

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IASI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**Simularea funcționalităților 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

Avizat,

Îndrumător lucrare de licență,

Asist. Dr. Croitoru Eugen.

Data:

Semnătura:

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Iacob Sergiu** domiciliat în România, jud. Iași, comuna Tomești, sat Tomești, strada Văzduh, nr. 73, născut la data de **03 martie 1997**, identificat prin CNP **1970301226742**, absolvent al Facultății de Informatică, **Facultatea de Informatică** specializarea **informatică**, promoția 2020, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Simularea funcționalităților unui mouse folosind repere faciale** elaborată sub îndrumarea domnului **Asist. Dr. Croitoru Eugen**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data:

Semnătura:

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Simularea funcționalităților unui mouse folosind repere faciale**, codul sursă al programelor și celealte conținuturi (grafice, multimedia, date de test, etc.) care însotesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Sergiu Iacob**

Data:

Semnătura:

Cuprins

Introducere	2
Context	2
Idee	3
Motivație	4
Obiective	5
Metodologie	6
TODO TODO TODO	6
Structura lucrării	6
1 Problema	7
1.1 Definirea problemei	7
1.1.1 Neuronul Artificial	8
1.1.2 Rețele neuronale	9
1.2 Strategie	10
1.2.1 Obținerea datelor de antrenament	10
1.2.2 Rețeaua de tip perceptron multistrat	10
1.2.3 Rețele neuronale convolutionale	10
1.2.4 Cercetare și analizare a bibliotecilor folosite	11
2 Informații preliminare	12
2.1 Maniera de lucru	12
2.2 Configurație utilizată	12
2.3 Informații tehnice	12
2.4 Limite și constrângeri	13
2.5 Date necesare	14
2.6 Obținerea datelor	14
2.6.1 Moduri de obținere	14

2.6.2 Salvarea datelor	15
3 Procesarea datelor	17
3.1 Avantajele unei preprocesări	17
3.2 Folosirea exclusivă a ochilor	17
3.3 Folosirea întregii feți	19
3.4 “Bandă oculară”	20
3.5 Retele MLP	22
3.6 Rețele neuronale de conoluție	24
3.6.1 Utilizarea întregii feți	24
3.6.2 Utilizarea “benzilor oculare” ca date de antrenament	25
3.6.3 Îmbunătățirea arhitecturii CNN	26
3.7 Regresie folosind CNN	27
3.8 Cum sunt extrase reperele faciale	28
Concluzii	36

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 definiție¹, 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 interpreze conținutul unui ecran digital sau unei persoane 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.

¹Definiție preluată din DEX 2009

Idee

Dacă este să analizăm laptop-urile care sunt acum pe piață, am constată că toate sunt echipate cu o cameră frontală de luat vederi pentru videoconferințe, denumită ușual *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”.

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*.

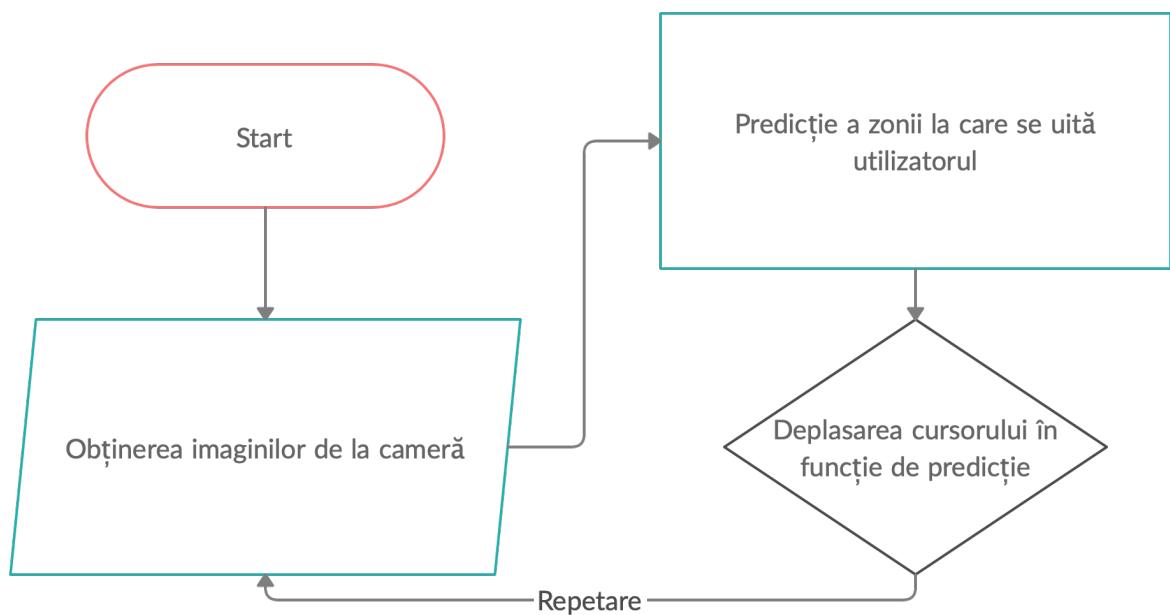


Figura 1: Procedură generală pentru deplasarea mouse-ului

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 aditionale.

*Camera Mouse*² este o propunere viabilă care urmărește o portiune fixată a feței (spre exemplu un ochi) și, când acea portiune își schimbă poziția, se schimbă și poziția cursorului. Oferă și funcționalități de simulare a apăsării pe butoanele mouse-ului, însă funcționează doar pentru sistemele care rulează Windows. Printre alternative se mai găsesc *IntelliGaze*³ și produse dezvoltate de *Tobii Dynavox*⁴, dar acestea necesită în primul rând hardware adițional și rulează doar pe Windows.

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 intelligentă 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.

²<http://www.cameramouse.org>

³<https://www.intelligaze.com/en/>

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

Obiective

Obiectivul principal al aplicației este acela de a simula folosirea unui mouse. Așadar, aplicația dezvoltată suportă cele mai importante funcționalități ale acestuia:

- mișcarea cursorului
- apăsarea butonului stâng
- apăsarea butonului drept⁵

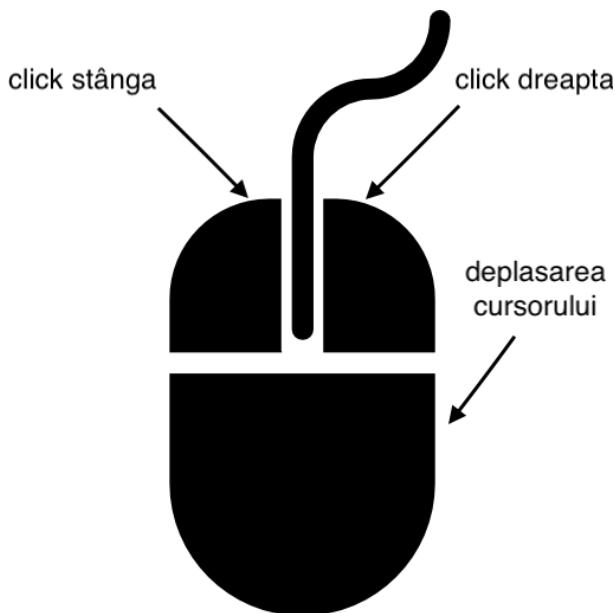


Figura 2: Funcționalități principale ale unui mouse. Imagine preluată și adaptată de pe Flaticon, autor: Kiranshastry

În urma analizei produselor deja existente, am vrut să dezvolt o aplicație care constituie un pachet atractiv de beneficii. Am construit o listă cu obiectivele principale ale aplicației:

- să fie cross-platform: aplicația ar trebui să ruleze pe toate sistemele de operare populare: Windows, Linux și MacOS
- să țină cont de diferențele fizionomice dintre utilizatori
- să aibă o interfață grafică
- să fie ușor și intuitiv de folosit
- să fie ușor de instalat

⁵Aceste funcționalități mai sunt denumite ușual și "click stânga/dreapta"

Metodologie

Descriere sumară a soluției (Eventual principalele probleme?)

Structura lucrării

Capitolul 1 Aici va fi prezentată... iar apoi...

Capitolul 2 Aici este prezentată...

Capitolul 3 Aici este prezentată...

Capitolul 4 Aici este prezentată...

Capitolul 1

Problema

1.1 Definirea problemei

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*), mai exact *Urmărire ochilor*, după cum a fost menționat și în introducere. Conform Wikipedia, 2020, urmărire ochilor este “procesul de măsurare a punctului de privire (unde se uită o persoană) sau a mișcării unui ochi relativ la cap”.¹

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”². 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 mulțimi de date, structură care la rândul ei influențează abordările de învățare.

¹Textul 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”

²Traducere liberă; text original: “ML is data-driven programming”

În acest caz, m-am concentrat asupra învățării supervizate, care presupune cunoașterea unei multimi de date

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

unde i reprezintă o instanță, un exemplu care conține o informație care trebuie prezisă, calculată, iar o reprezintă acea informație, un *adevăr de bază* deja cunoscut. Astfel, primul pas spre rezolvarea problemei este *colectarea datelor de antrenament*.

TODO ;————— asta de mutat in cap 2, la adunare de date Pentru această aplicație, mulțimea I de mai sus va fi alcătuită din imagini alea utilizatorului, iar mulțimea O va conține, spre exemplu, coordonatele (x, y) ale poziției la care se uita utilizatorul pe ecran atunci când acea imagine va fi capturată. Vom considera că pentru orice imagine din mulțimea I nu conține imagini în care utilizatorul nu se uita la ecran.

Învățarea profundă se preocupă, printre altele, de procesarea și analizarea imaginilor prin folosirea unor *arhitecturi profunde* bazate pe *rețele neuronale*. Învățarea poate fi la rândul ei supervizată, semi-supervizată sau nesupervizată. În termeni simpli, aceste tehnici încearcă să prezică sau să producă un rezultat pe baza unor exemple sau a unor relații între datele care trebuie procesate.

1.1.1 Neuronul Artificial

Înainte de a analiza arhitecturi de învățare profundă mai sofisticate, trebuie să aruncăm o privire rapidă asupra neuronului artificial. Pe scurt, este un model formal, simplificat al unui neuron biologic. Ne poate ajuta să realizăm *clasificare binară* folosind următoarea formulă:

$$y = \varphi(w * x + b)$$

În formula de mai sus, x este un vector de valori reale, reprezentând datele de intrare, iar w este un vector de valori reale denumit *vectorul ponderilor* (în engleză *weights*) și b este *partialitatea* (în engleză *bias-ul*). Produsul $w * x$ reprezintă produsul scalar și este egal cu $w * x = \sum_{i=1}^n w_i x_i$, n fiind lungimea vectorului x .

Rezultatul clasificării este dat de φ , denumit *funcție de activare*. Dacă aceasta se comportă ca un prag, o limită, un *threshold*, atunci funcția de activare se va traduce într-o

clasificare binară, cu rezultat 0 sau 1. Putem folosi și $\varphi(x) = x$, ecuație care ne poate ajuta la rezolvarea problemelor de tip *regresie liniară*.

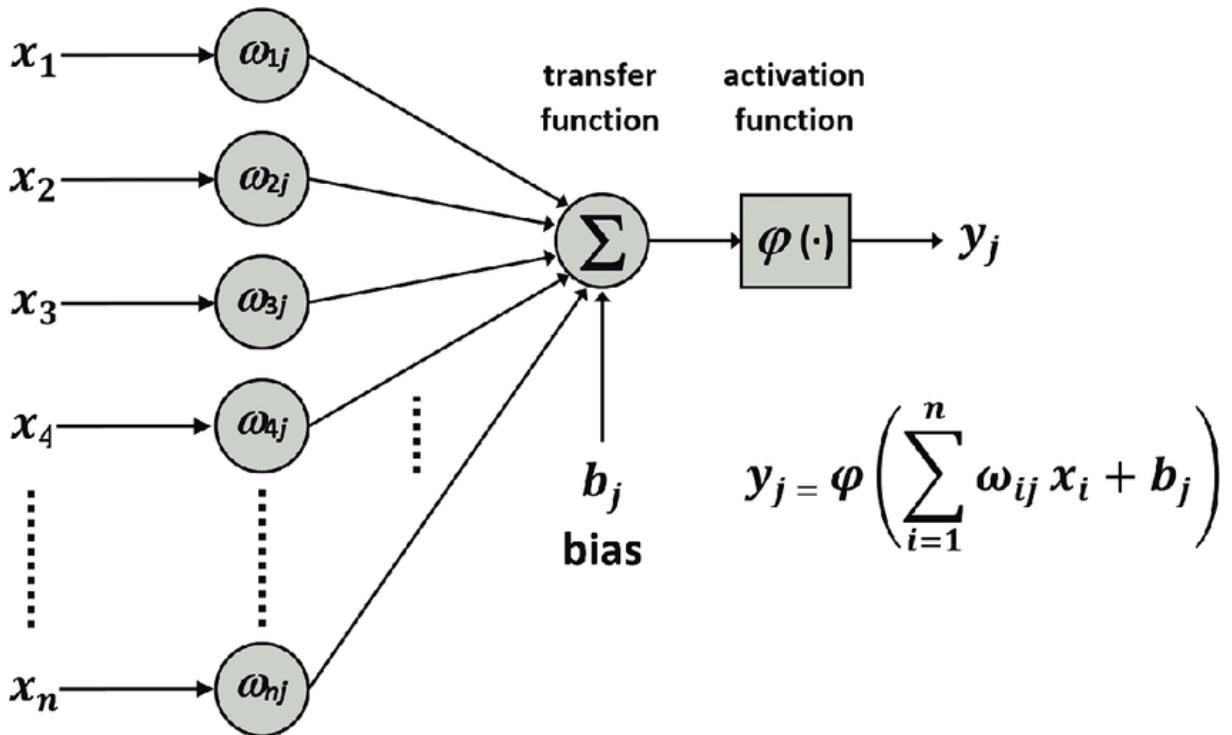


Figura 1.1: Algoritmul perceptron, bazat pe neuronul artificial. Imagine preluată de pe site-ul ResearchGate

1.1.2 Rețele neuronale

“Calul de bătaie” pentru problemele de Viziune Computerizată este Rețeaua Neuronală Artificială. Ea este, după cum sugerează și numele, compusă din mai mulți neuroni artificiali, distribuiți pe mai multe *straturi*.

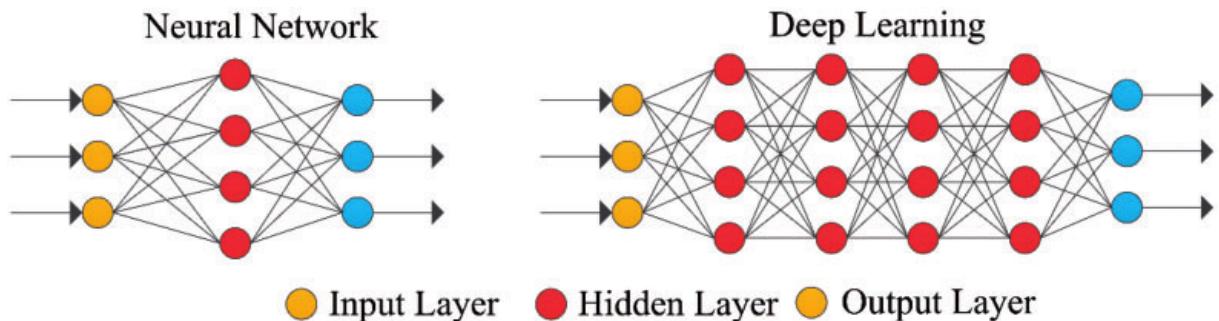


Figura 1.2: Structură generală a unei rețele neuronale. Imagine preluată de pe site-ul ResearchGate

1.2 Strategie

1.2.1 Obținerea datelor de antrenament

Este foarte bine cunoscut că un algoritm de Învățare Automată este pe atât de bun pe cât sunt datele pe care i le furnizăm. Acestea sunt incredibil de importante, aşa că m-am concentrat pe a dezvolta niște modalități facile de a aduna o multime consistentă de date

O altă etapă esențială este *procesarea de date*. Aceasta se preocupă 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.

1.2.2 Rețeaua de tip perceptron multistrat

Un tip special de rețea neuronală este *Perceptronul Multistrat (Multilayer Perceptron Network)*. Am folosit această arhitectură 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.

1.2.3 Rețele neuronale convolutionale

Un următor pas a fost studierea și aplicarea *retelelor 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 poziționare.

1.2.4 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

Informații preliminare

2.1 Maniera de lucru

Pentru dezvoltarea, structurarea și versionarea codului lucrării, am folosit platforma gratuită Github¹. Repository-ul proiectului poate fi accesat la această adresă: <https://github.com/sergiuiacob1/iClicker/>.

2.2 Configurație utilizată

Pentru rezultatele prezentate aici, este important de cunoscut că au fost create folosind următorul laptop: “MacBook Pro A1502, Early 2015”. Acest laptop dispune de un webcam capabil de o rezoluție maximă HD (1280x720). Experimentele realizate au fost făcute în general în medii bine luminate, întrucât o dată cu scăderea intensității luminii suferă și utilitatea aplicației prezentate aici.

2.3 Informații tehnice

Unul dintre cele mai populare limbaje de programare când vine vorba de Învățare Profundă este *Python*. Astfel, am ales să dezvolt aplicația folosind acest limbaj, deoarece suportul din partea comunității este unul foarte bun și resursele găsite online pentru a rezolva probleme comune sunt vaste. Aici este o listă a tehnologiilor pe care le-am folosit împreună cu Python:

¹<https://github.com>

Nume	Versiune	Link-uri utile
Python3	3.7	https://www.python.org
Conda	4.8.0	https://conda.io/

TODO asta trebuie sa dispara de aici. De pus in README pe github. Pt fiecare biblioteca sa spun la ce am folosit-o.

Trebuie să precizez, de asemenea, câteva biblioteci din Python care au adus funcționalități cruciale acestui proiect. Una dintre ele este *OpenCV*, pe care am folosit-o pentru captura de imagini. Pentru a construi o interfață grafică am folosit *PyQt5*.

Nume bibliotecă	Versiune	Link-uri utile
Keras	2.2.4	https://keras.io
PyTorch	1.4	https://pytorch.org
OpenCV	4.1.2	https://opencv.org
PyQt5	5.14	https://pypi.org/project/PyQt5/
dlib	19.19.0	https://pypi.org/project/dlib/
imutils	0.5.3	https://pypi.org/project/imutils/

2.4 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 însumi, deci trebuie luat 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. 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.

2.5 Date necesare

Datele de antrenament sunt extrem de importante. O expresie populară² 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. Apoi, am extras fie doar față acestuia din imagine, fie doar un ochi, fie o porțiune 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.

2.6 Obținerea datelor

2.6.1 Moduri de obținere

Pentru colectarea datelor am implementat 2 metode: o metodă “activă” și una “pasivă”. Folosind-o pe cea activă, utilizatorul trebuie să urmărească cu ochii cursorul mouse-ului în timp ce acesta se mișcă pe ecran 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 s-a efectuat o miscare a cursorului este capturată și o imagine și este salvată împreună cu poziția cursorului pe ecran la acel moment.

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. 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*.

```
1 def start_collecting(self, collection_type):  
2     dc_logger.info(f'Start collecting data in {collection_type} mode')
```

²Expresia este “garbage in, garbage out”, iar o expresie românească similară ar fi “semeni vânt, culegi furtună”

```

3     WebcamCapturer.start_capturing()
4
5     self.gui.start()
6     dc_logger.info('DataCollectorGUI started')
7
8     if collection_type == 'background':
9         self.mouse_listener.start_listening()
10    dc_logger.info('Mouse listener started')
11
12    elif collection_type == 'active':
13        threading.Thread(target=self.start_active_collection).start()

```

Listing 2.1: Colectarea datelor

2.6.2 Salvarea datelor

Când colectarea datelor este gata, acestea sunt salvate sub forma unei “sesiuni”. Fiindcare sesiune este definită de numărul imaginilor care au fost capturate, de rezoluția ecranului și rezoluția webcam-ului. Mai jos se poate observa structura directoarelor și modul în care aceste date sunt salvate, fără a aplica vreo modificare.

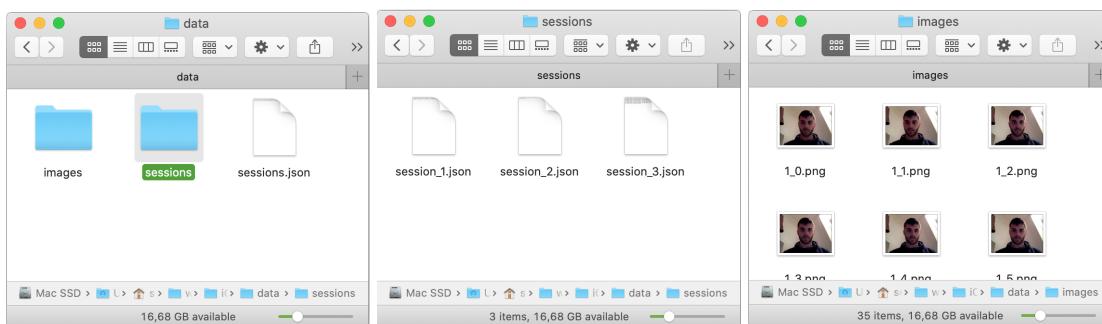


Figura 2.1: Cum am structurat datele

```

1 def save_collected_data(self):
2
3     if len(self.collected_data) == 0:
4
5         return
6
7     dc_logger.info(
8
9         f'Acquiring lock for data collection. Locked = {self.
10
11         collect_data_lock.locked() }')
12
13     self.collect_data_lock.acquire()
14
15     dc_logger.info('Lock acquired')
16
17     session_no = self.get_session_number()
18
19     dc_logger.info(f"Saving data for session_{session_no}")
20
21     self.save_session_info(session_no)
22
23     self.save_images_info(session_no)
24
25     self.save_images(session_no)

```

```
13     self.collect_data_lock.release()
14     self.collected_data = []
15     dc_logger.info('Saving data done')
```

Listing 2.2: Salvarea datelor

Capitolul 3

Procesarea datelor

3.1 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 multimea 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.

3.2 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??, ??, am extras doar porțiunile imaginilor care delimită ochii.

```
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
7     global _face_detector, _face_predictor
8
9     if _detectors_are_initialised() == False:
10         _initialize_detectors()
11
12     gray_image = Utils.convert_to_gray_image(cv2_image)
13     rects = _face_detector(gray_image, 0)
```

```

11 if len(rects) > 0:
12     shape = _face_predictor(gray_image, rects[0])
13     shape = face_utils.shape_to_np(shape)
14
15     eyes = []
16     for eye in ["left_eye", "right_eye"]:
17         # get the points for the contour
18         (eye_start, eye_end) = face_utils.FACIAL_LANDMARKS_IDXS[eye]
19         contour = shape[eye_start:eye_end]
20         # get the upper left point, lower right point for this eye
21         start = [min(contour, key=lambda x: x[0])[0],
22                  min(contour, key=lambda x: x[1])[1]]
23         end = [max(contour, key=lambda x: x[0])[0],
24                  max(contour, key=lambda x: x[1])[1]]
25         # extract the current eye
26         eyes.append(cv2_image[start[1]:end[1], start[0]:end[0]])
27
28
29     return eyes

```

Listing 3.1: 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.

TODO despre binary threshold, ce am folosit

```

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

```

Listing 3.2: Application d'un seuil binaire

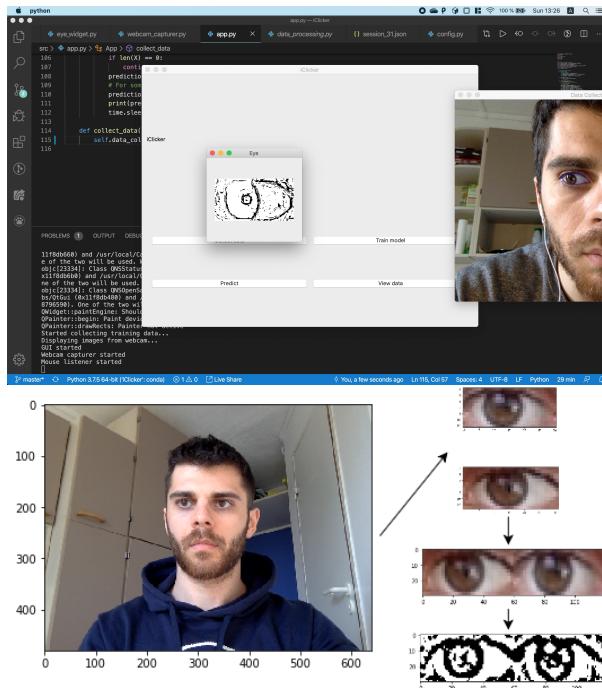


Figura 3.1: Obținerea de informații din imaginile ochilor

3.3 Folosirea întregii feți

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



Figura 3.2: Extraire le visage

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.

```
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 3.3: Extragerea feței dintr-o imagine

3.4 “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:

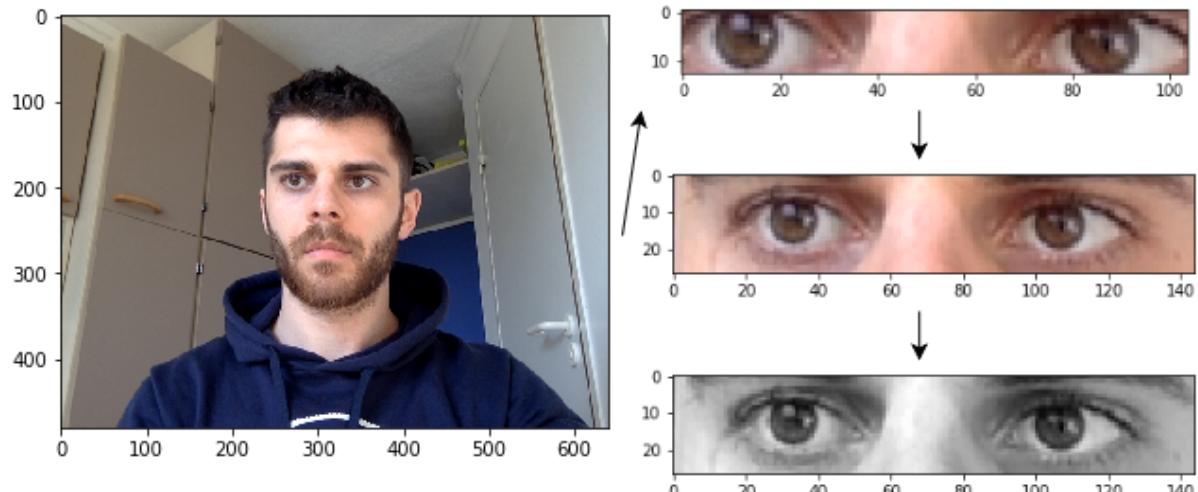


Figura 3.3: Extragerea “bandei oculare”

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.

```

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        # only for the first face found
16        shape = _face_predictor(gray_image, rects[0])
17        shape = face_utils.shape_to_np(shape)
18
19        (left_eye_start,
20         left_eye_end) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
21
22        (right_eye_start,
23         right_eye_end) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
24
25        # get the contour
26
27        start, end = min(left_eye_start, right_eye_start), max(
28            left_eye_end, right_eye_end)
29
30        strip = shape[start:end]
31
32        # get the upper left point, lower right point
33
34        start = [min(strip, key=lambda x: x[0])[0],
35                  min(strip, key=lambda x: x[1])[1]]
36
37        end = [max(strip, key=lambda x: x[0])[0],
38                  max(strip, key=lambda x: x[1])[1]]
39
40        # go a little outside the bounding box, to capture more details
41
42        distance = (end[0] - start[0], end[1] - start[1])
43
44        # 20 percent more details on the X axis, 60% more details on the Y
45        # axis
46
47        percents = [20, 60]
48
49        for i in range(0, 2):
50            start[i] -= int(percents[i]/100 * distance[i])
51            end[i] += int(percents[i]/100 * distance[i])
52
53        return cv2_image[start[1]:end[1], start[0]:end[0]]
54
55    return None

```

Listing 3.4: Extragerea “bandei oculare” în Python3

3.5 Rețele MLP

Primele experimente pe care le-am efectuat au fost efectuate folosind imaginile din care am extras ochii, pe care i-am convertit în imagini alb-negru (binary threshold), aşa cum a fost menţionat în prima parte a secţiunii de procesare de date^{3.2}. Fiecare ochi a fost redimensionat la 60x30 pixeli apoi au fost uniți orizontal, formând o imagine alb-negru de dimensiune 120x30 pixeli. De asemenea, am etichetat fiecare imagine cu numărul celulei în care se uita utilizatorul, conform dimensiunii grilei folosite^{??}. Scopul a fost de a prezice, pentru imagini noi, numărul acelei celule. Iată prima arhitectură pe care am încercat-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 3.5: MLP 1

Important Pentru primele experimente din această secţiune am folosit doar imagini în care utilizatorul se uita în colturile 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 menţiune este că datele de antrenament reprezintă 80% din multimea totală de date, restul fiind date de testare.

Experiment	Dim. grilă	Nr. 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 disperse 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.

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 că 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, alternante 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.

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.

3.6 Rețele neuronale de convoluție

3.6.1 Utilizarea întregii feți

Mai departe am vrut să experimentez cu rețelele convoluționale și am început prin a utiliza fețele extrase din imagini^{3.2}, pe o grilă de dimensiune 2×2 ?? . 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 3.6: Prima arhitectură CNN

Important Pentru primele experimente din această secțiune am folosit doar imaginiile în care utilizatorul se uita în colțurile ecranului. De exemplu, pentru o grilă de dimensiune 3×3 , 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: TODOTRANSLATE

Expérience	Grille taille	Taille des données	Epoques	Optimizer	Learning rate	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

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, ceea ce m-a făcut să constată că performanța unui model crește direct proporțional cu dimensiunea mulțimii de date pe care o avem la dispoziție. Putem vedea de asemenea semne de overfit în aceste modele.

3.6.2 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.^{3.3} Primele modele au fost antrenate doar cu acele imagini în care utilizatorul se uita în colțurile ecranului.

Expérience	Grille taille	Taille des données	Epoques	Optimizer	Learning rate	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 indepartam 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, încrucișat 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.

3.6.3 Îmbunătățirea arhitecturii CNN

Experimentele 17–22 au fost rezultatul încercării de a îmbunătăți arhitectura initială a rețelei de conoluție pe care am folosit-o 3.6.1. Am incercat fie să elimin straturi de conoluție, fie să schimb numărul filtrelor pentru a observa care este comportamentul acesteia.

Am remarcat că atunci când suprimez al doilea strat de conoluție, 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 conoluție, 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 3.7: Arhitectura CNN îmbunătățită

3.7 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 $f(x) = x$. 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 (ceea 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:

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

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.

Expérience	Grille taille	Taille des données	Epoques	Optimizer	Learning rate	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

3.8 Cum sunt extrase reperele faciale

Utilizând cu succes reperele faciale pentru a localiza și a extrage ochii, ca apoi să folosesc aceste date pentru o rețea convezională 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, 2020, 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).

J'ai essayé de ne prédire que le centre de l'oeil à partir d'une image de l'oeil. Pour obtenir des données de formation, j'ai utilisé le jeu de données de crowdpupil, qui consiste en 792 images, chaque image étant étiquetée avec la position du centre de l'oeil. Il existe deux méthodes pour essayer de prédire les points de repère du visage : essayer de trouver les coordonnées exactes (x, y) par régression, ou essayer de reconstruire des cartes thermiques qui contiennent ces points de repère du visage.

J'ai choisi la deuxième option, et j'ai donc généré une carte thermique pour chaque image de l'oeil qui codait le centre de l'oeil. J'ai mesuré une moyenne pour la distance entre le centre de la pupille et sa marge, et je l'ai utilisée comme “variance” pour une Distribution Gaussienne 2D, centrée dans l'oeil.

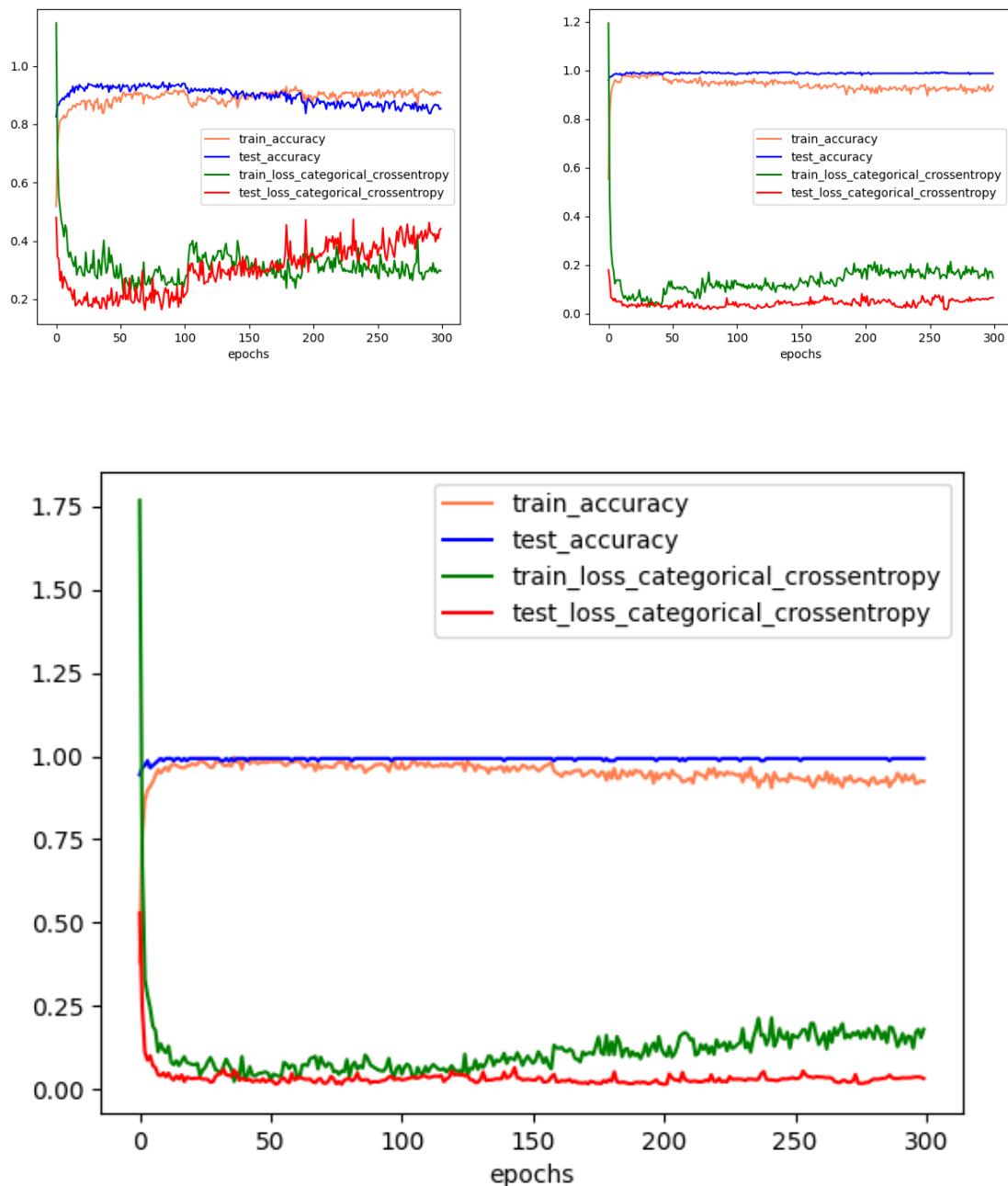


Figura 3.4: Modelele 1, 2 și 3

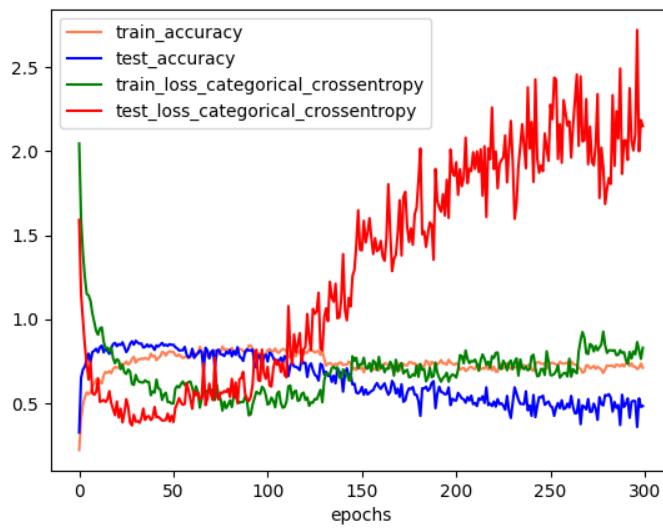


Figura 3.5: Modelul 4

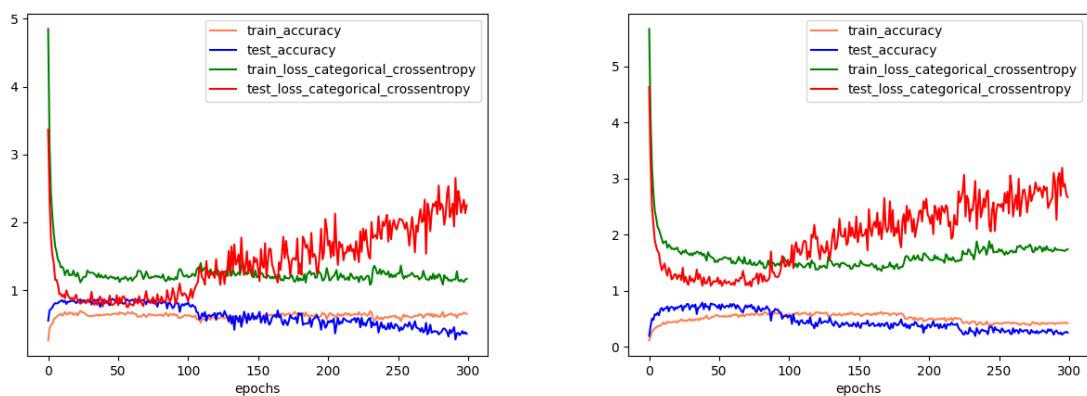


Figura 3.6: Modelele 5 și 6

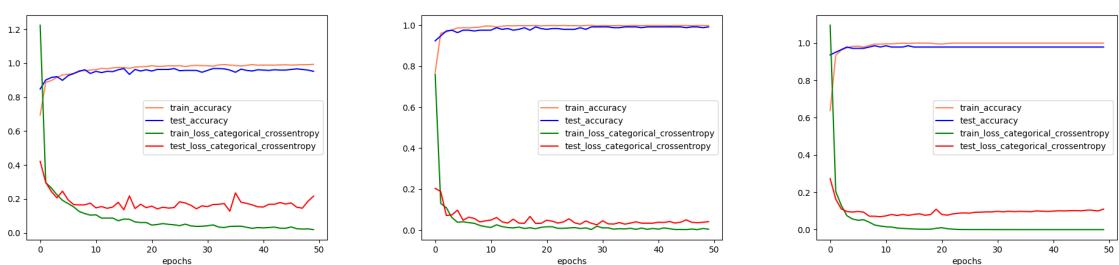


Figura 3.7: Modelele 7, 8 și 9

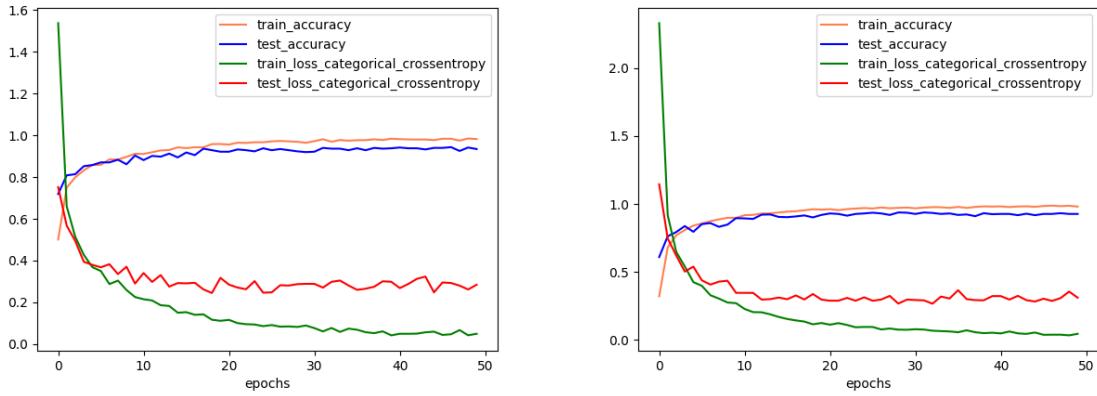


Figura 3.8: Modelele 10 și 11

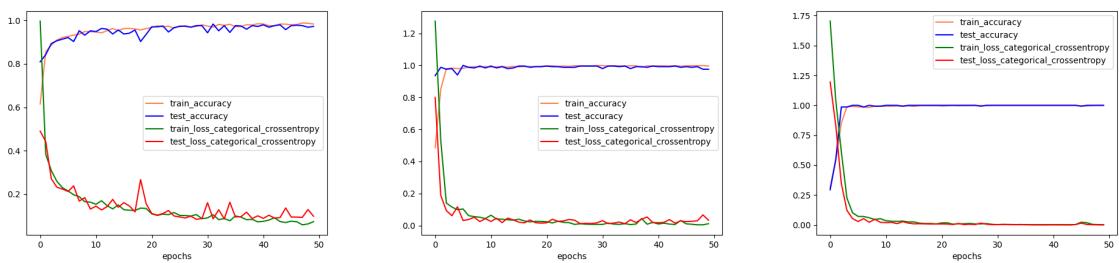


Figura 3.9: Modelele 12, 13 și 14

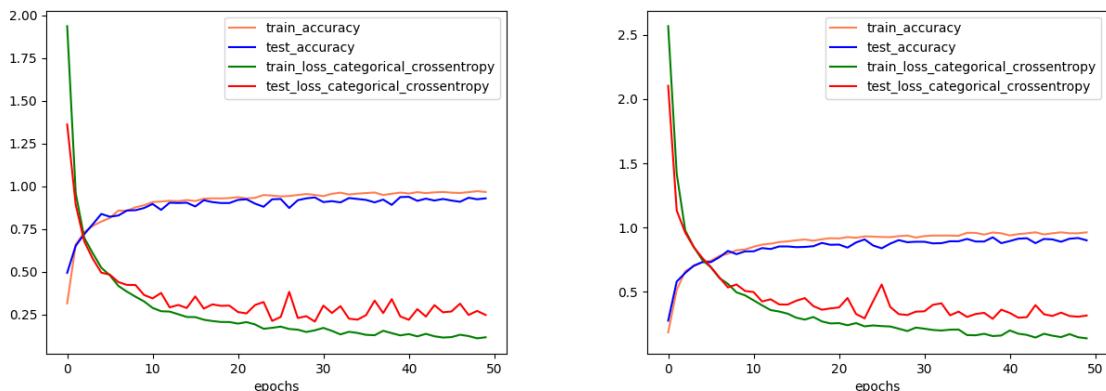


Figura 3.10: Modelele 15 și 16

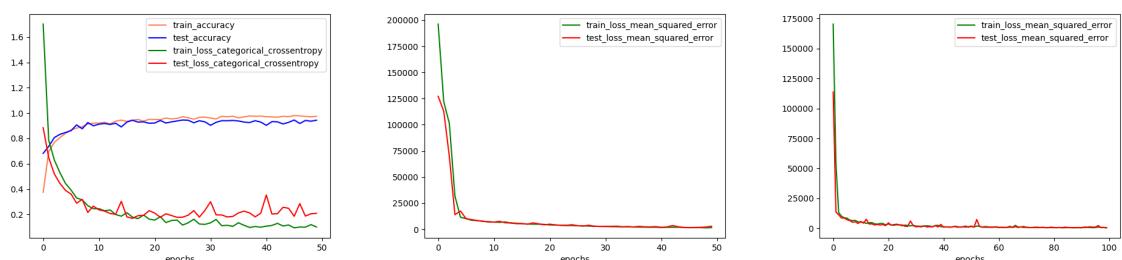


Figura 3.11: Modelele 22, 23 și 24

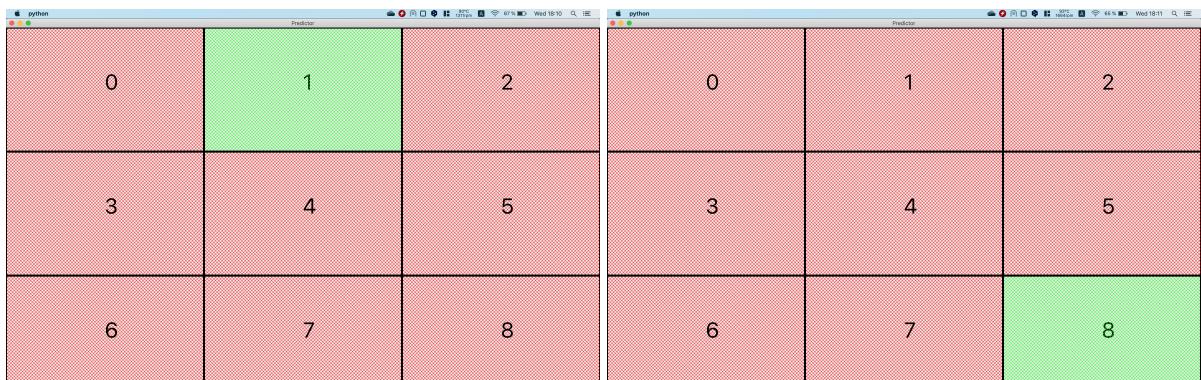


Figura 3.12: Utilizarea rețelei CNN pentru urmărirea ochilor

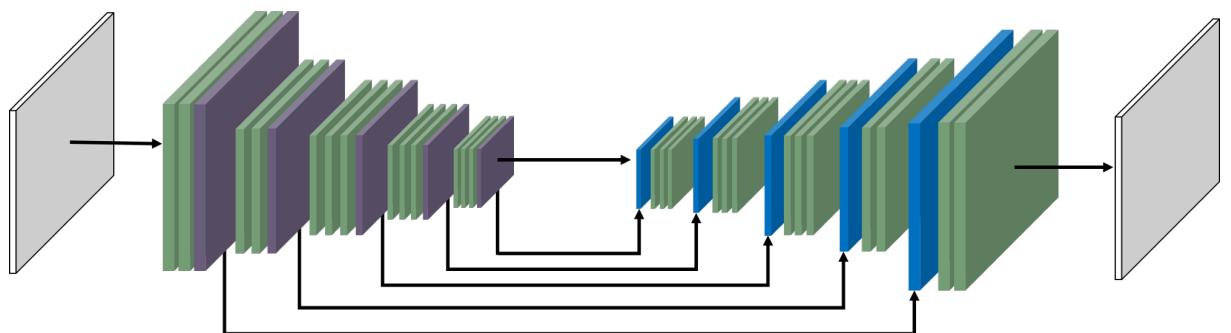


Figura 3.13: Exemple d'architecture Hourglass

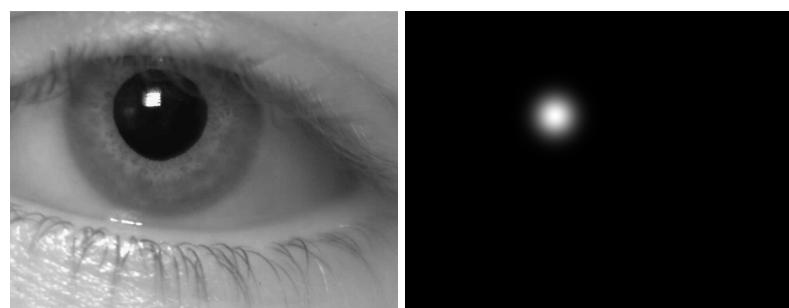


Figura 3.14: Centrul ochiului codificat printr-o hartă termografică

Après diverses expériences, j'ai découvert que je pouvais assez bien reconstruire les cartes thermiques en utilisant une *architecture simplifiée* de CNN Hourglass :

```
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        # encoding done, starting decoding
29        self.layer2 = nn.Sequential(
30            nn.Upsample(size=(28, 35), mode='bilinear'),
31            nn.ConvTranspose2d(filters[3], filters[2], kernel_size=2,
32            stride=1, padding=1),
33            nn.ReLU(inplace=True),
34            nn.BatchNorm2d(filters[2]),
35
36            nn.Upsample(size=(54, 68), mode='bilinear'),
37            nn.ConvTranspose2d(filters[2], filters[1], kernel_size=3,
38            stride=1, padding=2),
```

```

37         nn.ReLU(inplace=True),
38         nn.BatchNorm2d(filters[1]),
39
40         nn.Upsample(size=(104, 132), mode='bilinear'),
41         nn.ConvTranspose2d(filters[1], filters[0], kernel_size=2,
42                         stride=1, padding=2),
43         nn.ReLU(inplace=True),
44         nn.BatchNorm2d(filters[0]),
45
46         nn.Upsample(size=(202, 258), mode='bilinear'),
47         nn.ConvTranspose2d(filters[0], 1, kernel_size=3, stride=1,
48                         padding=2),
49         nn.ReLU(inplace=True),
50         nn.BatchNorm2d(1),
51     )

```

J'ai formé le réseau pendant 12 époques, en utilisant comme optimiseur Adam et comme score MSE. Pour la dernière époque, la moyenne des erreurs sur l'ensemble des données de test (environ 20% du total des données) était de 8,425983. Voici le réseau essayant de reconstruire les cartes thermiques à partir des images des yeux :

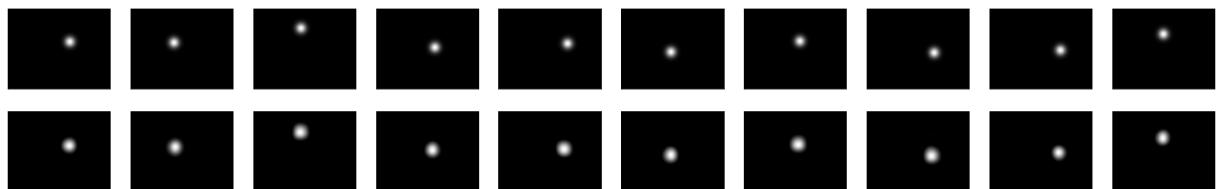


Figura 3.15: Reconstruirea hărtilor termografice. Prima linie reprezintă adevărul de bază, iar a doua linie rezultatele modelului antrenat

Cependant, l'utilisation du réseau sur des images de mes propres yeux n'a pas vraiment donné de bons résultats. L'une des raisons est que la qualité de ma webcam n'est pas très bonne et que les images des yeux extraites étaient très bruyantes.

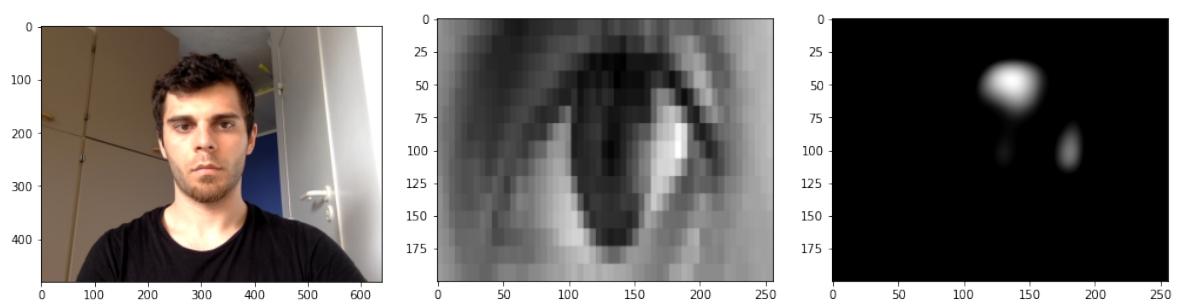


Figura 3.16: Tester le réseau Hourglass sur des images de moi-même

Concluzii

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nunc mattis enim ut tellus elementum sagittis vitae et. Placerat in egestas erat imperdiet sed euismod. Urna id volutpat lacus laoreet non curabitur gravida. Blandit turpis cursus in hac habitasse platea. Eget nunc lobortis mattis aliquam faucibus. Est pellentesque elit ullamcorper dignissim cras tincidunt lobortis feugiat. Viverra maecenas accumsan lacus vel facilisis volutpat est. Non odio euismod lacinia at quis risus sed vulputate odio. Consequat ac felis donec et odio pellentesque diam volutpat commodo. Etiam sit amet nisl purus in. Tortor condimentum lacinia quis vel eros donec. Phasellus egestas tellus rutrum tellus pellentesque eu tincidunt. Aliquam id diam maecenas ultricies mi eget mauris pharetra. Enim eu turpis egestas pretium.

Bibliografie

- Ciortuz, C. D. 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. (2020). 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
- Karim, R. (2020). Intuitions on L1 and L2 Regularisation. Retrieved April 17, 2020, from <https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261>
- Pierre Baldi, P. S. (2020). The dropout learning algorithm. Retrieved April 10, 2020, from <https://www.sciencedirect.com/science/article/pii/S0004370214000216>
- Wikipedia. (2020). Eye Tracking. Retrieved March 15, 2020, from https://en.wikipedia.org/wiki/Eye_tracking