

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IASI

**FACULTATEA DE INFORMATICA**



LUCRARE DE LICENTA

**Simularea functionalitatilor unui mouse  
folosind repere faciale**

propusă de

**Sergiu Iacob**

**Sesiunea:** iulie, 2020

Coordonator științific

**Asist. Dr. Croitoru Eugen**

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IASI

**FACULTATEA DE INFORMATICA**

**Simularea funcționalităților unui  
mouse folosind repere faciale**

**Sergiu Jacob**

**Sesiunea:** iulie, 2020

Coordonator științific

**Asist. Dr. Croitoru Eugen**

# Cuprins

<b>Introducere</b>	<b>2</b>
Context . . . . .	2
Idee . . . . .	3
Motivație . . . . .	4
Obiective . . . . .	5
Descriere sumară a soluției . . . . .	5
Structura lucrării . . . . .	7
<b>Contribuții</b>	<b>8</b>
<b>1 Prezentarea problemei</b>	<b>10</b>
1.1 Urmărirea ochilor . . . . .	10
1.2 Simularea funcționalităților mouse-ului . . . . .	11
1.3 Strategie . . . . .	12
1.3.1 Obținerea datelor de antrenament . . . . .	12
1.3.2 Predictia zonei de privire . . . . .	13
1.3.3 Simularea funcționalităților mouse-ului . . . . .	13
1.3.4 Cercetare și analizare a bibliotecilor folosite . . . . .	13
<b>2 Soluția propusă</b>	<b>14</b>
2.1 Informații preliminare . . . . .	14
2.1.1 Informații tehnice . . . . .	14
2.1.2 Python & Conda . . . . .	14
2.1.3 OpenCV . . . . .	15
2.1.4 PyQt5 . . . . .	15
2.1.5 Keras & PyTorch . . . . .	15
2.1.6 dlib . . . . .	16

2.2	Limite și constrângeri . . . . .	16
2.3	Structura aplicației . . . . .	16
<b>3</b>	<b>Date de antrenament</b>	<b>21</b>
3.1	Colectarea și salvarea datelor . . . . .	21
3.2	Procesarea datelor . . . . .	23
3.2.1	Avantajele unei preprocesări . . . . .	23
3.2.2	Folosirea exclusivă a ochilor . . . . .	24
3.2.3	Folosirea întregii fețe . . . . .	26
3.2.4	“Bandă oculară” . . . . .	27
<b>4</b>	<b>Antrenament</b>	<b>29</b>
4.1	Rețele MLP . . . . .	29
4.2	Rețele neuronale de convoluție . . . . .	34
4.2.1	Utilizarea întregii fețe . . . . .	34
4.2.2	Utilizarea “benzilor oculare” ca date de antrenament . . . . .	35
4.2.3	Îmbunătățirea arhitecturii CNN . . . . .	37
4.3	Regresie folosind CNN . . . . .	37
<b>5</b>	<b>Folosirea modelelor antrenate</b>	<b>40</b>
5.1	Deplasarea cursorului . . . . .	40
5.2	Apăsarea butoanelor mouse-ului . . . . .	41
<b>6</b>	<b>Analiză asupra identificării reperelor faciale</b>	<b>43</b>
	<b>Concluzii</b>	<b>49</b>

# Introducere

## Context

Știință și tehnologie. Acest duo se regăsește la orice pas al secolului XXI și interacționăm cu el zilnic, mai mult sau mai puțin, prin intermediul multor dispozitive precum telefonul, televizorul sau calculatorul personal. Într-o formă sau alta, dispozitivele de acest fel (*împreună* cu multimea de aplicații software pe care le rulează) fac lumea mai *accesibilă* pentru utilizatorii lor – de pildă, să verificăm vremea zilei de mâine pe un laptop sau să ne bazăm pe comenzi online în timpul unei pandemii. Astfel de exemple ne arată cum tehnologia modelează felul în care ne desfășurăm activitățile zilnice și “scurtăturile” pe care le putem lua pentru a îndeplini anumite sarcini – desigur, în anumite cazuri, cu niște costuri aferente.

În paragraful de mai sus este accentuat termenul “*accesibil*” care, prin definiție<sup>1</sup>, înseamnă ceva “care este la îndemâna cuiva; care poate fi ușor procurat”. Am văzut cum lumea poate fi “mai la îndemâna cuiva” – dar cum facem ca tehnologia să fie, la rândul ei, *accesibilă*? Cum ar putea, spre exemplu, o persoană paralizată să folosească un laptop?

După o analiză retrospectivă putem constata că oamenii au lucrat dintotdeauna la modalități (de exemplu la dezvoltarea de software) pentru a face tehnologia mai *accesibilă* oamenilor. Un exemplu ar fi Cititorul de ecran (în engleză *Screen reader*), care este incorporat în majoritatea smartphone-urilor recent lansate, sau asistentul inteligent precum Siri, Bixby sau Google Assistant. Acestea pot permite persoanelor fără vedere să interpreteze conținutul unui ecran digital sau persoanelor imobilizate să asculte muzică, să afle noutăți s.a.m.d. Acest tip de software este un factor cheie pentru a permite unor categorii diverse de oameni să poată profita de avantajele tehnologiei.

---

<sup>1</sup>Definiție preluată din DEX 2009

## Idee

Analizând laptop-urile care sunt acum pe piață, am constată că toate sunt echipate cu o cameră frontală de luat vederi pentru videoconferințe, denumită uzual *webcam*. Pentru calculatoarele obișnuite, precum un “sistem desktop” cu un monitor care nu dispune de această cameră integrată, există webcam-uri care se pot conecta printr-un port USB (de cele mai multe ori) și aduc aceeași funcționalitate și unui calculator “tradițional”.



Figura 1: Exemplu de webcam ce poate fi montat pe monitorul unui calculator. Imagine preluată de pe PC Garage

Lucrarea de față ia în considerare popularitatea acestui webcam și propune o soluție pentru a putea folosi parțial un calculator fără ajutorul mâinilor. O mare parte din interacțiunea dintre om și calculator se petrece *prin intermediul mouse-ului*, aşadar m-am concentrat pe simularea comportamentului acestuia *folosind doar caracteristici ale feței*. Ideea de bază constă în a prelua imagini ale utilizatorului de la webcam și, pe baza trăsăturilor faciale, de a simula funcționalități ale acestuia, spre exemplu de a muta cursorul în direcția în care privește utilizatorul. Exemplul cel din urmă este cunoscut în litera științifică drept “*Urmărire ochilor*” (din engleză, “Eye tracking”) și problema poate fi abordată prin tehnici de *Învățare Automată*, o ramură a *Inteligentei Artificiale*.

Această idee nu este nouă și există deja soluții pentru această problemă, bazate pe aceeași idee. Totuși, cele mai multe dintre ele sunt create ori doar pentru un anumit sistem de operare (în acest caz, Windows), ori nu sunt gratuite și au un cost atașat semnificativ, sau chiar necesită componente hardware adiționale.

*Camera Mouse*<sup>2</sup> este o propunere viabilă care urmărește o portiune fixată a feței (spre exemplu vârful nasului) și, când acea portiune își schimbă poziția, se schimbă și

---

<sup>2</sup><http://www.cameramouse.org>

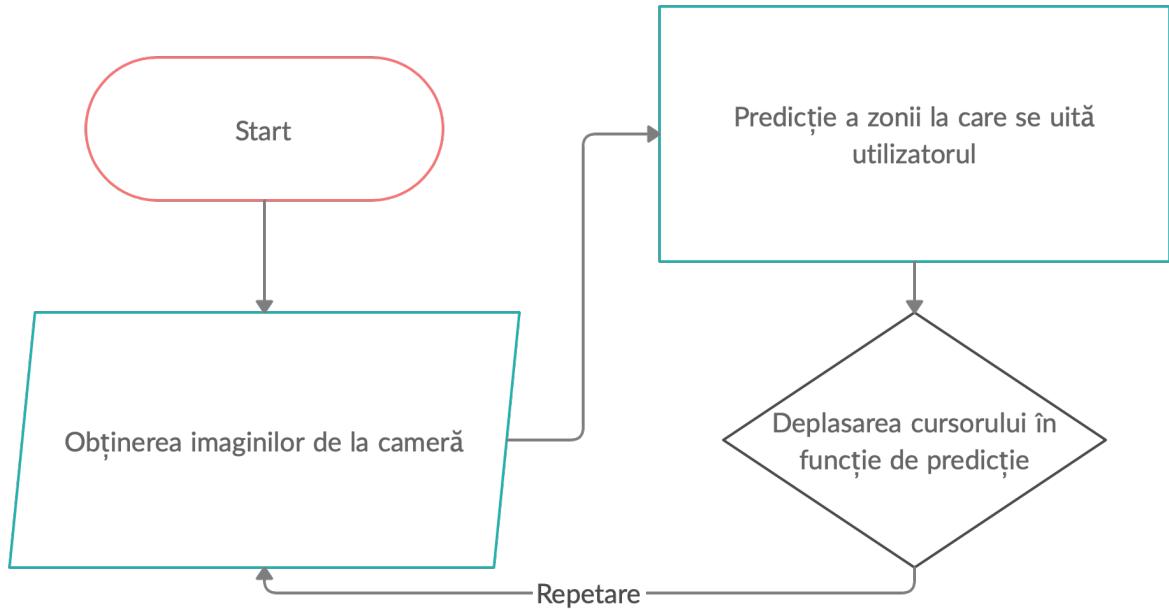


Figura 2: Schemă generală pentru deplasarea cursorului

poziția cursorului. Pentru ca acest lucru să funcționeze, utilizatorul trebuie să efectueze mișcări ale capului. Oferă și funcționalități de simulare a apăsării pe butoanele mouse-ului, însă aplicația funcționează doar pentru sistemele care rulează Windows. Printre alternative se mai găsesc *IntelliGaze*<sup>3</sup> și produse dezvoltate de *Tobii Dynavox*<sup>4</sup>, dar acestea necesită în primul rând hardware adițional, lucrează doar pentru sistemul de operare Windows și au și un cost atașat.

## Motivație

Când a trebuit să mă decid asupra temei lucrării de licență, am luat în calcul doi factori cheie: viitoarea mea carieră profesională și utilitatea proiectului. Mi-am dorit să lucrez la un proiect care mi-ar alimenta interesul în Inteligența Artificială și care mi-ar oferi șansa de a aplica cercetarea pe care aş face-o în acest domeniu. Mai mult, mi-am dorit de asemenea să am și o abordare practică asupra lucrării, astfel încât să construiesc ceva ce ar fi folositor.

Cât despre Inteligența Artificială, este inutil să-i subliniem importanța contemporană. De la aplicabilitatea medicală, conducere/pilotare autonomă, agricultură

<sup>3</sup><https://www.intelligaze.com/en/>

<sup>4</sup><https://www.tobiidynavox.com/software/windows-software/windows-control-2/>

inteligentă până la frigidere inteligente care-ți spun când ai rămas fără lapte, Inteligența Artificială este larg răspândită și extinderea ei nu se va opri prea curând. Pentru mine, acesta este un motiv în plus pentru a o studia și a o înțelege mai bine, mai ales că o găsim integrată în viața noastră de zi cu zi.

## Obiective

Cel mai important obiectiv al acestei lucrări de licență este înțelegerea și aplicarea cu succes a cunoștințelor și a noțiunilor dobândite ca student al Facultății de Informatică, Iași. Toate noțiunile teoretice și practice acumulate trebuie să se regăsească într-o combinație armonioasă pentru a putea permite dezvoltarea unui software de calitate. Am încercat, în acest sens, să țin cont de concepte precum principii de *software design* (spre exemplu principiile de programare *SOLID*), de *design patterns* precum *MVP*, de concepte de analizare a complexității timp/spațiu, de noțiuni legate de învățarea automată (spre exemplu tratarea *overfitting-ului*) și altele.

Obiectivul principal al aplicației este acela de a simula folosirea unui mouse doar prin gesturi ale feței. Prima parte a obiectivului se traduce în urmărirea ochilor utilizatorului, pentru a ști în ce direcție să deplasăm cursorul, și este partea pe care se concentreză cel mai mult această lucrare. Având această capacitate, aplicația trebuie apoi să pună la dispoziție utilizatorului modalități de a simula, prin folosirea feței, cele mai importante funcționalități ale unui mouse:

- mișcarea cursorului
- apăsarea butonului stâng
- apăsarea butonului drept<sup>5</sup>

## Descriere sumară a soluției

Fața umană poate fi analizată pe baza mai multor *repere faciale*. Acestea sunt reprezentate prin anumite puncte de pe față precum centrul ochiului, centrul nasului, centrul buzei superioare etc. Cunoscând acestea, putem delimita zone ale feței precum un singur ochi. În figura de mai jos<sup>4</sup>, ochiul stâng al unei persoane este delimitat de înfășurătoarea convexă a punctelor cu indicii [43, 48].

---

<sup>5</sup>Aceste funcționalități mai sunt denumite ușual și “click stânga/dreapta”

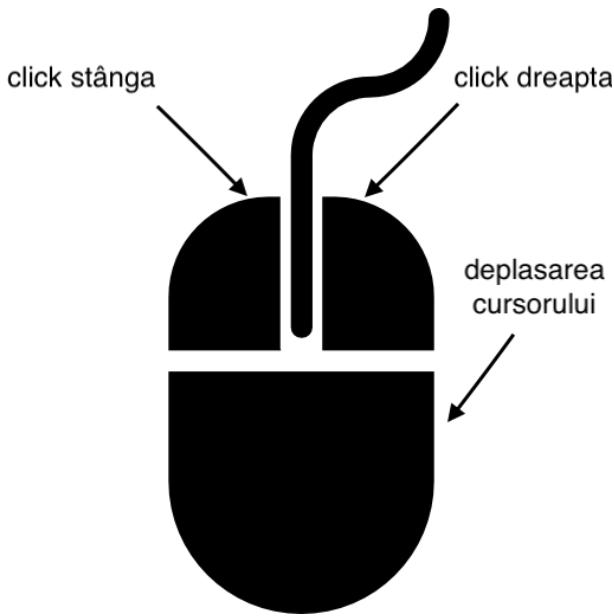


Figura 3: Funcționalități principale ale unui mouse. Imagine preluată și adaptată de pe Flaticon, autor: Kiranshastry

Folosind anumiți algoritmi deja existenți am detectat aceste puncte faciale din imagini ale utilizatorului aplicației (în acest caz, imagini cu mine însuși) iar apoi am extras regiuni dreptunghiulare care conțineau fie întreaga față, fie ochii persoanei. Apoi am etichetat aceste imagini decupate cu coordonatele punctelor de pe ecran pe care le privea utilizatorul în fiecare imagine și am dezvoltat un model matematic care poate calcula zona ecranului pe care o privește utilizatorul.

Mai departe am făcut uz de restul reperelor faciale pentru a putea atinge obiectivele prezentate mai devreme. Fiind capabilă să urmărească ochii, aplicația trebuie și să poată deplasa cursorul, dar într-un mod controlat, care să nu deranjeze privirea normală a ecranului (nu întotdeauna când privim ecranul dorim să deplasăm cursorul). Astfel, deplasarea cursorului se realizează doar în momentul în care distanța dintre buza superioară și cea inferioară este nenulă și reprezintă un anumit procent din distanța dintre extremitățile buzelor (care coincid). În termeni mai populari și mai simpli, relația se traduce prin gestul de deschidere a gurii.

Pentru apăsarea butoanelor am avut o abordare similară. Prin închiderea ochiului stâng pentru o anumită perioadă de timp se realizează apăsarea butonului stâng. Analog, apăsarea butonului drept se realizează similar. Aceste lucruri au dus la înăpereea obiectivelor prezentate mai devreme, întrucât aplicația poate fi manipulată exclusiv prin față utilizatorului.

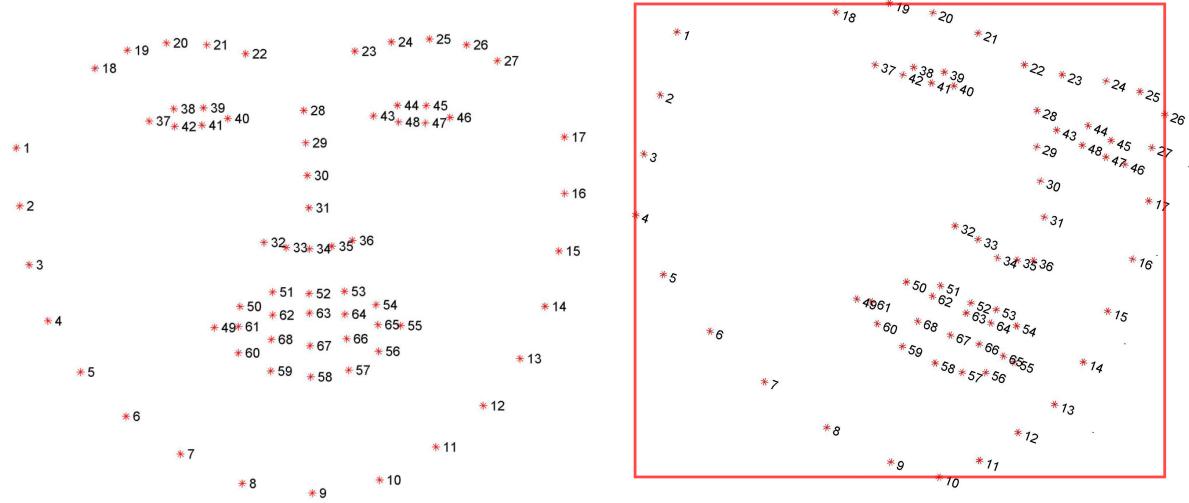


Figura 4: Repere faciale. Imagini preluate de aici: <https://ibug.doc.ic.ac.uk/resources/300-W/>

## Structura lucrării

În primul capitol am prezentat principala problemă pe care o analizează această lucrare, și anume urmărirea ochilor (*eye tracking*). Este prezentată de asemenea strategia pe care am abordat-o spre rezolvarea acestei probleme.

Următorul capitol prezintă soluția propusă împreună cu limitele și constrângerile acesteia. Am ilustrat structura acesteia, componentele principale și şablonul arhitectural (*design pattern-ul*) folosit pentru a ghida dezvoltarea aplicației.

Capitolele 3, 4 și 5 urmăresc pașii luați în rezolvarea unei probleme de învățare automată. Am început cu modul în care am obținut datele de antrenament, modul în care le-am procesat iar apoi am continuat cu prezentarea modului de antrenare și de dezvoltare a arhitecturilor de învățare profundă pe care le-am folosit. Capitolul 5 prezintă modul în care am folosit predicțiile de urmărire a ochilor în combinație cu repere faciale pentru a simula funcționalitatea unui mouse.

În cele din urmă, capitolul 6 ilustrează un experiment pe care l-am făcut pentru identificarea reperelor faciale. Acesta m-a ajutat la o înțelegere mai bună a modului în care reperele faciale sunt identificate.

# Contribuții

Analizând produsele deja existente bazate pe o idee similară am constatat că fiecare dintre ele prezintă un neajuns. Multe aplicații sunt concepute, spre exemplu, doar pentru sistemul de operare Windows. O analiză a cotei de piață pentru sistemele de operare pentru sisteme desktop ne indică faptul că există un număr semnificativ de utilizatori care nu folosesc mașini ce rulează Windows. Așadar, aplicația propusă este *cross-platform*, putând fi rulată pe cele mai importante 3 sisteme de operare: Windows, MacOS și Linux.

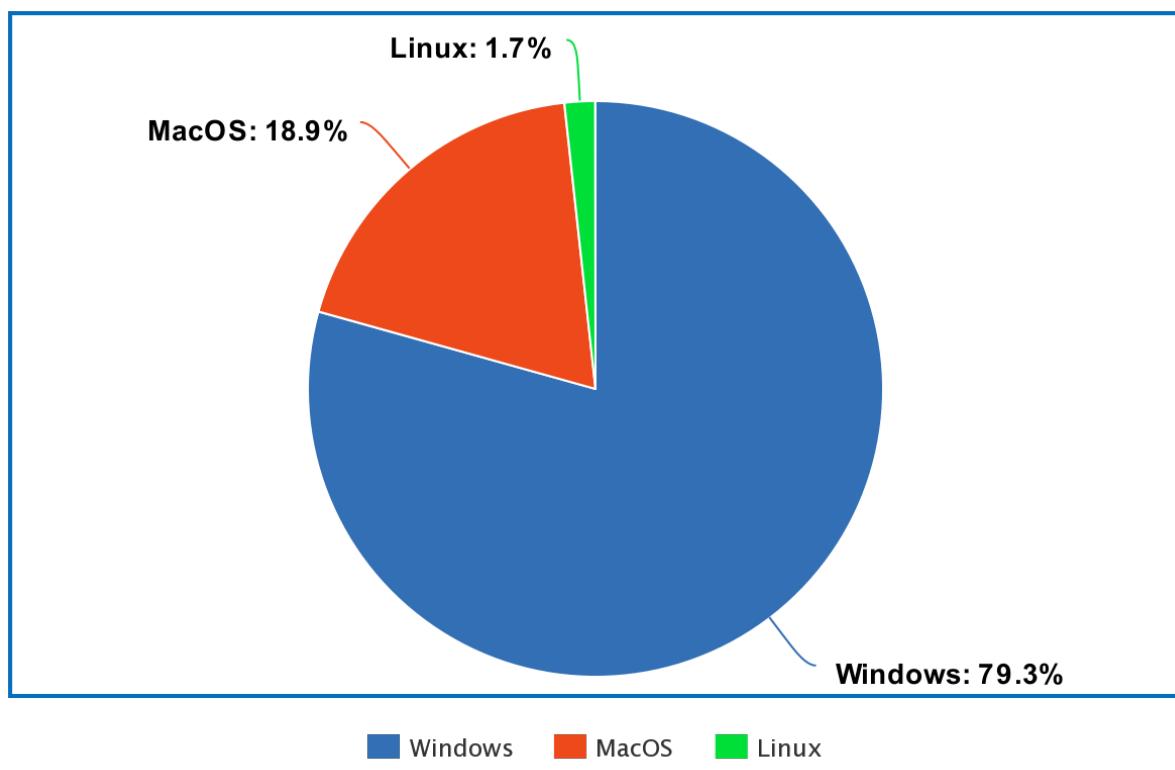


Figura 5: Distribuție a sistemelor de operare desktop în Mai, 2020. Date preluate de pe StatCounter GlobalStats

Un alt punct important este că unele aplicații necesită camere speciale sau senzori

speciali, ceea ce aduce un cost în plus și poate fi un impediment în utilizarea aplicației. Lucrarea de față nu necesită decât o cameră webcam, hardware care este deja existent pe orice laptop comercializat astăzi sau care poate fi atașat unui calculator obișnuit pentru un cost redus.

O ultimă mențiune este legată de diferențele fizionomice dintre persoane. În acest sens, aplicația se poate “adapta” fizionomiei fiecărui utilizator, întrucât pentru a analiza punctul de privire al utilizatorului sunt folosite imagini cu acesta. Acest lucru aduce un pas în plus pentru “configurarea” aplicației (colectarea de date), dar va rezulta într-o experiență mai robustă pentru fiecare utilizator în parte.

# Capitolul 1

## Prezentarea problemei

### 1.1 Urmărirea ochilor

Problema pe care am încercat să o rezolv constă în primul rând în a urmări cu acuratețe ochii utilizatorului, astfel încât cursorul să poată fi mișcat în concordanță cu privirea acestuia. Această problemă face parte dintr-o gamă mai largă de probleme de *Viziune Computerizată* (în engleză *Computer Vision*), denumită chiar *urmărirea ochilor*, după cum a fost menționat și în introducere. Conform Wikipedia, 2020, urmărirea ochilor este “procesul de măsurare a punctului de privire (unde se uită o persoană) sau a mișcării unui ochi relativ la cap”.<sup>1</sup>

Tehnicile de ultimă oră (*state of the art*) de a rezolva această problemă se bazează pe *Inteligenta Artificială* și sunt, mai exact, tehnici de *Învățare Automată*. Ciortuz, 2020 explică în termeni foarte simpli acest subdomeniu al informaticii: “Învățarea Automată este programare bazată pe date”<sup>2</sup>. Învățarea poate fi la rândul ei supervizată, semi-supervizată sau nesupervizată, aceste tehnici încearcând să prezică, să producă, să generalizeze niște rezultate pe baza unor exemple sau a unor relații dintr-o mulțime de date deja cunoscută. Diferența cheie între acestea constă în structura acestei multimi de date, structură care la rândul ei influențează abordările de învățare.

Această ramură a Inteligenței Artificiale poate fi mai departe divizată în mai multe secțiuni, una dintre ele fiind *Innvățarea Profundă*. Ea se preocupă, printre altele, de procesarea și analizarea imaginilor prin folosirea unor *arhitecturi profunde* bazate pe *rețele neuronale*.

---

<sup>1</sup>Textul original este din engleză: “the process of measuring either the point of gaze (where one is looking) or the motion of an eye relative to the head”

<sup>2</sup>Traducere liberă; text original: “ML is data-driven programming”

Am formulat problema ca una de învățare profundă supervizată, care presupune cunoașterea unei multimi de date

$$D = \{(i, o) | i \in I, o \in O\}$$

unde  $(i, o)$  reprezintă un exemplu, o asociere între un tip de date și un rezultat pe care vrem să îl prezicem, să îl reconstruim. Pentru fiecare element din mulțimea  $I$ , avem un element asociat în  $O$  care reprezintă mulțimea *adevărurilor de bază* deja cunoscute, pe care trebuie să le putem reproduce și prezice corect pentru alte elemente de tipul celor din mulțimea  $I$ .

Am definit obiectivul ca fiind acela de a prezice zona de pe ecran pe care o privește utilizatorul. Pentru acest lucru am plasat o “grilă” pe ecran și am numerotat fiecare celulă corespunzătoare. Am experimentat cu mai multe dimensiuni ale grilei, precum 2x2, 3x2 și 4x4.

0	1	0	1	2	0	1	2	3
		3	4	5	4	5	6	7
2	3	6	7	8	8	9	10	11
					12	13	14	15

Figura 1.1: Exemple de grile plasate pe ecran

Așadar, mulțimea  $I$  de mai sus va fi alcătuită din imagini ale utilizatorului, care ar permite “extragerea privirii” acestuia, iar mulțimea  $O$  va conține, spre exemplu, numărul celulei pe care o privea utilizatorul în imaginea respectivă. Mulțimea  $O$  conține valori discrete și este finită, așadar problema inițială este una de *clasificare*. Vom considera că mulțimea  $I$  nu conține imagini în care utilizatorul nu se uită la un punct de pe ecran.

Prima parte a problemei se traduce, așadar, astfel: dându-se o imagine cu utilizatorul privind ecranul, să se calculeze numărul celulei (în funcție de grila folosită) pe care utilizatorul o privește.

## 1.2 Simularea funcționalităților mouse-ului

A doua parte a problemei ridică următoarea întrebare: cum putem simula un mouse folosind reperele faciale? Cum asigurăm o mișcare fluidă a cursorului spre o

“zonă țintă”, dar în același timp cursorul să poată rămâne fix? Dacă am deplasa constant cursorul pe ecran, în funcție de punctul de privire, atunci utilizatorul ar putea fi distras atunci când doar ar privi ecranul (de exemplu când ar citi un document) sau ar efectua operația de click stânga/dreapta accidental, asupra altor elemente grafice.

Fiecare zonă a ecranului conține un număr finit de pixeli pe care utilizatorul i-ar putea privi. Trebuie să-i putem oferi astfel utilizatorului posibilitatea de a deplasa cursorul asupra acestor pixeli. Un buton plasat pe câțiva pixeli ar fi aproape imposibil de apăsat pe orice ecran digital contemporan și ar și obosi ochii, așa că orice element grafic al unei aplicații software se întinde pe un număr mai mare de pixeli. Profitând de acest lucru, putem spune că aplicația trebuie să poată deplasa cursorul *aproape* de orice pixel al ecranului.

În cele din urmă mai rămâne doar simularea apăsării butoanelor de pe mouse. Utilizatorul trebuie să poată solicita într-un mod explicit acest lucru, folosindu-se doar de față, iar aplicația trebuie să asigure evitarea cazurilor de tipul *false positive*, adică a apăsărilor butoanelor atunci când acest lucru nu este dorit.

## 1.3 Strategie

### 1.3.1 Obținerea datelor de antrenament

Primul pas spre rezolvarea problemei este *colectarea datelor de antrenament*. Este foarte bine cunoscut că un algoritm de învățare automată este pe atât de bun pe cât sunt datele pe care îi le furnizăm. Acestea sunt incredibil de importante, așa că m-am concentrat pe a dezvolta niște modalități facile de a aduna o mulțime consistentă de date.

O altă etapă esențială este *procesarea de date*. Aceasta se ocupă cu simplificarea și curățarea setului de date, cu eliminarea instanțelor de antrenament care sunt inutile și cu extragerea exclusiv a informațiilor care sunt de folos și cu înlăturarea a ceea ce rămâne. Optional, în această etapă se mai realizează și diferite transformări pentru a aduce datele dintr-o formă neprelucrată într-o formă convenabilă algoritmilor pe care îi folosim. În acest sens, am procesat datele neprelucrate în diverse moduri pentru a determina cea mai utilă stare în care le pot aduce.

### **1.3.2 Predicția zonei de privire**

Următorul pas a fost cel de antrenare și de dezvoltare a unui model matematic capabil să prezică zona de privire a utilizatorului. Pentru acest lucru m-am concentrat pe studiul *rețelei neuronale*, văzută drept “calul de bătaie” și baza pentru problemele de Viziune Computerizată. Am folosit o arhitectură bazată pe aceasta (rețea de tip *MLP*) ca un punct de pornire spre a rezolva problema și ca un prim experiment.

Folosind acest tip de rețea, am încercat să ating cel mai important obiectiv al acestei lucrări al licenței, și anume de a prezice aproximativ zona în care se uită utilizatorul, pentru a putea deplasa cursorul în acea zonă. Din experimentele efectuate, această arhitectură a rezultat într-o primă soluție promițătoare, fiind capabilă să urmărească, într-o anumită măsura, privirea utilizatorului.

Apoi a urmat studierea și aplicarea *rețelelor neuronale convolutionale*, o arhitectură de bază pentru multe metode *state of the art* pentru problemele din domeniul Viziunii Computerizate. Cu ajutorul acestora, urmărirea ochilor a devenit mai robustă și mai puțin sensibilă la diferențele dintre imagini, precum lumină sau poziționarea persoanei în fața webcam-ului.

### **1.3.3 Simularea funcționalităților mouse-ului**

Având un model care poate satisface nevoia de mai sus, de urmărire a ochilor, am analizat apoi cum pot simula efectiv funcționalitatea unui mouse. Pentru aceasta, am analizat modalități de a folosi reperele faciale<sup>4</sup> pentru a folosi gesturi ale feței și a simula aceste funcționalități. Soluția a constat în a folosi gesturi precum închiderea unui ochi și gestul de deschidere a gurii.

### **1.3.4 Cercetare și analizare a bibliotecilor folosite**

În lucrarea de licență am folosit diverse biblioteci printre care și `dlib`, pe care am folosit-o pentru a identifica reperele faciale ale utilizatorului. Am fost curios despre cum anume funcționează această bibliotecă și am făcut un mic experiment în care am încercat să reconstruiesc funcționalitatea acestei librării.

# Capitolul 2

## Soluția propusă

### 2.1 Informații preliminare

#### 2.1.1 Informații tehnice

Pentru dezvoltarea, structurarea și versionarea codului sursă al lucrării, am folosit platforma gratuită Github<sup>1</sup>. *Repository-ul* proiectului poate fi accesat la această adresă: <https://github.com/sergiuiacob1/iClicker/>.

Rezultatele prezentate aici au fost bazate pe imagini capturate prin intermediul unei camere webcam capabile de o rezoluție maximă HD (1280x720 pixeli). Experimentele realizate au fost făcute în general în medii bine luminate, întrucât o dată cu scăderea intensității luminii suferă și utilitatea aplicației prezentate aici din cauza calității webcam-ului.

#### 2.1.2 Python & Conda

Unul dintre cele mai populare limbi de programare când vine vorba de învățare profundă este Python<sup>2</sup>. Astfel, am ales să dezvolt aplicația folosind acest limbaj, deoarece suportul din partea comunității este unul foarte bun și resursele online pentru a rezolva probleme comune sunt vaste. De asemenea, este un limbaj potrivit pentru teste, experimente și pentru o prototipizare rapidă a anumitor funcționalități, iar pe parcursul dezvoltării am făcut numeroase astfel de teste.

Am facut uz de asemenea de tehnologia Conda<sup>3</sup> care ajută în gestionarea mediilor

---

<sup>1</sup><https://github.com>

<sup>2</sup><https://www.python.org>

<sup>3</sup><https://docs.conda.io/en/latest/>

de dezvoltare. Cu ajutorul acestora am putut crea un mediu de dezvoltare separat, care se poate instala cu ușurință pe orice calculator fără a crea conflicte cu alte pachete deja existente pe mașina unui utilizator. Această combinație a ajutat la înăperearea necesității aplicației de a rula pe mai multe sisteme de operare, întrucât Python deja satisface această nevoie. Versiunile folosite au fost Python 3.7 și Conda 4.8.0.

### 2.1.3 OpenCV

Una dintre bibliotecile Python care au adus funcționalități cruciale acestui proiect este OpenCV<sup>4</sup>. Aceasta conține diferite funcționalități legate de Viziunea Computerizată, precum captura de imagini prin intermediul webcam-ului. Am folosit-o de asemenea pentru a realiza redimensionări de imagini, pentru a le converti în gri sau în imagini alb-negru prin aplicarea unui *binary threshold*. Versiunea folosită a fost 4.1.2.

### 2.1.4 PyQt5

Aplicația dezvoltată are și o interfață grafică ce a fost implementată folosind biblioteca PyQt5<sup>5</sup>. Unul dintre avantajele acestei biblioteci este acela că oferă un nivel înalt de abstractizare și componente (butoanele, ferestrele etc.) grafice au un aspect diferit în funcție de sistemul de operare pe care rulează aplicația, fără a fi nevoie ca acest lucru să fie implementat de dezvoltator. Versiunea folosită a fost 5.14.

### 2.1.5 Keras & PyTorch

Keras<sup>6</sup> și PyTorch<sup>7</sup> au fost folosite pentru a facilita antrenarea rețelelor neuronale (de tip *MLP* sau *CNN*). Acestea au mărit cu mult realizarea arhitecturilor pentru învățarea automată, Keras având un nivel de abstractizare decât PyTorch. Am folosit cele două tehnologii pentru a căpăta experiență în utilizarea amânduror, întrucât nu există o “cea mai bună unealtă”, ci mai degrabă contextul nevoii dictează uneltele, tehnologiile ce ar trebui folosite. Versiunile folosite au fost Keras 2.2.4 și PyTorch 1.4.

---

<sup>4</sup><https://pypi.org/project/opencv-python/>

<sup>5</sup><https://pypi.org/project/PyQt5/>

<sup>6</sup><https://keras.io>

<sup>7</sup><https://pytorch.org>

### **2.1.6 dlib**

Cea mai importantă bibliotecă de care am făcut uz în această lucrare este dlib<sup>8</sup>. Cu ajutorul acesteia, am putut realiza identificarea reperelor faciale într-o imagine care conținea fața unei persoane. Reperele respective sunt furnizate de către sub forma unei liste de coordonate  $(x, y)$  corespunzătoare pixelilor acelor caracteristici faciale. Mai departe, pe baza acestora, am putut decupa fie fața, fie ochii persoanei în imagini mai mici pe care le-am folosit mai apoi ca date de antrenament. Versiunea folosită a fost 19.19.0.

## **2.2 Limite și constrângeri**

Aplicația are niște limite și lucrează de asemenea cu niște presupuneri, precum faptul că utilizatorul folosește un singur monitor și un singur webcam. De asemenea, imaginile folosite sunt cu mine însuși, deci trebuie luat în vedere acest lucru pentru orice rezultat prezentat.

Aplicația este menită să se poată “mula” pe fizionomia utilizatorului, însă este posibil să aibă performanțe mai slabe pentru persoanele care poartă ochelari, spre exemplu. Motivul pentru care se întâmplă acest lucru este acela că aplicația lucrează cu ochii utilizatorului, iar dacă lumina se reflectă în lentilele ochelarilor, ochii ar putea fi indistinctibili. Ca o ultimă mențiune, aplicația se concentrează majoritar pe poziția pupilelor relativ la ochi (glob ocular + anexe ale globului ocular), deci se va considera că poziția capului nu va suferi schimbări majore între datele de antrenament și datele de test.

## **2.3 Structura aplicației**

Aplicația are 4 mari componente, corespunzătoare pașilor luați în rezolvarea unei probleme de învățarea automată: colectarea de date, procesarea acestora, antrenarea unui model care realizează urmărirea ochilor și simularea funcționalităților mouse-ului. Utilizatorul poate interacționa cu aceste componente prin intermediul interfaței grafice realizată în PyQt5. Fereastra principală a aplicației are și o parte în care vor fi afișate informații utile.

---

<sup>8</sup><https://pypi.org/project/dlib/>

Pentru a construi aplicația m-am ghidat după modelul de proiectare MVP (*Model–View–Presenter*). Fiecare componentă care poate fi folosită de utilizator are o interfață grafică atașată (*View*) ce poate cere aplicației (prin intermediul unui *Presenter*) să realizeze anumite proceduri. Partea de *Model* al acestui şablon de dezvoltare constă în datele colectate neprelucrate și a celor rezultate din partea de procesare a datelor, precum și în modelele antrenate folosind aceste date.

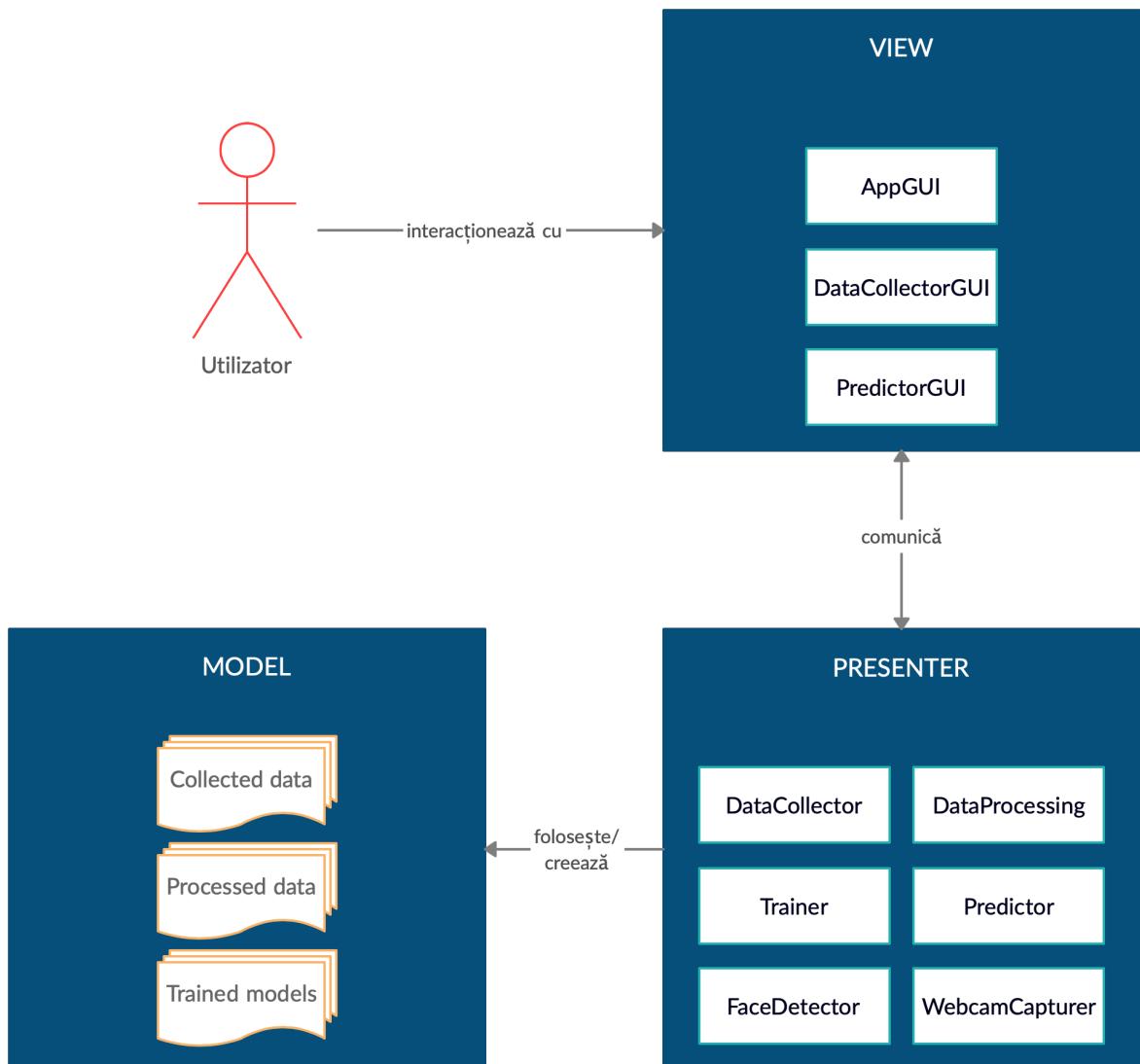


Figura 2.1: Arhitectura MVP a aplicației

Pentru a realiza funcția de colectare a datelor, utilizatorul trebuie să apese pe butonul denumit sugestiv “Colectare date”. Există două moduri în care poate fi realizată această funcție: modul activ și modul pasiv.

Modul activ presupune ca utilizatorul să urmărească cursorul mouse-ului pe ecran

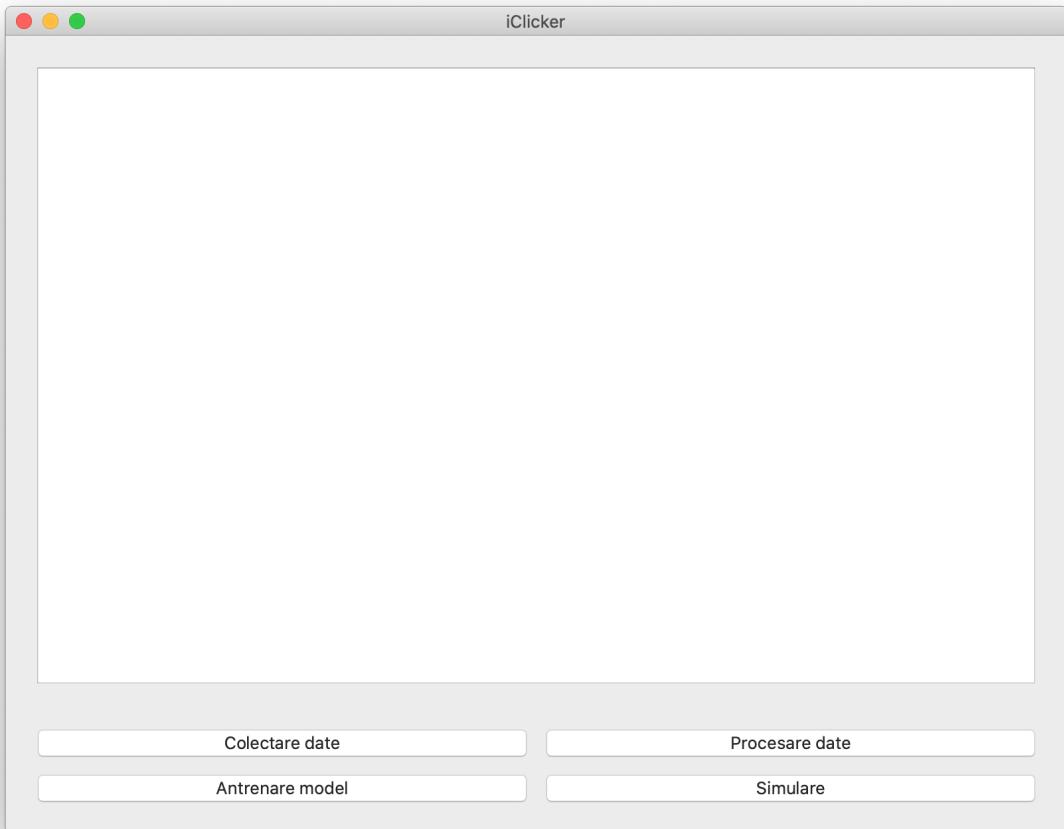


Figura 2.2: Fereastra principală a aplicației

în timp ce acesta se mișcă pe un traseu fix, prin mișcări glisante, de la stânga la dreapta, apoi de la dreapta la stânga, astfel încât să acopere toată suprafața ecranului. De fiecare dată când cursorul își schimbă poziția, aplicația salvează o imagine preluată de la webcam împreună cu poziția cursorului în acel moment. Pentru a evita obosirea ochilor și a asigura o colectare de date consistentă, utilizatorul poate lua o pauză (întreruperea mișcării cursorului și a colectării datelor) prin apăsarea tastei *SPACEBAR*. De asemenea, viteza cu care glisează cursorul poate fi schimbată prin apăsarea tastelor *SĂGEATĂ SUS* și *SĂGEATĂ JOS*.

Metoda pasivă este menită să nu deranjeze rutina utilizatorului, astfel încât de fiecare dată când utilizatorul apasă pe butonul stâng al mouse-ului, aplicația salvează, în același mod ca mai sus, o imagine capturată prin intermediul webcam-ului și poziția cursorului pe ecran. Această variantă poate fi contraintuitivă, având în vedere că aplicația ar trebui folosită fără ajutorul mâinilor, dar m-a ajutat pe mine în obținerea



Figura 2.3: Mesaje de eroare

datelor în timp ce realizam alte sarcini. Este important de menționat că de multe ori nu ne uităm acolo unde apăsăm cu mouse-ul, aşa că, deși este o metodă gândită să ruleze pe fundal, este bine ca utilizatorul să țină cont de prezența acesteia și să realizeze apăsări de buton acolo unde se uită, pentru a construi *date consistente*.

Partea de procesare de date și de antrenare a modelelor se folosesc în același mod, prin apăsarea butoanelor denumite sugestiv. Acestea vor informa utilizatorul despre progresul realizat prin intermediul ferestrei principale.

Toate aceste funcționalități rulează pe câte un *fir de execuție* separat. Dacă acest lucru nu s-ar petrece, atunci firul principal de execuție (care se ocupă și de afișarea componentelor grafice) ar fi “blocat” până la terminarea execuției acelor funcționalități și aplicația ar “îngheța”.

Ultima componentă este și cea mai importantă, cea de simulare a funcționalităților mouse-ului. Prin intermediul acesteia, aplicația începe să urmărească ochii utilizatorului și să-i analizeze gesturile faciale. Utilizatorului i se va arăta pe ecran și o grilă (de dimensiune 3x3) pentru a vedea în timp real predicțiile rețelei convolutionale legate de zona ecranului pe care acesta o privește.

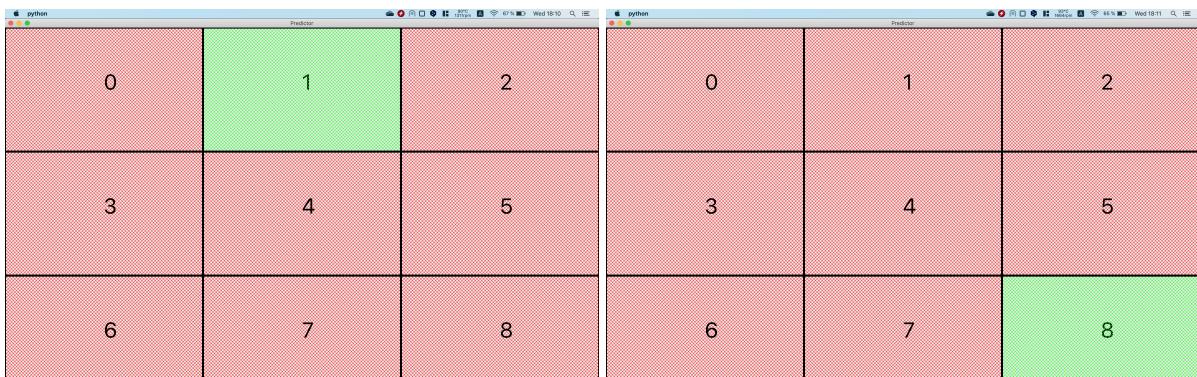


Figura 2.4: Testarea predicțiilor ale zonei de privire ale utilizatorului

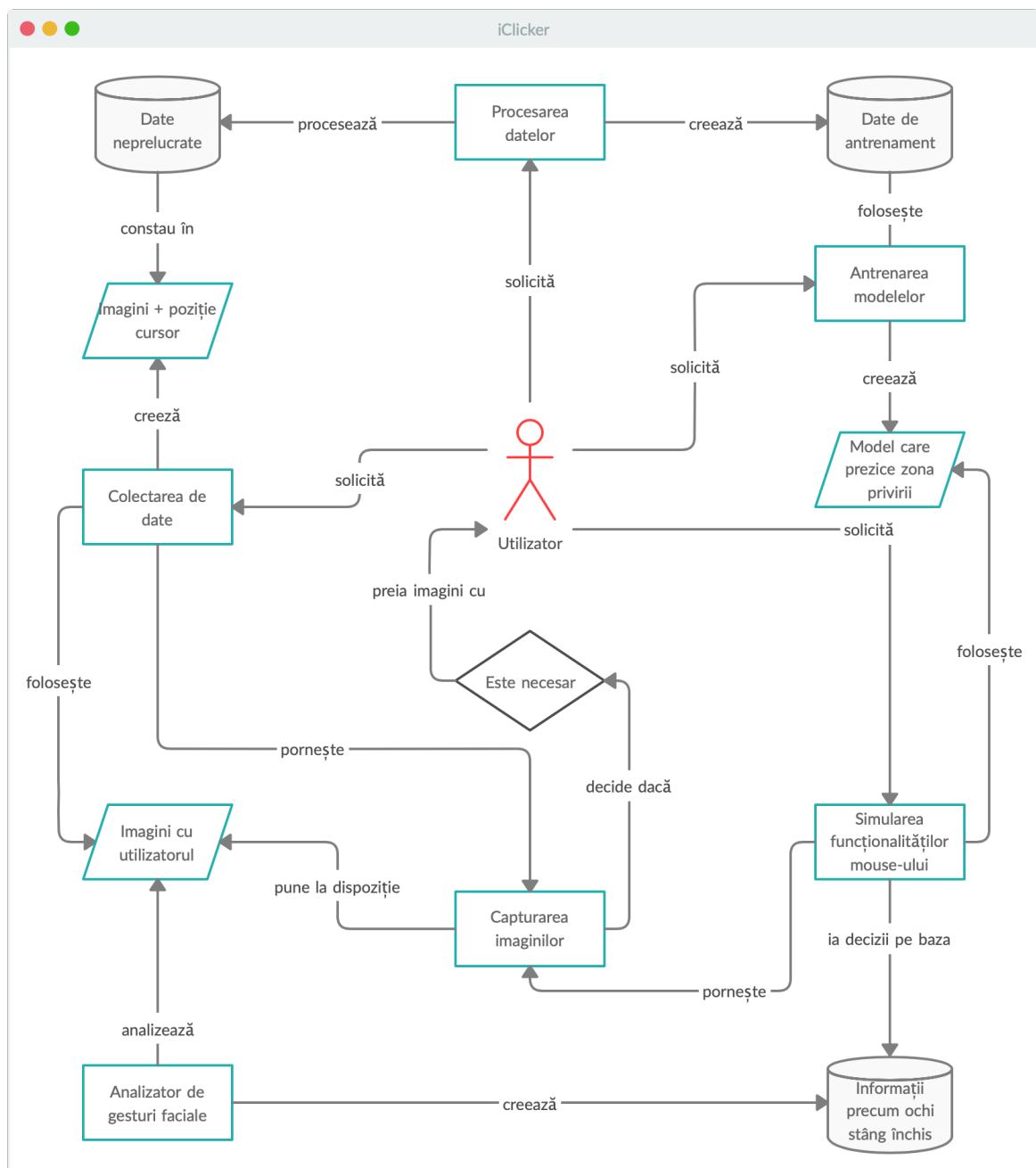


Figura 2.5: Organigramă aplicației

# Capitolul 3

## Date de antrenament

### 3.1 Colectarea și salvarea datelor

Datele de antrenament sunt extrem de importante. O expresie populară (*garbage in, garbage out*) ne spune că dacă datele furnizate algoritmilor de învățare automată sunt de proastă calitate, atunci aşa va fi și performanța acestor algoritmi, sau mai bine spus a modelelor matematice și a generalizațiilor create de aceștia. Multimea de date trebuie să conțină varietate, în cazul de față însemnând poze cu fundaluri diferite, condiții de iluminare diverse și.a.m.d.

Am lucrat cu imagini cu utilizatorul, capturate prin webcam. Când colectarea datelor este gata, acestea sunt salvate sub forma unei “sesiuni”. Fiecare sesiune este definită de numărul imaginilor care au fost capturate, de rezoluția ecranului și rezoluția webcam-ului. Fiecare sesiune conține, pentru fiecare imagine, poziția de pe ecran la care se uita utilizatorul în momentul capturării imaginii.

```
1 def start_collecting(self, collection_type):
2     dc_logger.info(f'Start collecting data in {collection_type} mode')
3     WebcamCapturer.start_capturing()
4     self.gui.start()
5     dc_logger.info('DataCollectorGUI started')
6     if collection_type == 'background':
7         self.mouse_listener.start_listening()
8         dc_logger.info('Mouse listener started')
9     elif collection_type == 'active':
10        threading.Thread(target=self.start_active_collection).start()
```

Listing 3.1: Colectarea datelor

```

1 def save_collected_data(self):
2     if len(self.collected_data) == 0:
3         return
4     dc_logger.info(
5         f'Acquiring lock for data collection. Locked = {self.
6         collect_data_lock.locked() }')
7     self.collect_data_lock.acquire()
8     dc_logger.info('Lock acquired')
9     session_no = self.get_session_number()
10    dc_logger.info(f"Saving data for session_{session_no}")
11    self.save_session_info(session_no)
12    self.save_images_info(session_no)
13    self.save_images(session_no)
14    self.collect_data_lock.release()
15    self.collected_data = []
dc_logger.info('Saving data done')

```

Listing 3.2: Salvarea datelor

Toate datele folosite le-am plasat în directorul *data*. Acesta conține două subdrectoare (*images* și *sessions*) și un fișier *sessions.json* cu informații despre sesiunile de colectare. Imaginele sunt denumite sugestiv (*NumărSesiune\_NumărImagine.png*) și sunt salvate în formatul *PNG* (*Portable Network Graphics*). Directorul *sessions* conține fișiere în formatul *JSON* cu informațiile despre imaginile obținute în acea sesiune.

The image shows two side-by-side code editor panes. The left pane displays the contents of the file `sessions.json` with the following JSON structure:

```

data > {} sessions.json > ...
1  {
2      "total_sessions": 9,
3      "session_1": {
4          "items": 2575,
5          "timestamp": 1584973094.580029,
6          "screen_size": [
7              1440,
8              900
9          ],
10         "webcam_size": [
11             640,
12             480
13         ]
14     },

```

The right pane displays the contents of the file `session_4.json` with the following JSON structure:

```

data > sessions > {} session_4.json > ...
1  [
2      "0": {
3          "mouse_position": [
4              614.99609375,
5              215.87890625
6          ]
7      },
8      "1": {
9          "mouse_position": [
10             493.20703125,
11             436.5546875
12         ]
13     },
14     "2": {

```

Figura 3.1: Fisiere cu informații despre sesiuni

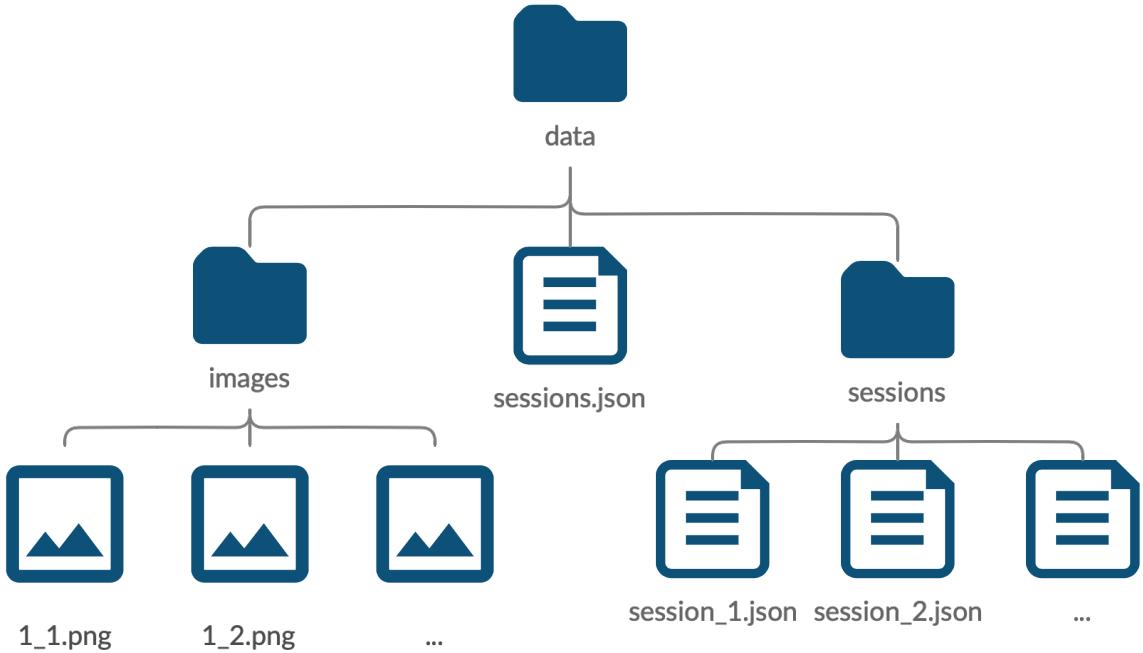


Figura 3.2: Structurarea datelor

## 3.2 Procesarea datelor

Din imaginile colectionate am extras fie doar față acestuia din imagine, fie doar un ochi, fie o portiune dreptunghiulară în care se regăsesc ambii ochi. Pentru fiecare experiment realizat, am menționat ce date au fost folosite și în ce mod au fost procesate acestea.

### 3.2.1 Avantajele unei preprocesări

Procesarea datelor joacă un rol important în soluționarea unei probleme de învățare automată. În primul rând, este să eliminăm orice fel de “zgomot” sau informație neesențială din mulțimea noastră de date. Apoi, putem deriva alte informații din datele inițiale, neprelucrate, informații care pot aduce un plus de performanță în soluția finală a problemei.

În cadrul unui experiment realizat m-am confruntat cu o situație care a dovedit chiar necesitatea (nu doar avantajul) de a folosi procesarea de date. În lipsa normalizării datelor, rețeaua a fost incapabilă să învețe, indiferent de timpul de antrenare pe care îl-am acordat.

### 3.2.2 Folosirea exclusivă a ochilor

În unele experimente, care urmează a fi prezentate în capitolul următor, m-am axat pe extragerea ochilor din imagini. Pentru asta, am făcut uz de două biblioteci Python: dlib și imutils. Bazându-mă pe reperele faciale care au fost prezentate în introducere<sup>4</sup>, am extras doar porțiunile imaginilor care delimităază ochii. Fiecare ochi a fost redimensionat la 60x30 pixeli apoi au fost uniți orizontal, formând o imagine alb-negru de dimensiune 120x30 pixeli.

```
1 def extract_eyes(cv2_image):
2     """Returns a list of images that contain the eyes extracted from the
3     original image.
4
5     First result is the left eye, second result is the right eye."""
6     global _face_detector, _face_predictor
7     if _detectors_are_initialised() == False:
8         _initialize_detectors()
9
10    gray_image = Utils.convert_to_gray_image(cv2_image)
11    rects = _face_detector(gray_image, 0)
12    if len(rects) > 0:
13        shape = _face_predictor(gray_image, rects[0])
14        shape = face_utils.shape_to_np(shape)
15
16        eyes = []
17        for eye in ["left_eye", "right_eye"]:
18            # get the points for the contour
19            (eye_start, eye_end) = face_utils.FACIAL_LANDMARKS_IDXS[eye]
20            contour = shape[eye_start:eye_end]
21            # get the upper left point, lower right point for this eye
22            start = [min(contour, key=lambda x: x[0])[0],
23                      min(contour, key=lambda x: x[1])[1]]
24            end = [max(contour, key=lambda x: x[0])[0],
25                      max(contour, key=lambda x: x[1])[1]]
26            # extract the current eye
27            eyes.append(cv2_image[start[1]:end[1], start[0]:end[0]])
28
29    return eyes
30
31    return None
```

Listing 3.3: Extragerea ochilor dintr-o imagine

Pentru prezicerea zonei în care se uită utilizatorul ne interesează mai mult contrastul dintre pupilă și iris. Pentru aceasta, am aplicat un prag binar (*binary threshold*) imaginilor ochilor (care au fost convertite în prealabil în imagini gri) pentru a scoate în evidență poziția pupilei, relativ la întregul ochi.

Un *binary threshold* simplu constă în a alege o constantă  $C \in [0, 255]$  și, pentru o imagine gri  $I$ , creăm o nouă imagine  $\hat{I}$  în care înlocuim valorile fiecărui pixel  $p \in I$  astfel:

$$\hat{p} = \begin{cases} 0, & \text{dacă } p < C \\ 1, & \text{altfel} \end{cases} \quad (3.1)$$

Din păcate, această metodă nu a dat cele mai bune rezultate. În funcție de claritatea imaginilor și de modul în care se reflectă lumina în ochi, imaginile alb-negru rezultate nu erau foarte consistente din punct de vedere al informației pe care o ofereau.

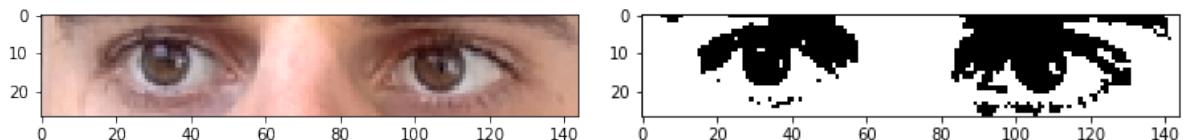


Figura 3.3: Aplicarea unui *binary threshold* simplu cu  $C = 127$

O soluție mai bună ar fi ca pentru fiecare pixel să ținem cont și de valorile pixelilor vecini deoarece ar fi o soluție mai robustă pentru imaginile care conțin mai multe variații de lumină. O astfel de metodă este *Gaussian Adaptive Thresholding* care, conform OpenCV, 2020, calculează o sumă gaussiană ponderată a pixelilor vecini, din care apoi se scade valoarea  $C$ . Metoda este net superioară celei simple și are rezultate mult mai bune.

Cu ajutorul bibliotecii OpenCV am aplicat mai întâi un filtru denumit *Median Filter* cu scopul de a netezi imaginea (*smoothing*). Apoi am aplicat un prag de tipul *Gaussian Adaptive Thresholding*.

```

1 def get_binary_thresholded_image(cv2_image):
2     img = convert_to_gray_image(cv2_image)
3     img = cv2.medianBlur(img, 5)
4     img = cv2.adaptiveThreshold(
5         img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
6     return img

```

Listing 3.4: Aplicarea unui *Gaussian Adaptive Thresholding*

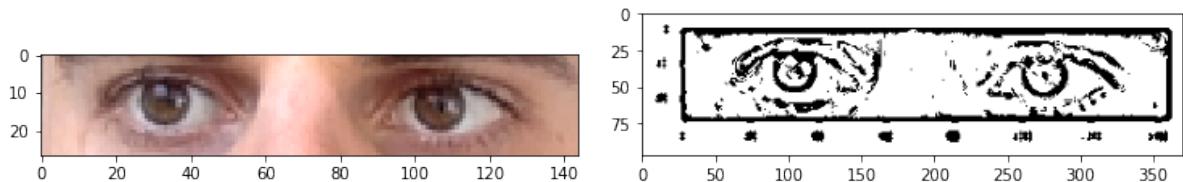


Figura 3.4: Thresholding Adaptiv Gaussian. Conturul ochilor este mai bine definit

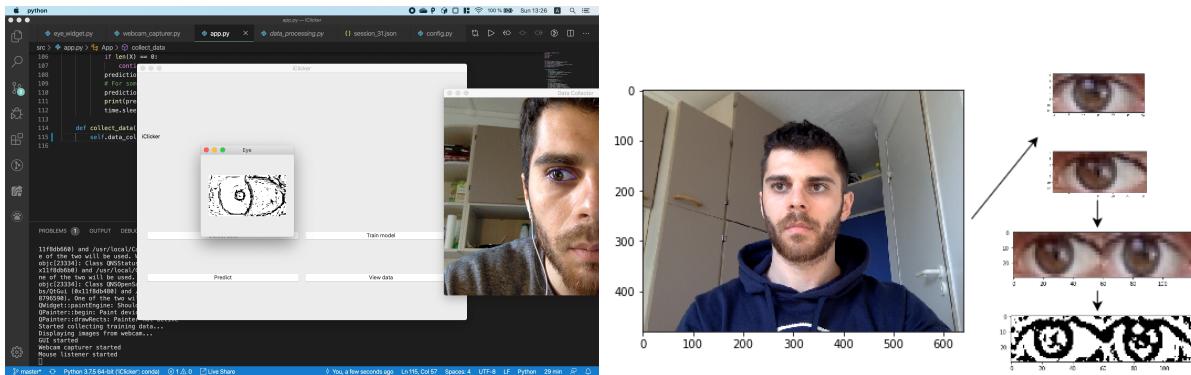


Figura 3.5: Obținerea de informații din imaginile ochilor

### 3.2.3 Folosirea întregii fețe

Următoarea idee a fost să folosesc față completă a utilizatorului, apoi să o transmit unei rețele neuronale convoluționale. Procesul de extragere a feței arată astfel:



Figura 3.6: Extragerea feței

Imaginea finală este în forma unui pătrat, conținând față extrasă din imagine, convertită apoi în gri. Este, de asemenea, normalizată, pentru a lua valori între 0 și 1.

Pentru a extrage față am folosit biblioteca dlib. Aceasta pune la dispoziție un detector facial care identifică un *bounding box* în care se regăsește față unei persoane.

```

1 def extract_face(cv2_image):
2     """Returns the face part extracted from the image"""
3     global _face_detector
4     if _detectors_are_initialised() == False:
5         _initialize_detectors()
6
7     gray_image = Utils.convert_to_gray_image(cv2_image)
8     rects = _face_detector(gray_image, 0)
9     if len(rects) > 0:
10         # only for the first face found
11         (x, y, w, h) = face_utils.rect_to_bb(rects[0])
12         return cv2_image[y:y+h, x:x+w]
13     return None

```

Listing 3.5: Extragerea feței dintr-o imagine

### 3.2.4 “Bandă oculară”

Următoarea opțiune și cea care a rămas integrată în soluția finală a fost să folosesc ambii ochi, fără modificări adiționale. Procesul de extragere a ochilor arată astfel:

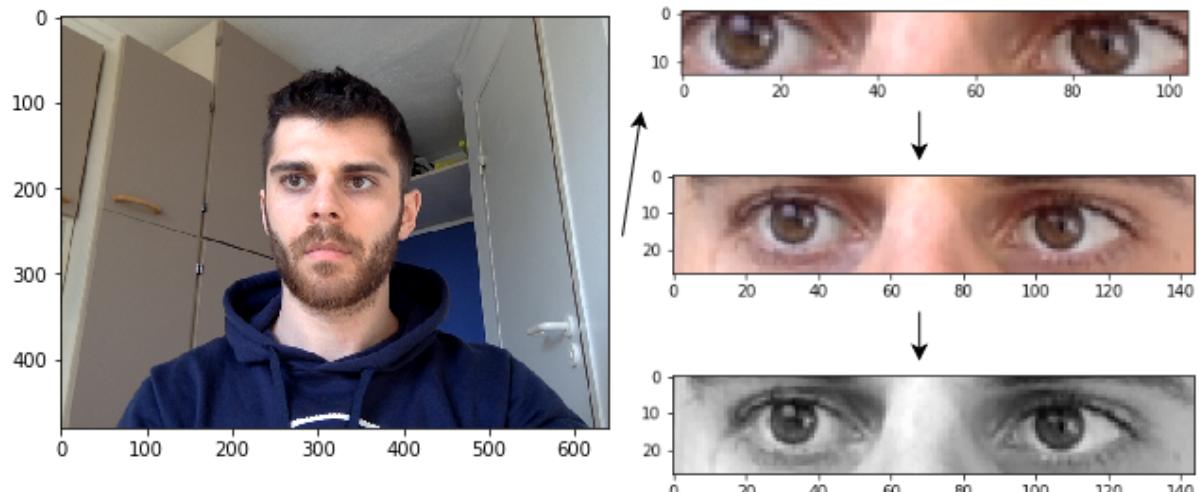


Figura 3.7: Extragerea “bandei oculare”

După ce am detectat față, am extras ambii ochi, am mărit zona dreptunghiulară care încadrează ochii, pentru a avea mai multă informație, apoi am convertit imaginea în gri și am normalizat valorile pixelilor. Zona care încadrează ochii a fost extinsă cu 20% pe orizontală și cu 60% pe verticală.

```

1 def extract_eye_strip(cv2_image):
2     """Returns a horizontal image containing the two eyes extracted from
3     the image"""
4
5     global _face_detector, _face_predictor
6
7     if _detectors_are_initialised() == False:
8         _initialize_detectors()
9
10
11    gray_image = Utils.convert_to_gray_image(cv2_image)
12    rects = _face_detector(gray_image, 0)
13
14    if len(rects) > 0:
15
16        # only for the first face found
17        shape = _face_predictor(gray_image, rects[0])
18        shape = face_utils.shape_to_np(shape)
19
20        (left_eye_start,
21         left_eye_end) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
22
23        (right_eye_start,
24         right_eye_end) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
25
26        # get the contour
27
28        start, end = min(left_eye_start, right_eye_start), max(
29            left_eye_end, right_eye_end)
30
31        strip = shape[start:end]
32
33        # get the upper left point, lower right point
34
35        start = [min(strip, key=lambda x: x[0])[0],
36                 min(strip, key=lambda x: x[1])[1]]
37
38        end = [max(strip, key=lambda x: x[0])[0],
39                 max(strip, key=lambda x: x[1])[1]]
40
41        # go a little outside the bounding box, to capture more details
42
43        distance = (end[0] - start[0], end[1] - start[1])
44
45        # 20 percent more details on the X axis, 60% more details on the Y
46        # axis
47
48        percents = [20, 60]
49
50        for i in range(0, 2):
51
52            start[i] -= int(percents[i]/100 * distance[i])
53            end[i] += int(percents[i]/100 * distance[i])
54
55        return cv2_image[start[1]:end[1], start[0]:end[0]]
56
57    return None

```

Listing 3.6: Extragerea “bandei oculare” în Python 3.7

# Capitolul 4

## Antrenament

**OBSERVAȚIE** Valorile de pe axa verticală a graficelor prezentate în acest capitol au fost logaritmice. Acest lucru a ajutat la o vizualizare mai bună a graficelor în care se analizează evoluția antrenării unui model.

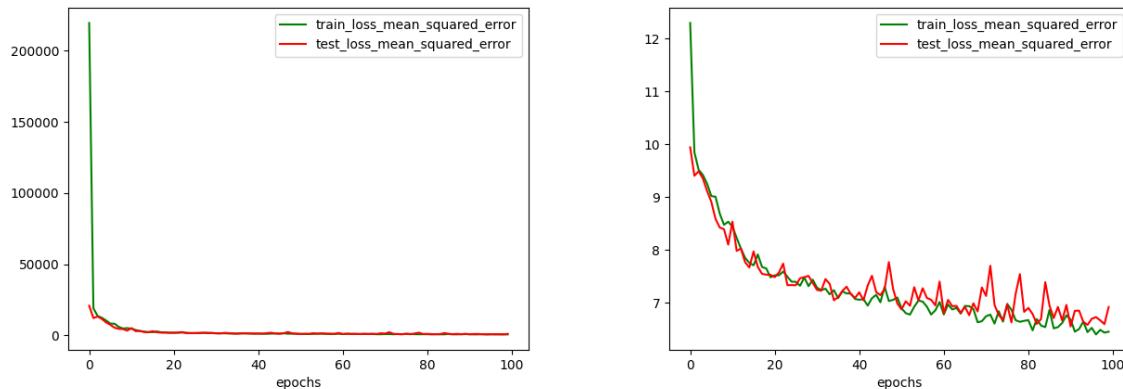


Figura 4.1: Graficul aceluiași model, înainte și după logaritmarea valorilor de pe axa verticală

Modelele au fost antrenate folosind *framework-ul* Keras. Dacă valoarea unui parametru nu este menționată, atunci valoarea respectivă este cea de prestabilită de Keras (versiunea 2.2.4).

### 4.1 Rețele MLP

Primele experimente pe care le-am efectuat au fost efectuate folosind imaginile din care am extras ochii, pe care apoi i-am convertit în imagini alb-negru (folosind un *adaptive gaussian threshold*) normalize, aşa cum a fost menționat în prima parte

a secțiunii de procesare de date 3.2.2. De asemenea, am etichetat fiecare imagine cu numărul celulei în care se uita utilizatorul, conform dimensiunii grilei folosite 1.1. Scopul a fost de a prezice, pentru imagini noi, numărul acelei celule. Iată arhitectură pe care am folosit-o:

```

1 model = Sequential([
2     Dense(100, input_shape=(n,), kernel_initializer='glorot_uniform'),
3     Dropout(0.5),
4     ReLU(),
5     Dense(128, kernel_initializer='glorot_uniform'),
6     Dropout(0.5),
7     ReLU(),
8     Dense(64, kernel_initializer='glorot_uniform'),
9     # Dropout(0.5),
10    ReLU(),
11    Dense(Config.grid_size * Config.grid_size, activation='softmax')
12 ])

```

Listing 4.1: Arhitectura MLP

Fiind o problemă de clasificare, rețeaua trebuie să furnizeze pentru fiecare zonă a ecranului  $z_i$  o probabilitate  $p_i$  însemnând probabilitatea ca în acea imagine utilizatorul să privească zona  $z$ . Evident,  $\sum(p_i) = 1$ . Pentru acest lucru am folosit ca funcție de activare pe ultimul strat funcția softmax, definită prin

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

unde  $x$  este vectorul care coincide valorilor de pe ultimul strat (9 valori pentru o grilă 3x3) înainte de a trece prin funcția de activare softmax.

Pentru restul straturilor am folosit funcția de activare *ReLU* (*Rectified Linear Unit*), aceasta fiind preferată în *deep learning* datorită performanțelor ei. Pentru a evalua performanța modelelor am definit eroarea ca fiind *Categorical Crossentropy Loss*, definită prin:

$$E = - \sum_i \sum_c t_{ic} * \log(y_{ic})$$

unde  $t_{ic}$  este 1 doar dacă instanța  $i$  aparține clasei  $c$  și  $y_{ic}$  este probabilitatea ca instanța  $i$  să aparțină clasei  $c$ . Aceasta se comportă foarte bine în combinație cu funcția softmax, care ne furnizează exact probabilitățile  $y_{ic}$  de care avem nevoie în formula de mai sus.

**OBSERVATIE** Pentru primele experimente din această secțiune am folosit doar imaginile în care utilizatorul se uita în colțurile ecranului. De exemplu, pentru o grilă de dimensiune 3x3, am ales doar imaginile în care eticheta corespunzătoare a imaginii este 0, 2, 6 sau 8. Ultima mențiune este că datele de antrenament reprezintă 80% din multimea totală de date, restul fiind date de testare.

Experiment	Dimensiune grilă	Număr imagini	Epoci	Optimizator	Rată învățare	Batch size
1	2x2	2184	300	Adam	0.001	32
2	3x3	1247	300	Adam	0.001	32
3	4x4	711	300	Adam	0.001	32

Experimentele 2 și 3 au beneficiat de mai puține imagini de antrenament deoarece, dimensiunea grilei fiind mai mare, imaginile sunt mai dispersate și fiecare celulă are mai puține imagini corespunzătoare ei. Evident, pentru o grilă 2x2, fiecare imagine se află într-o celulă din colț (fiind doar 4), deci au fost folosite toate imaginile.

Primul rezultat pe care l-am constatat a fost că doar primul model putea prezice bine zona în care mă uitam. Rețeaua MLP se descurca cu atât mai bine cu cât deschideam ochii mai mult, întrucât putea realiza mai bine conturul ochiului și să delimitizeze pupila.

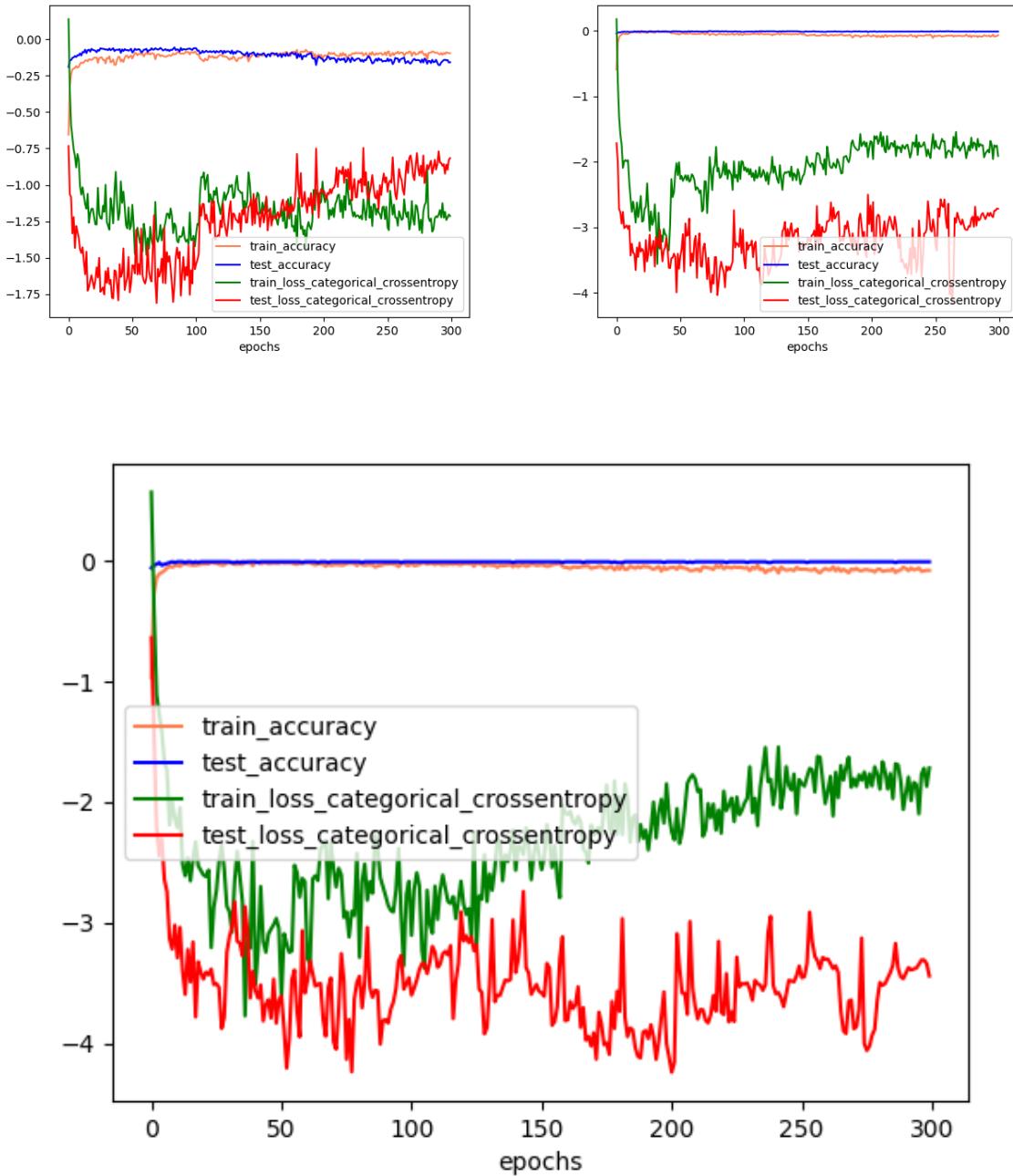


Figura 4.2: Modelele 1, 2 și 3

Acuratețea este foarte bună în cele 3 cazuri, însă nu se comportă atât de bine pe imagini noi, într-un mediu diferit. De asemenea, am constatat ca luminozitatea și modul în care utilizatorul este plasat față de webcam sunt foarte importante și influențează performanța semnificativ.

Un lucru important de observat, care se va regăsi în majoritatea graficelor, sunt acele oscilații, alternanțe ale acurateței, care urcă și coboară. Stratul *Dropout* este cel care cauzează acest comportament, prin modul în care funcționează: "by randomly dropping

units during training to prevent their co-adaptation”, precum este menționat de Pierre Baldi, 2020. Acesta elimină neuroni ai unui strat, în mod aleatoriu, pentru a permite tuturor neuronilor să participe la învățare și să sporească performanța rețelei. Astfel, atunci când sunt eliminați neuroni “buni”, adică aceia care conțin multă informație utilă pentru rezultatul final, performanța (acuratețea, eroarea) pot fi afectate negativ. Pe de cealaltă parte, atunci când sunt eliminați neuroni care nu ajută foarte mult pentru predicție, performanța nu este afectată prea mult.

Am continuat prin a antrena aceeași rețea, pe o grilă de dimensiune 3x3, dar de data aceasta folosind toate imaginile disponibile. Totuși, acest lucru nu a ajutat foarte mult și am observat și că după o anumită epocă, rețeaua nu se mai comportă bine pe datele de test și apărea fenomenul de *overfit*. Acest lucru se poate observa pe graficul modelului 4, unde acuratețea pentru datele de antrenament și cea pentru datele de testare începe să diveargă.

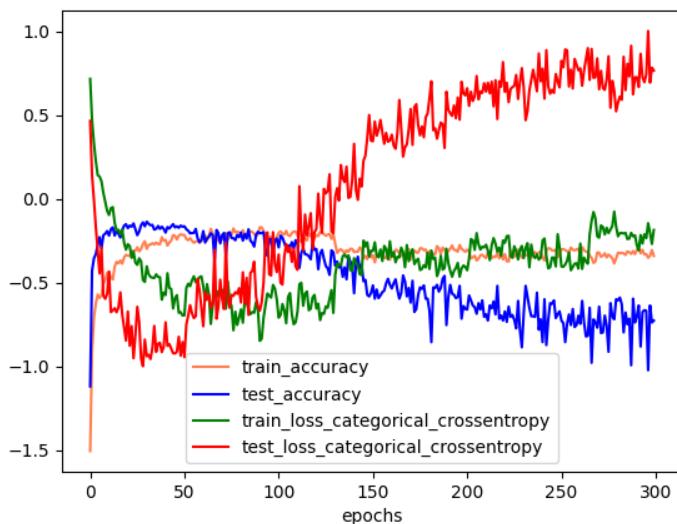


Figura 4.3: Modelul 4. Se observă fenomenul de *overfitting*

Pentru a rezolva problema overfit-ului, am introdus o *regularizare L2* care, conform Karim, 2020, ar trebui să ajute în acest caz. Într-adevăr, situația a fost ameliorată, însă nu a ajutat modelul să prezică mai bine zona în care mă uitam.

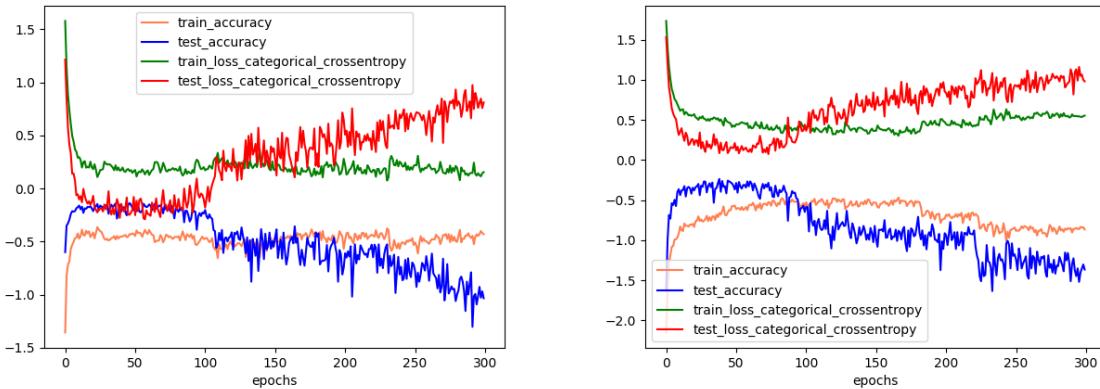


Figura 4.4: Modelele 5 și 6. *Overfit-ul* este mai puțin pronunțat

## 4.2 Rețele neuronale de conoluție

### 4.2.1 Utilizarea întregii fețe

Mai departe am vrut să experimentez cu rețelele convolutionale și am început prin a utiliza fețele extrase din imagini 3.6, pe o grilă de dimensiune  $2 \times 2$  1.1. Din păcate, nu am avut rezultate deloc bune în primă fază. După o analiză a codului și a datelor, am descoperit că aveam o problemă în cod și că datele nu erau normalize, fapt care împiedica rețeaua din a învăța. Am rezolvat această problemă și am continuat experimentarea.

Iată arhitectura cu care am obținut rezultatele ce urmează a fi prezentate:

```

1 model = Sequential()
2 model.add(Conv2D(16, kernel_size=(3, 3),
3                 input_shape=input_shape))
4 model.add(MaxPooling2D(pool_size=(2, 2)))
5 model.add(ReLU())
6 model.add(Conv2D(32, kernel_size=(3, 3)))
7 model.add(MaxPooling2D(pool_size=(2, 2)))
8 model.add(ReLU())
9 model.add(Flatten())
10 model.add(Dense(128, activation='relu'))
11 model.add(Dense(4, activation='softmax'))
12 # optimizer used
13 opt = Adam()
```

Listing 4.2: Prima arhitectură CNN

**OBSERVATIE** Pentru primele experimente din această secțiune am folosit doar imaginile în care utilizatorul se uita în colțurile ecranului. De exemplu, pentru o grilă de dimensiune 3x3, am ales doar imaginile în care eticheta corespunzătoare a imaginii este 0, 2, 6 sau 8. Ultima mențiune este că datele de antrenament reprezintă 80% din multimea totală de date, restul fiind date de testare.

Mai jos sunt parametrii de antrenare și rezultatele antrenării:

Experiment	Dimensiune grilă	Număr imagini	Epoci	Optimizator	Rată învățare	Batch size
7	2x2	2184	50	Adam	0.001	32
8	3x3	1247	50	Adam	0.001	32
9	4x4	711	50	Adam	0.001	32

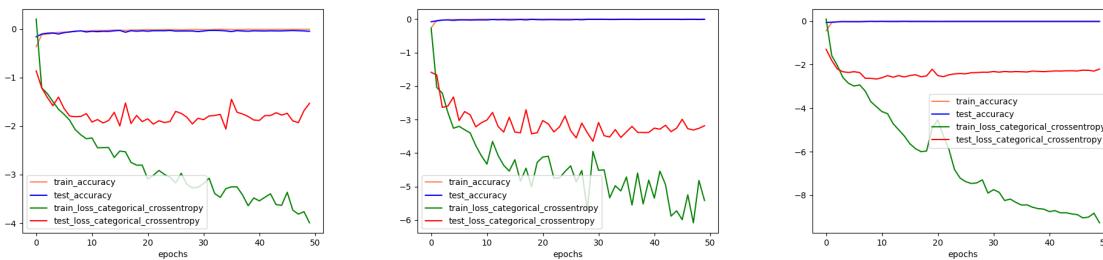


Figura 4.5: Modelele 7, 8 și 9

Modelele 8 și 9 au performanțe bune, dar sunt instabile când le folosesc în condiții reale. Modelul 7 s-a descurcat mai bine. Dacă mă poziționez la un unghi prielnic față de cameră, acesta poate prezice destul de bine unde privesc. Acest lucru este probabil datorat faptului că a fost antrenat folosind mai multe date.

Pentru experimentele 10 și 11, am adunat mai multe imagini și am antrenat aceeași arhitectură CNN pe toate datele (nu doar cele corespunzătoare colțurilor ecranului), ceea ce a însemnat 2736 de imagini, pentru grilele de dimensiune 3x3 și 4x4. Rezultatele au fost mai bune în condiții reale de testare, ceea ce m-a făcut să constat că performanța unui model crește direct proporțional cu dimensiunea multumii de date pe care o avem la dispoziție. Putem vedea de asemenea semne de *overfit* în aceste modele.

#### 4.2.2 Utilizarea “benzilor oculare” ca date de antrenament

Am folosit aceeași arhitectură de rețea convolutională ca mai sus, dar de această dată folosind doar porțiunile dreptunghiulare care încadrează ambii ochi 3.7. Primele

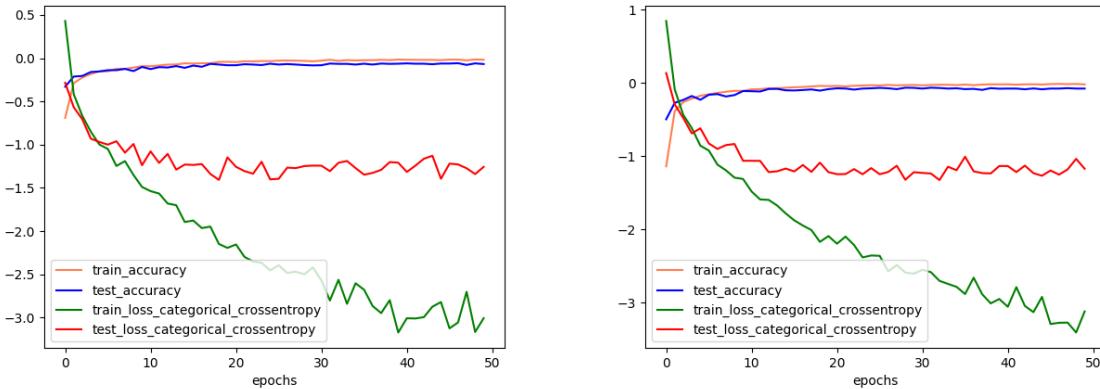


Figura 4.6: Modelele 10 și 11. Valoarea erorii este mai mare decât modelele 8 și 9 deoarece am folosit mai multe date, fiecare exemplu participând cu o valoare pozitivă la valoarea finală a erorii

modele au fost antrenate doar cu acele imagini în care utilizatorul se uita în colțurile ecranului.

Experiment	Dimensiune grilă	Număr imagini	Epoci	Optimizator	Rată învățare	Batch size
12	2x2	2184	50	Adam	0.001	32
13	3x3	1247	50	Adam	0.001	32
14	4x4	711	50	Adam	0.001	32

Experimentul 12 a fost unul reușit. În cadrul acestuia am putut prezice relativ bine unde mă uitam în condiții reale, chiar dacă mă indepărțam sau apropiam de ecran.

Am observat, de asemenea, aceeași tendință de scădere a performanței atunci când modelele sunt antrenate pe mai o mulțime de date mai mică, motiv pentru care experimentele 13 și 14 nu au fost atât de reușite în condiții reale. Faptul acesta a fost confirmat, întrucât prezicerea a fost mai stabilă, dar tot nu era utilizabilă pentru un produs finit.

**Comentariu intermediar** Din experimentele prezentate până acum am concluzionat că cea mai bună variantă este folosirea rețelelor convolutionale împreună cu "benzile oculare". Mai mult, atunci când am folosit același algoritm pe mai multe date, modelul realizat a fost mai stabil și a avut performanțe mai bune. Acest fapt subliniază, din nou, importanța datelor într-un algoritm de învățare automată.

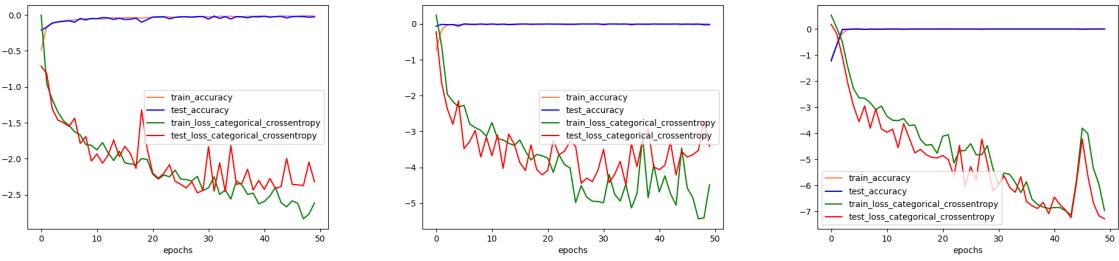


Figura 4.7: Modelele 12, 13 și 14

### 4.2.3 Îmbunătățirea arhitecturii CNN

Experimentele 17–22 au fost rezultatul încercării de a îmbunătăți arhitectura inițială a rețelei de convoluție pe care am folosit-o 4.2.1. Am incercat fie să elimin straturi de convoluție, fie să schimb numărul filtrelor pentru a observa care este comportamentul acesteia.

Am remarcat că atunci când suprimez al doilea strat de convoluție, rețeaua nu mai poate spune atunci când mă uit la anumite celule, deși înainte nu avea probleme în acest sens (spre exemplu celula numărul 2, pe o grilă de dimensiune 3x3). De asemenea, prin creșterea numărului de filtre de convoluție, performanța a crescut, așadar am decis să continui cu această arhitectură:

```

1 model = Sequential()
2 model.add(Conv2D(64, kernel_size=(3, 3),
3                  input_shape=input_shape))
4 model.add(MaxPooling2D(pool_size=(2, 2)))
5 model.add(ReLU())
6 model.add(Conv2D(128, kernel_size=(3, 3)))
7 model.add(MaxPooling2D(pool_size=(2, 2)))
8 model.add(ReLU())
9 model.add(Flatten())
10 model.add(Dense(128, activation='relu'))
11 model.add(Dense(Config.grid_size * Config.grid_size, activation='softmax'))
```

Listing 4.3: Arhitectura CNN îmbunătățită

## 4.3 Regresie folosind CNN

O altă idee pe care am avut-o a fost să transform problema inițială, din clasificare în regresie. Am încercat să prezic exact poziția (coordonatele) la care utilizatorul se uită,

iar apoi să o traduc în numărul celulei corespunzătoare.

Pentru acest lucru, am adaptat arhitectura precedentă, astfel încât ultimul strat să fie format din doar 2 neuroni corespunzători coordonatelor  $(x, y)$  ale cursorului de pe ecran. Am schimbat și funcția de activare de pe acest ultim strat într-o funcție liniară  $f(x) = x$ .

```
1 model.add(Dense(2, activation='linear'))
```

Listing 4.4: Schimbarea ultimului strat pentru problema de regresie

De precizat este faptul că, în acest caz, predicțiile rezultate dintr-o astfel de rețea pot depăși limitele ecranului. Spre exemplu rețeaua poate prezice că utilizatorul se uită la coordonatele  $(-5, 100)$  sau  $(2000, 800)$  pentru o rezoluție a ecranului *Full HD*. În acest caz, am “reglat” valorile astfel încât să nu iasă din ecran (de exemplu  $(-5, 100)$  devine  $(0, 100)$ ).

O altă schimbare necesară constă în definirea erorii. Pentru această regresie am folosit *eroarea medie pătratică* (*Mean Squared Error*, abreviat MSE). Astfel, formula de calcul a erorii devine:

$$E = \frac{1}{n} * \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

unde  $y$  este vectorul de coordonate reale (cea ce trebuie să prezicem), iar  $\hat{y}$  este vectorul rezultat, predicția modelului nostru.

Am introdus și o scădere treptată a ratei de învățare (learning rate decay), menită să ajute rețeaua să învețe mai mult și să conveargă mai precis spre parametrii optimi. În cazul tehnologiei Keras pe care am folosit-o, rata de învățare se adaptează astfel:

```
1 lr = initial_lr * (1 / (1 + decay * iteration))
```

Listing 4.5: Modul în care Keras actualizează rata de învățare

Avantajul unei astfel de tehnici este că putem seta rata de învățare mai mare la început, pentru a invăța mai repede, apoi să o micșorăm pentru a face pași mai mici, dar mai precisi. Metoda aceasta a dat rezultate pentru un număr mai mare de epoci și a rezultat într-un model care se descurca bine în a atinge obiectivul principal al acestei lucrări.

Experiment	Dimensiune grilă	Număr imagini	Epoci	Optimizer	Rată învățare	Batch size
22	3x3	2736	50	Adam	0.001	32
23	3x3	2736	100	Adam decay= $10^{-4}$	0.001	32
24	3x3	2736	100	Adam decay= $10^{-4}$	0.01	32

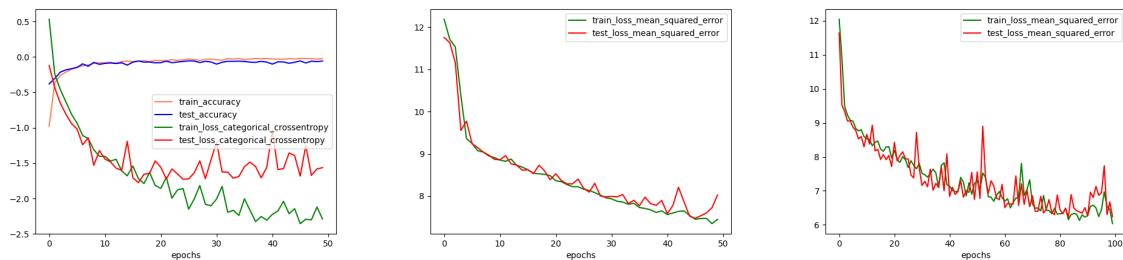


Figura 4.8: Modelele 22, 23 și 24

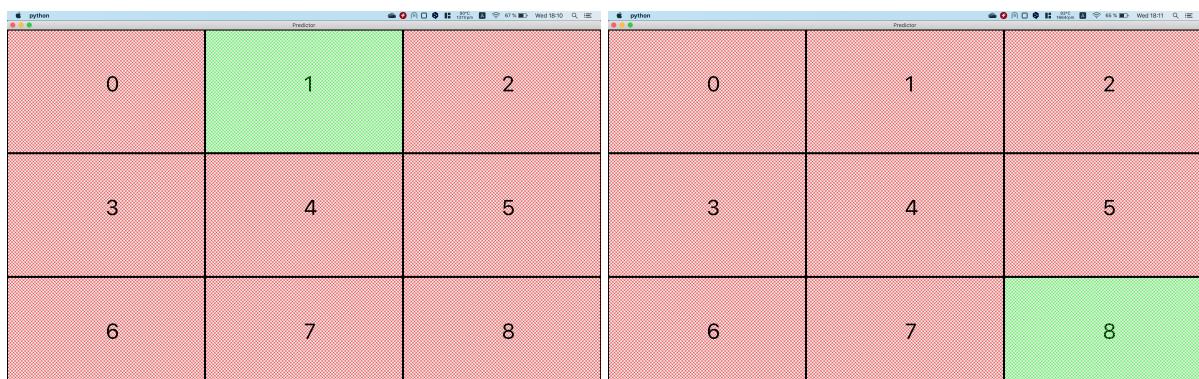


Figura 4.9: Utilizarea rețelei CNN pentru urmărirea ochilor

# Capitolul 5

## Folosirea modelelor antrenate

### 5.1 Deplasarea cursorului

Funcționalitatea de bază a mouse-ului constă în deplasarea cursorului pe ecran. Am implementat, în primul rând, deplasarea acestuia pe 8 direcții: N, NE, E, SE, S, SV, V, NV. Dacă utilizatorul vrea să mute cursorul în direcția stânga-sus, atunci acesta va trebui să se uite în acea parte a ecranului (corespunzătoare celulei cu numărul 0 pe o grilă de dimensiune 3x3). Analog, pentru a-l deplasa în sus, va trebui să se uite pe centrul ecranului în partea de sus (celula cu numărul 1) și.a.m.d.

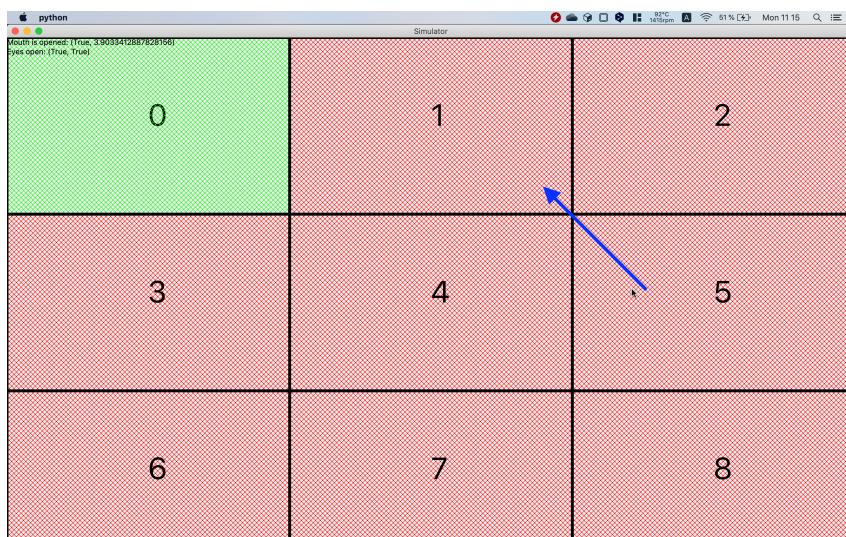


Figura 5.1: Deplasarea cursorului în direcția NV

Întrebarea care reieșe din paragraful de mai sus este: cum poate solicita utilizatorul deplasarea cursorului? Fiind o aplicație care se bazează doar pe gesturi ale feței, am decis ca deplasarea cursorului să se realizeze în momentul deschiderii gurii.

Un articol interesant (Ashlesha Singh, 2018) prezintă metode de identificare a oboselii unui șofer. Un factor cheie este detectarea căscatului, care se rezumă la același gest pe care a trebuit să îl identific și eu. Articolul propune folosirea reperelor faciale situate pe buze pentru a detecta un raportul de aspect al gurii (*MAR, Mouth Aspect Ratio*). Formula propusă este:

$$MAR = \frac{|CD| + |EF| + |GH|}{3|AB|}$$

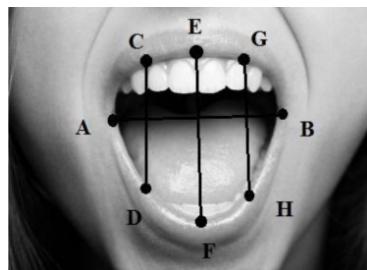


Figura 5.2: Repere faciale folosite pentru a calcula valoarea MAR. Imagine preluată din Ashlesha Singh, 2018

Astfel, dacă valoarea *MAR* depășește un anumit prag (*MAR threshold*) care este configurabil, atunci aplicația consideră că acesta solicită deplasarea cursorului. Avantajul configurării acestui prag este că poate fi setat astfel încât utilizatorul să poată și conversa în timp ce aplicația este deschisă, fără a mișca accidental mouse-ul.

## 5.2 Apăsarea butoanelor mouse-ului

Un gest util ce poate fi folosit în același fel ca mai sus este închiderea ochiului. Am decis să folosesc închiderea ochiului pentru a simula funcțiile de “click stânga” și “click dreapta”. Tereza Soukupová, 2016 propune un calcul al raportului de aspect al ochiului (*EAR, Eye Aspect Ratio*) pentru a detecta când o persoană are ochiul închis sau nu. Formula propusă este:

$$EAR = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2||p_1 - p_4||}$$

Când utilizatorul menține ochiul stâng închis pentru un anumit interval de timp  $\delta_t$ , aplicația va efectua simularea apăsării butonului stâng. Analog, închiderea ochiului drept va determina simularea apăsării butonului drept. Intervalul de timp  $\delta_t$  a fost

folosit pentru a evita apăsarea butoanelor într-un mod inutil atunci când utilizatorul doar clipește.



Figura 5.3: Simularea apăsării butonului stâng de pe mouse folosind valoarea EAR. Prima imagine ilustrează detectarea clipirii și este preluată din Tereza Soukupová, 2016

# Capitolul 6

## Analiză asupra identificării reperelor faciale

Utilizând cu succes reperele faciale pentru a localiza și a extrage ochii, ca apoi să folosesc aceste date pentru o rețea convoluțională ca să urmăresc ochii utilizatorului, am devenit interesat de cum funcționează biblioteca Python pe care am folosit-o (dlib) pentru a găsi aceste repere faciale. Am decis să studiez puțin această problemă și, conform Jing Yang, 2017, metoda *state of the art*, de ultimă oră, pentru detectarea reperelor faciale se bazează pe arhitectura de tipul *Hourglass* care este, într-o formă simplă a ei, o arhitectură de tipul *Autoencoder* (Encoder-Decoder).

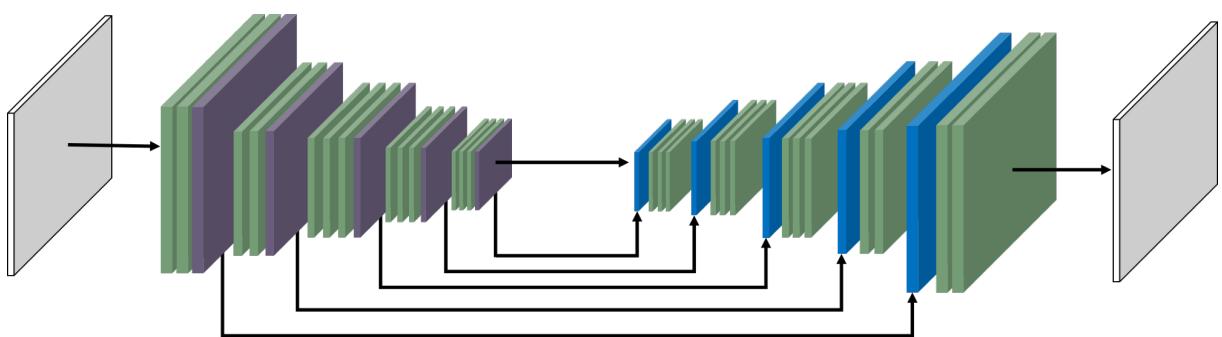


Figura 6.1: Exemplu de arhitectură Hourglass. Imagine preluată de pe Medium

M-am concentrat doar pe prezicerea centrului unui ochi, plecând de la o mulțime de date care conține imagini ale ochiului. Ca date de antrenament am folosit mulțimea de date pusă la dispoziție pusă la dispoziție de *crowdpupil*<sup>1</sup>, care constă în 792 de imagini, fiecare imagine fiind etichetată cu poziția centrului ochiului în acea imagine.

<sup>1</sup><http://cs.uef.fi/pupoint/>

O metodă interesantă pentru identificarea reperelor faciale este prezentată de [ccnn paper](#), care creează o arhitectură compusă din rețele convoluționale puse în cascadă, menite să creeze o *estimare* a acestor puncte și apoi o *rafinare* a lor. Această metodă se numește *coarse-to-fine estimation*, iar partea de rafinare este tradusă într-o problemă de regresie.

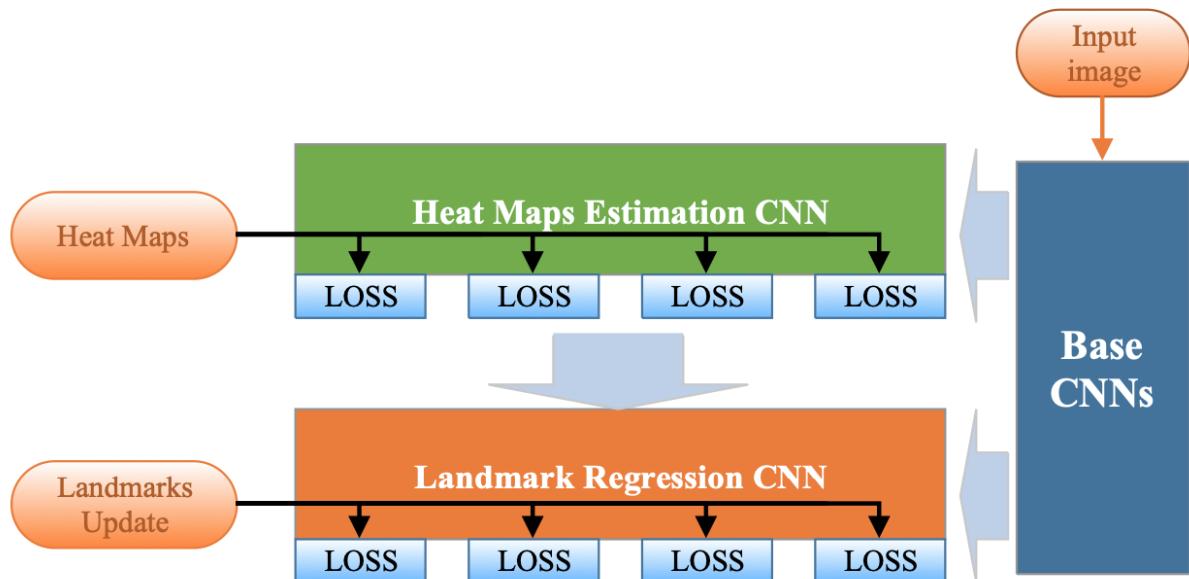


Figura 6.2: Arhitectura CCNN propusă de [ccnn paper](#). Consistă într-o combinație între *heatmap estimation* și *landmark regression*.

Din căutările pe care le-am efectuat, am descoperit că există două metode de a încerca prezicerea reperelor faciale. Prima metoda se ocupă cu găsirea coordonatelor exacte ( $x, y$ ) a fiecărui reper facial prin regresie, iar a doua metodă încearcă construirea unor hărți termografice (*heatmaps*) care codifică acele reperuri faciale.

Am ales cea de-a doua opțiune, astfel că am generat, pentru fiecare imagine a ochiului din mulțimea de date [crowdpupil](#), o hartă termografică care codifică centrul ochiului. Acest lucru l-am făcut prin a construi o *Distribuție Gaussiană* multivariată (2D), definită prin

$$\mathcal{N}(\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp\left(-\frac{1}{2}(X - \mu)^T \Sigma^{-1} (X - \mu)\right)$$

În cazul de față am definit  $\mu$  și  $\Sigma$  astfel:

$$\mu = (x, y), (x, y)$$

$$\Sigma = (f(r), f(r))$$

$\mu$  fiind coordonatele centrului ochiului iar  $r$  raza pupilei.

Am calculat o medie pentru valoarea lui  $r$  uitându-mă la o parte din imagini iar apoi am definit  $f(r) = 1.5 * r$ . Având acestea definite, am putut codifica centrul ochiului fiecărei imagini:

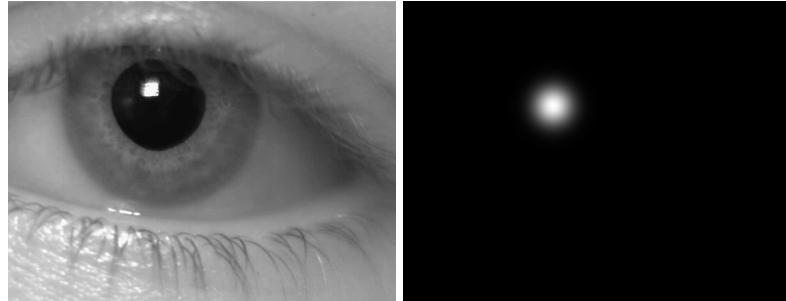


Figura 6.3: Centrul ochiului codificat printr-o hartă termografică

Am folosit o formă simplă a arhitecturii Hourglass pentru a încerca reconstruirea acestei hărți termografice. Arhitectura se bazează pe 2 părți corespunzătoare unei arhitecturi *Encoder–Decoder*. Prima parte produce o *compresie* a informației, iar a doua parte o *decompresie* pentru a încerca reconstruirea imaginii inițiale. În acest fel, rețeaua învăță să folosească doar acele informații care sunt de folos din imaginea originală și care aduc un plus de informație pentru rezultatul final. Iată arhitectura pe care am folosit-o:

```

1 class MyCNN(nn.Module):
2     def __init__(self, input_size):
3         super(MyCNN, self).__init__()
4         ## eye image -> encoder -> decoder -> heatmap
5         filters = [16, 32, 64, 128]
6         # starting encoding
7         self.layer1 = nn.Sequential(
8             nn.Conv2d(1, filters[0], kernel_size=3, padding=2),
9             nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
10            nn.ReLU(inplace=True),
11            nn.BatchNorm2d(filters[0]),
12
13            nn.Conv2d(filters[0], filters[1], kernel_size=2, padding=2),
14            nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
15            nn.ReLU(inplace=True),
16            nn.BatchNorm2d(filters[1]),
17
18            nn.Conv2d(filters[1], filters[2], kernel_size=3, padding=2),

```

```

19         nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
20         nn.ReLU(inplace=True),
21         nn.BatchNorm2d(filters[2]),
22
23         nn.Conv2d(filters[2], filters[3], kernel_size=2, padding=1),
24         nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
25         nn.ReLU(inplace=True),
26         nn.BatchNorm2d(filters[3]),
27     )
28
29     # encoding done, starting decoding
30     self.layer2 = nn.Sequential(
31         nn.Upsample(size=(28, 35), mode='bilinear'),
32         nn.ConvTranspose2d(filters[3], filters[2], kernel_size=2,
33                           stride=1, padding=1),
34         nn.ReLU(inplace=True),
35         nn.BatchNorm2d(filters[2]),
36
37         nn.Upsample(size=(54, 68), mode='bilinear'),
38         nn.ConvTranspose2d(filters[2], filters[1], kernel_size=3,
39                           stride=1, padding=2),
40         nn.ReLU(inplace=True),
41         nn.BatchNorm2d(filters[1]),
42
43         nn.Upsample(size=(104, 132), mode='bilinear'),
44         nn.ConvTranspose2d(filters[1], filters[0], kernel_size=2,
45                           stride=1, padding=2),
46         nn.ReLU(inplace=True),
47         nn.BatchNorm2d(filters[0]),
48
49         nn.Upsample(size=(202, 258), mode='bilinear'),
50         nn.ConvTranspose2d(filters[0], 1, kernel_size=3, stride=1,
51                           padding=2),
52         nn.ReLU(inplace=True),
53         nn.BatchNorm2d(1),
54     )

```

Partea de compresie corespunde operației de convoluție care reduce dimensiunea imaginii de la un strat la altul în funcție de parametrii convoluției – spre exemplu pasul (saltul) pe care îl realizează filtrul de convoluție. În partea de decompresie am folosit operația de *upsampling* care merge înapoi pe pașii convoluției și mărește

dimensiunea imaginii, aducându-o la dimensiunea originală (în acest caz imaginile au fost redimensionate la 200x256 pixeli). Stratul *Upsample* folosește metoda *bilinear* pentru a multiplica pixelii imaginii luând în considerare vecinii acelor pixeli.

Calcularea dimensiunii imaginii rezultată ( $W_o$  și  $H_o$ ) după o operație de convoluție se face astfel:

$$W_o = \frac{W - F_w + 2P}{S_w} + 1$$

$$H_o = \frac{H - F_h + 2P}{S_h} + 1$$

unde  $(W, H)$  este dimensiunea înainte de operația de convoluție,  $(F_w, F_h)$  este dimensiunea filtrului de convoluție,  $P$  este *padding-ul* adăugat imaginii iar  $(S_w, S_h)$  dimensiunea pasului pe care îl realizează filtrul de convoluție.

Am antrenat rețeaua pentru 12 epoci, folosind optimizatorul *Adam*, iar eroarea a fost calculată folosind *MSE*. Pentru ultima epocă, media scorului *MSE* pentru toate datele de test (în jur de 20% din datele totale) a fost de 8,425983.

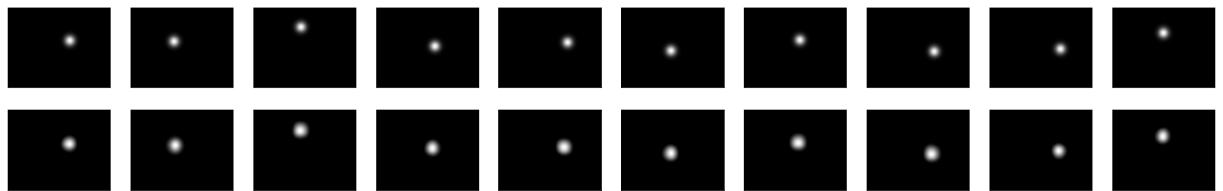


Figura 6.4: Reconstruirea hărților termografice. Prima linie reprezintă adevărul de bază, iar a doua linie rezultatele modelului antrenat

Desi rețeaua a reușit să reconstruiască bine hărțile termografice de mai sus, folosind imagini cu mine în condiții reale nu a dat rezultate bune. Unul dintre motive ar fi rezoluția webcam-ului care nu este foarte bună și imaginile ochilor extrași sunt de o calitate inferioară. Este interesant de observat în figura de mai jos că rețeaua a putut identifica aproximativ centrul ochiului pe o imagine care conținea fața mea întreagă, deși nu a fost antrenată în acest sens.

Acest experiment a servit pentru înțelegerea funcționalității bibliotecilor precum dlib. Experimentul pe care l-am efectuat a fost unul foarte simplu care poate fi extins la a prezice câte o hartă termografică pentru fiecare reper facial (uzual sunt 68) și la a folosi o arhitectură de tip Hourglass mai avansată, care ar folosi, de exemplu, legături reziduale între straturile de convoluție. Chiar dacă rezultatele finale nu au fost reușite,

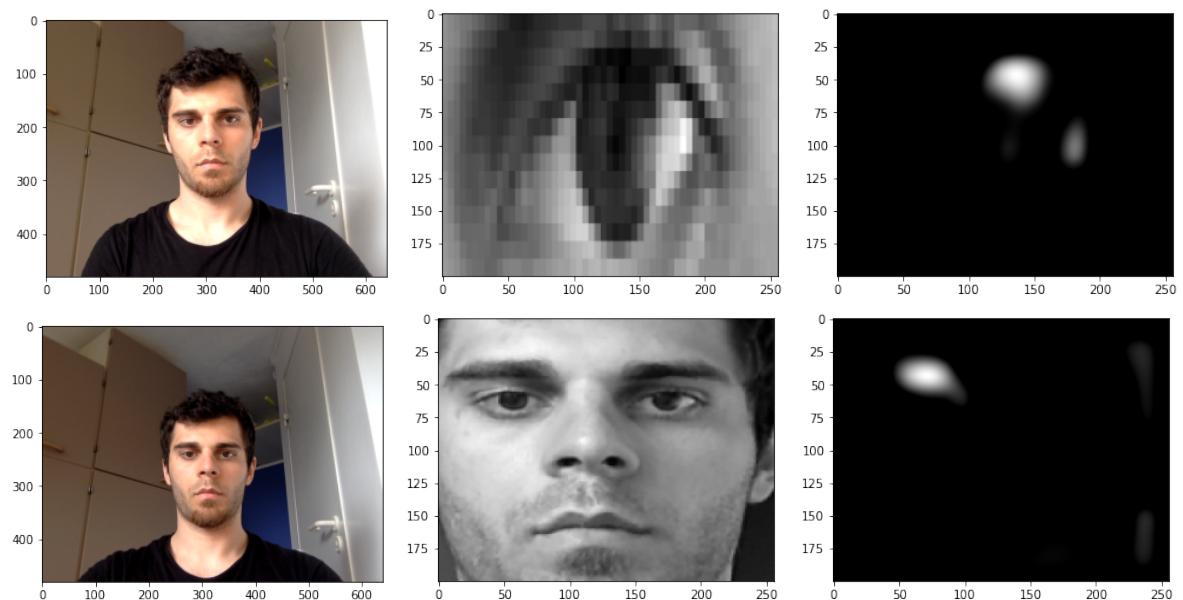


Figura 6.5: Testarea rețelei de tip Hourglass pe imagini cu mine însumi

experimentul m-a ajutat la înțelegerea metodelor *state of the art* pentru identificarea reperelor faciale.

# Concluzii

Am început această lucrare de licență cu un anumit set de obiective în minte. În acest sens am reușit să le transpun în dezvoltarea unei aplicații care poate realiza, în primul rând, sarcina de *gaze tracking*. Am reușit apoi să simulez cu succes deplasarea cursorului mouse-ului și apăsarea butoanelor acestuia, ceea ce a dus la atingerea obiectivelor inițiale.

Pe parcursul lucrării m-am confruntat cu probleme tipice învățării automate, precum *overfitting-ul*, și am învățat importanța datelor și a procesării acestora. Aceste lucruri se extind și la partea de antrenare și la versionarea modelelor antrenate, pentru a avea evidența evoluției performanței. Documentarea acestor modele și a fiecărei schimbare este foarte importantă, lucru pe care inițial nu l-am valorificat suficient de mult.

Am realizat și un experiment în care am încercat să reconstruiesc funcționalitatea unei biblioteci Python pe care m-am bazat, care ajută la identificarea reperelor faciale. Deși experimentul nu a avut rezultate promițătoare, am dobândit mai multe cunoștințe despre metodele *state of the art* de a localiza reperele faciale și despre cum arată procedeul de a realiza asta. Acest lucru m-a învățat și necesitatea de “antrenament” în a citi mai multe lucrări științifice.

În concluzie, această lucrare de licență mi-a permis înglobarea aptitudinilor și cunoștințelor dobândite ca student dar, în același timp, a pus în lumină și lucrurile pe care trebuie să le îmbunătățesc. Aplicația poate fi îmbunătățită în continuare prin adresarea lipsurilor, precum funcționalități de apăsare dublă/lungă a butoanelor mouse-ului.

# Bibliografie

- Ashlesha Singh, P. P., Chandrakant Chandewar. (2018). Driver Drowsiness Alert System with Effective Feature Extraction. Retrieved May 15, 2020, from <http://ijrest.net/downloads/volume-5/issue-4/pid-ijrest-54201808.pdf>
- Ciortuz, L. (2020). Curs de Învățare Automată. Retrieved June 11, 2020, from <https://profs.info.uaic.ro/~ciortuz/teaching.html>
- Jing Yang, K. Z., Qingshan Liu. (2017). Stacked Hourglass Network for Robust Facial Landmark Localisation. Retrieved May 20, 2020, from [http://openaccess.thecvf.com/content\\_cvpr\\_2017\\_workshops/w33/papers/Yang\\_Stacked\\_Hourglass\\_Network\\_CVPR\\_2017\\_paper.pdf](http://openaccess.thecvf.com/content_cvpr_2017_workshops/w33/papers/Yang_Stacked_Hourglass_Network_CVPR_2017_paper.pdf)
- Karim, R. (2020). Intuitions on L1 and L2 Regularisation. Retrieved April 17, 2020, from <https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261>
- OpenCV. (2020). Image Thresholding. Retrieved March 7, 2020, from [https://docs.opencv.org/3.4/d7/d4d/tutorial\\_py\\_thresholding.html](https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html)
- Pierre Baldi, P. S. (2020). The dropout learning algorithm. Retrieved April 10, 2020, from <https://www.sciencedirect.com/science/article/pii/S0004370214000216>
- Tereza Soukupová, J. Č. (2016). Real-Time Eye Blink Detection using Facial Landmarks. Retrieved May 15, 2020, from <http://vision.fe.uni-lj.si/cvww2016/proceedings/papers/05.pdf>
- Wikipedia. (2020). Eye Tracking. Retrieved March 15, 2020, from [https://en.wikipedia.org/wiki/Eye\\_tracking](https://en.wikipedia.org/wiki/Eye_tracking)