XoY, valoarea X-ului minim al cutiei semaforului.
- **y_max** – reprezintă în sistem de coordonate XoY, valoarea Y-ului maxim al cutiei semaforului.

y_min – reprezintă în sistem de coordonate XoY, valoarea Y-ului maxim al cutiei semaforului.

Mai sus, la ultimele patru puncte, am descris ce înseamnă anumite coordonate găsite în fișierul yaml. Să ne imaginăm că un semafor, este reprezentat într-o imagine binară ca un chenar dreptunghic cu valoarea albă (valoarea pixelului va fi 255). Cele patru colțuri ale dreptunghiului vor putea fi reprezentate de cele patru puncte descrise mai sus. Valorile celor doi de **x**, vor reprezenta valorile minime și maxime ale înălțimii chenarului (dacă ne imaginăm că este reprezentat în sistem de coordonate XoY), iar cei doi de **y** vor reprezenta valorile lungimii. Altfel spus, făcând o referință la reprezentarea imaginii sub formă matriceală, x-ul va avea valorile liniilor, iar y-ul va avea valorile coloanelor.

La figura 5.2, avem descrise două imagini. În prima imagine nu avem prezent niciun semafor, de asta vectorul boxes este gol, iar în a doua imagine avem reprezentate trei semafoare, fiecare dintre ele conținând: tipul semaforului, dacă este pornit sau nu și coordonatele lui. În plus,

pentru fiecare imagine în parte, avem și locația exactă a ei, adică calea către acel semafor.

5.3.1. Setul de antrenare

Având în vedere că domeniul în care s-a realizat proiectul este acela al învățării automate, adică Machine Learning și se vor folosi câțiva algoritmi de clasificare pentru a prezice locațiile semafoarelor, asta înseamnă că avem nevoie de a împărți setul de date în două: set de antrenare și set de test.

Pentru realizarea proiectului de față, s-au creat directoare separate pentru fiecare tip de semafor în parte (asta înseamnă că s-a făcut clasificare pentru fiecare tip de semafor), dar și un director pentru avea imagini de antrenare care să conțină toate tipurile de semafoare. Pe lângă aceste directoare, s-au creat directoare separate în care să se introducă, cu ajutorul procedurilor specializate, toate fișierele de tip Excel generate de aplicație, cu ajutorul cărora se vor face clasificările de semafoare. Deci, pentru setul de antrenare avem următoarea structură ce se aplică pentru fiecare tip de semafor în parte:

- **director** pentru imaginile binare în care vor exista imagini generate prin cod, în care sunt evidențiate nuanțele de roșu, verde sau galben (după caz), dar în culoarea alb (fundalul va fi negru, iar locul acestor nuanțe va fi plin de pixeli cu valoarea albă).
- **director** pentru fișierele în format Excel aferente imaginilor de mai sus.
- **director** în care vor fi stocate toate imaginile originale ce vor fi prelucrate. Aceste imagini vor reprezenta bazele seturilor de antrenament, iar ele vor fi în număr de 80 de imagini pentru fiecare semafor în parte.
- **director** în care vor fi stocate imagini binare, generate cu ajutorul anumitor proceduri, în care vor fi evidențiate semafoarele cu ajutorul pozițiilor lor exacte date de fișierul yaml ce va fi amintit ulterior.
- **director** pentru fișierele Excel generate în care sunt descrise imaginile din punct de vedere al pixelilor (vor fi puse aici trăsăturile, adică pixelii în format RGB, HSV, valoarea luminozității lor și histograma gradienților).
- **director** pentru fișierele Excel generate în care avem stocate pozițiile fiecărui pixel în parte.
- **director** pentru fișierele Excel care conțin patch-urile imaginilor binare generate cu ajutorul pozițiilor semafoarelor.
- **director** în care vor fi fișierele Excel formate cu ajutorul tuturor celorlalte fișiere Excel de dinainte. Aceste fișiere cu extensia **.csv**, vor fi date clasificatorului (după ce vor fi concatenate și aranjate astfel încât să se acopere și partea de optimizare).
- **fișier cu extensia .yaml** ce va conține toate semafoarele din directorul general cu toate imaginile selectate pentru procesul de antrenare.

5.3.2. Setul de test

Setul de antrenare, explicat la sub-capitolul 5.3.1. va fi folosit pentru antrenarea clasificatorilor ce vor fi folosiți pe parcursul aplicației. După ce au fost antrenați, acești clasificatori, vor genera niște modele, modele ce vor fi folosite pentru predicția semafoarelor ce se află în seturile de test. Seturile de test vor reprezenta deci, un set de date care vor fi date algoritmilor

de clasificare pentru a le putea fi testată performanța. S-au creat directoare speciale pentru fiecare tip de semafor în parte, ce vor reprezenta directoare de stocare al datelor de test pentru obiectele de interes. Acestea vor avea următoarea structură:

- **director** special pentru stocarea imaginilor binare generate cu ajutorul unor proceduri prin care se manipulează pixelii. De pe urma acestor imagini se vor genera fișiere Excel aferente fiecărui patch în parte din imagini.
- **director** pentru stocarea imaginilor binare, în care vor fi evidențiate doar semafoarele, fundalul acestor imagini fiind negru. Aceste imagini vor fi generate cu ajutorul unor proceduri prin care se vor se lua pozițiile semafoarelor și între acele coordonate se vor pune doar pixeli albi, în rest fundalul fiind negru.
- **director** pentru stocarea tuturor fișierelor Excel aferente. Vor fi aceleași fișiere cu extensia .csv ca pentru imaginile din setul de antrenare.
- **fișier cu extensia .yaml**, care va stoca toate semafoarele din imaginile de test sub format de cod scris, aici fiind pozițiile lor, tipul semaforului și calea către acele imagini de test.
- Un set de 10-15 imagini de test pe care se va testa aplicația.

5.4. Structura proiectului

Aici se va descrie structura aplicației din punct de vedere al codului. Nu se vor detalia procedurile folosite, pentru asta având sub-capitole aferente, ci se va detalia modul în care a fost structurat proiectul. În ajutorul acestei secțiuni vor veni și două diagrame UML, în sub-capitolul **5.8.** Până atunci, aplicația de față a fost structurată în patru mari fișiere:

1. **Main-ul aplicației –** din acest fișier se va porni aplicația. Aici se vor afla procedurile aferente GUI-ului aplicației (din engleză Graphical User Interface – Interfața Grafică) dar și o metodă din care se vor apela alte funcții care se vor ocupa cu pornirea clasificatorilor. Se va reveni cu detalii în secțiunea unde se va explica cum s-au implementat aceștia.
2. **Utility –** acest fișier este pentru stocarea tuturor procedurilor care se ocupă cu manipularea setului de date, atât cel de antrenare, cât și de test, extragerea cutiilor reprezentative pentru semafoare din fișierul .yaml și formarea fișierelor cu extensia .csv care se vor da clasificatorilor.
3. **ImageFunctions –** un alt fișier cu proceduri importante, aici vor fi stocate funcții prin care s-au manipulat pixelii din imagini. S-au implementat metode prin care s-au realizat acele patch-uri cu ajutorul cărora se vor grupa datele din fișierele de antrenament și de test, precum și alte funcții pentru construirea histogramelor, extragerea trăsăturilor din imagini, conversii dintr-un tip de imagine în alt tip, formări de fișiere cu extensia .csv, etc. Se va reveni cu detalii în sub-capitolele următoare.
4. **TrafficLight –** ultimul fișier reprezintă de fapt o clasă. Am considerat că un semafor poate fi reprezentat din punct de vedere al codului printr-o clasă și din acest motiv s-a creat acest fișier. Aici, avem doar un constructor prin care se inițializează un semafor și o funcție de afișare.

5.5. Procesarea imaginilor și extragerea trăsăturilor

La acest sub-capitol se vor explica principalele module și metode folosite pentru partea de

procesare de imagini, dar și pentru extragerea trăsăturilor care vor ajuta la procesul de clasificare. Procesarea imaginilor acoperă următoarele puncte importante: convertirea imaginilor în diferite tipuri (din RGB în HSV, din RGB în grayscale sau chiar din grayscale sau RGB într-o imagine binară), manipularea pixelilor dintr-o imagine (cum se calculează valorile din modelul de culoare HSV sau luminozitatea unui pixel, etc), dar și manipularea punctelor dintr-o imagine astfel încât să se poată trage linii între diferite puncte reprezentate în coordonate XoY pentru a forma dreptunghiuri sau cercuri (acestea vor ajuta la procesul de vizualizare al rezultatelor finale, obținute de pe urma predicțiilor). Extragerea trăsăturilor dintr-o imagine presupune atingerea următoarelor puncte: extragerea histogramelor aferente spațiului de culoare RGB, folosirea histogramei gradienților sau extragerea valorilor importante din pixeli (valori obținute de pe urma procesării imaginilor).

5.5.1. Convertirea imaginilor în diferite tipuri disponibile

La acest sub-capitol se vor acoperi principalele conversii între tipurile de imagini, ce au fost folosite și pentru realizarea proiectului de față. Nu se vor descrie procedurile la nivel de cod, adică linie cu linie, ci doar se vor pune antetele metodelor folosite și principale puncte forte ale lor, sau ideea prin care s-au realizat acestea.

1. **Conversia RGB-HSV**

Acest tip de conversie se vrea a fi o modalitate prin care să se convertească o imagine color (adică în format RGB), într-o imagine în modelul de culoare HSV. Pentru acest lucru, am realizat o metodă cu antetul **convert_image_from_rgb_to_hsv(imagePath)**. După cum se poate observa, numele metodei este sugestiv pentru ceea ce vrea aceasta să facă. În primul rând, metoda primește ca și parametru o variabilă, ce va fi un string (adică un șir de caractere) ce va reprezenta calea absolută către imaginea pentru care se face conversia. În această metodă se va citi o imagine cu ajutorul funcției predefinite **imread** din librăria **OpenCV**, iar cu metoda ce se află și ea în această librărie și anume **cvtColor**, ce primește ca și parametri imaginea citită și constanta **cv2.COLOR_BGR2HSV** s-a realizat această conversie între RGB și modelul de culoare HSV. Procedura va returna imaginea rezultată în urma conversiei descrise.

2. **Conversia RGB – Grayscale**

Acest tip de conversie face ca o imagine în modelul de culoare RGB să devină o imagine grayscale. În implementarea proiectului de față, s-a realizat procedura cu următorul antet: **convert_an_rgbImage_to_grayscale(imagePath)**. Această metodă, după cum sugerează și numele, face conversia unui imagini din modelul de culoare RGB în tipul grayscale. Primește ca parametru calea absolută a imaginii cu pricina și returnează imaginea grayscale rezultată. Pentru această conversie s-a folosit funcția predefinită din librăria **OpenCV** și anume **imread**, căreia i s-au furnizat ca și parametri calea către imaginea în format color și constanta **cv2.IMREAD_GRAYSCALE**.

3. **Conversia RGB – Binary**

Acest tip de conversie transformă o imagine din modelul de culoare RGB într-o imagine binară. În implementarea proiectului de față, pentru o conversie de acest gen, s-a realizat procedura cu următorul antet: **convert_an_rgbImage_to_binary(imagePath)**. Această metodă, după cum sugerează și numele, face conversia unui imagini din modelul de culoare RGB într-o imagine binară (adică alb-negru). Metoda, primește ca parametru calea absolută a imaginii cu pricina și returnează imaginea binară rezultată în urma conversiei. Pentru această conversie, s-a citit

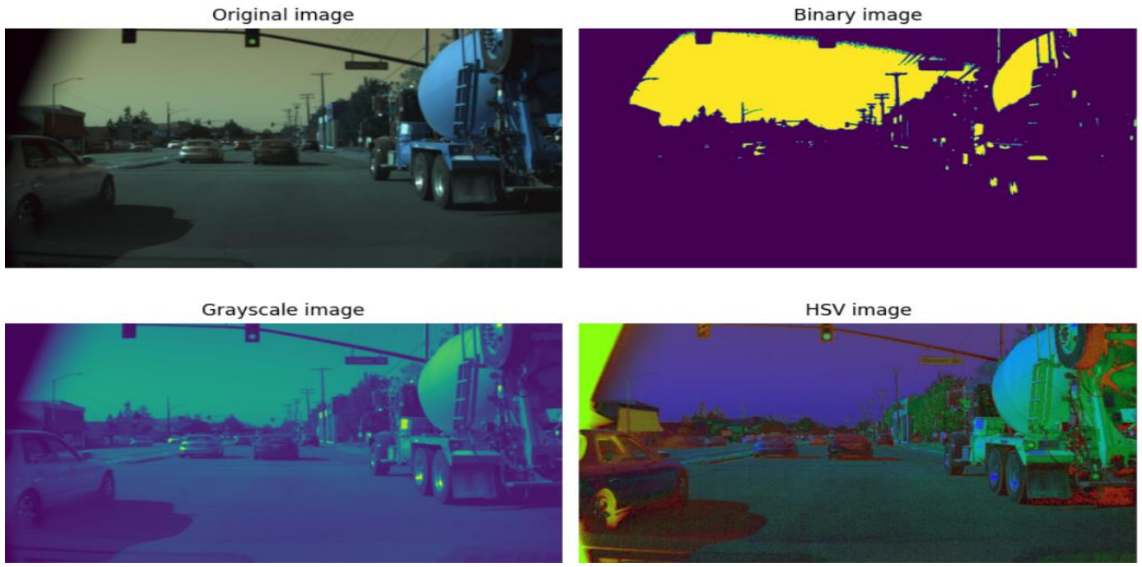
imaginea curentă cu ajutorul funcției **imread**, ce primește ca parametri calea către directorul imaginii și constanta 2. Mai departe, s-a convertit imaginea citită într-una binară cu ajutorul funcției predefinite din librăria **OpenCV** și anume **cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)**. După cum se poate observa, funcția primește ca parametri imaginea RGB, pragul după care să se facă conversia și anume 127, valoarea 255 care reprezintă valoarea pixelului alb și constanta **cv2.THRESH_BINARY**.

4. **Conversia Grayscale – Binary**

Conversia descrisă acum transformă o imagine grayscale într-o imagine binară. Procedura creată de mine, are următorul antet: **convert_an_grayscaleImage_to_binaryImage(imagePath)**. O fost ales un nume sugestiv pentru ca urmăritorii codului să știe la ce vrea să facă referire această metodă, iar parametrul imagePath reprezintă calea absolută către această imagine la care se face conversia. Pentru această conversie, s-a citit imaginea curentă cu ajutorul funcției **imread**, ce primește ca parametru calea către directorul imaginii. Mai departe, s-a convertit imaginea citită într-una binară cu ajutorul funcției predefinite din librăria **OpenCV** și anume **cv2.threshold(image, 127, 256, cv2.THRESH_BINARY)**. După cum se poate observa, funcția primește ca parametri imaginea RGB, pragul după care să se facă conversia și anume 127, valoarea 256 care reprezintă limita paletei de culori și constanta **cv2.THRESH_BINARY**.

5. **Crearea unei imagine cu fundal negru**

În cadrul acestui sub-capitol, se va descrie și o procedură prin care s-a creat o imagine cu fundal negru. Aceasta, are antetul **create_blank_image(width, height, rgb_color = (0,0,0))** și după cum specifică și numele va crea o imagine cu pixeli negri. Această metodă se folosește de funcția predefinită din librăria **Numpy** și anume **np.zeros((height, width, 3), np.uint8)** care crează un vector cu valori de 0, un vector în trei dimensiuni, cu ajutorul dimensiunilor specificate ca parametri, în care toate datele pot avea doar valori de tip întreg, pe 8 biți și cu semn. Imaginea rezultată în urma acestor operației se va returna din funcție.



[Figura 5.3 – Diverse conversii de imagini]

În figura de mai sus, putem observa câteva conversii de imagini, mai exact trei dintre ele și anume: conversia unei imagini color într-o imagine binară, grayscale și în modelul de culoare

HSV. Pentru a afișa aceste patru imagini s-a folosit o procedură, numită **plot_images_conversions**, ce primește ca parametru calea unei imagini pentru care se dorește a se face aceste conversii.

5.5.2. Extragerea trăsăturilor din imagini

În acest sub-capitol se vor acoperi principalele metode și proceduri care s-au realizat în cadrul aplicației, pentru a putea extrage trăsăturile ce vor fi date clasificatorilor. Aceste trăsături, reprezintă valorile din spațiile de culoare RGB și HSV, valoarea luminozității unui pixel, media valorilor HSV, precum și proceduri prin care s-au manipulat valorile pixelilor dintr-o zonă specifică a imaginii. În plus, s-au creat funcții pentru extragerea celor mai importante histograme ale unei imagini, dintre care și histograma gradienților.

5.5.2.1. Extragerea valorilor RGB, HSV și intensității pixelilor

În cadrul proiectului de față s-au realizat proceduri pentru extragerea valorilor pixelilor din spațiile de culoare RGB și HSV. Aceste modele de culoare nu vor mai fi descrise încă o dată, deoarece au fost detaliate pe larg la secțiunea 4.4.1. În schimb se va detalia cum anume s-au realizat metodele prin care s-au extras aceste valori. În cele ce urmează, pentru început, se va reprezenta printr-un pseudocod modalitatea prin care s-au convertit valorile din spațiul RGB în modelul de culoare HSV.

```
convert_rgb_values_to_hsv_values(red, green, blue): procedură
        red <- red/255.0
        green <- green/255.0
        blue <- blue/255.0

        calculate_maxim_minim(red, green, blue): procedură
        diffBetweenMaximMinim <- maxim - minim

        dacă maxim = minim atunci
                hue <- 0
        altfel dacă maxim = red, atunci
                hue <- (60 * ((green - blue)/diffBetweenMaximMinim) + 360) % 360
        altfel dacă maxim = green, atunci
                hue <- (60 * ((blue - red)/diffBetweenMaximMinim) + 120) % 360
        altfel dacă maxim = blue, atunci
                hue <- (60 * ((red - green)/diffBetweenMaximMinim) + 240) % 360

        dacă maxim = 0, atunci
                saturation <- 0
        altfel
                saturation <- (diffBetweenMaximMinim / maxim) * 100

        value <- maxim * 100

        returnează hue, saturation, value
```

[Figura 5.4 – Pseudocod ce descrie conversia RGB - HSV]

În figura 5.4, este detaliată metoda prin care se poate face conversia a trei valori R, G, B (roșu, verde și albastru), în valorile corespunzătoare spațiului de culoare HSV. Matematic vorbind, aceste formule au fost descrise pe larg la sub-capitolul 4.4.1. Pentru început, va trebui să împărțim cele trei valori RGB, la valoarea 255, pentru a putea obține valori în virgulă flotantă. Următorul pas este de a calcula maximul și minimul acestor trei valori obținute. Se poate face acest calcul cu ajutorul unor funcții predefinite din librăriile limbajului Python. Se va calcula pe urmă și diferența corespunzătoare dintre maximul aflat și valoarea minimă . La final, cu ajutorul formulelor

matematice descrise la sub-capitolul cu pricina se vor afla valorile corespunzătoare pentru Hue, Saturation și Value, urmând să fie cele returnate din funcție. S-a realizat o procedură, în limbajul Python corespunzător pseudocodului descris mai sus.

În continuare, au fost implementate două metode cu nume sugestive, care extrag valorile de pe canalele R, G și B, respectiv H, S și V. Prima dintre metode se numește **rgb_channels** și primește ca parametru calea către imaginea pentru care se vrea a fi extrase valorile de pe canale. Cu ajutorul eficienței limbajului Python, putem dintr-un vector, vector pe trei dimensiuni în cazul de față, să extragem valorile de interes din ele. Folosind sintaxa **image[:, :, X]**, unde X, un număr întreg din intervalul [0-2], este canalul de interes din RGB, se va extrage vectorul corespunzător acestuia. Se vor returna la finalul acestei metode trei vectori, ce vor reprezenta vectorii valorilor din fiecare canal corespunzător modelului de culoare RGB. Pe aceeași metodă s-a realizat și funcția cu numele **hsv_channels**, care primește ca parametru tot calea imaginii ce se vrea a fi prelucrată. Această procedură returnează vectorii corespunzători celor trei canale din spațiul de culoare HSV.

Metoda cu numele **brightness**, va extrage culoarea medie ce apare în imaginea primită ca parametru. Folosind câteva funcții predefinite din librăriile de specialitate din limbajul Python, metoda va afișa și va returna la final culoarea medie care se poate observa în imaginea dată de utilizator.

Aceste metode au fost folosite pentru prima parte a acestui proiect, și anume clasificarea semafoarelor folosind imagini în care erau prezente doar ele. Se va reveni cu detalii în sub-capitolele următoare.

5.5.2.2. Folosirea histogramelor și a histogramei gradienților

În proiectul de față s-au folosit histogramele pentru a observa anumite detalii în legătură cu densitatea pixelilor din imagini dar și pentru extragerea trăsăturilor care vor ajuta la clasificarea semafoarelor. În rândurile următoare, se va descrie mai întâi o funcție predefinită din librăria **OpenCV**, ce a fost des folosită pentru calcularea histogramelor. Aceasta este **cv2.calcHist**, se folosește pentru a construi histogramelor imaginilor și are următorul antet:

cv2.calcHist(images, channels, mask, histSize, ranges)

- **images** – reprezintă calea absolută către imaginea pentru care se va calcula histograma sau chiar o imagine în sine, asta dacă ea a fost citită cu ajutorul unor altor funcții predefinite.
- **channels** – acest parametru reprezintă un vector, de fapt o listă, care va reprezenta vectorul canalelor disponibile. Pentru a construi histograma unei imagini color, avem trei canale: roșu, verde și albastru, deci lista canalelor va fi [0, 1, 2].
- **mask** – pentru acest parametru, se va detalia într-un sub-capitol următor ce înseamnă o mască în domeniul procesării imaginilor. Pe scurt, în cazul de față, dacă se aplică o mască, histograma se va construi doar pentru pixelii din acea mască. Dacă nu se dorește folosirea măștii, parametrul va avea valoarea **None**.

  histSize – parametrul acesta va reprezenta numărul de bin-uri (adică cât de mare să fie histograma, sau lungimea maximă a vectorului histogramei). Dar și acest parametru este o listă, asta înseamnă că dimensiunea histogramei se va aplica pentru fiecare canal în parte. De exemplu, dacă alegem o dimensiune de 16 (adică bins = 16) și cum avem și trei canale reprezentative pentru spațiul de culoare RGB, asta înseamnă că vom avea o histogramă totală cu dimensiunea egală cu valoarea 48 (adică câte 16 lungimi pentru fiecare canal în parte).

Vor fi două funcții ce vor fi explicate în cadrul acestui sub-capitol. Prima dintre ele este o funcție numită **create_histogram_for_color_image** ce are ca parametru calea imaginii pentru care va fi creată histograma. Această procedură, descrisă și de figura 21 unde este reprezentat pseudocodul ei, se împarte în principal în trei părți. În prima parte, s-au citit imaginile color și în modelul HSV, aferente căii absolute dată ca și parametru la funcție, s-au creat două liste (o listă pentru caracterele corespunzătoare fiecărui canal în parte din modelul RGB și analog, o listă pentru caracterele spațiului HSV) și 6 liste, inițializate cu liste vide, pentru fiecare canal în parte (red, green, blue, hue, saturation, value). În a doua parte, cu ajutorul a două bucle for, s-au calculat histogramele pentru fiecare canal în parte. La prima buclă, s-a calculat histograma pentru modelul RGB, iar valorile din histogramă s-au pus într-o listă aferentă acestui model, iar la a doua buclă, analog ca la prima, s-a realizat histograma pentru spațiul HSV și s-au pus valorile din ea într-un vector (listă) separat. În ultima parte a acestei proceduri, din listele formate în pasul anterior, s-au format liste cu valori din histogramă pentru toate canalele în parte (și asta s-a realizat cu ajutorul a trei bucle, în fiecare dintre ele formându-se liste pentru perechile de canale: blue-hue, green-saturation și red-value). Mai jos, se poate observa pseudocodul procedurii descrise până acum.

```
create_histogram_for_color_image(imagePath): procedură
        citește imaginea originală
        convertește imaginea originală în format HSV

        culoriRGB <- ('b', 'g', 'r')
        culoriHSV <- ('h', 's', 'v')
        blueChannel, greenChannel, redChannel, hueChannel, saturationChannel, valueChannel <- listă vidă

        vectorRGB <- listă vidă
        pentru fiecare canal și culoare aferentă din culoriRGB, execută
                histogramă <- calculareHistogramă: procedură predefinită
                listăCuValoriDinHistogramă <- extragere_valori_din_histogramă(histogramă)

                pentru x din listăCuValoriDinHistogramă, execută
                        vectorRGB.lipește(x)

        vectorHSV <- listă vidă
        pentru fiecare canal și culoare aferentă din culoriHSV execută
                histogramă <- calculareHistogramă: procedură predefinită
                listăCuValoriDinHistogramă <- extragere_valori_din_histogramă(histogramă)

                pentru x din listăCuValoriDinHistogramă, execută
                        vectorHSV.lipește(x)

        pentru i din intervalul [0, 16), execută
                blueChannel.lipește(vectorRGB[i])
                hueChannel.lipește(vectorHSV[i])

        pentru i din intervalul [16, 32), execută
                greenChannel.lipește(vectorRGB[i])
                saturationChannel.lipește(vectorHSV[i])

        pentru i din intervalul [32, lungime(vectorRGB)), execută
                redChannel.lipește(vectorRGB[i])
                valueChannel.lipește(vectorHSV[i])

        returnează blueChannel, greenChannel, redChannel, hueChannel, saturationChannel, valueChannel
```

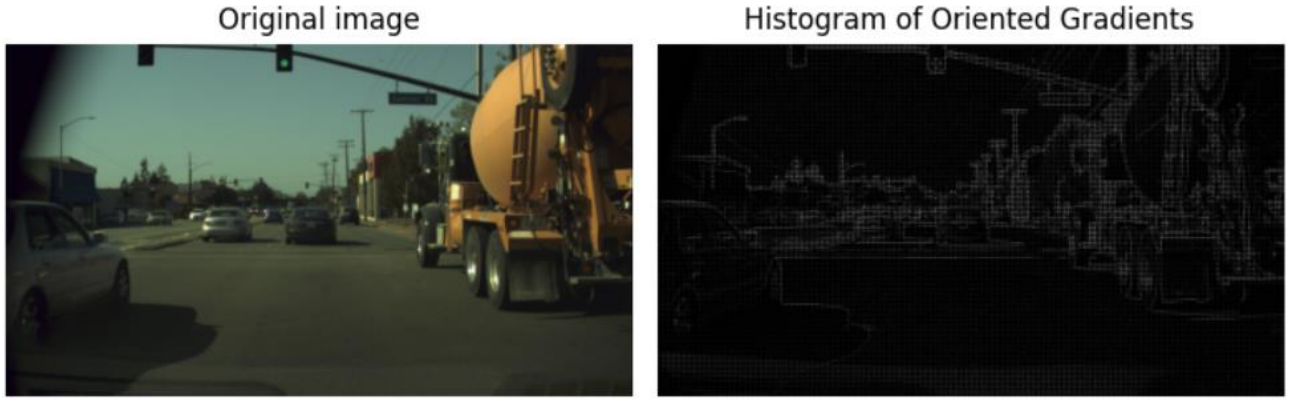
[Figura 5.5 – Pseudocod ce descrie construirea histogramelor pentru RGB și HSV]

În figura de mai sus, este descrisă prin pseudocod, procedura prin care se creează histograme pentru o imagine color și pentru cea din modelul de culoare HSV. A se observa în acest pseudocod că pentru calculul histogramelor s-a folosit o funcție **calculareHistogramă**. S-a ales denumirea asta doar pentru partea de pseudocod, în codul propriu-zis, această funcție fiind cea predefinită de librăria **OpenCV**, și anume **calcHist**, funcție explicate la începutul acestui sub-capitol. Această funcție a fost folosită în prima parte a problemei clasificării abordate în acest proiect, și anume clasificarea imaginilor ce conțin doar semafoare. Se va reveni cu detalii la un sub-capitol următor. Următoarea procedură ce va fi discutată este **create_hog_histogram**, ce

ca parametru calea către imaginea pentru care se va crea histograma. Această funcție va construi histograma HOG (adică histograma gradienților), iar aceasta va reprezenta un set foarte important de trăsături, ele fiind folosite în procesul de clasificare. Pentru început, la această funcție, se va citi o imagine cu ajutorul funcției predefinite **open** din librăria **Image**. La realizarea histogramei gradienților a luat parte și o funcție predefinită din librăria **Skimage** și anume:

hog(image, orientations, pixels_per_cell, cells_per_block, visualize, multichannel)

- **image –** acest parametru va reprezenta calea către imaginea pentru care se vrea a fi construită histograma gradienților.
- **orientations –** reprezintă dimensiunea histogramei, inițial ea este setată la 9 bin-uri (adică, are lungimea 9).
- **pixels_per_cell –** acest parametru descrie dimensiunea unei celule, pentru început, el va fi setat la 8x8.
- **cells_per_block –** după cum sugerează și numele, acesta va reprezenta numărul de celule de pixeli dintr-un bloc.
- **visualize –** acesta este un parametru de tip boolean pentru partea de vizualizare a histogramei.
- **multichannel –** după cum sugerează și numele, ultimul parametru va reprezenta dacă histograma gradienților va fi făcută pentru toate canalele de culoare sau nu, deci va avea o valoare booleană.



[Figura 5.6 – Crearea histogramei gradienților]

În figura de mai sus este reprezentată imaginea histogramei gradienților pentru secvența din trafic capturată în stânga. Metoda descrisă în prima parte a acestei secțiuni va crea vectorul caracteristic histogramei, în care sunt surprinși toți pixelii imaginii din stânga, dar și o imagine aferentă histogramei pentru a se vedea distribuția fiecărui pixel din imagine. Vectorul caracteristic pixelilor va conține grupuri de câte 9 valori, cuprinse toate între [0, 1], de tipul flotant, ce vor fi puse în fișiere cu extensia .csv care vor reprezenta datele de antrenare și de test pentru clasificatori.

5.5.2.3. Proceduri folosite pentru manipularea pixelilor dintr-o imagine

În acest sub-capitol se vor descrie implementări de proceduri care s-au folosit pentru manipularea pixelilor din diferite zone ale imaginii. În proiectul de față, aceste metode s-au folosit

în prima parte a aplicației, în care s-a încercat clasificarea semafoarelor cu ajutorul unor imagini în care apar doar ele. Aceste imagini au rezultat din imaginile originale din setul de date, decupând cu ajutorul pozițiilor semafoarelor (acele valori din fișierul yaml aferent proiectului) zonele de interes pentru utilizator. Pentru început se va descrie cum au fost extrase cutiile semafoarelor din fișierul yaml conținut de setul de date).

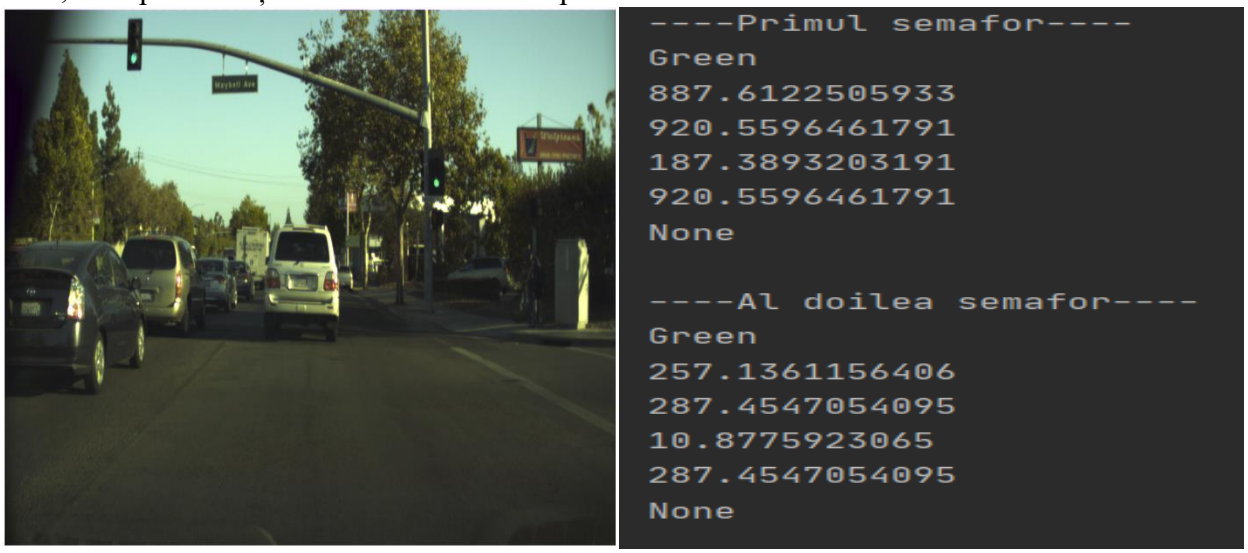
Procedura cu numele **create_a_list_of_object_traffic_lights** primește ca și argumente trei parametrii:

- **numberOfValidImages** – ce reprezintă numărul de imagini valide din fișierul yaml. Aici, ne referim la acele imagini care conțin semafoare, în capitolul 5.3. fiind descris fișierul yaml și structura acestuia. O imagine validă este acea imagine care în câmpul **boxes** are valori, adică are semafoare. Dacă **boxes** conține o listă vidă, înseamnă că imaginea nu este validă.
- **onlyTwoColors** – acest parametru se referă la imaginile care conțin semafoare cu culorile: roșu și verde, sau măcar una dintre ele. Metoda la final va returna o multitudine de liste, una dintre ele fiind o listă în care se află toate semafoarele ce conțin culorile menționate mai sus.
- **checker** – asemănătoare cu clasica variabilă "ok" acest parametru validează funcționalitatea specifică a metodei. Se va descrie în detaliu imediat pentru ce a fost folosită această valoare.

Cu ajutorul checker-ului explicat imediat mai sus, se va executa una dintre cele două funcționalități: extragerea tuturor semafoarelor cu ajutorul fișierului .yaml, sau extragerea doar a semafoarelor care au ca și culoare reprezentativă verdele. În amândouă direcții cursul metodei va fi aproximativ același, dar diferă puțin la volumul de date stocate prin această metodă. Se va deschide mai întâi fișierul .yaml, aferent modului de execuție (vor fi două fișiere, unul din directorul setului de date descărcat de pe internet și celălalt, format de mine, pentru o mai bună precizie la culegerea semafoarelor pentru culoarea verde). Pentru această deschidere de fișier s-a folosit funcția predefinită open. Tot cu ajutorul unei funcții predefinite s-au încărcat datele găsite în acele fișiere și s-au introdus într-o variabilă cu numele **documents**, funcția numindu-se **yaml.safe_load** și se găsește în librăriile aferente manipulării acestor tipuri de fișiere. Cu trei bucle **for**, s-au parcurs datele de acolo și s-au extras doar acele porțiuni reprezentative dintr-un **boxes**. Dacă acest câmp era gol (adică avea ca și valoare o listă vidă), se trecea la elementul următor din document, dacă nu, s-au parcurs toate semafoarele din el. Din fiecare semafor, s-au extras pe rând valorile de la următoarele câmpuri: **label**, **occluded**, **x_max**, **x_min**, **y_max**, **y_min** și **path-ul**. Cu aceste valori s-au format obiecte de tipul clasei TrafficLight, adică s-au format semafoare, ele fiind introduse apoi în liste aferente execuției curente. Adică, dacă checker-ul de la început, avea valoarea 0, se va forma o listă în care vor fi toate semafoarele cu culorile roșu și verde. Dacă checker-ul avea valoarea 1, vom avea o listă doar pentru semafoarele cu culoarea verde. În plus, se vor număra câte imagini valide, adică cu semafoare s-au găsit. La finalul acestei metode se vor returna rezultatele obținute.

Următoare metodă care se va discuta în acest sub-capitol se ocupă cu extragerea semafoarelor din fișierul **.yaml** din setul de date. Procedura se numește **extract_specific_box** și primește ca parametri path-ul imaginii de unde să se extragă semafoarele, un checker care va avea

câteva funcționalități și tipul semaforului dorit a fi extras. În prima parte a acestei metode s-a ales fișierul de unde să se extragă semafoarele. Pentru fiecare tip de semafor în parte avem câte un fișier separat cu aceste obiecte de interes, atât pentru partea de antrenare cât și pentru test. Deci, o să avem într-un final patru instrucțiuni de decizie mari (câte una pentru fiecare tip de semafor în parte, ultima fiind pentru cazul în care se vrea a fi clasificate toate semafoarele în același timp) și câte două instrucțiuni de tipul -if- prin care se alege documentul aferent ori datelor de antrenament, ori acela pentru partea de testare. Pe mai departe, se va parcurge documentul curent cu ajutorul a trei bucle -for-, asemănător modului prin care a fost creată metoda explicată înainte. Dacă câmpul boxes conține semafoare, se vor extrage datele aferente acestora, se va forma un obiect de tipul clasei **TrafficLight** și se va introduce acest obiect într-o listă cu semafoare. Această listă de la final, va reprezenta și rezultatul returnat de procedura curentă.

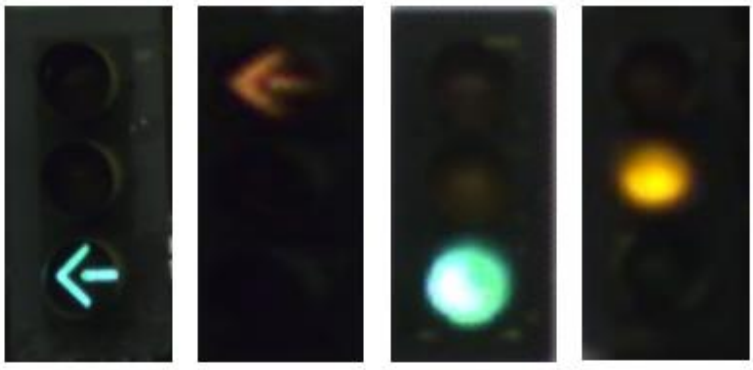


[Figura 5.7 – Rezultatul metodelor de extragere a semafoarelor din yaml]

În figura 5.7, se poate observa rezultatul metodelor explicate mai sus. Se poate observa că pentru imaginea din stânga, au fost observate două semafoare cu culoarea verde, iar pozițiile lor sunt în formatul virgulei flotante.

În continuare se va explica procedura prin care s-au decupat semafoarele din imaginile originale de se găsesc în setul de date al proiectului. Aceasta, are un antet cu nume sugestiv, și anume, **crop_images_from_list_of_objects**. Primește ca și argumente următorii parametrii: o listă în care se vor afla toate semafoarele din imagini și o variabilă **onlyTwoColors**, care va specifica dacă decupările se vor face doar pe acele imagini de conțin semafoare în două culori. Pentru început, se va folosi una dintre cele două metode explicate anterior, pentru a putea extrage toate semafoarele de interes pentru utilizator. Această listă se va trimite ca și parametru la procedura în curs de explicare. În prima parte a metodei se vor defini câteva liste și variabile care vor ajuta pe parcursul ei. După care, cu o buclă -for- se vor parcurge toate obiectele din lista de semafoare și se va sări peste acele obiecte care nu au o cale absolută validă. Dacă este ok până acum, se va citi imaginea în care se găsește semaforul curent și cu ajutorul altor variabile se vor stoca pozițiile la care se găsește acel semafor. Cu ajutorul funcției din librăria **Image** și anume **.crop** se va decupa imaginea după pozițiile semaforului. Următorul pas este de a salva rezultatul,

iar pentru acest lucru s-a folosit funcția predefinită **save** la care s-a trimis ca și parametru calea absolută a locației unde să se salveze imaginea rezultată în urma decupării. În cazul în care utilizatorul transmite prin parametrul **onlyTwoColors**, valoarea 1, asta înseamnă că se vor decupa doar acele semafoare care au culoarea roșie sau verde. Pentru acest caz, s-a folosit o instrucțiune de decizie în care s-a repetat salvarea imaginii dar în altă locație. Pentru o mai bună observare a rezultatelor, s-au salvat semafoarele decupate, ca și obiecte în diferite liste, care la final au fost returnate din procedură.



[Figura 5.8 – Rezultatul decupării semafoarelor]

În figura 5.8 se pot observa câteva semafoare decupate cu ajutorul acestei metode. Deoarece aceste obiecte se găsesc la poziții diferite în imagini, unele fiind la mare depărtare de prim-planul văzut de utilizator, foarte multe imagini vor fi la o dimensiune mică de pixeli. Această metodă a fost folosită la clasificarea semafoarelor în cazul în care am doar imagini de acest tip.

Următoarea procedură ce va fi discutată s-a folosit pentru extragerea valorilor medii de pe fiecare canal RGB în parte, dintr-o imagine. Această funcție are un antet, cu nume sugestiv, și anume, **extract_average_value_from_each_RGBchannel**, și primește ca și parametru calea absolută pentru imaginea la care se va face această prelucrare a pixelilor. În primul rând, se va citi imaginea cu ajutorul funcției predefinite **open** din librăria **Image**, iar pe urmă se va forma o listă a pixelilor sub format de vector cu ajutorul tot a unei funcții predefinite **np.asarray**, din librăria **Numpy**. În continuare, ne putem imagina că se vrea a se face media valorilor a trei vectori, pentru că așa vor fi privite canalele de roșu, verde și albastru din imagine. Asta înseamnă că pentru început vom avea nevoie de trei variabile, care vor reprezenta sumele valorilor din aceste canale. Se va parcurge imaginea cu două bucle -for- (de fapt, se va parcurge matricea de liste formată anterior) și s-a pus constrângerea ca un pixel curent să aibă valoarea canalelor [100, 100, 100], pentru a putea restrânge imaginea la doar anumite valori de pixeli căutate. Dacă pixelii găsiți au valorile de roșu, verde și albastru mai mari decât cele din constrângerea amintită anterior, se vor calcula sumele respective și se va contoriza pixelul găsit pentru a ști la final numărul total de pixeli găsiți. În ultima parte a acestei metode, se vor calcula valorile medii pentru fiecare canal în parte, cu ajutorul sumelor calculate anterior și a numărului de pixeli găsiți. Aceste valori medii se vor returna la final din procedură.

Această funcție este descrisă mai jos în figura 5.9, unde este prezentat pseudocodul ei.

```
extract_average_value_from_each_RGBchannel(imagePath): procedură
        citireImage: cu ajutorul unei funcții predefinite
        transformareImagineInVector: cu ajutorul unei funcții predefinite

        sumRedChannel, sumGreenChannel, sumBlueChannel <- 0
        numberOfValidPixels <- 0

        pentru fiecare i mergând pe înălțimea, execută:
                pentru fiecare j mergând pe lățime, execută:
                        dacă pixelValue > 100, atunci:
                                sumRedChannel <- sumRedChannel + pixel_redChannel
                                sumGreenChannel <- sumGreenChannel + pixel_greenChannel
                                sumBlueChannel <- sumBlueChannel + pixel_blueChannel

                                numberOfValidPixels <- numberOfValidPixels + 1

        dacă numberOfValidPixels > 0, atunci:
                redChannel <- sumRedChannel/numberOfValidPixels
                greenChannel <- sumGreenChannel/numberOfValidPixels
                blueChannel <- sumBlueChannel/numberOfValidPixels

        returneaza redChannel, greenChannel, blueChannel
```

[Figura 5.9 – Extragerea valorilor medii de pe canalele RGB]

În figura de mai sus a fost descrisă prin pseudocod funcția amintită curent. Se poate observa modul în care se parcurge imaginea, cu două bucle, prima cu o variabilă pe linii și a doua pe coloane. Modul de implementare al acestei funcții a dus la realizarea ultimei proceduri discutate la acest sub-capitol.

O altă metodă, asemănătoare cu cea de dinainte, se ocupă cu extragerea valorilor medii din zonele colorate ale semafoarelor (acele zone marcate cu roșu, galben sau verde). În plus, această metodă va extrage și valorile canalelor din spațiul de culoare HSV, tot din acele zone amintite anterior. Pentru început, antetul funcției este **get_pixel_feature_from_color_zone** și primește ca și parametru o listă de obiecte de tip **TrafficLight**. În prima parte a acestei proceduri se vor forma și inițializa două liste: una pentru valorile medii ale zonelor colorate și încă una pentru valorile din modelul de culoare HSV. Cu o altă variabilă **labelsFromTrafficLights** ne-am ajutat pentru a stoca toate câmpurile de tip label din **semafoare**. Pe mai departe, s-a folosit o buclă -for- pentru a putea parcurge lista de obiecte primită ca și parametru la funcție și a putea extrage toate label-urile din ele. Ideea de bază la această metodă este de a calcula acele liste de valori medii și HSV pentru cât mai multe imagini posibile. Pentru asta, cu o altă buclă -for- s-au parcurs toate label-urile din lista aferentă lor (de fapt, este același lucru cu parcurgerea semafoarelor) și cu ajutorul unei variabile **imageNumber** s-a citit fiecare imagine din directorul cu imagini decupate în care am doar semafoare cu culori de roșu și verde. Această citire s-a pus într-o structură de tip **try-catch** pentru a ne asigura că nu se primește vreo eroare. În alte două variabile, **width** și **height** s-au stocat dimensiunile imaginii curente ce s-a citit. Respectând același principiu pe care s-a mers și la implementarea procedurii explicate anterior s-au calculat valorile medii pentru fiecare canal din spațiul RGB. În plus, față de metoda anterioară, aici extragem și valorile hue, saturation și value pentru pixelii aflați. Pentru aceste calcule s-a folosit metoda explicată la sub-capitolul 5.5.2.1. și anume **convert_rgb_values_to_hsv_values**. Trebuie menționat că operațiile făcute până acum în această metodă, mai exact în această buclă -for- s-au realizat atât pentru semafoarele cu culoarea verde, cât și pentru cele cu culoarea roșie, pentru acest lucru folosindu-se o instrucțiune de decizie. La finalul metodei s-au returnat rezultatele obținute.

Metodele acestea de manipulare a pixelilor s-au folosit pentru partea în care s-au clasificat semafoarele folosindu-se de imaginile decupate ale lor

Primele două metode s-au folosit și la partea a doua, în care s-a implementat și versiunea finală a aplicației și anume cea în care s-au folosit imaginile originale din setul de date.

În continuare se vor explica două metode care s-au folosit pentru manipularea pixelilor dintr-o imagine sau dintr-un set de imagini, pentru a putea evidenția doar semafoarele din acestea.

Prima metodă, folosită pentru extragerea valorii maxime dintre patru puncte, a apărut în contextul în care trebuia găsită o soluție pentru pregătirea clasificatorilor într-una din variantele de implementare ale aplicației. Metoda, care are antetul **get_maxim_value_from_four_points** și primește ca și parametri calea imaginii pentru care se caută acea valoare maximă și patru puncte, în sistem de coordonate XoY, ce reprezintă pozițiile între care se află semafoarele din imagine. Pentru început, se va deschide imaginea cu ajutorul funcții predefinite open, din librăria Image. După care, se va transforma această imagine într-un vector (de fapt matrice) care va conține toate valorile pixelilor. După modelul calculării maximului dintr-un vector, se va calcula valoarea maximă și din această imagine. De menționat, variabila maxValue, care va reprezenta maximul din imagine, se va inițializa cu valoarea [-1, -1, -1], pentru a se putea actualiza în timpul parcurgerii imaginii. Cu două bucle -for-, se va parcurge imaginea, dar nu până la capătul dimensiunilor ei, ci în anumite intervale de puncte, și anume: cu primul -for- se va parcurge pe linii în intervalul [yMin, yMax], iar cu al doilea -for- se vor parcurge coloanele în intervalul [xMin, xMax]. Cu ajutorul unei instrucțiuni de decizie, se va compara fiecare valoare a pixelilor din imagine cu valoarea maximului curent (adică cu maxValue). Dacă găsim un pixel ce are valoarea mai mare decât cea a maximului, se va actualiza valoarea variabilei din urmă. La finalul metodei se va returna acest maxim.

Următoarea metodă, care folosește funcția descrisă mai sus, creează imagini cu ajutorul cărora să se poată clasifica semafoarele. Această metodă, folosește un nume sugestiv, și anume, **create_image_with_traffic_lights_from_points_coordinates_forAll** și primește ca și argumente doar o valoare ce reprezintă o limită de imagini pentru care să se aplice algoritmul. Pentru început, în această procedură, se va defini o variabilă care va reprezenta calea absolută a directorului în care se găsesc imaginile. Cu o buclă -for- se vor parcurge toate imaginile din acel director. Se va verifica cu o instrucțiune de decizie dacă s-a ajuns la limita de imagini care să fie prelucrate, iar cu o alta dacă fișierul de se prelucrează curent este o imagine, adică dacă are extensia .png. Dacă imaginea este validă, se va salva calea absolută a acesteia într-o variabilă, iar cu ajutorul metodei explicate la sub-capitol 5.5.2.3. și anume **extract_specific_box**, s-au extras semafoarele aferente acelei imagini din fișierul .yaml (s-a extras de fapt o listă ce conține toate semafoarele). S-a definit o variabilă pentru valoarea maximă găsită între coordonatele fiecărui semafor din imagine. Cu o altă buclă -for- s-a parcurs lista de semafoare găsită anterior și s-au extras pozițiile în sistem de coordonate XoY, pentru a fi prelucrate ulterior. În această buclă s-a apelat procedura explicată mai sus, adică cea în care s-a găsit pixelul cu valoarea maximă între patru puncte din imagine. Următorul este de a forma o imagine cu ajutorul acestor valori maxime găsite, pentru fiecare semafor în parte. Folosind funcția predefinită **new** din librăria Image, s-a creat o imagine în format RGB, cu toți pixelii setați pe negru (adică [0, 0, 0]). S-au folosit în continuare trei bucle -for-: primele două pentru parcurgerea imaginii create anterior, iar ultima pentru parcurgerea listei

de semafoare găsită cu ajutorul funcției **extract_specific_box**. S-a luat semaforul curent din listă și cu ajutorul a trei instrucțiuni de decizie în care s-a verificat tipul semaforului, toți pixelii ce se află între pozițiile semafoarelor s-au setat pe valoarea acelui pixel maxim aferent fiecărui semafor în parte. La final, s-a salvat imaginea formată cu ajutorul funcției predefinite **save** și s-a trecut la prelucrarea imaginii următoare.

[Figura 5.10 – Evidențierea semafoarelor din imagine folosind valoarea pixelului maxim]

Figura de mai sus, reprezintă rezultatul celor două proceduri descrise în ultima parte al acestui sub-capitol. Se pot observa în imaginea din dreapta, toate semafoarele ce se regăsesc și în stânga, doar că ele sunt evidențiate aici cu valoarea maximă a pixelului ce se găsește între coordonatele lor. Se poate observa că aceste culori nu sunt foarte reprezentative pentru semafoare, verdele de exemplu, având nuanțe de albastru sau chiar de gri închis.

5.5.3. Soluții găsite pentru încadrarea semafoarelor prezise

În acest sub-capitol se vor acoperi câteva proceduri folosite pentru partea finală a aplicației, adică pentru cea în care va trebui să se observe rezultatele obținute. Se vor discuta aici metode prin care se extrag valorile maxime ale pixelilor aflați între diferite puncte, de creare a unor imagini binare în care sunt evidențiate semafoarele sau metode prin care se încadrează semafoare pe imaginile originale luate din setul de date.

Pentru început se va descrie o metodă folosită pentru extragerea punctelor din fișierele .csv rezultate în urma predicțiilor. Ca să dau un context asupra a ceea ce se va discuta mai departe, clasificatorii explicați în ultima parte a acestui capitol vor genera ca și rezultate în urma predicțiilor efectuate de ei, câteva fișiere cu extensia .csv. Aceste fișiere vor conține trăsăturile acelor puncte (pixeli) din imagini care vor fi considerați de ei ca fiind parte din semafoare. Deci, în această metodă se vor prelucra datele din fișiere ce conține trăsături prezise. În primul rând, numele acestei metode este: **detect_the_array_of_points_from_predicted_csv**, și primește ca și argumente doi parametrii: calea absolute către fișierul .csv ce va conține punctele prezise și o variabilă pentru a specifica tipul semaforului pentru care se vrea detecția acestor puncte. Se va deschide acest fișier primit ca parametru cu ajutorul unor funcții predefinite ce se folosesc în manipularea acestor tipuri de fișiere (funcția pentru deschidere, se va numi **read_csv**). Datele din acest fișier vor fi sub formă vectorială și s-a ales extragerea doar primei coloane, în care vor fi puse pozițiile patch-urilor (se va reveni cu detalii asupra acestui aspect la sub-capitolul 5.6.). În funcție de tipul semaforului, folosindu-se aici patru instrucțiuni de decizie pentru fiecare tip de semafor în parte și pentru cazul în care clasificarea a fost făcută pentru toate și folosindu-se și variabila dată ca parametru la funcție

pentru specificarea semaforului, se va deschide un singur fișier cu extensia .csv care va conține pozițiile fiecărui pixel în parte (se va reveni și aici cu detalii, în sub-capitolul 5.6., unde se va descrie structura fiecărui fișier de acest tip folosit). Din acest fișier, se vor reține pozițiile pixelilor din imagine într-o listă definită anterior. Partea finală a acestei metode cuprinde parcurgerea listei extrase anterior și prelucrarea ei. Aici, s-a folosit o buclă -for- cu care s-a parcurs toată lista. S-a reținut prima valoare din fiecare linie în parte din listă (pentru că va fi o listă de liste, fiecare listă în parte având pozițiile pixelilor din sub-matricea respectivă). Cu o altă buclă -for- se va parcurge și lista de poziții din fișierul prezis de clasificator, reținându-se și aici fiecare poziție în parte în altă variabilă. Dacă cele două poziții reținute până acum sunt egale, înseamnă că avem un patch de poziții prezis bine, iar acesta va fi reținut într-o altă listă. Aceasta din urmă se va returna la finalul procedurii.

Pe aceeași direcție a fost implementată și metoda **clean_the_predicted_csv** care primește ca și parametrii calea către fișierul cu extensia .csv prezis și tipul semaforului pentru care se vor face următoarele prelucrări. La fel, se va deschide fișierul cu trăsăturile prezise și se vor returna liste cu pozițiile din patch-urile bune. În plus față de metoda anterioară, la această procedură se va curăța fișierul .csv inițial astfel încât să se șteargă acele trăsături ale zonelor din imagine care nu prezintă un interes.

În ultima parte a acestui sub-capitol se vor discuta metode prin care s-au afișat anumite rezultate de imagini prelucrate cu punctele rezultate din primele două metode.
Următoarele două metode au aceeași funcționalitate, dar diferă prin eficiență. Prima, are numele create_image_from_classifier_image_result și primește ca și parametrii o listă de puncte ce va fi rezultată printr-o metodă explicată anterior (acea metodă care extrage pozițiile pixelilor prezișI de către clasificator). Se creează o imagine RGB, care se umple cu pixeli negri (adică, [0, 0, 0]). Apoi, cu două bucle -for- se va parcurge această imagine, iar cu o a treia, se va parcurge lista de poziții prezise. Aceste poziții fiind în format de listă de caractere, de forma [x, y], unde x și y sunt inițial valori de tipul **String**, se va folosi funcția **findall** care va extrage atât x-ul, cât și y-ul din această listă. Pe urmă, se vor converti aceste valori la tipul întreg și dacă indicii primelor două bucle -for- reprezintă de fapt pozițiile din listă (adică sunt egali cu x-ul și y-ul din listă), atunci pe acea poziție din imagine se pune valoarea pixelului alb, adică [255, 255, 255], altfel se menține pixelul negru

În figura de mai jos, se poate observa pseudocodul de la a două metodă ce va fi explicată ulterior.

```
create_first_image_result_from_classifier_predicted_points(points): procedură
        image <- create_rgb_image: funcție predefinită

        pentru fiecare x din lista de puncte (points), execută:
                i, j <- extragerePuncteDinLista (funcție predefinită)
                image(i)(j) <- 255 (valoarea pixelului alb)

        afisareImagine: funcție predefinită
```

[Figura 5.11 – Pseudocodul procedurii pentru formarea imaginii binare din punctele prezise]

Procedura descrisă prin pseudocodul din figura 5.11, este implementată cam pe același principiu ca metoda anterioară, dar este mult mai simplă. Aceasta, având numele de mai sus,

primește și ea o listă de puncte, dar spre deosebire de prima metodă, aici se va parcurge doar această listă. Pe pozițiile extrase din aceasta, se va introduce un pixel alb în imaginea creată înaintea parcurgerii. La final, se va afișa imaginea rezultată.

[Figura 5.12 – Rezultatul metodelor descrise anterior]

Figura de mai sus prezintă rezultatul metodelor descrise anterior. Se poate observa cum semaforul a fost prezis (se va descrie imediat, în sub-capitolele următoare procesul clasificării) și punctele lui au fost evidențiate într-o imagine binară, pixelii albi fiind punctele prezise de clasificator.

Ultimele două metode din această secțiune, ce vor fi descrise, s-au folosit pentru încadrarea semafoarelor cu ajutorul pozițiilor prezise de către clasificator pe imaginile originale. S-au creat două proceduri: una pentru încadrarea semafoarelor în forme sferice, iar a doua pentru încadrarea lor în forme rectangulare (mai exact dreptunghiuri). Se va descrie mai detaliat a doua metodă pentru că în versiunea finală aceasta a fost folosită. Ideea din spatele acestor metode este că ele primesc o listă de puncte prezise de clasificator și manipulează pixelii din imaginea originală cu ajutorul lor. În primul rând, se citește imaginea originală cu ajutorul funcției predefinite **imread**, din librăria **OpenCV**, ce primește ca și parametru calea către aceasta. După care, s-a presupus că la modul în care acest clasificator a fost implementat se vor prezice maxim 2-3 semafoare de aceeași culoare pe imagine. Din acest fapt, s-a realizat cu ajutorul unei bucle -for- o sortare pe punctele prezise, după modul în care acestea ar fi așezate într-un sistem de coordonate XoY. Cu cât diferențele dintre poziții erau mai mari, cu atât exista posibilitatea ca în punctele prezise să se găsească un alt semafor în plus față de cel curent care se prelucrează. Astfel, după această sortare, a rezultat o listă de liste, în care fiecare listă are coordonatele specifice fiecărui semafor găsit. Mai rămâne acum doar să se încadreze acestea. Pentru partea finală a metodei, în care se prelucrează punctele găsite și sortate, s-a mers pe două variante: prima în care semafoarele se încadrează în cercuri și a doua, cea în care semafoarele se încadrează în dreptunghiuri. Astfel, s-a parcurs cu o buclă -while- lista de liste formată anterior și s-a luat fiecare listă în parte găsită în aceasta. Pentru fiecare dintre ele, s-au definit patru variabile: xMinimPosition, xMaximPosition, yMinimPosition, yMaximPosition. Acestea reprezintă pozițiile minime și maxime, în coordonate XoY. Într-o buclă -for- s-au parcurs toate punctele găsite în lista curentă ce se prelucrează și cu ajutorul unor

instrucțiuni de decizie, se actualizează valorile variabilelor definite anterior. Pentru final, dacă s-a ales încadrarea cu forme sferice, s-au calculat două valori de mijloc, una pentru axa OX și una pentru axa OY, iar cu ajutorul funcției **cv2.circle** din OpenCV, s-a trasat un cerc de rază 50, pornind din centrul calculat anterior. Cercul are circumferința de culoare neagră și grosimea acesteia de valoare 5. Dacă s-a ales trasarea unui dreptunghi, cu acele patru valori minime și maxime, definite anterior, se vor forma patru puncte. Între aceste puncte se vor trage linii, folosindu-se funcția predefinită **cv2.line**, din librăria OpenCV. Aceste două proceduri, se numesc: **create_image_from_classifier_image_result_final_inCircle**, care încadrează semafoarele în cercuri și a doua **create_image_from_classifier_image_result_final_inRectangle**, care încadrează semafoarele în dreptunghiuri.

[Figura 5.13 – Încadrarea semafoarelor în diferite forme geometrice]

În figura de mai sus, se poate observa cum semafoarele din imagine au fost încadrate, cu ajutorul celor două metode descrise anterior, în cercuri (imaginea din stânga) sau dreptunghiuri (imaginea din dreapta).

5.6. Formarea seturilor de antrenare și de test

La acest sub-capitol se vor discuta principalele proceduri folosite pentru implementarea fișierelor ce se vor da clasificatorilor, atât pentru partea de antrenare, cât și pentru cea de test. Se va discuta modul în care s-au format sub-matrici (patch-uri) din imaginile originale, folosind trăsăturile extrase cu ajutorul metodelor folosite la sub-capitolele anterioare, dar și modul în care s-au combinat toate aceste fișiere cu extensia .csv, obținute, cu scopul de a forma setul de date de antrenare, cât și de test.

5.6.1. Împărțirea imaginilor în patch-uri

Aici, se va descrie modalitatea prin care o imagine, sub formă matriceală, a fost împărțită în sub-matrici, acestea din urmă având denumirea tehnică de patch-uri. Pentru această operațiune s-au creat mai multe proceduri, pentru diferite stagii ale proiectării proiectului, toate făcând în schimb același lucru, dar folosind valorile pixelilor într-un mod diferit. Se va detalia mai pe larg metoda prin care se împarte matricea unei imagini în patch-uri, fiecare dintre acestea conținând principalele trăsături alese pentru lucrul cu clasificatorii din următorul capitol.

Procedura care se va discuta are antetul **create_csv_with_patches_longVersion** și primește ca și argumente patru valori: calea imaginii pentru care se face această operație, o variabilă de tip OK (checker se va numi) prin care se specifică pentru ce tip de seturi de date se

face prelucrarea imaginii (dacă pentru setul de date de antrenament, sau pentru cel de test), o variabilă pe nume csvNumber prin care se va specifica numărul de ordine al fișierului cu extensia .csv format (se vor prelucra toate imaginile – din setul de antrenament sau cel de test – folosind aceeași metodă, adică se va reapela metoda asta de un număr finit de ori într-o altă metodă importantă discutată la sub-capitolul următor) și o variabilă prin care se specifică tipul semaforului (roșu, verde, galben sau all, pentru clasificarea tuturor tipurilor de semafor în același timp). În prima parte a acestei metode se va forma lista cu indicii fiecărei coloane din fișierul .csv. Pentru acest lucru, s-a folosit o buclă -for- în care s-a parcurs un interval, de la 0 la 200, iar în lista aferentă acestor indici menționați mai sus s-a introdus fiecare x din această buclă. Următorul pas este de a citi imaginea, pentru această operație folosindu-se funcții predefinite menționate și la alte metode ce au fost explicate. Pentru o mai bună viziune asupra modului în care a fost implementat acest algoritm de împărțire în sub-matrici se poate observa pseudocodul de mai jos:

```
create_patches(image): procedură
        height, width <- image_dimensions

        x <- 0
        y <- 0
        cât timp nu s-a atins limita de imagini procesate, execută:
                linesNumber <- 0

                pentru fiecare i ce parcurge liniile matricii imaginii, execută:
                        dacă linesNumber = 5, atunci:
                                ieși

                        countPosition <- 0
                        pentru fiecare j ce parcurge coloanele matricii imaginii, execută:
                                dacă countPosition = 5, atunci:
                                        ieși
                                altfel
                                        extragere_trăsături: proceduri explicate anterior
                                        formare_vector_ce_se_va_pune_în_csv

                                        countPosition <- countPosition + 1

                        linesNumber <- linesNumber + 1

                        lipire_vector_cu_trăsături_în_csv

                        dacă j=dimensiunea_matricii_pe_coloane, atunci:
                                x <- x + 5
                                y <- 0
                        altfel
                                y <- y + 5

                        dacă i = dimensiunea_matricii_pe_linii - 1  și j=dimensiunea_matricii_pe_coloane - 1, atunci:
```

[Figura 5.14 – Pseudocodul formării sub-matricelor imaginii]

Pseudocodul descris de figura 5.14 reprezintă ideea din spatele algoritmului de descompunere a matricei imaginilor în sub-matrici. Ca și constrângere, acest algoritm va merge doar pe matrici care au produsul (Lățime x Înălțime) % 5, el construind doar patch-uri de dimensiunea 5x5. Ca o viitoare îmbunătățire ce o poate avea această procedură, este adăugarea ca și parametru a unei variabile prin care să se specifice dimensiunea dorită a sub-matricelor. Pentru observarea clară a algoritmului s-a dorit realizarea unui pseudocod în care a fost prezentat doar el, ca și procedură separată. În realitate, această separare în sub-matrici a făcut parte dintr-un proces mai mare și nu o fost realizată o procedură specială doar ea. S-a folosit algoritmul prezentat mai

sus în implementarea unor alte metode de construcție a fișierelor cu extensia .csv ce conțin trăsăturile. Revenind la metoda **create_csv_with_patches_longVersion**, pentru că aceasta se explica, după ce s-a împărțit matricea imaginii curentă în sub-matrici va urma partea salvării fișierului Excel. Ca și observație, pentru procedura curentă, la introducerea trăsăturilor în fișiere .csv, s-au introdus următoarele: valorile modelelor RGB și HSV, valoarea luminozității și valoarea medie a fiecărui canal din spațiul HSV, toate acestea făcându-se pentru fiecare pixel în parte din fiecare sub-matrice. Pentru salvarea fișierelor rezultate, s-au folosit variabile din antetul funcției, și anume, cea care specifică tipul semaforului și cea pentru tipul setului de date pentru care se fac fișierele Excel. Cu ajutorul a patru instrucțiuni de decizie s-au acoperit toate tipurile de semafor, iar în lor, s-au mai realizat încă un set de câte alte două instrucțiuni de decizie pentru tipul setului de date. Salvarea propriu-zisă se face cu ajutorului funcției predefinite save. Diferența între toate aceste salvări o face calea absolută specificată ca și parametru la această modalitate de salvare.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA
Position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
11865	0.1098	0.12157	0.0902	82.5	25.8065	0.04767	0.10719	36.118	0.11373	0.12941	0.0902	84	30.303	0.05075	0.11111	38.1179	0.11765	0.12941	0.09412	80	27.2727	0.05075	0.11373	35.7745	0.12157	0.14118
16492	0.05882	0.05882	0.07451	240	21.0526	0.02922	0.06405	87.0273	0.05882	0.0549	0.07451	252	26.3158	0.02922	0.06275	92.7817	0.05882	0.05098	0.07451	260	31.5789	0.02922	0.06144	97.2027	0.05882	0.05098
28014	0.18039	0.2	0.15686	87.2727	21.5686	0.07843	0.17908	36.3066	0.18039	0.18824	0.16863	84	10.4167	0.07382	0.17908	31.4968	0.18039	0.19608	0.15686	84	20	0.07689	0.17778	34.6923	0.18039	0.19608
9530	0.15686	0.18039	0.15686	120	13.0435	0.07074	0.16471	44.3714	0.15686	0.17647	0.15686	120	11.1111	0.0692	0.1634	43.7268	0.15686	0.18039	0.14902	105	17.3913	0.07074	0.16209	40.8207	0.15686	0.18039
1549	0.03922	0	0.06275	277.5	100	0.02461	0.03399	125.842	0.03922	0.00392	0.06275	276	93.75	0.02461	0.03529	123.258	0.03922	0.00392	0.06275	276	93.75	0.02461	0.03529	123.258	0.03922	0.00392
5333	0.06667	0.08235	0.07451	150	19.0476	0.0323	0.07451	56.36	0.07843	0.0902	0.08235	140	13.0435	0.03537	0.08366	51.0263	0.08627	0.09804	0.08627	120	12	0.03845	0.0902	44.0128	0.09412	0.09804
15924	0.06667	0.07843	0.07059	140	15	0.03076	0.0719	51.6769	0.06667	0.07843	0.07059	140	15	0.03076	0.0719	51.6769	0.06667	0.07843	0.07059	140	15	0.03076	0.0719	51.6769	0.07059	0.08235
30753	0.15294	0.16863	0.15294	120	9.30232	0.06613	0.15817	43.1228	0.15686	0.16863	0.14902	96	11.6279	0.06613	0.15817	35.898	0.16078	0.17255	0.1451	85.7143	15.9091	0.06767	0.15948	33.897	0.16078	0.17255
15005	0.13725	0.15294	0.12941	100	15.3846	0.05998	0.13987	38.4815	0.12941	0.13333	0.11765	75	11.7647	0.05229	0.1268	28.939	0.12549	0.13725	0.1098	85.7143	20	0.05383	0.12418	35.256	0.12549	0.13725
1626	0.88627	1	0.80392	94.8	19.6078	0.39216	0.89673	38.2667	0.88235	1	0.79608	94.6154	20.3922	0.39216	0.89281	38.4666	0.88235	1	0.80392	96	19.6078	0.39216	0.89542	38.6667	0.87451	1
9337	0.49412	0.61569	0.51765	131.613	19.7452	0.24145	0.54248	50.5332	0.58039	0.66275	0.56078	108.462	15.3846	0.2599	0.60131	41.3687	0.65882	0.78431	0.58824	98.4	25	0.30757	0.67712	41.2359	0.7098	0.79608
26673	0.06275	0.06667	0.07059	210	11.1111	0.02768	0.06667	73.7129	0.06275	0.06667	0.07059	210	11.1111	0.02768	0.06667	73.7129	0.06275	0.06667	0.07059	210	11.1111	0.02768	0.06667	73.7129	0.06275	0.06667
20363	0.06667	0.06275	0.07843	255	20	0.03076	0.06928	91.6769	0.05882	0.05882	0.06667	240	11.7647	0.02614	0.06144	83.9303	0.05882	0.0549	0.06667	260	17.6471	0.02614	0.06013	92.5577	0.05882	0.0549
5829	0.09804	0.12941	0.11765	157.5	24.2424	0.05075	0.11503	60.5977	0.09804	0.12941	0.13333	186.667	26.4706	0.05229	0.12026	71.0632	0.09412	0.12941	0.12941	180	27.2727	0.05075	0.11765	69.1078	0.09412	0.12549
9422	0.17647	0.13333	0.17647	300	24.4444	0.0692	0.16209	108.171	0.1098	0.11373	0.13333	230	17.6471	0.05229	0.11895	82.5665	0.09412	0.10196	0.0902	100	11.5385	0.03998	0.09542	37.1928	0.0902	0.09804
30974	0.17647	0.19608	0.16471	97.5	16	0.07689	0.17908	37.859	0.17255	0.19608	0.16863	111.429	14	0.07689	0.17908	41.8352	0.17647	0.19216	0.17255	108	10.2041	0.07536	0.18039	39.4265	0.17647	0.19608
5835	0.08627	0.12941	0.12157	169.091	33.3333	0.05075	0.11242	67.4917	0.08627	0.10588	0.11765	202.5	26.6667	0.04614	0.10327	76.4043	0.08235	0.09804	0.10196	192	19.2308	0.03998	0.09412	70.4236	0.07451	0.0902
13123	0.05882	0.04314	0.07843	266.667	45	0.03076	0.06013	103.899	0.05882	0.04314	0.07843	266.667	45	0.03076	0.06013	103.899	0.05882	0.04314	0.07843	266.667	45	0.03076	0.06013	103.899	0.05882	0.04706
490	0.08627	0.09412	0.04706	70	50	0.03691	0.07582	40.0123	0.0902	0.10588	0.03137	72.6316	70.3704	0.04152	0.07582	47.6812	0.0902	0.09804	0.04314	68.5714	56	0.03845	0.07712	41.5366	0.08627	0.09412
16964	0.35686	0.58039	0.77255	207.736	53.8071	0.30296	0.56993	87.282	0.22745	0.41176	0.5451	205.185	58.2734	0.21376	0.39477	87.8908	0.1451	0.20392	0.27451	212.727	47.1429	0.10765	0.20784	86.6593	0.11765	0.18431
11126	0.15294	0.16471	0.16078	160	7.14286	0.06459	0.15948	55.7358	0.15686	0.16078	0.17647	228	11.1111	0.0692	0.16471	79.7268	0.15294	0.17255	0.16471	156	11.3636	0.06767	0.1634	55.8104	0.15294	0.18824
29439	0.38824	0.38039	0.17255	57.8182	55.5556	0.15225	0.31373	37.842	0.39216	0.37647	0.19608	55.2	50	0.15379	0.32157	35.1179	0.39216	0.37647	0.18431	55.4717	53	0.15379	0.31765	36.2085	0.39216	0.37647
13187	0.19216	0.24706	0.23529	167.143	22.2222	0.09689	0.22484	63.154	0.17647	0.29804	0.18431	123.871	40.7895	0.11688	0.21961	54.9258	0.18824	0.23529	0.24314	188.571	22.5806	0.09535	0.22222	70.4158	0.17255	0.25098
5597	0.37647	0.46667	0.32941	99.4286	29.4118	0.18301	0.39085	43.0078	0.36863	0.44314	0.33725	102.222	23.8938	0.17378	0.38301	42.0966	0.35686	0.42745	0.33333	105	22.0183	0.16763	0.37255	42.3953	0.34902	0.42353
22047	0.78039	0.85098	0.63137	79.2857	25.8064	0.33372	0.75425	35.142	0.72157	0.8549	0.56078	87.2	34.4037	0.33526	0.71242	40.6463	0.66667	0.77255	0.52549	85.7143	31.9797	0.30296	0.6549	39.3323	0.59608	0.67843
13372	0.13725	0.14902	0.14118	140	7.89474	0.05844	0.14248	49.3177	0.14118	0.15294	0.13333	96	12.8205	0.05998	0.14248	36.2935	0.14118	0.15686	0.13725	108	12.5	0.06151	0.1451	40.1872	0.14118	0.15686
27986	0.12549	0.1451	0.16471	210	23.8095	0.06459	0.1451	77.958	0.12157	0.1451	0.15686	200	22.5	0.06151	0.14118	74.1872	0.12157	0.1451	0.15294	195	20.5128	0.05998	0.13987	71.8576	0.12157	0.14118
11617	0.10196	0.13333	0.09804	113.333	26.4706	0.05229	0.11111	46.6187	0.1098	0.13333	0.11765	140	17.6471	0.05229	0.12026	52.5664	0.11373	0.12549	0.10196	90	18.75	0.04921	0.11373	36.2664	0.10588	0.11765
31492	0.17647	0.19608	0.14902	85	24	0.07689	0.17386	36.359	0.18039	0.19608	0.16078	86.6667	18	0.07689	0.17908	34.9145	0.18039	0.19608	0.16471	90	16	0.07689	0.18039	35.359	0.18039	0.19608
6541	0.31765	0.48627	0.38039	142.326	34.6774	0.1907	0.39477	59.0646	0.32157	0.46275	0.41176	158.333	30.5085	0.18147	0.39869	63.0078	0.31373	0.48627	0.37647	141.818	35.4839	0.1907	0.39216	59.1642	0.32941	0.46667
3987	0.08627	0.08627	0.11765	240	26.6667	0.04614	0.09673	88.9043	0.08235	0.0902	0.09412	200	12.5	0.03691	0.08889	70.8456	0.08235	0.08235	0.10196	240	19.2308	0.03998	0.08889	86.4236	0.07843	0.0902
1786	0.15294	0.15294	0.09412	60	38.4615	0.05998	0.13333	32.8405	0.15686	0.1451	0.1098	45	30	0.06151	0.13725	25.0205	0.14902	0.15294	0.10196	64.6154	33.3333	0.05998	0.13464	32.6696	0.14902	0.1451
4539	0.11765	0.12941	0.11373	105	12.1212	0.05075	0.12026	39.0573	0.11373	0.11373	0.10196	60	10.3448	0.0446	0.1098	23.4631	0.1098	0.09804	0.10588	320	10.7143	0.04306	0.10458	110.252	0.1098	0.10196
14239	0.07451	0.06275	0.08235	276	23.8095	0.0323	0.0732	99.9473	0.07059	0.06275	0.06667	330	11.1111	0.02768	0.06667	113.713	0.06667	0.06667	0.05882	60	11.7647	0.02614	0.06405	23.9303	0.07451	0.06667

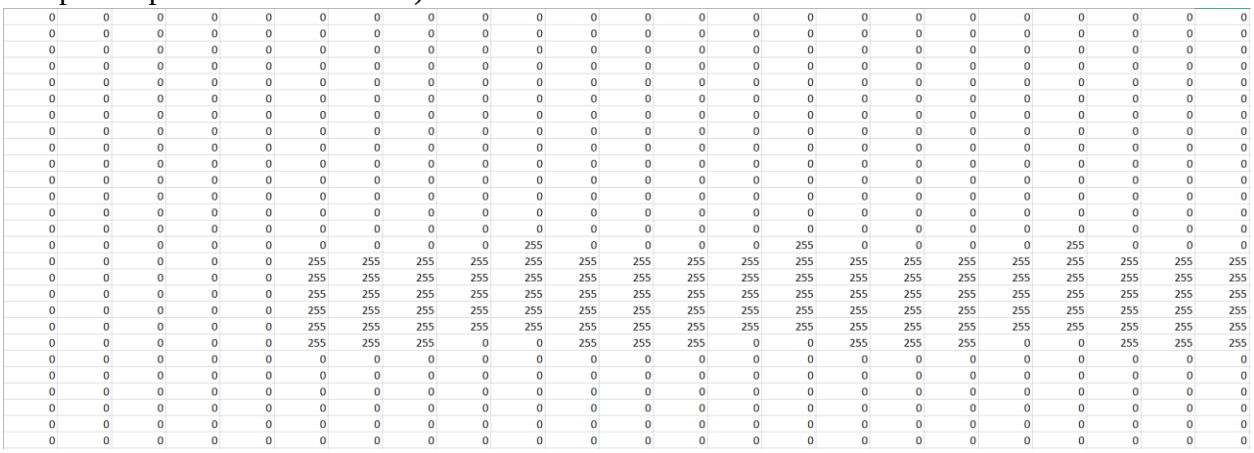
[Figura 5.15 – Fișierul Excel rezultat]

La figura de mai sus, se poate observa cum arată un fișier cu extensia .csv, rezultat printr-o metodă asemănătoare cu cea descrisă anterior. Aici, se pot observa doar primele 26 de coloane, ea având în total doar 200 (deocamdată doar atât, la realizarea fișierelor finale se vor mai adăuga câteva coloane). Prima coloană nu a fost adăugată prin metoda de mai sus, ci printr-o altă metodă ce va fi explicată la sub-capitolul următor. În rest, se pot observa multe valori de tipul flotant, reprezentând fiecare câte o trăsătură: RGB, HSV (pentru acestea două s-a luat fiecare canal în parte), valoarea luminozității și media valorilor canalelor de pe modelul de culoare HSV. S-a ales transformarea acestor valori în tipul flotant, pentru a eficientiza spațiul de memorie ocupat și pentru mai bune rezultate la clasificare (clasificatorii lucrând mai bine cu valori de tipul flotant).

Asemănătoare cu această metodă sunt următoarele:

- **create_csv_with_patches_short –** la această procedură, aproape la fel cu cea ce a fost explicată în detaliu, s-a ales a se folosit ca și trăsături doar modelul HSV. Se vor pune în fișierele Excel doar cele trei canale din acest model de culoare.
- **create_csv_with_position –** puțin diferită față de metoda discutată, acest fișier Excel se ocupă cu stocarea pozițiilor fiecărui pixel în parte din patch-urile formate. Aceste poziții vor veni fiecare în format [x, y], fiind o listă ce conține coordonatele XoY luate din parcurgerea matricei imaginii.
- **create_csv_with_binary_values –** pentru această metodă s-a ales a se pune în csv-uri, pentru fiecare pixel în parte, valorile pixelilor din intervalul [0, 255].
- **create_csv_only_for_traffic_lights –** această metodă va dori implementarea unui algoritm prin care, dintr-o imagine binară, asemănătoare cu cea din figura 28, unde sunt clar evidențiate semafoarele folosind pixeli albi, se vor pune în fișierele Excel valori de 255, pentru acele patch-uri în care există pixeli care în componența lor să aibă canale cu această valoare.

Pentru ultima metodă, se poate observa figura 5.16. Aici, avem un fișier cu extensia .csv, în care valorile celulelor din aceste sub-matrici pot lua doar două valori (0 pentru pixel negru și 255 pentru pixel cu valoarea alb).



[Figura 5.16 – Fișierul Excel pentru imagini binare]

Ca o ultimă observație, aceste fișiere pot fi diferite din punct de vedere al numărului de coloane, dar au același număr de linii (deci, avem o dimensiune totală de 36863xN unde N este numărul de coloane aferent fiecărui tip de fișier Excel în parte).

5.6.2. Soluții găsite pentru formarea fișierelor ce vor fi manipulate

La acest sub-capitol se vor descrie procedurile prin care s-au format fișierele Excel ce vor fi date clasificatorilor. S-au implementat mai multe variante de fișiere dar numai una a fost aleasă pentru a fi folosite de algoritmii de clasificare. Toate metodele în schimb merg pe aceeași idee de implementare și de asta s-a ales discutarea unei singure metode, restul fiind descrise doar prin ceea ce diferă ele.

Procedura ce se va discuta acum, se ocupă cu crearea fișierelor de clasificare și se numește **final_method_to_create_csv_for_classification_of_green_traffic_light**. Ea prelucrează doar

acele semafoare care au culoarea verde. În primul rând, aceasta va primi ca și parametru o variabilă ce specifică limita de imagini ce vor fi prelucrate. La începutul metodei, se vor defini căile absolute în care se vor găsi imaginile cu semafoare (aici, vom avea două căi: prima pentru imaginile originale în care avem semafoare, iar a doua, un director în care sunt stocate imagini binare în care sunt evidențiate doar semafoarele cu ajutorul pixelilor albi). Următorul pas este de a crea aceste imagini binare specificate mai devreme, pentru această operațiune folosindu-se procedura **create_image_with_traffic_lights_from_points_coordinates**. Un pas foarte important este acum de a forma fișiere cu extensia .csv de pe urma tuturor acestor imagini formate. Cu ajutorul unei bucle -for- se va lua fiecare imagine originală din director și se vor crea fișiere cu trăsături și cu pozițiile pixelilor, cu ajutorul sub-matricelor, aceste operațiuni fiind descrise la sub-capitolul anterior. Aceeași operațiune se va aplica și pe imaginile binare create anterior. În următoarea parte se va descrie detaliat restul procedurii. Într-o buclă -for- se va parcurge fiecare imagine în parte. Scopul acestei bucle este ca la final, să se formeze fișiere Excel cu toate aceste trăsături combinate. Pentru început, se vor stoca informațiile din aceste fișiere în diferite liste, deci, mai întâi ele se vor deschide (pentru asta folosindu-se funcția open), iar pe urmă folosind alte bucle -for-, pentru fiecare fișier cu trăsături în parte, se vor pune în liste aceste valori. Cu ajutorul funcției discutate la sub-capitolul de extrage de trăsături, și anume **create_hog_histogram**, și ajutându-mă și de eficacitatea limbajului Python de a parcurge liste, într-o listă cu numele **myListFromHog** s-au pus acele trăsături extrase din histograma gradienților. Cu ajutorul unei bucle -for- se va forma acum o listă de dimensiune 236, care va conține header-ul fiecărei coloane (sau numele fiecărei coloane): prima coloană se va numi Position și aici se vor pune toate pozițiile sub-matricelor, pentru a putea fi extrași ușor pixelii din acestea, următoarele 234 de coloane reprezintă doar trăsături, iar ultima și cea mai importantă, o coloană cu numele Clasă, care va reprezenta coloana valorilor de 0 sau 1, care vor specifica dacă sub-matricea liniei curente reprezintă un patch dintr-un semafor. O altă buclă -for- este folosită pentru combinarea tuturor trăsăturilor existente. La finalul acesteia cu ajutorul funcției predefinite **concat**, se concatenează toate fișierele Excel obținute. Proceduri similare cu aceasta discutată anterior sunt următoarele:

- **create_csv_from_hog** – funcție prin care s-au creat fișiere .csv cu ajutorul trăsăturile obținute din histograma gradienților.
- **create_version_2_of_Dataset** – procedură prin care s-a creat un fișier .csv, cu ajutorul histogramelor și a valorilor din modelele de culoare RGB și HSV. Acest tip de fișier are 257 de coloane și a fost folosit pentru clasificarea implementată în prima parte, și anume cea a semafoarelor ajutându-ne de imaginile decupate ale lor.
- **create_version_3_of_Dataset** -- procedură prin care s-a creat un fișier .csv, cu ajutorul histogramelor, pe diferite canale din modelul RGB și punând și valorile din spațiile de culoare RGB, HSV, media culorii roșii din partea de sus a imaginii, din partea de jos și media culorii de verde din partea de sus și jos a imaginii. Acest fișier, la fel ca și cel generat de metoda anterioară, a fost folosit pentru implementarea clasificării culorilor din semafoare din imaginile decupate și are un număr total de 395 de coloane.
- **create_version_4_of_Dataset** -- procedură prin care s-a creat un fișier .csv, cu ajutorul histogramelor, pe diferite canale din modelul RGB și punând și valorile din spațiile de culoare RGB, HSV, media culorii roșii din partea de sus a imaginii, din partea de jos și

și media culorii de verde din partea de sus și jos a imaginii. Acest fișier, la fel ca și cel generat de metoda anterioară, a fost folosit pentru implementarea clasificării culorilor din semafoare din imaginile decupate și are un număr total de 105 de coloane. Diferența dintre cea discutată anterior este că la aceasta, dimensiunea histogramelor este de 16 celule.

Celelalte metode ce au fost implementate în această secțiune a codului sunt asemănătoare cu procedura de a fost detaliată, diferența între ele fiind că au fost făcute pentru culori diferite. Acestea au fost folosite toate pentru versiunea finală a acestui proiect. S-au realizat metode de creare a fișierelor atât pentru setul de date de antrenament, cât și pentru cel de test.

5.7. Clasificarea semafoarelor și testarea modelelor obținute

Ultimul sub-capitol în care se va descrie implementarea propriu-zisă, acoperă partea de clasificare. Aici, se vor explica cum s-au folosit principalii algoritmi de clasificare acoperiți în această lucrare, dar și metode de eficientizare a acurateței. Se vor prezenta comparații între acești algoritmi, precum și modelele obținute de pe urma lor.

Următoarele patru metode s-au realizat pentru clasificarea semafoarelor:

- **adaboost_algorithm** – ce realizează clasificarea cu ajutorul algoritmul de clasificare Adaboost.
- **random_forest_classifier_algorithm –** metodă prin care se realizează clasificarea semafoarelor cu ajutorul algoritmul Random Forest.
- **classifier_knn –** această metodă implementează algoritmul de clasificare K-NN.
- **naïve_bayes_classifier –** ultima metodă, ajută la folosirea algoritmului Naïve Bayes pentru a clasifica semafoarele.

Aceste patru proceduri sunt în mare parte la fel, dar folosesc doar algoritmi diferiți. Se va detalia doar metoda prin care s-a realizat clasificarea cu algoritmul K-NN, deoarece aceasta a fost și cea folosită pentru implementarea proiectului. Toate celelalte metode sunt similare din punct de vedere al codului. Deci, procedura se numește classifier_knn și primește ca și parametru tipul semaforului ce se vrea a fi clasificat. Pentru început, s-a folosit o instrucțiune de decizie -if- pentru a alege calea absolută a fișierului ce conține setul de date de antrenare ce se va folosi pentru antrenare. Se va deschide acest fișier cu ajutorul funcțiilor predefinite ce se folosesc de obicei în lucrul cu fișiere cu extensia .csv (funcția de față folosită se numește **read_csv**).

În clasificarea obiectelor va trebui să se formeze atât seturi de antrenare, cât și de test. Seturile de antrenare se vor numi **x_train** și **y_train**, iar cele de test **x_test** și **y_test**. Diferența dintre acestea se face în modul în care vor fi folosite. De obicei, când se clasifică pe setul de antrenare, pentru a putea construi un model, distribuția datelor va fi 90-10, adică 90% din aceste date de antrenare se vor folosi pentru antrenarea clasificatorului, iar 10% pentru testarea lui, deci, chiar dacă se folosește un set de date de antrenare chiar și acesta va fi împărțit în date de antrenare și de test. Această operație se va realiza cu ajutorul funcției predefinite **train_test_split**, ce primește ca și parametrii un **x**, **y** și dimensiunea datelor de test, după distribuția antrenament-test. Cele două valori **x** și **y**, vor fi setul de date original din care: se vor extrage toate coloanele, mai puțin cea cu numele de **Clasă**, pentru variabila **x**, iar pentru **y** se va folosi doar coloana **Clasă**.

Pe mai departe, s-a creat o variabilă **classifier** ce va primi un clasificator clasic, K-NN,

folosind funcția predefinită KneighborsClassifier din librăria aferentă. Se vor folosi ca parametrii **n_neighbors = 5**, adică se vor căuta 5 vecini în apropierea punctului curent prelucrat, **metric = 'manhattan'**, adică ca și distanță ce se va calcula se va folosi distanța Manhattan și **weights = 'distance'**, ce reprezintă ponderile care vor conta în votul punctelor. Algoritmul de clasificare K-NN a fost explicat la sub-capitolul 4.6.3., iar din acest motiv nu se va reveni cu detalii la modul lui de funcționare. După crearea clasificatorului, se va defini o variabilă model, ce va reprezenta contopirea datelor din setul de antrenare. Se va folosi pentru această operație metoda **.fit** din librăria predefinită a clasificatorului.

Următorul pas este de a prezice clasele din volumul de 10% date alese din setul de date, pentru partea de test, aici folosindu-se funcția predefinită **.predict** care primește ca și parametru **x_test**. Aceste operații implementate până acum în procedura curentă vor forma un model, acesta putând fi salvat ulterior, pentru a putea fi folosit în clasificarea pe imagini de test. Pentru salvarea acestor modele, s-au folosit patru instrucțiuni de decizie pentru a specifica clar pentru ce tip de semafor este modelul obținut.

K-NN Algorithm

	precision	recall	f1-score	support
0	0.88	0.86	0.87	1500
1	0.85	0.87	0.86	1349
accuracy			0.87	2849
macro avg	0.87	0.87	0.87	2849
weighted avg	0.87	0.87	0.87	2849

AdaBoost Algorithm

	precision	recall	f1-score	support
0	0.91	0.89	0.90	1496
1	0.89	0.90	0.90	1353
accuracy			0.90	2849
macro avg	0.90	0.90	0.90	2849
weighted avg	0.90	0.90	0.90	2849

Random Forest Algorithm

	precision	recall	f1-score	support
0	0.95	0.89	0.92	1522
1	0.88	0.94	0.91	1327
accuracy			0.91	2849
macro avg	0.91	0.92	0.91	2849
weighted avg	0.92	0.91	0.91	2849

Naive Bayes Algorithm

	precision	recall	f1-score	support
0	0.77	0.44	0.56	1512
1	0.57	0.85	0.69	1337
accuracy			0.63	2849
macro avg	0.67	0.65	0.62	2849
weighted avg	0.68	0.63	0.62	2849

[Figura 5.17 – Rezultatele diferitelor metode de clasificare]

Se pot observa în figura de mai sus diferite rezultate în urma procesului de clasificare folosind algoritmii K-NN, AdaBoost, Random Forest și Naive Bayes. Aceste metrici de performanță s-au afișat folosind funcții predefinite cum ar fi: **confusion_matrix** sau **classification_report**. Din imaginea de mai sus rezultă că algoritmul Naive Bayes furnizează cele mai slabe rezultate. De menționat este că aceste rezultate sunt pe baza unei versiuni de set de date care nu a fost folosită într-un final la realizarea proiectului. Versiunea finală de set de date, cuprinde datele, adunate cu ajutorul metodelor următoare:

- **final_method_to_create_csv_for_classification_of_green_traffic_light**
- **final_method_to_create_csv_for_classification_of_red_traffic_light**

- **final_method_to_create_csv_for_classification_of_yellow_traffic_light**

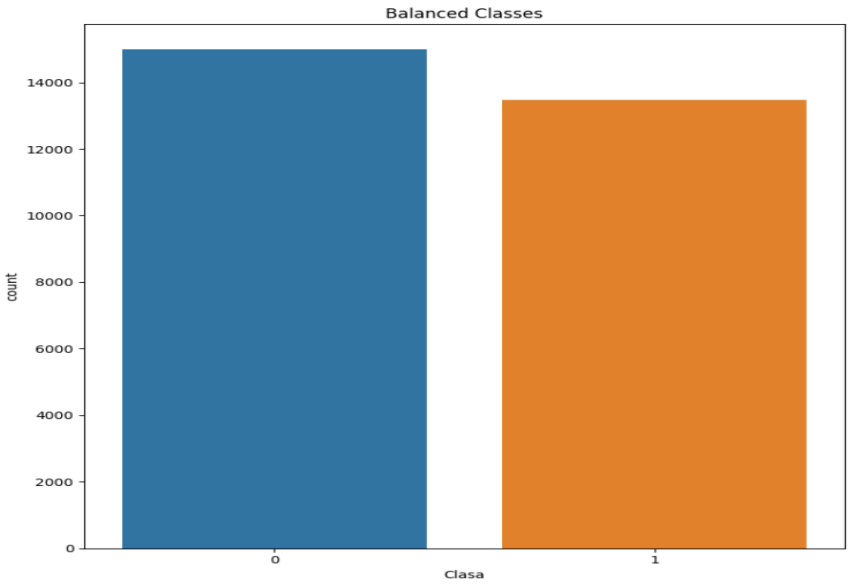
Prima metodă a fost explicată în detaliu la sub-capitolul anterior, din acest motiv nu cred că mai este necesar să se mai descrie încă o dată procedurile. Rezultatul clasificărilor folosind setul de date complet și optimizat se poate observa la figura 35 ce este descrisă pe următoarea pagină.

Pe lângă funcțiile care au ajutat la formarea clasificatorilor, s-au creat două proceduri prin care să se optimizeze rezultatele obținute. Prima dintre ele, s-a folosit pentru a extrage un număr de date care au clasa 0 și a forma cu toate celelalte de clasa 1 setul de date final ce va folosit pentru clasificare. Această funcție se numește **suffle_data_frame** și primește ca și parametrii fișierul cu date format din metodele explicate la sub-capitolul anterior și o variabilă ce va specifica tipul semaforului. O astfel de metodă este foarte necesară deoarece, setul de antrenament va cuprinde datele culese din 80 de poze cu semafoare. Acestea vor genera un fișier (format din concatenarea altor 80 de fișiere) ce va avea o dimensiune totală de aproximativ 2.940.000x236, iar de exemplu, pentru culoarea verde acesta va fi foarte nebalansat. Dintr-un număr de aproximativ 3 milioane de linii, dacă facem rotunjire în sus, vom avea doar aproximativ 13500 de linii ce vor fi de clasă 1.

Soluția găsită se numește "**resampling**" și înseamnă a lua linii random cu clasa 0 și a le lipi într-un set de date ce conține doar linii cu clasa 1. Adică, luăm toate datele ce sunt în clasa minoritară și le punem într-un nou set de date, iar în acesta, la final vom concatena date din clasa majoritară. În această procedură ce se descrie, se va citi mai întâi setul complet de date cu ajutorul funcției predefinite **read_csv**. Se vor amesteca liniile din setul date cu ajutorul tot unei funcției predefinite, pe nume **sample**. Cu ajutorul unei instrucțiuni de decizie, se va implementa cazul în care utilizatorul dorește să clasifice toate semafoarele, iar pentru asta dorindu-și în această metodă să extragă datele claselor fiecărui tip de semafor în parte (1 pentru roșu, 2 pentru verde și 3 pentru galben). Într-un alt set de instrucțiuni de decizie se vor lua seturi (de dimensiuni diferite) de linii cu clasa 0, din clasa majoritară. La final se vor concatena cele două seturi de date (cu clasa minoritară și cea majoritară) cu ajutorul funcției **concat** și se va salva fișierul Excel obținut. La figura 34 se poate observa distribuția claselor în setul de date obținut.

O altă metodă folosită este **suffle_lines_in_data_frame**, care primește ca și parametri setul de date obținut de pe urma metodei precedente și tipul semaforului ce se vrea a fi clasificat. Această procedură amestecă liniile din setul de date, astfel încât să se obțină un rezultat cât mai clar (nu este chiar atâta de corect ca liniile cu clasa 1 să fie toate, una după alta, în una din jumătățile setului de date, ci toate liniile trebuie să fie cât mai amestecate). Pentru această operație, s-a citit setul de date și s-au introdus toate liniile într-o listă. Pe urmă s-au amestecat aceste date cu ajutorul funcției predefinite **suffle** din librăria Random. Rezultatul final a fost salvat într-un nou fișier Excel și va fi dat clasificatorului.

În figura 5.18, se poate observa cum au fost distribuite clasele din setul de date inițial după algoritmul de "**resampling**". Fișierul Excel final va avea un număr de aproximativ 29000 de linii, dintre care 13800 cu clasa 1, iar restul fiind de clasa 0.



[Figura 5.18 – Distribuția claselor din setul de date după algoritmul de optimizare]

În următoare imagine se poate observa rezultatul clasificării imaginilor cu semafoare verzi, după toate metodele de optimizare și folosind setul de date final, cu toate trăsăturile ce s-au decis a fi extrase.

	precision	recall	f1-score	support
0	0.96	1.00	0.98	1500
1	0.99	0.96	0.98	1349
accuracy			0.98	2849
macro avg	0.98	0.98	0.98	2849
weighted avg	0.98	0.98	0.98	2849

Accuracy: 97.70%

[Figura 5.19 – Rezultatul clasificării semaforului verde după optimizare]

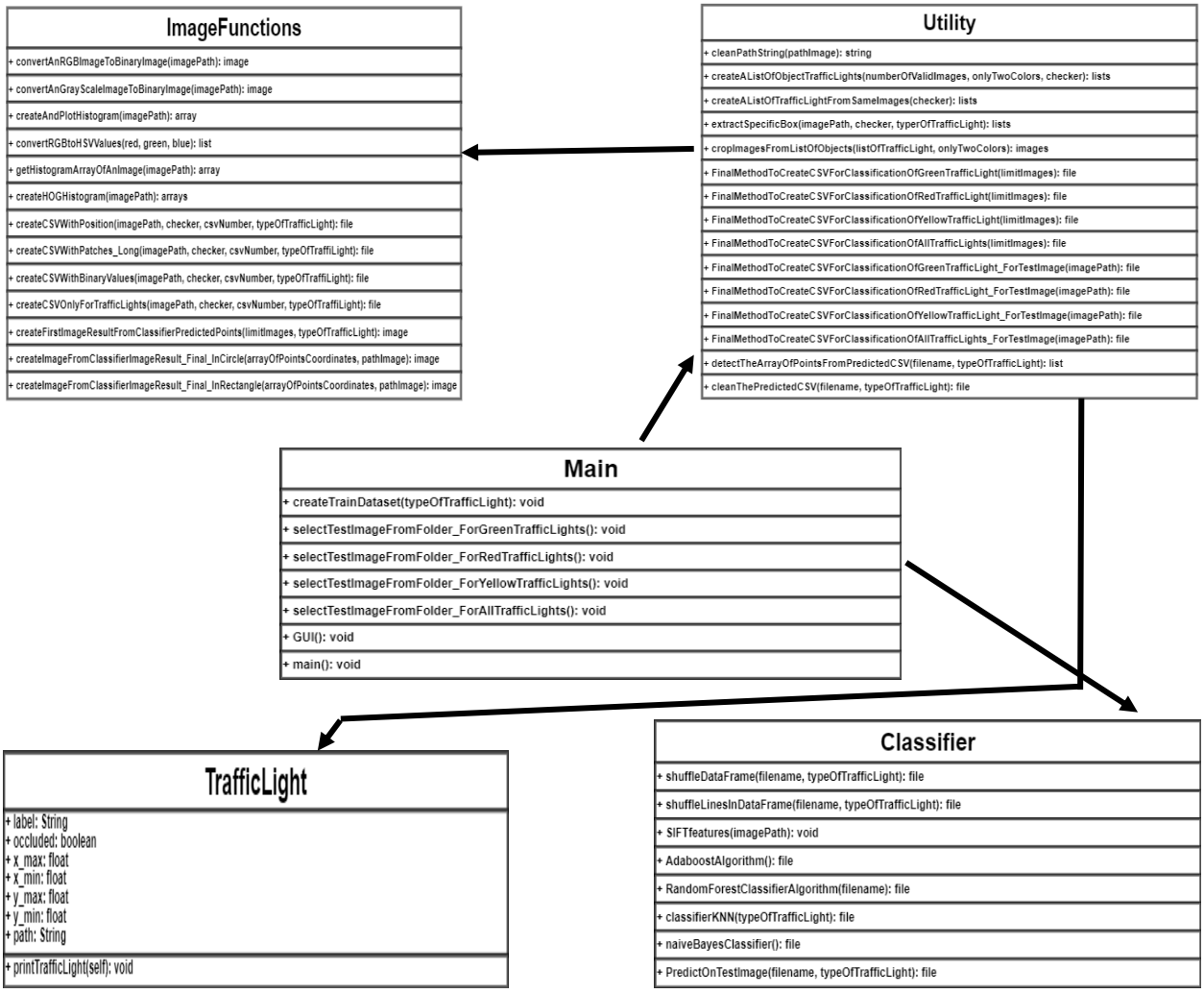
Ultima metodă ce se va discuta în acest sub-capitol se numește **predict_on_test_image** și are ca parametrii calea absolută a fișierului .csv cu toate datele extrase din imaginea de test și o variabilă ce va specifica tipul semaforului prezis. Pentru început, într-un set de instrucțiuni de decizie se va alege calea către modelul antrenat cu ajutorul clasificatorului descris în paragrafele de mai sus, acesta fiind câte unul pentru fiecare tip de semafor în parte. Se vor repeta pașii de existau și în procedurile cu algoritmii de clasificare și anume: deschiderea fișierului cu datele de test și împărțirea datelor în set de antrenare și set de test. Diferența majoră este că aici, pe imagine de test, distribuția datelor va fi 1-99, adică 1% date de antrenare și restul de 99% date de test. S-a decis acest lucru pentru că se va folosi modelul deja antrenat cu clasificatorul descris în paginile anterioare, nefiind necesare alte date consistente de antrenament. Cu ajutorul funcției **load** se va încărca modelul aferent tipului de semafor și se va face predicție pe datele de test existente. Rezultatele se vor salva într-un fișier Excel (aceste rezultate reprezintă de fapt liniile ce au fost prezise cu ajutorul modelului, ca fiind de clasă 1). Metricile de performanță pentru imaginea de test se vor afișa la capitolul următor.

5.8. Diagramele UML ale aplicației

La acest sub-capitol se vor reprezenta diagramele UML aferente acestui proiect. UML (Unified Modeling Language) este un limbaj folosit pentru descrierea aplicațiilor software. Aceste diagrame au fost la bază dezvoltate pentru a reprezentata complexitatea aplicațiilor construite pe principiul programării orientate pe obiecte, al căror temelie sunt clasele și obiectele. Dar, cu toate că această aplicație nu respectă decât foarte puțin principiile OOP (object oriented programming), s-a realizat o diagramă UML pentru fișierele aplicației, pentru a se prezenta legăturile dintre ele. Pentru această aplicație se vor oferi două diagrame UML:

- **Diagrama UML a componentelor** – aici, se vor reprezenta printr-o diagramă toate fișierele Python ce au dus la implementarea aplicației, precum și principalele proceduri din componența lor.
- **Diagrama de secvențe** – diagramă prin care se reprezintă interacțiunea dintre module.

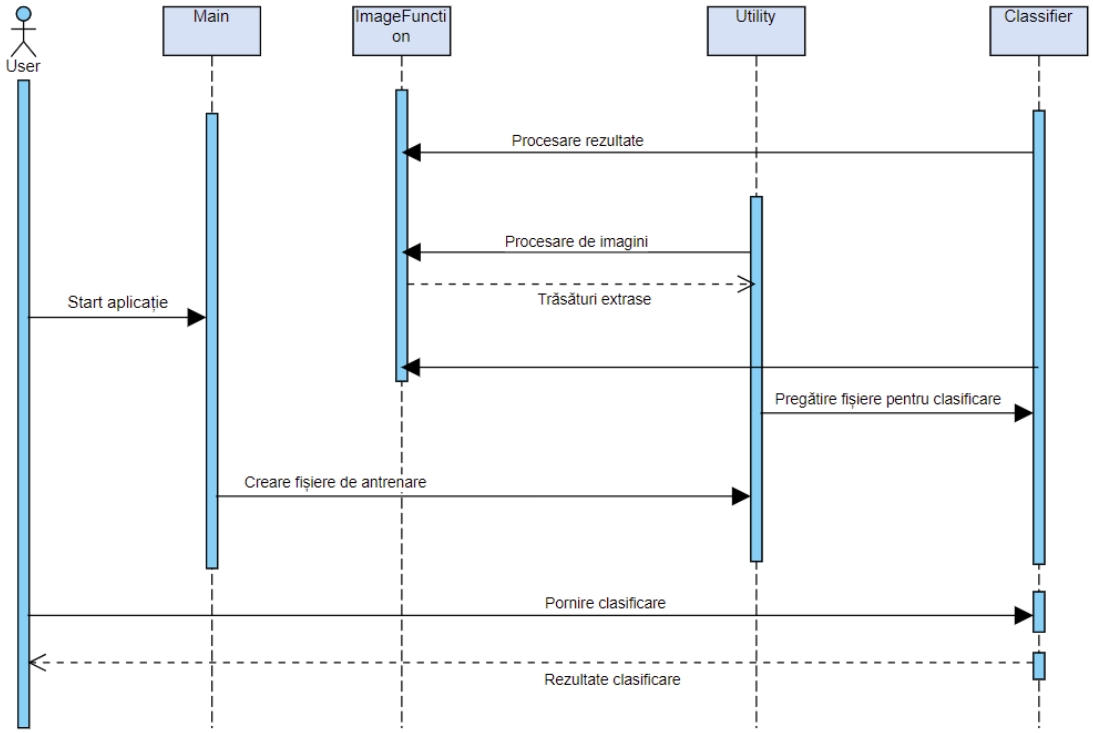
1. Diagrama UML a fișierelor aplicației (diagrama de componente)



[Figura 5.20 – Diagrama UML de componente a aplicației]

În figura de mai sus se poate observa diagrama UML de componente a aplicației. Aici, sunt descrise fișierele care au dus la implementarea proiectului, precum și principalele proceduri care sunt stocate în ele. S-a ales a nu se pune toate procedurile din fișierele Utility și ImageFunctions în diagramă deoarece nu ar mai fi fost destul de vizibilă pentru cititorul acestui document. Diagrama aceasta reprezintă arhitectura aplicației. Aceste șase fișiere descrise de modelul grafic de mai sus sunt într-o relație de asociere, pentru că se folosesc între ele pentru realizarea proiectului. De exemplu, în fișierul de Utility se regăsesc proceduri pentru manipularea tuturor imaginilor cu scopul de a forma seturile de date de antrenament și de test. Acest fișier folosește funcțiile din ImageFunctions, unde sunt stocate metode de extragere de trăsături, pentru a manipula fluxul de date primit.

2. Diagrama de secvențe



[Figura 5.21 – Diagrama de secvențe]

Se poate observa în figura de mai sus diagrama de secvențe pentru aplicația prezentată în documentul de față. În această diagramă avem următoarele module:

1. **Start aplicație** – utilizatorul va da start aplicației prin butoanele disponibile lui.
2. **Creare fișiere de antrenare** – se vor trimite comenzi modulului Utility pentru a putea crea fișierele cu datele de antrenare și de test.
3. **Procesare de imagini** – se vor apela proceduri din modulul ImageFunctions, pentru a se putea extrage principalele trăsături. Ca și răspuns la acest apel, din modulul ImageFunctions, vor veni seturi de valori importante din imagini (valorile pixelilor în spațiile de culoare RGB, HSV, valorile extrase din histograma gradienților etc).

4. **Pregătire fișiere pentru clasificare** – se vor da comenzi clasificatorilor pentru a putea pregăti fișierele de antrenare (se vor folosi metode de optimizare etc).
5. **Pornire clasificare** – se vor folosi diferiți algoritmi de clasificare pentru a se putea face această operațiune. Ca și răspuns la clasificări, se vor primi fișiere cu punctele prezise ca și rezultate.
6. **Procesare rezultate** – se vor procesa punctele prezise, iar aplicația va returna un set de imagini modificate în care se vor putea observa semafoarele încadrate în dreptunghiuri sau cercuri.

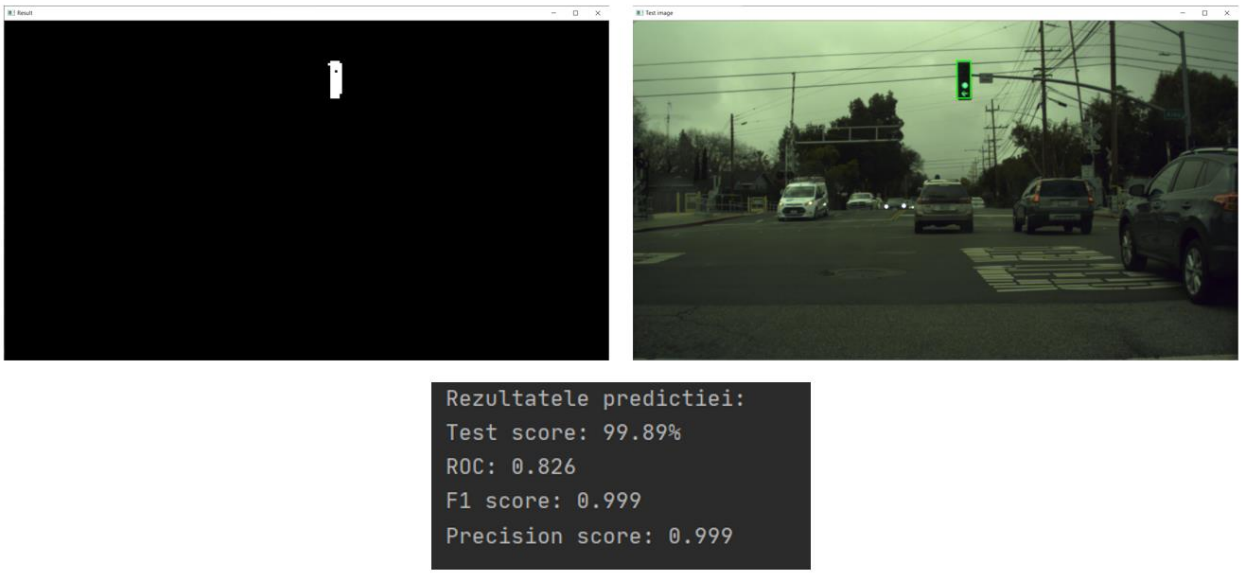
Capitolul 6. Testare și validare

La acest capitol se vor observa rezultatele obținute în urma algoritmilor discutați la capitolul anterior. Se vor prezenta atât rezultatele în urma clasificării folosind algoritmii AdaBoost, Naive Bayes și Random Forest, precum și rezultatele obținute de pe urma versiunii finale în care s-a folosit algoritmul K-NN. Se vor prezenta la acest capitol și rezultatele furnizate de algoritmii de procesare de imagini prin care s-a reușit încadrarea semafoarelor în forme dreptunghiulare sau sferice.

6.1. Testarea aplicației și observarea rezultatelor obținute

În cele ce urmează se vor prezenta principalele rezultate obținute în urma clasificării semafoarelor folosind diferitele tehnici puse la dispoziție de librăriile de specialitate. Nu se vor mai explica încă o dată metricile de performanță obținute în urma clasificărilor folosind primele versiuni ale setului de antrenament, deoarece ele au fost menționate la sub-capitolul 5.7. Mai jos, se pot observa rezultatele clasificării fiecărui semafor în parte, dar și a tuturor în același timp.

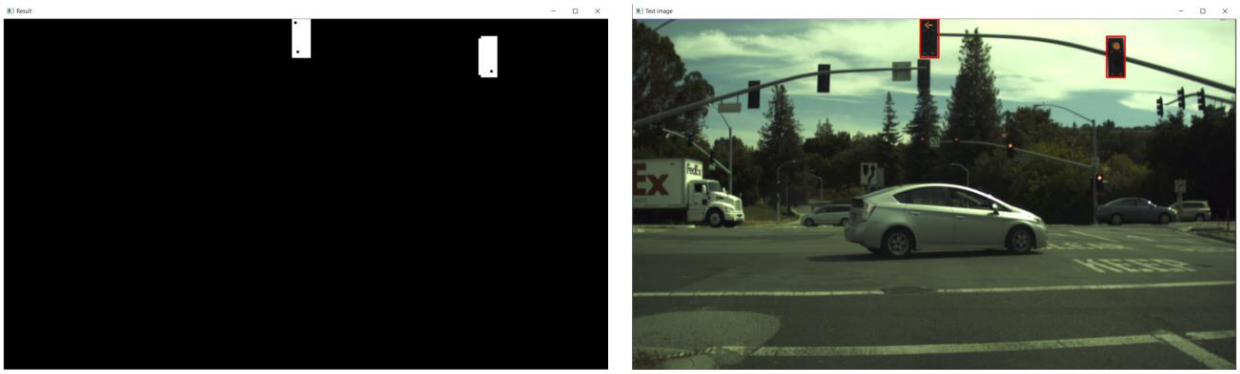
La figura 6.1, se poate observa rezultatul clasificării semaforului cu culoarea verde. În imaginea din stânga avem reprezentată predicția clasificatorului K-NN, folosind punctele prezise de el și formând o imagine binară în care pixelii prezișI să aibă valoarea 255, iar cei din fundal valoarea 0. În partea dreaptă avem prelucrarea acestor puncte prezise de algoritm pe imaginea de test. S-a încadrat semaforul verde din imagine într-un chenar dreptunghiular. În plus, se pot observa și metricile din punct de vedere al performanței clasificatorului.



[Figura 6.1 – Rezultatul clasificării semaforului verde]

La figura 6.2, se poate observa rezultatul clasificării semaforului cu culoarea roșie. În imaginea din stânga avem reprezentată predicția clasificatorului, folosind punctele prezise de el și formând o imagine binară în care pixelii prezișI să aibă valoarea 255, iar cei din fundal valoarea 0. În partea dreaptă avem prelucrarea acestor puncte prezise de algoritm pe imaginea de test. S-au încadrat toate semafoarele din prim-plan în chenare dreptunghiulare de culoare roșie. În plus, se

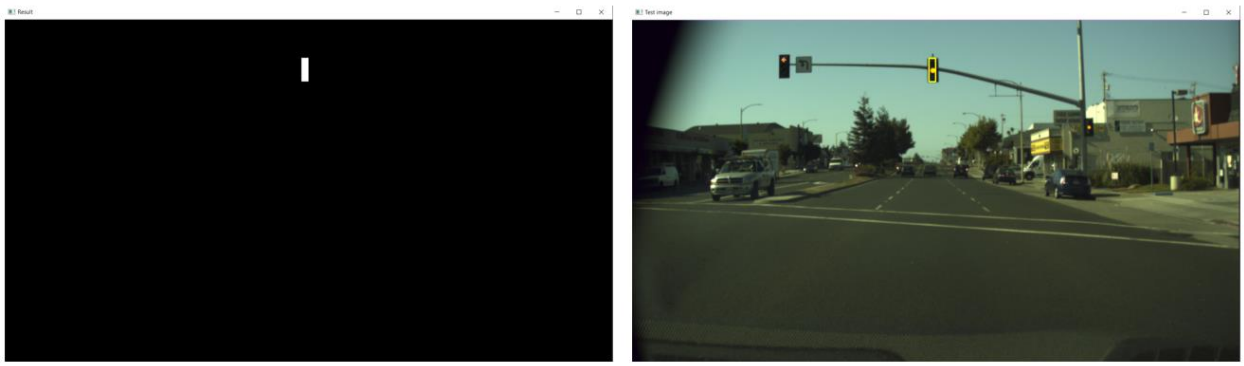
pot observa și metricile din punct de vedere al performanței clasificatorului.



```
Rezultatele predictiei:
Test score: 99.71%
ROC: 0.856
F1 score: 0.997
Precision score: 0.997
```

[Figura 6.2 – Rezultatul clasificării semaforului roșu]

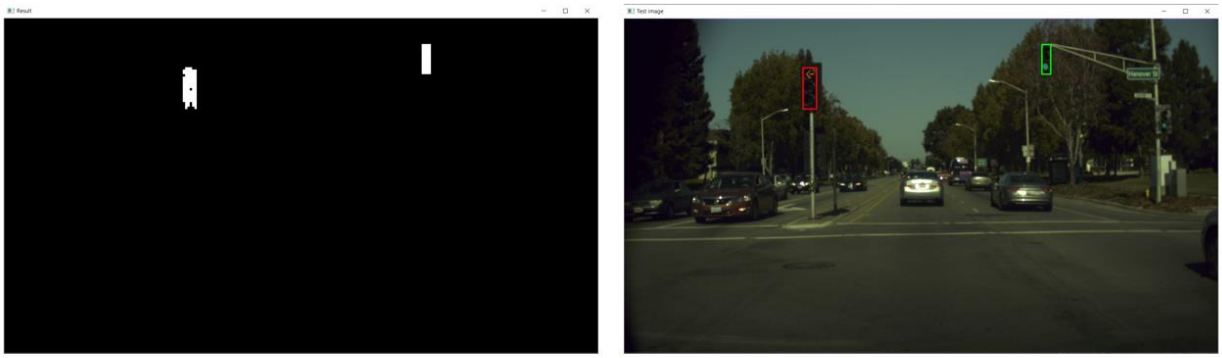
Mai jos, avem clasificarea și predicția semaforului de culoare galbenă. Putem observa în imaginea din stânga, în primul rezultat, o imagine binară în care avem semaforul galben prezis de algoritm evidențiat cu pixeli albi (cu valoarea 255), fundalul aici fiind negru. În dreapta, avem încadrat în imaginea de test originală, semaforul găsit de clasificator. Doar unul din două semafoare a fost găsit. În plus, avem și câteva metrici de performanță prin care se poate analiza modelul format cu ajutorul algoritmului K-NN și eficiența sa.



```
Rezultatele predictiei:
Test score: 99.85%
ROC: 0.674
F1 score: 0.998
Precision score: 0.998
```

[Figura 6.3 – Rezultatul clasificării semaforului galben]

În ultima figură care se referă la o imagine de test specifică (figura 6.4) se poate rezultatul predicției pe toate semafoarele găsite în imaginea de test. Avem, la fel ca și la celelalte figuri următoarele: o imagine binară în care se observă punctele prezise de clasificator, o imagine de test în care avem toate semafoarele prezise, încadrate în dreptunghiuri cu culori aferente tipului semaforului (dreptunghi roșu pentru semaforul cu statusul roșu și verde pentru semaforul cu culoarea verde) și metricile de performanță aferente acestui rezultat.



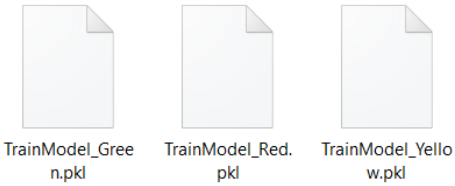
```
Rezultatele predictiei (---- Green Traffic Light ----):
Test score: 99.87%
ROC: 0.763
F1 score: 0.999
Precision score: 0.999

Rezultatele predictiei (---- Red Traffic Light ----):
Test score: 99.87%
ROC: 0.837
F1 score: 0.999
Precision score: 0.999
```

[Figura 6.4 – Rezultatul clasificării tuturor semafoarele din imaginea de test]

Din punct de vedere al metricilor, se pot observa îmbunătățiri semnificative din partea clasificatorului, aici făcând diferența setul de date de antrenament optimizat cu ajutorul a tuturor trăsăturilor extrase prin procesul de procesare de imagini și extragere de trăsături, dar și prin optimizarea clasificatorului balansând setul de date, astfel încât să avem în el stocate un număr relativ egal de clase de 1 și 0.

Aceste rezultate au fost obținute prin antrenarea clasificatorului K-NN și prin obținerea și salvarea a trei modele cu ajutorul cărora s-au făcut predicțiile pe imaginile de test. Acestea sunt cele trei din figura de mai jos:



[Figura 6.5 – Modele folosite pentru predicție]

Aceste modele au fost create cu ajutorul librăriei predefinite **Pickle**. S-a observat că, cu cât modelele au o dimensiune mai mare, cu atât timpul de procesare a predicțiilor crește, iar acest lucru se poate preveni prin două variante: ori se reduce setul de date de antrenament (reducând

numărul de imagini de antrenament), ori renunțând la anumite trăsături, astfel încât fișierele .csv să fie cât mai mici ca și dimensiuni.

În următoarele două tabele (tabelele 6.1 și 6.2) se poate observa și analiza o statistică globală, rezultată în urma testării modelelor formate pe un set de imagini de test format din 19 imagini. Această statistică reprezintă validitatea și funcționarea algoritmilor implementați în proiectul de față. Imaginile testate conțin atât semafoare cu culoarea verde, cât și cu culoarea roșie. Nu s-au găsit imagini care să conțină toate cele trei tipuri de semafoare în același timp.

Metrică	Valoare
Acuratețe	0.98 ⇔ 98%
ROC	0.72 ⇔ 72%
F1	0.98 ⇔ 98%
Precizie	0.98 ⇔ 98%

Metrică	Valoare
Acuratețe	0.98 ⇔ 98%
ROC	0.68 ⇔ 68%
F1	0.98 ⇔ 98%
Precizie	0.98 ⇔ 98%

[Tabelele 6.1 și 6.2 – Statistici globale pentru imaginile de test]

Se pot observa în aceste două tabele următoarele metrici de performanță acoperite: scorul general – Acuratețea-, ROC-ul, scorul F1 și precizia clasificatorului. Diferența foarte puțin sesizabilă se face la metrica ROC. Aceste metrici globale au fost obținute cu ajutorul unei metode prin care s-au parcurs imaginile de test și s-a făcut predicție pe fiecare dintre semafoarele din acestea.

Primul tabel (6.1 - stânga) a rezultat în urma predicțiilor făcute pe semafoarele de culoare verde din cadrul imaginilor de test, iar cel de-al doilea (6.2 - dreapta) cuprinde datele obținute pentru semaforul de culoare roșie.

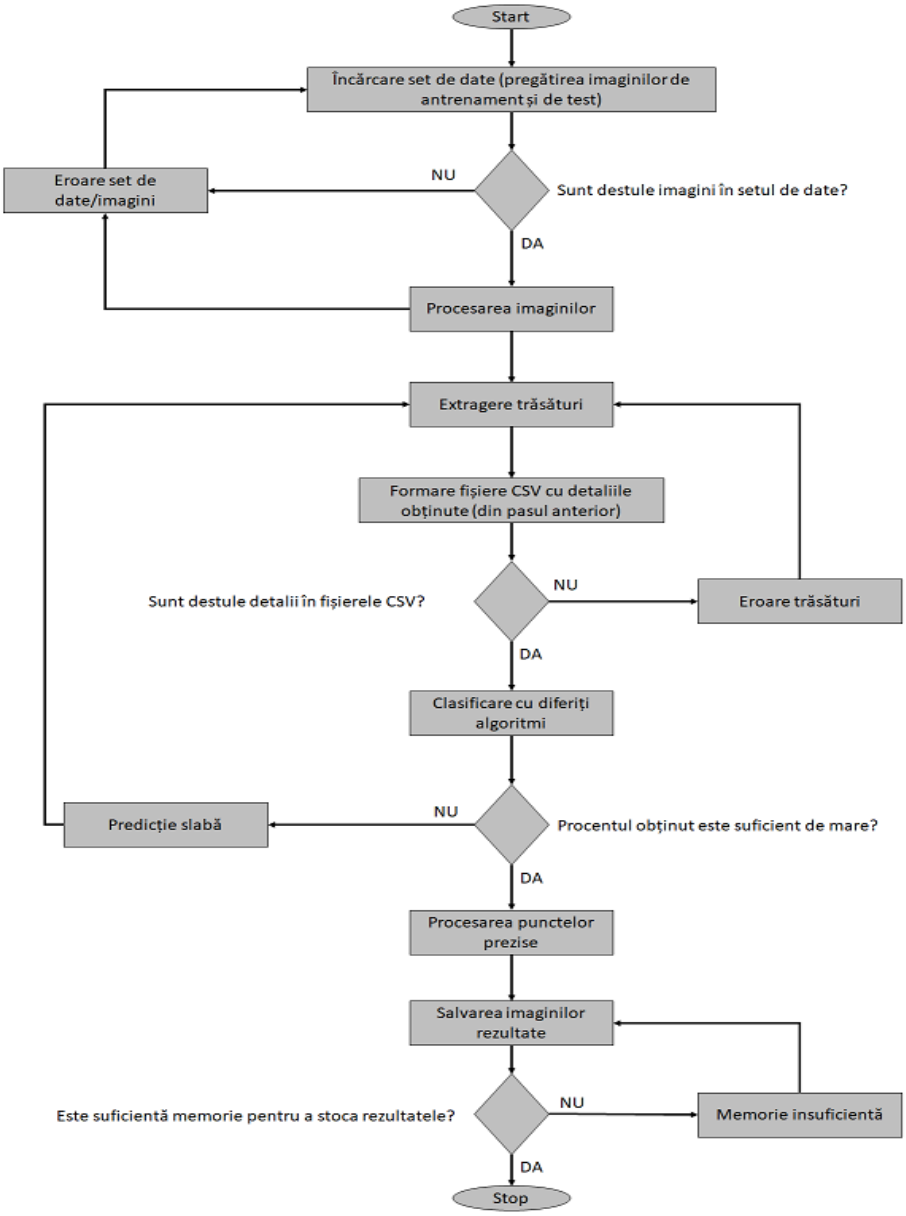
Trebuie menționat că din cauza dimensiunii fișierelor ce vor fi folosite pentru predicția semafoarelor (acele fișiere cu extensia CSV ce cuprind toate trăsăturile extrase), dar și a modelelor folosite în procedurile de predicție (acestea au dimensiuni considerabile, în jur de 25-35 de MB) timpul de execuție al predicției este de aproximativ 30-35 de secunde. Două soluții posibile ce ar putea rezolva problema timpului de execuție: eliminarea unor trăsături din setul de date de test (avem 236 de coloane în fișierele Excel, deci un timp de procesare a tuturor acestor date mai mare) sau găsirea unor metode prin care să se reducă dimensiunea modelelor obținute (pentru aceasta s-a implementat o soluție posibilă).

Capitolul 7. Manualul de instalare și utilizare

La acest capitol se vor prezenta, pe rând, diagrama Flow Chart (adică diagrama ce descrie fluxul de utilizare al aplicației) și manualul de instalare și utilizare al produsului software implementat de mine.

7.1. Fluxul de utilizare al aplicației

În acest sub-capitol se va prezenta Flow Chart-ul aplicației. Aceasta (tradusă din engleză – o diagramă de flux), este definită ca o reprezentare schematică a unui algoritm, a unui proces. Ea poate descrie și rezolvarea pas cu pas a unei sarcini. Pentru proiectul de față, această diagramă reprezintă fluxul de sarcini pe care le cuprinde aplicația, pornind de la punctul de start și trecând pas cu pas prin toată implementarea ei, până la sfârșitul acesteia.



[Figura 7.1 –Diagrama fluxului de utilizare al aplicației]

În figura de la pagina anterioară se pot observa principalele module prin care trec datele în această aplicație software. De la punctul de start se pornește aplicația. Primul pas este de a pregăti setul de date, aici fiind incluse atât imaginile de antrenare, cât și cele de test. Ca o primă instrucțiune de decizie, sau ca și condiție, fluxul va trece printr-un -if- ce va verifica dacă setul de date este valid, sau dacă imaginile sunt suficiente ca și număr. Dacă nu este respectată condiția, utilizatorul va fi nevoit să caute ori un alt se de date, ori mai multe imagini. Dacă da, se va trece la partea de procesare de imagini. Aici, împreună cu următorul pas, adică extragerea de trăsături, se vor reține cele mai importante detalii din imaginile de antrenare sau de test, pentru a se putea pregăti setul de fișiere ce se vor da clasificatorilor. La pasul de formarea fișierelor CSV, se va încerca construirea acestor tipuri de fișiere, folosind datele extrase la pașii anteriori. O altă instrucțiune condițională va întreba sistemul dacă sunt valide trăsăturile. Dacă se va returna valoarea fals, se va reveni la pașii anteriori, căutându-se alte trăsături specifice imaginilor. Dacă acest pas este valid, se va continua cu următorul. Urmează partea de clasificare a semafoarelor. Aici, se vor folosi principalii algoritmi de clasificare, descriși și în restul aplicației, pentru a avea rezultate cât mai bune. O instrucțiune condițională va verifica dacă rezultatul predicțiilor este suficient de bun. Dacă nu este pe placul utilizatorului, se va reveni la pașii în care se extrag trăsăturile și se vor căuta noi detalii mai potrivite care să furnizeze rezultate mai bune. Dacă predicțiile sunt acceptabile, se vor procesa punctele prezise de clasificatori formând astfel imagini cu semafoarele încadrate în forme dreptunghiulare. Dacă memoria este insuficientă, utilizatorul va fi rugat să mai elibereze din memoria sistemului. Dacă se vor salva cu succes imaginile rezultate, acesta va putea verifica dacă aplicația a funcționat în parametrii normal și semafoarele dacă au fost prezise și încadrate cu succes.

7.2. Manualul de instalare și utilizare

La acest sub-capitol se vor prezenta pașii de instalare și utilizare a aplicației implementate de mine. Aceștia vor fi descriși pe puncte cât mai clare pentru a nu pune probleme utilizatorilor. Pentru început, în cadrul aplicației au fost identificate următoarele resurse care vor fi necesare:

1. **Resurse Hardware:**

- Un sistem PC/Laptop care să un procesor relativ modern, cu frecvență ridicată și memorie RAM (cel puțin 8 GB) și spațiu de stocare bun (minim 250-500 GB).

2. **Resurse Software:**

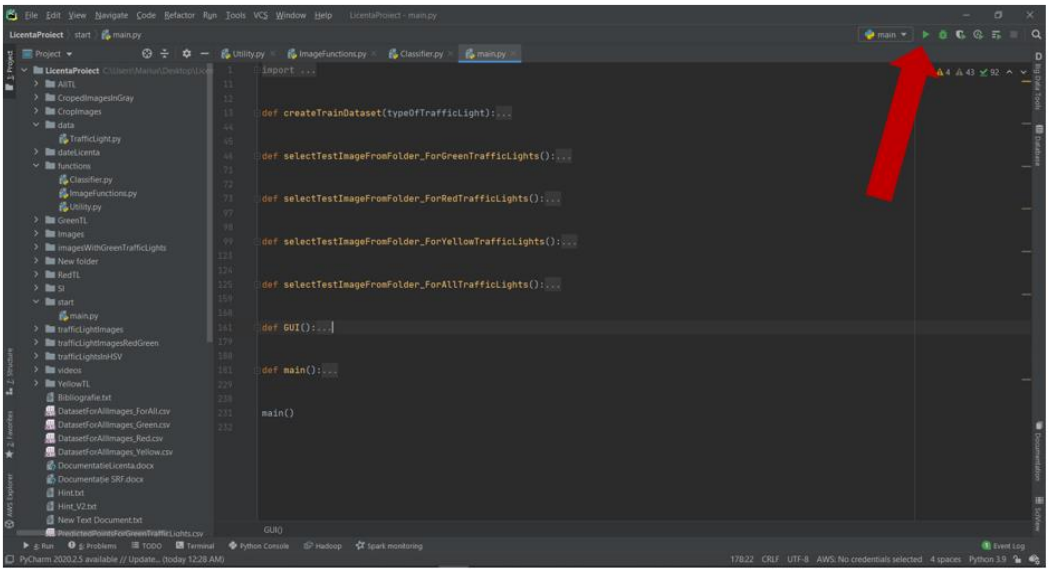
- Sistem de Operare: Windows sau Mac OS
- Mediul de dezvoltare software PyCharm
- Set de date cu imagini/video-uri/locații cu semafoare, pentru antrenarea rețelelor/clasificatorilor și realizarea detecțiilor de semafoare și ale culorilor lor (minim 4500-5000 de imagini necesare).

Pașii pentru instalarea și utilizarea aplicației de față sunt următorii:

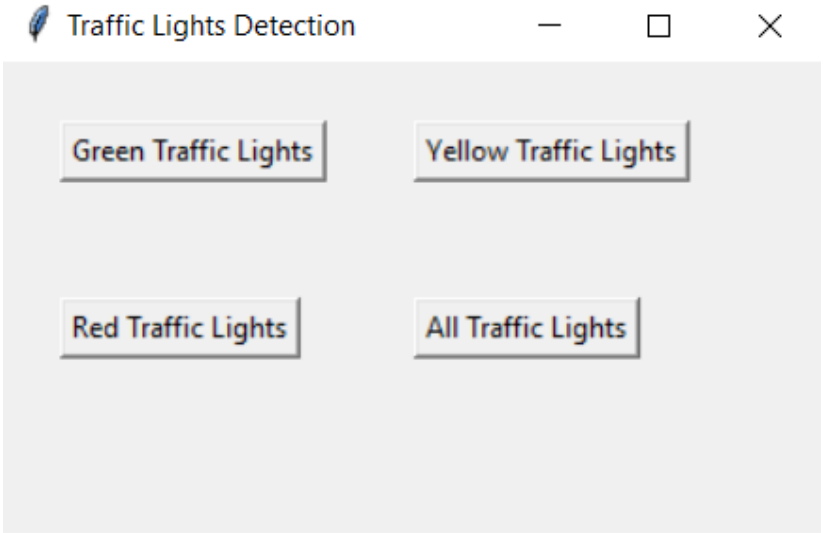
1. **Instalare mediului de dezvoltare PyCharm –** utilizatorul va fi nevoit să-și instaleze un mediu de dezvoltare pentru a putea testa aplicația.
2. **Găsire set de date de antrenament și de test –** utilizatorul va trebui să găsească un set de date, de imagini, pentru a putea fi procesate și date apoi algoritmilor de clasificare.
3. **Pornire aplicație PyCharm –** după ce a fost instalat mediul de dezvoltare și au fost găsite

imagini pentru aplicația software de față, utilizatorul va trebui să deschidă aplicația PyCharm.

4. **Start aplicație –** pentru acest pas, se poate urmări imaginea următoare. Utilizatorul va trebui să apese pe butonul indicat de săgeata din figură.



5. **Selectare opțiune de clasificare și predicție –** se va deschide o fereastră, iar utilizatorul va fi nevoit să selecteze una dintre cele patru opțiuni disponibile: "**Green Traffic Lights**", "**Yellow Traffic Lights**", "**Red Traffic Lights**" și "**All Traffic Lights**".

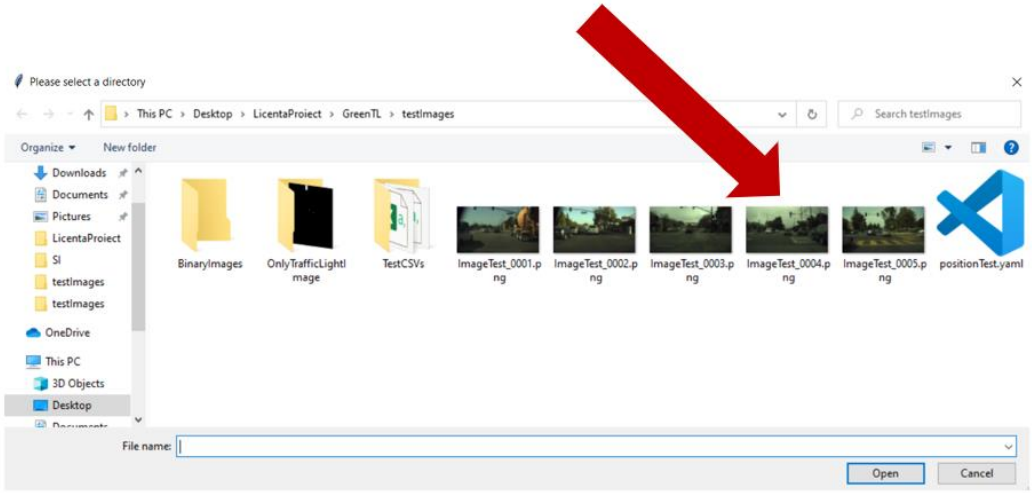


6. **Selectare imagine de test pentru a fi prelucrată și să se clasifice semafoarele din ea –** indiferent de opțiunea selectată la pasul anterior, utilizatorului i se va deschide o nouă fereastră de unde va fi nevoit să aleagă o imagine pentru a fi data aplicației în vederea

prelucrării ei. Pentru exemplul de față, se va selecta butonul aferent semafoarelor de culoare verde.

a) Selectare buton - Green Traffic Lights -

b) Selectare imagine de test



7. **Așteptarea rezultatelor –** utilizatorul va trebui să aștepte în jur de 30-35 de secunde pentru a se putea face următoarele:
   - Pregătirea fișierelor .csv aferente imaginii selectate.
   - Încărcarea modelului și prezicerea semafoarelor pe aceste fișiere construite
   - Încadrarea semafoarelor în chenare dreptunghiulare cu ajutorul punctelor prezise.
8. **Selectarea unei noi imagini** – după analizarea rezultatelor oferite de aplicație, utilizatorul va putea selecta o nouă imagine de test sau o nouă opțiune din cele patru, prezentate de imaginea de la pasul 5. Deci, se repetă pașii 5-7, intrându-se într-un ciclu.
9. **Oprirea aplicației** – dacă utilizatorul dorește să oprească aplicația o va putea opri prin butonul de "**Stop Main"**, furnizat de mediul de dezvoltare PyCharm.

Capitolul 8. Concluzii

La acest capitol se vor prezenta concluziile cu privire la aplicația realizată, dar și eventualele dezvoltări ale ei ce pot avea loc.

Scopul acestei aplicații a fost de la bun început detecția semafoarelor din traficul urban. Acest obiectiv a fost realizat cu succes cu ajutorul tehnicilor de clasificare și de procesare de imagini. Proiectul a beneficiat de un larg set de imagini ce au reprezentat temelia setului de antrenare și de test pentru întreaga aplicație. Din acest set, s-au extras imagini relevante cu intersecții semaforizate, din ele obținându-se fișiere cu extensia .csv în care se regăseau doar trăsăturile pixelilor. Astfel se formaseră seturile de antrenament și de test cu care s-au făcut detecțiile de semafoare. Pe urmă, prin algoritmi de procesare de imagini și prelucrare a pixelilor s-au încadrat toate semafoarele detectate în chenare dreptunghiulare.

Pentru o performanță ridicată și o recunoaștere cât mai bună a semafoarelor se recomandă a se folosi imagini sau secvențe video cât mai clare. A se reaminti că setul de date care s-a folosit pentru realizarea și antrenarea acestei aplicației nu este suficient de mare astfel încât să poată fi făcută o precizie cât mai ridicată în timp real. Pentru asta ar fi fost necesar un set de date care să cuprindă zeci de mii de poze, poate chiar sute de mii astfel încât aplicația să poată să recunoască cât mai ușor semafoarele, în toate situațiile posibile.

Prin această versiune de aplicație, am oferit o soluție clară pentru detecția semafoarelor din trafic, folosind cunoștințele acumulate pe parcursul anilor de studiu, dar și prin documentarea individuală.

De asemenea, pe tot parcursul implementării aplicației descrise în acest document, au apărut și diferite probleme care au fost rezolvate mai greu sau mai ușor de mine. Acestea au ținut de procesarea imaginilor (cum pot extrage cât mai bine trăsăturile pixelilor), dar și de partea de clasificare (am avut probleme la creșterea acurateței algoritmilor folosiți). Toate acestea au fost rezolvate cu succes, putând să continui drumul implementării acestei aplicații.

8.1. Dezvoltări ulterioare

Această aplicație poate fi dezvoltată pe mai multe direcții. Dacă prin modul în care este ea implementată în momentul de față se pot face detecții numai pe semafoarele din trafic, se pot adăuga module care să conțină proceduri și pentru detecția altor obiecte. De exemplu, un alt proiect similar cu acesta face detecția marcajelor longitudinale de pe drumuri. Se poate implementa acest lucru și la proiectul de față prin adăugarea unor noi proceduri prin care să se acopere și acest tip de clasificare. În plus, în viitor, se pot implementa module și pentru următoarele: detecția pietonilor, a semnelor de circulație, a mașinilor etc.

Se poate pune întrebarea dacă aplicația software de față poate fi integrată pe autovehicule. Răspunsul este da. Dacă se va face o colaborare cu o firmă specializată în construcția dispozitivelor hardware, se va putea implementa un modul fizic, cu procesor, care să conțină acest sistem software.

Capitolul 9. Bibliografie

[1] Classification, Self-driving car, https://en.wikipedia.org/wiki/Self-driving_car

[2] The 6 Levels of Vehicle, https://www.synopsys.com/automotive/autonomous-driving-levels.html

[3] Introduction to conditional automation in driving, https://www.section.io/engineering-education/conditional-automation-in-driving/

[4] Advantages & Disadvantages of Self-Driving Cars, https://environmental-conscience.com/self-driving-cars-pros-cons/

[5] Steering Assist, https://www.computerlanguage.com/results.php?definition=steering

[6] Levels of Autonomous Driving, https://www.jdpower.com/cars/shopping-guides/levels-of-autonomous-driving-explained

[7] Asistență pentru șofer, Cruise Control, Adaptive Cruise Control, Pilot Assist

https://www.volvocars.com/ro/support/manuals/s60-twin-engine/2019w46/asistenta-pentru-sofer/

[8] Turing Machine, Description, History, https://en.wikipedia.org/wiki/Turing_machine

[9] Enigma Machine, History, Design, https://en.wikipedia.org/wiki/Enigma_machine

[10], [11] Inteligență Artificială, Curs - Anca Mărginean și Artificial Intelligence: A Modern Approach: Russell, Norvig, Prentice Hall, 2002

[12] Rețele Neurale, Structură, https://ro.wikipedia.org/wiki/Re%C8%9Bea_neural%C4%83

[13] R. R. Slavescu - Sisteme Inteligente 2019/2020 - Lecture Notes

[14] Bayesian network, Classification methods, Basic Algorithms for Data Mining

https://www.sciencedirect.com/topics/mathematics/bayesian-network

[15] Algorithms, Classification methods, https://en.wikipedia.org/wiki/Pattern_recognition

[16] This BMW will stop at red lights for you, https://www.whichcar.com.au/car-news/bmw-pioneers-traffic-light-obedient-cars

[17] Tesla Launches Autopilot Traffic Light And Stop Sign Control, https://insideevs.com/news/414820/tesla-autopilot-traffic-light-recognition/

[18] Properties, Classic data sets, https://en.wikipedia.org/wiki/Data_set

[Figura 1.1] https://www.europarl.europa.eu/news/ro/headlines/economy/

[Figura 2.1] https://www.semanticscholar.org/paper/Deep-Convolutional-Traffic-Light-Recognition-for-Bach-Stumper/

[Figura 3.2] https://haltakov.net/blog/how-can-tesla-improve-traffic-light-and-stop-sign-control/

[Figura 4.2] https://www.researchgate.net/figure/Example-of-a-mixed-attribute-dataset-with-two-perturbed-biclusters-highlighted-There_tbl1_320321472

[Figura 4.6] https://towardsdatascience.com/histograms-in-image-processing-with-skimage-python-be5938962935

[Figura 4.10] https://www.sciencedirect.com/topics/engineering/adaboost