

**UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA  
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ  
SPECIALIZAREA INFORMATICĂ ROMÂNĂ**

**LUCRARE DE LICENȚĂ**

**Restaurarea imaginilor vechi și  
eliminarea obiectelor dintr-o poză**

**Conducător științific  
Lect. Ph.D. Radu D. Gaceanu**

*Absolvent  
Iachim Mihai-Bogdan*

**2023**



---

## ABSTRACT

---

Restoring and modifying images is a very old practice, either from the time of manual restoration or by filling in any gaps that have damaged the artwork over the years. Traditionally, artists hand-filled images, which is a tedious and time-consuming process. In modern times, the digitization of analog images often leads to unwanted defects, such as scratches, blurring, etc., which should be eliminated. This practice is known as inpaint.

The original contribution of this paper consists in the generation of a new and improved inpainting algorithm. To achieve this, we study how an inpainting algorithm works and we identify a few different methods at every step of inpainting. Then, we generate a few algorithms based on combinations of these methods. And in the end, we understand the pros and cons of every method, thus allowing us to create the best algorithm using these methods.

The first chapter briefly presents the history of image inpainting and the objective of this paper.

The second chapter presents a few well known inpainting algorithms in the first part. Afterwards it dictates about a few methods at every step of inpainting. In the end of the chapter we generate a few algorithms using these methods, we test them and based on these results we create the best possible algorithm using these methods.

The third chapter goes over the theory necessary to understand how to create and use a Mask R-CNN. We will use this Mask R-CNN in our application to facilitate the creation of masks over objects known by the ai.

The fourth chapter covers the technologies used to develop the application separated into Client and Server. The technologies on both the client side and server side are well known.

The fifth chapter presents the architecture of the application, its use case diagram and a manual for the user where all the buttons are mapped and there are a few notes about the behaviour of the application.

The sixth chapter summarises our work and gives a few ideas on how it can be expanded and improved.

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

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>1</b>
<b>2</b>	<b>Metode de umplere a imaginii si teoria lor</b>	<b>3</b>
2.1	Introducere . . . . .	3
2.2	Algoritmi cunoscuti de umplere a imaginilor . . . . .	3
2.2.1	Algoritmul lui Oliveira . . . . .	4
2.2.2	Hadhoud, Moustafa, and Shenoda's Algorithm . . . . .	4
2.2.3	Algoritmul lui Efros si Leung . . . . .	5
2.2.4	Algoritmul lui Criminisi . . . . .	6
2.3	Primul pas dintr-un algoritm de umplere . . . . .	7
2.3.1	Liniar(L) . . . . .	8
2.3.2	Aleatoriu(A) . . . . .	8
2.3.3	Calculand prioritatea regiuni pixelilor(CD) . . . . .	8
2.4	Al doilea pas dintr-un algoritm de umplere . . . . .	10
2.4.1	Difuzie(D) . . . . .	10
2.4.2	Folosind regiuni similare(S) . . . . .	11
2.5	Implementarea algoritmilor . . . . .	12
2.6	Algoritmi generati . . . . .	14
2.7	Evaluarea Algoritmilor . . . . .	17
2.8	Algoritmul final . . . . .	18
<b>3</b>	<b>Fundamente teoretice inteligență artificială</b>	<b>19</b>
3.1	Perceptron . . . . .	19
3.1.1	Neuron biologic . . . . .	19
3.1.2	Neuron artificial . . . . .	20
3.1.3	Perceptron . . . . .	21
3.2	Rețele neuronale . . . . .	21
3.2.1	Ce este o RN? . . . . .	21
3.2.2	Retropagarea erorii(backpropagation) . . . . .	22
3.2.3	Funcția de activare . . . . .	22
3.2.4	Funcția cost . . . . .	23

3.2.5	Tipologii . . . . .	24
3.3	Rețele neuronale convolutionale . . . . .	25
3.3.1	Ce este o RNC? . . . . .	25
3.3.2	Stratul convolutional . . . . .	25
3.3.3	Stratul de agregare(pooling layer) . . . . .	26
3.3.4	Stratul complet conectat . . . . .	27
3.3.5	Stratul de abandon(dropout layer) . . . . .	28
3.4	Mask R-CNN . . . . .	29
3.4.1	Segmentarea instanțelor . . . . .	29
3.4.2	Cum funcționează un Mask R-CNN? . . . . .	30
<b>4</b>	<b>Tehnologiile din aplicație</b>	<b>33</b>
4.1	Tehnologi client . . . . .	33
4.1.1	HTML . . . . .	33
4.1.2	CSS . . . . .	33
4.1.3	JavaScript . . . . .	34
4.1.4	jQuery . . . . .	34
4.2	Tehnologi Server . . . . .	35
4.2.1	Python . . . . .	35
4.2.2	Flask . . . . .	36
4.2.3	OpenCV . . . . .	36
4.2.4	Numpy . . . . .	38
4.2.5	Proiect Mask R-CNN . . . . .	39
<b>5</b>	<b>Aplicatia</b>	<b>41</b>
5.1	Arhitectura aplicatie . . . . .	41
5.2	Diagrama cazurilor de utilizare . . . . .	42
5.3	Manualul utilizatorului . . . . .	42
<b>6</b>	<b>Concluzii și eventuale îmbunătățiri</b>	<b>44</b>
<b>Bibliografie</b>		<b>45</b>

# Capitolul 1

## Introducere

Restaurarea și modificarea imaginilor în mod nedetectabil este o practică foarte veche, încă din momentul restaurării manuale sau prin completarea eventualelor goluri care au deteriorat opera de artă de-a lungul anilor. În mod tradițional, artiștii execuțau manual umplerea de imagini, ceea ce era un proces foarte obositor și consumator de timp. În vremurile moderne, digitizarea imaginilor analogice duce adesea la defecte nedorite, cum ar fi zgârieturi, blurari etc., care ar trebui eliminate. Această practică este cunoscută sub numele de inpainting. Scopul umplerii de imaginii este de a recupera porțiunile deteriorate ale imaginii. Necesitatea restaurării imaginii se extinde de la picturi la fotografii și filme pentru recondiționarea oricărora deteriorări, cum ar fi fisuri sau zgârieturi în picturi și fotografii, sau pentru a modifica imaginea prin adăugarea sau eliminarea elementelor. Scopul este de a produce o imagine modificată care este similară cu imaginea originală și regiunea pictată îmbinată nu poate fi distinsă de imaginea modificată. Aplicații de completare a imaginii, inclusiv recuperarea blocurilor pierdute în transmisia wireless a imaginii, îndepărțarea obiectelor ca efecte speciale în falsificarea imaginii, îndepărțarea zgârieturilor în restaurarea imaginilor istorice, îndepărțarea ocluziilor precum subtitrarea textului, și logo-uri, stampile, îndepărțarea unor obiecte nedorite din imagine.

Obiectivul acestei lucrări este de a analiza cum funcționează un algoritm de umplere a imaginii, identificând câteva metode de realizare și cum să se folosească aceste metode pentru ca rezultatul să fie cât mai bun.

Aplicația noastră pune la dispoziție 2 algoritmi de umplere a imaginii. Unul rapid, însă cu rezultate nu foarte bune și unul lent cu rezultate bune.

Primul algoritm este implementat în OpenCV "inapint" care urmează algoritmul lui Alexandru Telea. Al doilea este creat prin selectarea metodelor posibile de umplere a imaginilor și folosindu-le în câțiva algoritmi pentru a vedea punctele tari și punctele slabe a fiecarei metode. Apoi folosind aceste informații se încerca creearea celui mai bun algoritm.

Lucrarea este structurată astfel:

**Capitolul 2** prezintă teoria și practica din spatele creării celui de-al doilea algoritm de umplere a imaginii. În 2.2 sunt prezentate câțiva algoritmi cunoscuți. În 2.3 și 2.4 apar pașii și metodele de creare a unui algoritm de umplere a imaginilor. De la 2.5 la 2.8 este partea practică unde explic implementările și rezultatele lor.

În **Capitolul 3**, deoarece folosesc inteligență artificială pentru a ajuta utilizatorul cu desenarea obiectelor comune pentru masca, explic fundamentele teoretice necesare pentru a crea și înțelege un Mask R-CNN.

**Capitolul 4** surprinde tehnologiile necesare creării acestei aplicații web, împărțite pe client și server.

În **Capitolul 5** este prezentată aplicația. Aici putem găsi arhitectură ei, diagramele cazurilor de utilizare și un scurt manual al aplicație.

Lucrarea se încheie cu capitolul 6, care summarizează munca noastră și prezintă câteva idei pentru viitor.

# **Capitolul 2**

## **Metode de umplere a imaginii si teoria lor**

### **2.1 Introducere**

Procesul de umplere a regiunii în urma pierderii de informații din imaginile digitale reprezintă un aspect important în procesarea imaginilor. Image inpainting se referă la metodele de restaurare utilizate pentru a îndepărta daunele sau obiectele nedorite dintr-o imagine, într-un mod natural, astfel încât un observator neutru să nu observe nicio modificare și să considere rezultatul ca fiind imaginea originală.

Metodele de restaurare pot fi clasificate în trei mari categorii: tehnici de vopsire structurală, metode de vopsire texturală și metode hibride. În ciuda acestor trei categorii, metodele pot fi împărtite în algoritmi bazați pe ecuații cu derivate parțiale (PDE), metode semiautomate de inpainting, metode de sinteză a texturii, algoritmi bazați pe modele/șabloane și tehnici hibride în funcție de caracteristicile specifice.

### **2.2 Algoritmi cunoscuți de umplere a imaginilor**

Voi enumera 5 algoritmi de umplere a imaginilor. Primul algoritm a fost dezvoltat de Bertalmio et al. [MBB00] și reprezintă o metodă de referință în pictură. A doua, prezentată în [MMOC01], prezintă o soluție simplă, bazată pe o operatie de convoluție, urmată de a treia, o versiune adaptată a precedentei [MMHS09]. Următoarea tehnică a fost propusă de Efros și Leung [EL99] pentru sinteza texturii și ultimul algoritm considerat de la Criminisi și colab. [ACT04], combinând tehnici pentru pictura în interior structurată și reproducerea texturii.

### 2.2.1 Algoritmul lui Oliveira

Pe baza metodei lui Bertalmio, Oliviera et al. [MMOC01] au propus un algoritm de inpainting care se bazează exclusiv pe difuzie. Pașii de procesare constau în ștergerea informațiilor de culoare din interiorul măștii, urmată de detectarea marginilor pentru zona oclusă. Pornind de la pixelii de pe margine, se aplică apoi o operație de convoluție, folosind o vecinătate centrată pe fiecare pixel de contur și unul dintre nucleele propuse (Figura 2.1). Valorile lui a, b și pentru ambele nuclee sunt 0,073235, 0,176765 și, respectiv, 0,125 [MMOC01].

a	b	a
b	0	b
a	b	a

c	c	c
c	0	c
c	c	c

Figura 2.1:Kernel-urile de convoluție propuse de Oliveira [VB14]

### 2.2.2 Hadhoud, Moustafa, and Shenoda's Algorithm

Hadhoud și colab. [MMHS09] au propus o îmbunătățire a metodei lui Oliveira, atât în ceea ce privește imaginea finală, cât și timpul necesar de procesare. Unii pași au fost păstrați din metoda originală din [MMOC01], care implică selectarea măștii, urmată de eliminarea informațiilor de culoare existente în mască. Spre deosebire de algoritmul lui Oliveira, metoda folosește un nucleu de convoluție definit diferit. Ideea a fost de a folosi cât mai multe informații din afara regiunii, în vederea procesului de restaurare (Figura 2.2). Folosind mai mulți vecini cunoscuți, restaurarea poate fi realizată chiar și într-o singură iterație.

a	b	a	c	c	c
b	a	b	c	c	c
a	b	0	c	c	0

Figura 2.2:Kernel-urile de convoluție propuse de Hadhoud [VB14]

### 2.2.3 Algoritmul lui Efros si Leung

Pașii algoritmului includ definirea unei măști și specificarea unei zone sursă, urmată de detectarea marginilor pentru zona oclusă [EL99]. Toți pixelii de pe margine vor fi sortați în ordine descrescătoare după numărul de vecini cunoscuți. Va fi definit un şablon centrat pentru fiecare pixel ales pentru restaurare. Această fereastră are o dimensiune parametrizată și va fi folosită în căutarea de blocuri similare în zona sursă. Măsura similarității este dată de suma diferențelor pătrate (SSD). Pentru a păstra caracterul local al texturii, se folosește un nucleu gaussian, care are ca scop controlul influenței pixelilor aflați prea departe de zona oclusă. Considera:

$$d = d_{SSD} * G$$

În funcție de valoarea SSD, se va obține o colecție de blocuri candidate. Considera

$$\begin{aligned} w_{best} &= \arg_w \min d(w(p), w) c I'_{smp} \\ d(w(p), w) &< (1 + \epsilon) d(w(p), w_{best}), \epsilon = 0.1, \end{aligned}$$

unde pixelul procesat este  $p$ ,  $I_{smp}$  reprezintă zona sursă și  $d(w(p), w)$  descrie distanța de la o fereastră de dimensiune  $w$  centrată pe pixel  $p$  la un bloc de aceeași dimensiune  $w$ , găsit în sursă. Unul dintre blocurile candidate va fi ales aleatoriu și informațiile de culoare ale pixelului său central vor fi atribuite pixelului de pe margine (centrul şablonului de fereastră).

### 2.2.4 Algoritmul lui Criminisi

Atât sinteza texturii, cât și tehnicele de inpainting sunt avantajele combinate ale algoritmului Criminisi inpainting. La fel de demonstrat în Fig 2.3 (a, b, c) conform algoritmului Criminisi literatură porțiunea deteriorată din imagine care trebuie să fie umplut este denumit regiune ţintă și este indicat prin  $(\Omega)$  restul unei părți a imaginii  $I$  este cunoscută ca regiune sursă care mai departe poate fi explicitat ca  $(\phi = I - \Omega)$  „Frontul de umplere” este indicat prin  $(\&\Omega)$  este linia de graniță dintre regiunea sursă și ţintă regiune. Practic algoritmul Criminisi este un algoritm iterativ, astfel încât la fiecare iterare un patch centrat pe  $(p \in \Omega)$  pe frontul de umplere cu superioritate este atent ales să fie completat într-un mod corect, astfel de plasture este denumit plasture ţintă. Similar pentru a obține un patch asemănător cu patch-ul ţintă, regiunea sursă este patch căutată prin plasture. Se poate înțelege cu ușurință din Fig 2.3(b),(c) că plasturele sursă candidat se va așeza exact fie pe marginea respectivă sau margini cu aceeași culoare a marginii dacă patch-ul ţintă este de-a lungul uneia dintre marginile imaginii.

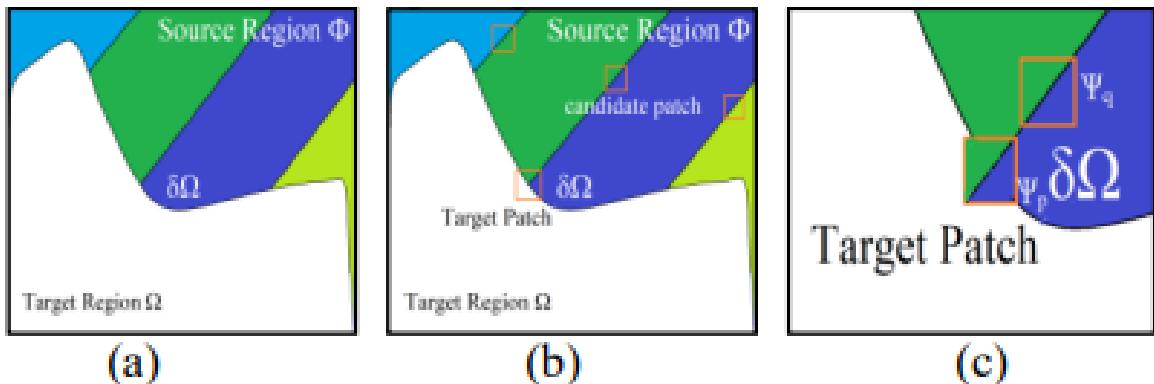


Figura 2.3: a) the original image. b) the target patch  $\Psi_p$  and the candidate patch  $\Psi_q$ . C) target patch filling with most similar patch from the source region [RY19]

În mod implicit, dimensiunea patch-ului utilizat în modelul Criminisi este de (9x9). acest Finalizarea patch cu patch reduce timpul de execuție al algoritm. Mărește și mai mult acuratețea structurii imaginii propagarea de asemenea. Îndepărarea obiectului nefavorabil și reconstructia imaginii după modelul Criminisi se produce cu trei trepte. Inițial, funcția de prioritate atribuie toate prioritățile valori pentru fiecare patch-țintă. Iată un patch cu cea mai mare prioritate este ales pentru a fi completat.

$$P(p) = C(p)D(p)(1)$$

Unde  $C(p)$  este încrederea, prioritatea patch-ului  $P(p)$  și  $D(p)$  este datele. Din nou

termenul de date și termenul de încredere pot fi recalculat ca

$$C(p) = \left( \sum_{q \in \Psi_p \cap (I - \Omega)} C(q) \right) / |\Psi_p|$$

$$D(p) = (|\nabla I_p^\perp n_p|) / \alpha$$

Unde zona de patch este prezentată de  $|\Psi_p|$  iar cel normalizat factor ( $\alpha = 255$ ) în imaginea în tonuri de gri. Vector ortonormal la punctul (p) este (np). ortogonalitatea se notează cu ( $\perp$ ).

În pasul următor, acest algoritm funcționează pe textura imaginii și propagarea structurii. Odată ce fiecare pixel de pe umplere-front este atribuită o valoare de prioritate, patch-ul cu cea mai mare prioritate este selectat ( $\Psi_p$ ). după ce regiunea sursă este căutată pentru a afla cea mai asemănătoare corecție sursă cu  $\Psi_p$ .

$$\Psi_q = \operatorname{argmin}_{\Psi_q \in \phi} d(\Psi_p, \Psi_q)$$

Unde  $d(\Psi_p, \Psi_q)$  este suma diferențelor pătrate. Apoi informațiile legate de pixelii patch-ului sursă sunt copiate pixelii corespunzători ai patch-ului țintă. Ca rezultat imaginea textura și structura sunt propagate petic cu petic în regiunea țintă. În cele din urmă, în al treilea pas este noua umplere față diferențiate astfel valorile de încredere sunt actualizate.

$$C(p) = C(p'), \forall p \in \Psi'_p \cap \Omega$$

[RY19][ACT04]

## 2.3 Primul pas dintr-un algoritm de umplere

Sa presupunem ca  $\omega$  reprezinta regiunea care trebuie umpluta si  $\varphi\omega$  conturul ei. Alegem ca propagarea informatiei din restul imaginii in  $\omega$  sa se faca de la exterior catre interior. Adica tot timpul vom incepe umplerea cu pixelii din  $\varphi\omega$ . Aceasta metoda este cea mai naturala si in cazul in care am incerca sa umplem imaginea manual. In cazul in care am alege sa pornim cu pixelii din afara lui  $\varphi\omega$  am fi obligatii sa ghicim la inceput valorile acestor pixeli, in asa fel incat , cand vom umple pixelii din  $\varphi\omega$  sa obtinem o similaritate intre ei si restul imaginii.

Odata ce umplem un pixel din  $\varphi\omega$  el este scos din  $\omega$  si va fi folosit pentru calcularea viitorilor pixeli.

### 2.3.1 Liniar(L)

In aceasta metoda vom alege toti pixeli din  $\varphi\omega$  in ordine crescatoare a pozitie. Aceasta metoda este cea mai simpla si cea mai rapida, in sensul in care aplicam algoritmul de umplere in ordinea in care gasim pixelii de pe  $\varphi\omega$ . Insa ea ar putea fi periculoasa daca pixeli care au in vecinatate multi pixeli care apartin de  $\omega$  sunt calculati la inceput, ei neavand foarte multa informatie.

### 2.3.2 Aleatoriu(A)

Aceasta metoda este similara cu cea de dinainte, diferenta fiind ca alegem la intamplare pixeli din  $\varphi\omega$  cu care sa incepem. Ea poate fi mai buna deoarece regiunile care trebuie umplute tind sa aiba putine zone cu vecinatati mici si mari de informatie valabila de folosit si multe zone cu vecinatati medii de informatie. In acest caz alegand aleatoriu pixelii avem o sansa mai mare sa nu folosim vecinatatile cu putina informatie.

### 2.3.3 Calculand prioritata regiuni pixelilor(CD)

Aceasta metoda este inspirata din "Region Filling and Object Removal by Exemplar-Based Image Inpainting" a lui Criminisi [ACT04]

Pentru toate regiunile  $\Psi_p$  centrate pe punctul p pentru  $p \in \varphi\omega$  definim prioritata  $P(p)$  ca produsul dintre  $C(p)$  si  $D(p)$ .

$$P(p) = C(p) * D(p)$$

Vom numi pe  $C(p)$  termenul de incredere si pe  $D(p)$  termenul de date.

$$C(p) = \left( \sum_{q \in \Psi_p \cap (I - \Omega)} C(q) \right) / |\Psi_p|$$

$$D(p) = (|\nabla I_p^\perp n_p|) / \alpha$$

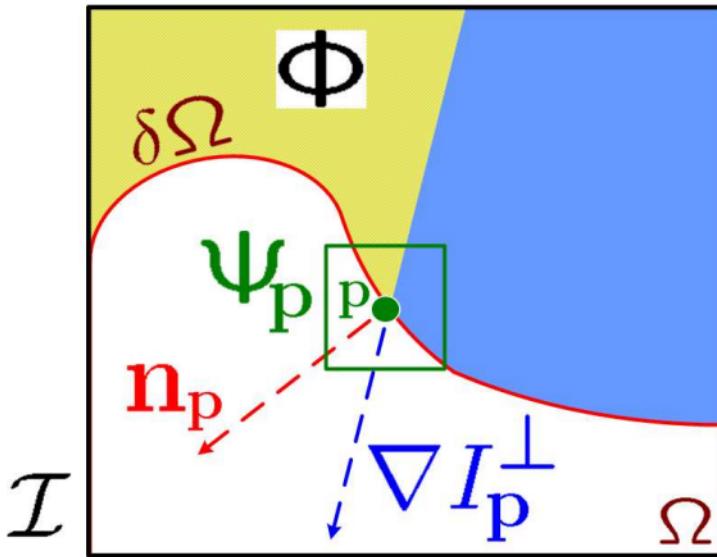


Figura 2.4: Având în vedere patch-ul  $\Psi_p$ ,  $n_p$  este normala conturului  $\varphi\omega$  din regiunea ţintă  $\omega$  și  $\nabla I_p^\perp$  este izofotul (direcția și intensitatea) la punctul  $p$ . Întreaga imagine este notată cu  $I$ .[RNC]

Unde  $|\Psi_p|$  este aria lui  $\Psi_p$ ,  $\alpha$  este un factor de normalizare (de ex.,  $\alpha = 255$  pentru o imagine tipică gri),  $n_p$  este un vector unitar ortogonal față  $\varphi\omega$  în punctul  $p$  și reprezintă operatorul ortogonal. Prioritatea  $P(p)$  este calculată pentru fiecare graniță patch, cu patch-uri distințe pentru fiecare pixel de la limita regiunea ţintă. În timpul inițializării, funcția  $C(p)$  este setată la  $C(p) = 0 \forall p \in \Omega$ , iar  $C(p) = 1 \forall p \in I - \Omega$ .

Termenul de încredere  $C(p)$  poate fi considerat ca o măsură a cantitatii de informație fiabila din jurul pixelului  $p$ . Intentia este de a umple mai întai acele petice care au mai mulți pixeli deja umpluți, cu preferință suplimentară acordată pixelilor care au fost completate devreme (sau care nu au făcut niciodată parte din regiunea ţintă). După cum va fi ilustrat în fig. 2.2, aceasta încorporează automat preferința față de anumite forme ale frontului de umplere. De exemplu, petice care includ colțuri și fire subțiri ale regiunii ţintă vor tinde să fie umplute primele, deoarece sunt înconjurate cu mai mulți pixeli ai imaginii originale. Aceste petice oferă informații mai fiabile cu care să se potrivească. Invers, petice la vârful „peninsulelor” de pixeli umpluți care ies în regiunea ţintă vor tinde să fie puse deoparte până când mai mulți pixeli din jur sunt completăți. La un nivel grosier, termenul  $C(p)$  impune aproximativ ordinea de umplere concentrică de dorit. Pe măsură ce umplerea continuă, pixelii din straturile exterioare ale regiunii ţintă vor avea tendința de a fi caracterizați de valori mai mari de încredere și, prin urmare, vor fi umpluți mai devreme; pixelii din centrul regiunii ţintă vor avea mai puține valori de încredere.

Termenul de date  $D(p)$  este o funcție a puterii izofotilor lovind frontul  $\varphi\omega$  la fiecare iterație. Acest termen sporește prioritatea unui patch în care „curge” un izofot. Acest factor este de importanță fundamentală în algoritmul nostru, deoarece

încurajează structuri liniare care urmează să fie sintetizate mai întâi și, prin urmare, propagate în siguranță în regiunea țintă.

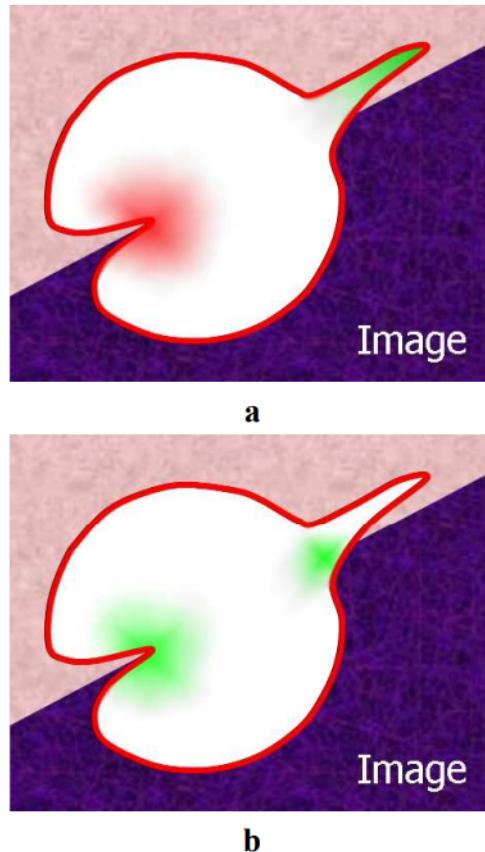


Figura 2.5: Efectele datelor și termenilor de încredere. (a) Termenul de încredere atribuie prioritate mare de umplere pentru apendicele care scot în evidență (în verde) și prioritate scăzută pentru cele îndreptate (în roșu), încercând astfel să se obțină o limită netedă și aproximativ circulară a țintei. (b) Termenul de date acordă prioritate mare pixelilor de pe continuarea structurilor de imagine (în verde) și are ca efect favorizarea anexelor de intrare în direcția structurilor de intrare. Combinarea celor doi termeni din Ec. lui P(p) produce echilibrul organic dorit între cele două efecte, unde este creșterea în interior a structurilor imaginii aplicat cu moderatie.[RNC]

## 2.4 Al doilea pas dintr-un algoritm de umplere

### 2.4.1 Difuzie(D)

Fie  $I_{xy}$  pixelul selectat pentru umplere cu coordonatele  $x$  și  $y$  din matricea imaginii,  $A_{n,n}$ ,  $n$  impar intreg, submatricea de dimensiune  $n$  și  $n$  din imagine cu centrul în coordonatele  $x$  și  $y$  și  $K_{n,n}$ ,  $k_{ij} \in K_{n,n}$ ,  $0 \leq k_{ij} \leq 1$  astfel incat  $\sum_{i=1}^n \sum_{j=1}^n k_{ij} = 1$ .

Pentru fiecare  $I_{xy}$  generam  $K'_{n,n} k'_{ij} \in K'_{n,n}$  unde  $k'_{ij} = 0$ , daca  $I_{x-((n-1)/2)+i, y-((n-1)/2)+j} \in \omega$  si  $k'_{ij} = k_{ij}$ , altfel.

Atunci

$$I_{xy} = \text{conv}(A_{n,n}, K'_{n,n}) / (\sum_{i=1}^n \sum_{j=1}^n k'_{ij})$$

Aceasta tehnica este destul de rapida si usor de implementat, insa daca  $\omega$  este mare, regiunea umpluta va fi blurata.

a	b	a
b	0	b
a	b	a

c	c	c
c	0	c
c	c	c

Figura 2.6:Kernel-urile de convoluție propuse de Oliveira [VB14]

#### 2.4.2 Folosind regiuni similare(S)

Aceasta tehnica presupune sa cautam pentru  $\Psi_p$  (regiunea din jurul unui pixel p)  $\Psi_q \in I - \Omega$  ( $I$  este imaginea), de aceleasi dimensiuni, astfel incat similaritatea lor sa fie maxima. Dupa care valorile pixelilor din  $\Psi_q$  sunt transmise in valorile pixelilor din  $\Psi_p$ .

Acest lucru este suficient pentru a realiza propagarea atât a structurii cât și informației de textură de la sursa I la regiunea țintă  $\Omega$ . De fapt, observăm că mai departe manipularea valorilor pixelilor (de exemplu, adăugarea de zgomot, netezire etc.) care nu depinde în mod explicit de statisticile sursei regiune, este mai probabil să degradeze similaritatea vizuală între regiunea umplută și regiunea sursă, decât pentru a o îmbunătăți.

## 2.5 Implementarea algoritmilor

Daca dorim sa implementam metodele precizate mai sus unele dintre ele sunt clare si usor de rezolvat, iar altele pot fi rezolvate in mai multe feluri. In acest sens voi detalia rezolvarea mea pentru partile abstracte din tehniciile de mai sus.

**Calcularea termenului de date  $D(p)$ .**

$$D(p) = (|\nabla I_p^\perp n_p|)/\alpha = \frac{|\nabla I_p^\perp| * |n_p|}{\alpha} * \cos(\phi)$$

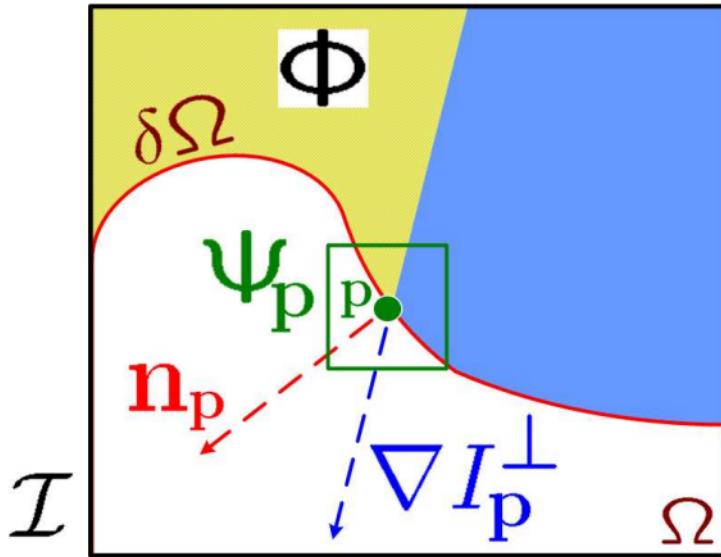


Figura 2.7:[RNC]

Din moment ce  $\alpha$  este un scalar ales pentru normalizare, putem observa ca ceea ce ne intereseaza defapt este unghiul dintre cei 2 vectori, care poate fi calculat cu formula:

$$\phi = \arctan\left(\frac{m1 - m2}{1 + m1 * m2}\right)$$

unde  $m1$  este panta dreptei formata de vectorul  $n_p$  si  $m2$  este panta dreptei formata de vectorul  $\nabla I_p^\perp$ .

Putem calcula ecuatia lui  $\varphi\Omega$  prin interpolare spline. Pentru a folosi interpolarea spline alegem cateva puncte  $q$  dintr-o regiune de  $nxn$  din jurul lui  $p$  de unde  $q \in \varphi\Omega$ . Odata ce stim ecuatia lui  $\varphi\Omega$  putem calcula panta  $m0$  a lui  $\varphi\Omega$  in punctul  $p$ . Din moment ce  $n_p$  este ortogonal pe  $\varphi\Omega$  il putem afla pe  $m1$ :

$$m1 = \frac{-1}{m0}$$

Filtrul Laplacian este folosit pentru a calcula derivatele secunde ale unei imagini, măsurând rata cu care se modifică primele derive. Aceasta determină dacă o modificare a valorilor pixelilor adiacenți este de la o margine sau o progresie continuă.

Kernel-urile de filtru laplacian conțin de obicei valori negative într-un model încrucișat, centrat în cadrul matricei. Colțurile sunt fie zero, fie valori pozitive. Valoarea centrală poate fi fie negativă, fie pozitivă. Următoarea matrice este un exemplu de nucleu 3x3 pentru un filtru Laplacian.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Figura 2.8:Exemplu de kernel Laplacian

Folosind kernel-ul de mai sus vom detecta muchiile din imagine unde este contrast de culoare. Apoi pentru a afla panta ecuației dreptei formate de vectorul  $\nabla I_p^\perp$  vom folosi tot interpolare spline pe pixeli q dintr-o regiune de  $n \times n$  din jurul lui p de unde  $q! = 0 \in L_{n,n}$ , unde  $L_{n,n}$  este o portiune din matricea cu detectia muchiilor. Odata ce stim ecuația m2 va fi derivată ei în punctul p.

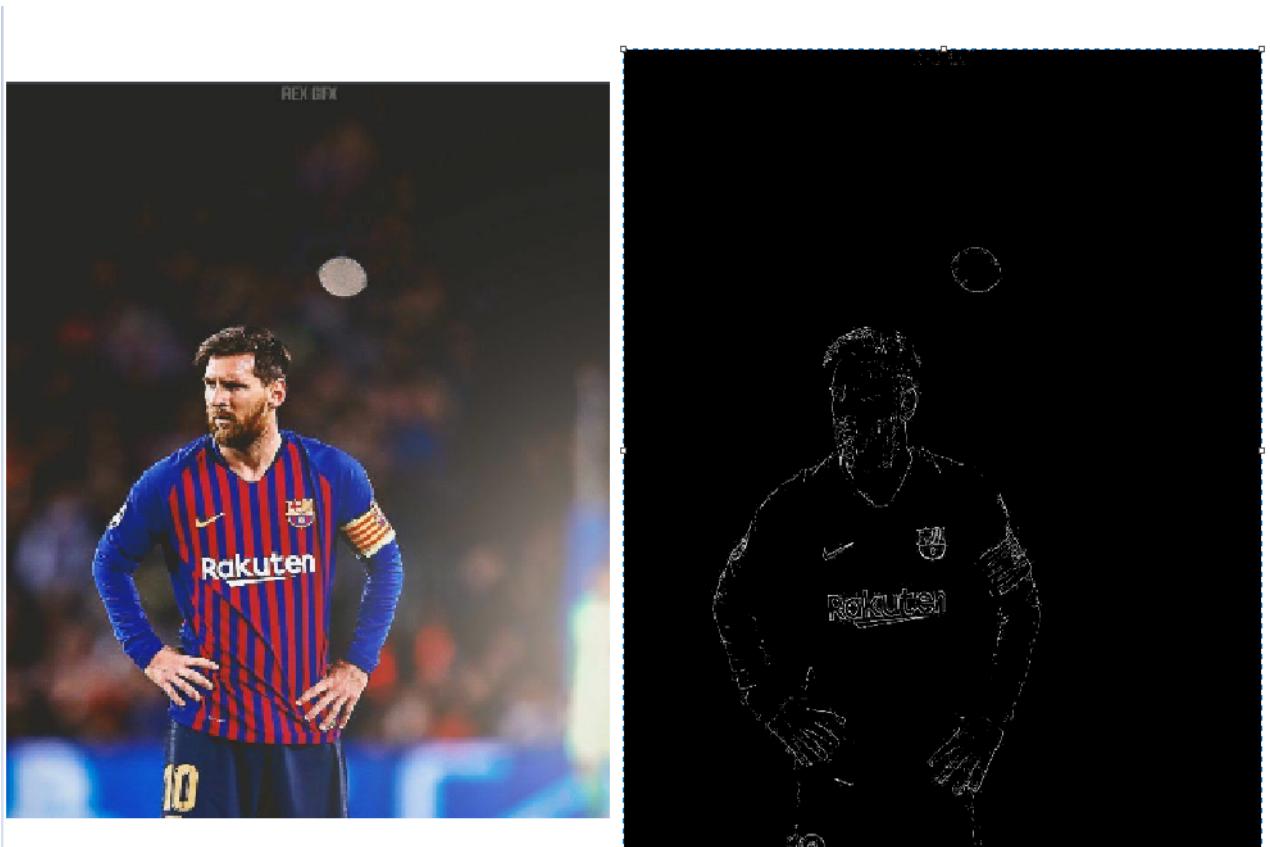


Figura 2.9:Exemplu de detectie al muchiilor

Acum ca stim m1 si m2 D(p) va deveni:

$$D(p) = \left| \cos\left(\arctan\left(\frac{m1 - m2}{1 + m1 * m2}\right)\right) \right|$$

### Calcularea similaritatilor dintre regiuni

In aplicarea metodei de umplere cu regiuni similare, voi folosi suma diferentelor patrate pentru a calcula similaritatea dintre fiecare regiune din imagine si regiunea ce trebuie umpluta.

Suma diferențelor pătrate (SSD) este o măsură de potrivire care se bazează pe diferențele de intensitate pixel cu pixel dintre cele două imagini. Aceasta calculează suma pătratului pentru produsul scăderii pixelilor dintre două imagini. Cu această măsură de similaritate, punctul de potrivire poate fi determinat luând în considerare locația valorii minime în matricele imaginii. În general, SSD utilizează direct formularea sumei erorii pătrate.

$$\iint_A (f - g)^2$$

Dacă ecuația aneroioara este convertită în formă digitală, atunci

$$SSD(i, j) = \sum_{i=0}^N \sum_{j=0}^M (f(i, j) - g(i + u, j + v))^2$$

unde M este dimensiunea liniilor din imaginea de referință și N este dimensiunea coloanei în timp ce u și v sunt variabile ce deplasează componenta de-a lungul direcției x și respectiv direcției y.

## 2.6 Algoritmi generati

Conform metodelor specificate mai sus voi crea 6 algoritmi astfel incat sa obtin toate combinatiile dintre ele pentru a vedea punctele slabe si punctele forte a metodelor discutate.

### Algoritmul LD

Algoritmul va alege toti pixeli din  $\varphi\omega$  liniar si va aplica o tehnica de umplere de difuzie(propaga informatia din pixelii vecini in el).

### Algoritmul LS

Algoritmul va alege toti pixeli din  $\varphi\omega$  liniar si va aplica o tehnica de umplere de selectare a regiunilor similare.

### **Algoritmul AD**

Algoritmul va alege la intamplare pixeli din  $\varphi\omega$  liniar si va aplica o tehnica de umplere de difuzie(propaga informatia din pixelii vecini in el).

### **Algoritmul AS**

Algoritmul va alege la intamplare pixeli din  $\varphi\omega$  liniar si va aplica o tehnica de umplere de selectare a regiunilor similare.

### **Algoritmul CDD**

Algoritmul va alege pixeli din  $\varphi\omega$  in functie de termenul de incredere si termenul de date din acel pixel si va aplica o tehnica de umplere de difuzie(propaga informația din pixelii vecini in el).

### **Algoritmul CDS**

Algoritmul va alege pixeli din  $\varphi\omega$  in functie de termenul de incredere si termenul de date din acel pixel si va aplica o tehnica de umplere de selectare a regiunilor similare.

Experimental si logic se poate observa ca algoritmii care folosesc tehnica de umplere de similaritate a regiunilor au mai mult succes cand masca este o regiunea mare si ampla. In cazul in care masca consista in mai multe linii subtiri efectul pare sa nu fie optim.



Figura 2.9.1: exemplu de umplere prin S



Figura 2.9.2: exemplu de umplere prin S

De asemenea algoritmii care folosesc tehnica de umplere de difuzie dau rezultate mai bune cand masca este mai subtire. In cazul in care regiunea este mai mare se creeaza un efect de blurare din cauza difuziei.

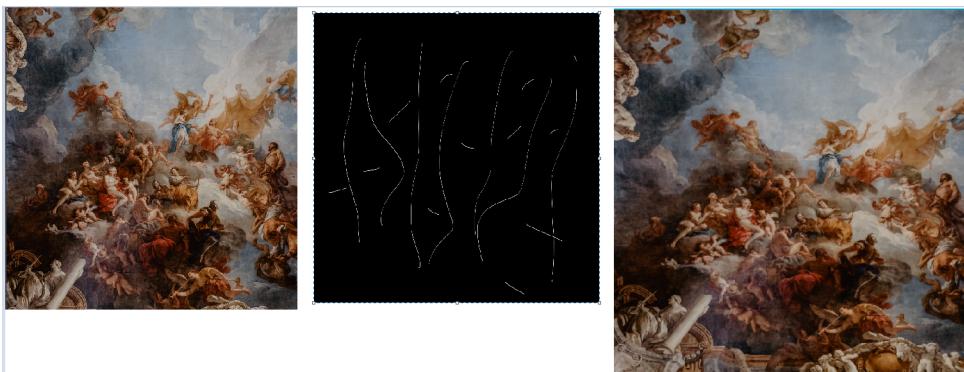


Figura 2.9.3: exemplu de umplere prin D



Figura 2.9.4: exemplu de umplere prin D

Conform celor evidențiate mai sus un algoritm bun de umplere a imaginilor ar trebui sa foloseasca o metoda de umplere de difuzie pentru masti subtiri si o metoda de umplere de inlocuire a regiunilor similare cand mastile au o grosime notabila.

## 2.7 Evaluarea Algoritmilor

Am luat 10 imagini oarecare si am creat 10 masti care sa surprinda mai multe situatii. Apoi pentru fiecare imagine am aplicat toti algoritmii prezentati mai sus cu fiecare din cele 10 masti. Pentru a verifica similaritatea intre imaginea initiala si imaginea rezultata dupa algoritmul de umplere voi folosi functia norm din OpenCV. Mai jos putem vedea rezultate:

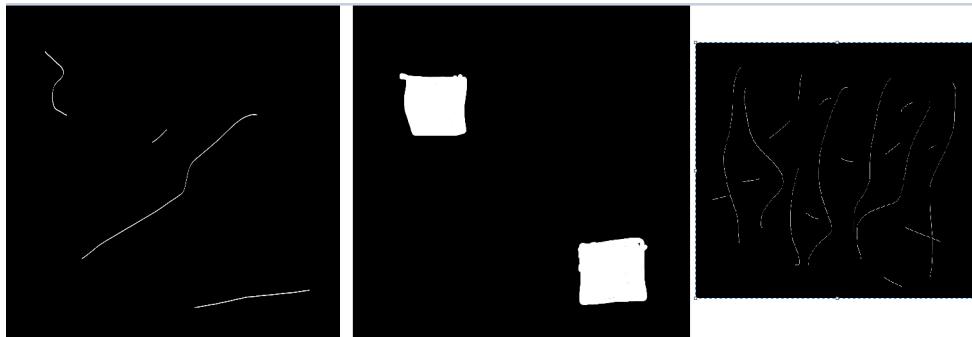


Figura 2.9.5:3 masti din cele 10 folosite pentru evaluateare

Algoritm	Timp(secunde)	Similaritate(%)
OpenCV	0.026279597282409667	97.83331131300278
LD	2.9688877415657045	99.19838849520018
LS	8.619754946231842	99.33276379133132
AD	2.951638197898865	99.39876743998852
AS	7.794377017021179	99.39488999567094
CDD	124.64162526130676	99.27163710469938
CDS	25.238749349117278	99.29382234759943

Timpul de executie si similaritatea calculata reprezinta timpul si similaritatea medie pe o imagine si o masca.

In urma acestor rezultate, se observa ca metodele care aleg pixelii la intamplare(A) au cel mai bun timp si cea mai buna similaritate, in medie. Algoritmii care folosesc CD au o similaritate care ar putea concura (cu A), mai ales daca un utilizator ar vrea sa foloseasca aceasta operatie o singura data, deoarece in algoritmii ce folosesc A este o sansa foarte mica, dar existenta, ca rezultatul sa fie mai rau ca rezultatul algoritmilor cu CD. Insa din cauza timpului imens de calculare din algoritmii CD pare ca cei mai viabili algoritmi sunt algoritmii AD si AS.

## 2.8 Algoritmul final

Conform ultimelor doua subcapitole am ajuns la concluzia ca cea mai buna metoda de alegere a pixelilor la inceput este aleatoriu si ca D si S trebuie folosite in functie de dimensiunea regiunilor separate care trebuie umplute.

Astfel algoritmul final va fi structurat astfel:

1. Se va detecta regiunea care trebuie umpluta  $\Omega$  si conturul ei  $\varphi\Omega$
2. Se va selecta aleatoriu un pixel de pe  $\varphi\Omega$

$$p = \text{rand}(q), q \in \varphi\Omega$$

3. Se verifica grosimea regiunii  $\Omega$  din jurul punctului p

$$t = \frac{\text{mask}(\Psi_p)}{|\Psi_p|} * 100\%$$

$$\text{mask}(\Psi_p) = \sum q, q \in \Psi_p, q \in \Omega$$

4. In functie de procentajul grosimii calculate aplicam ori tehnica de umplere de difuzie, ori cea de similaritate a regiunilor

$$p = \frac{\text{conv}(A_{n,n}, K'_{n,n})}{\sum_{i=1}^n \sum_{j=1}^n k'_{ij}}, t \geq \alpha$$

$$\Psi_q = \arg\min_{\Psi_q \in \phi} d(\Psi p, \Psi q), t < \alpha$$

5. Se repeta pasii 2.-4. pana cand se umple toata regiunea  $\Omega$

# Capitolul 3

## Fundamente teoretice inteligență artificială

### 3.1 Perceptron

#### 3.1.1 Neuron biologic

În creierul omenesc există un număr imens de neuroni, formând un număr imens de sinapse [Olt00]. Neuronii sunt celule diferențiate, care au obținut funcții specifice. Ei transmit, generează și recepționează impulsul nervos. [Olt00] Neuronii sunt alcătuși din corp celular (soma), dendrite și axon. Dendritele sunt prelungiri scurte ale neuronului, care primesc informații de la ceilalți neuroni. Corpul celular procesează informația primită de la dendrite, iar axonul este cel folosit de către neuron pentru a transmite informația mai departe. Sinaptele reprezintă conexiuni între un axon și dendritele altor neuroni. În figura 1.1 este reprezentat un neuron biologic.

Neuron

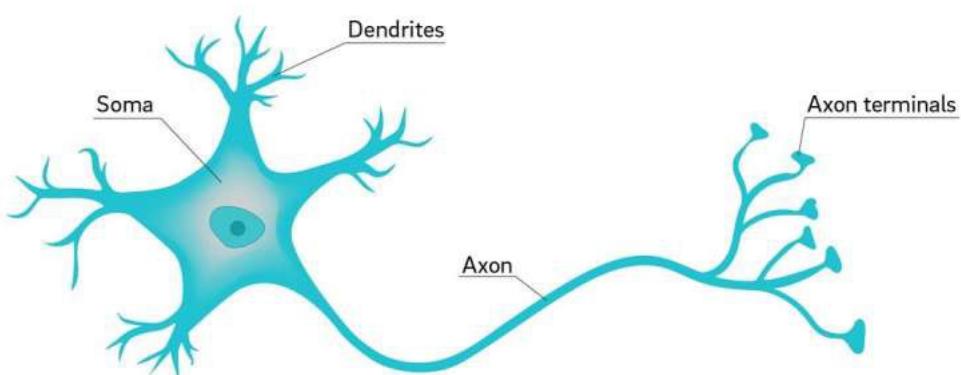


Figura 1.1: Neuron bilogic [Roo19]

### 3.1.2 Neuron artificial

Neuronul biologic servește că sursă de modelare pentru neuronul artificial. Astfel, pentru stocarea informațiilor, se folosesc ponderi sinaptice, care reprezintă conexiunile dintre neuroni. După ce se procesează local semnalul de intrare (prin înmulțirea valorilor de intrare cu ponderile sinaptice corespunzătoare), se realizează o sumă a rezultatelor obținute (proces asemănător cu cel produs de neuronul biologic în soma). Dacă rezultatul depășește un anumit prag, informația este transmisă mai departe. O reprezentare generală a neuronului artificial poate fi văzută în figura 1.2

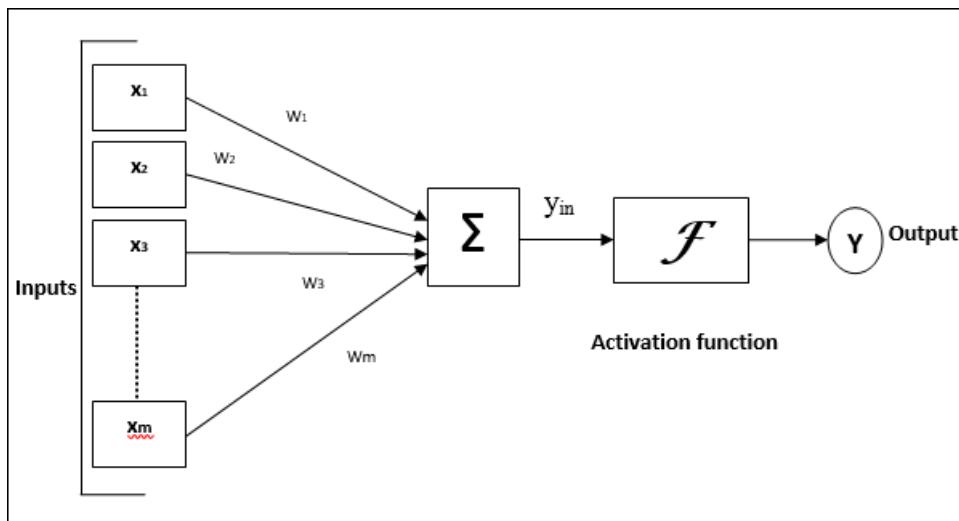


Figura 1.2: Neuron artificial [ANN]

Pentru modelul general prezentat mai sus,  $Y_{in}$  poate fi calculat astfel:

$$Y_{in} = x_1 * w_1 + x_2 * w_2 + x_3 * w_3 + \dots + x_m * w_m$$

$$Y_{in} = \sum_i^m x_i * w_i$$

Rezultatul final Y se calculează aplicând peste  $Y_{in}$  funcția de activare:

$$Y = F(Y_{in})$$

### 3.1.3 Perceptron

Perceptronul este cel mai simplu model de rețea neuronală, fiind format dintr-un singur neuron artificial. Acesta a fost inventat în anul 1958 de către psihologul Frank Rosenblatt [Ros57], iar inițial a fost menit să fie o mașinărie, nu un algoritm.

În domeniul inteligenței artificiale, perceptronul este un algoritm pentru învățare supervizată a clasificatorilor binari. Un clasificator binar este o funcție care decide dacă o intrare (reprezentată printr-un vector de numere) aparține sau nu unei clase.[FS99] Perceptronul folosește că funcție de activare funcția prag discutată anterior. Principiul de bază constă în “hrănierea” rețelei cu date de antrenament, una câte una. Erorile de clasificare conduc la actualizarea ponderilor. Algoritmul de învățare nu se va termină dacă setul de date nu este unul separabil liniar.

În figura 1.3 se poate observă atât un set de date liniar (există o dreaptă în plan care separă cele două categorii) cât și un set care nu este liniar (nu poate fi împărțit graficul în două categorii trasând o dreaptă).

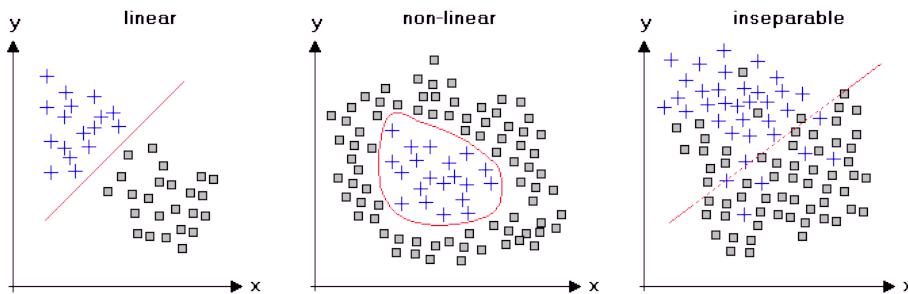


Figura 1.3: [Loh]

## 3.2 Rețele neuronale

### 3.2.1 Ce este o RN?

Rețelele neuronale, cunoscute și sub denumirea de rețele neuronale artificiale (ANN) sau rețele neuronale simulate (SNN), sunt un subset al învățării automate și se află în centrul algoritmilor de învățare profundă. Numele și structura lor sunt inspirate de creierul uman, imitând modul în care neuroni biologici se semnalează unul altuia.

Rețelele neuronale artificiale (ANN) sunt compuse din straturi de noduri, care conțin un strat de intrare, unul sau mai multe straturi ascunse și un strat de ieșire. Fiecare nod, sau neuron artificial, se conectează la altul și are asociate o greutate și un prag. [NNI]

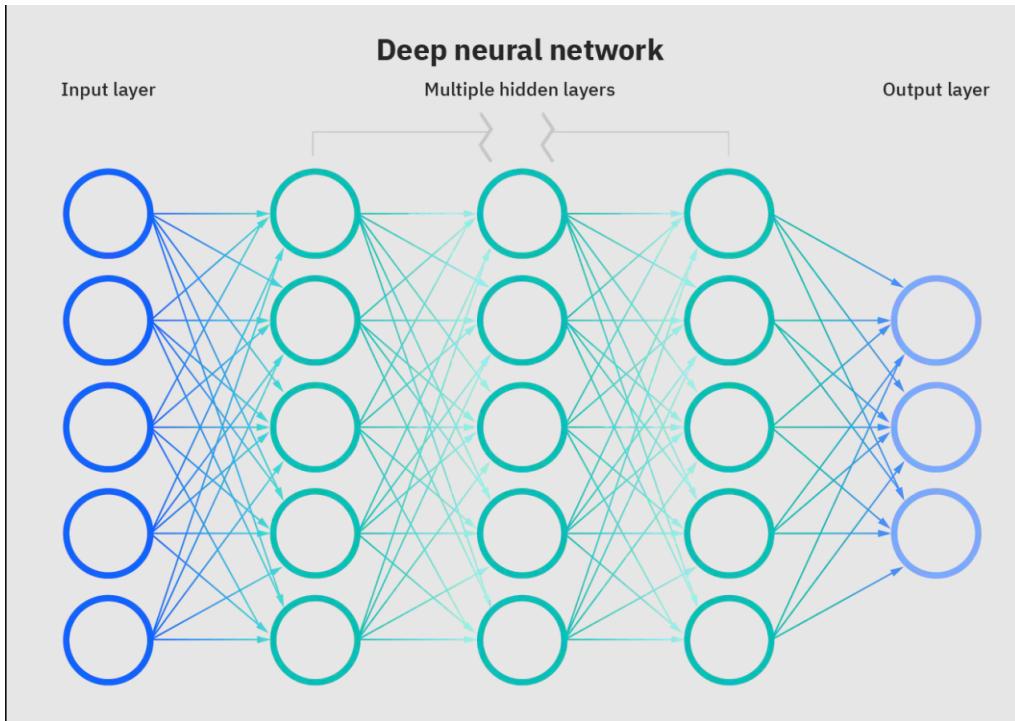


Figura 1.4: Rețea neuronală[NNI]

### 3.2.2 Retropagarea erorii(backpropagation)

Retropagarea erorii este esența antrenamentului rețelei neuronale. Este metodă de reglare fină a greutăților unei rețele neuronale pe baza ratei de eroare obținute în epoca anterioară (adică, iteratie). Reglarea corectă a greutăților va permite să reducem ratele de eroare și să facem modelul fiabil prin creșterea generalizării acestuia.

Retropagarea erorii în rețeaua neuronală este o formă scurtă pentru „propagarea înapoi a erorilor”. Este o metodă standard de antrenament a rețelelor neuronale artificiale. Această metodă ajută la calcularea gradientului unei funcții de pierdere în raport cu toate greutățile din rețea.

### 3.2.3 Funcția de activare

Funcția de activare decide dacă un neuron ar trebui activat sau nu prin calcularea sumei ponderate și adăugând în continuare bias cu această. Scopul funcției de activare este de a introduce neliniaritatea în ieșirea unui neuron.

Explicație:- Știm, rețeaua neuronală are neuroni care funcționează în corespondență între greutate, bias și funcția de activare a acestora. Într-o rețea neuronală, am actualiza ponderile și biasurile neuronilor pe baza erorii de la ieșire. Acest proces este cunoscut sub numele de retropagarea erorii(backpropagation). Funcțiile de activare fac posibilă retropagarea erorii, deoarece gradienții sunt furnizați împreună cu eroarea pentru a actualiza ponderile și părtinirile.

De ce avem nevoie de funcții de activare neliniară?:- O rețea neuronală fără funcție de activare este în esență doar un model de regresie liniară. Funcția de activare face transformarea neliniară la intrare, făcându-l capabil să învețe și să execute sarcini mai complexe.

### **3.2.4 Funcția cost**

O funcție de cost este un parametru important care determină cât de bine funcționează un model de învățare automată pentru un anumit set de date. Aceasta calculează diferența dintre valoarea așteptată și valoarea prezisă și o reprezintă că un singur număr real.

În învățarea automată, odată ce ne antrenăm modelul, vrem să vedem cât de bine funcționează modelul nostru. Deși există diverse funcții de precizie care spun cum funcționează modelul dvs., dar nu vă vor oferi informații pentru a le îmbunătăți. Deci, avem nevoie de o funcție care să găsească când modelul este cel mai precis, găsind locul dintre modelul subantrenat și supraantrenat.

### 3.2.5 Tipologii

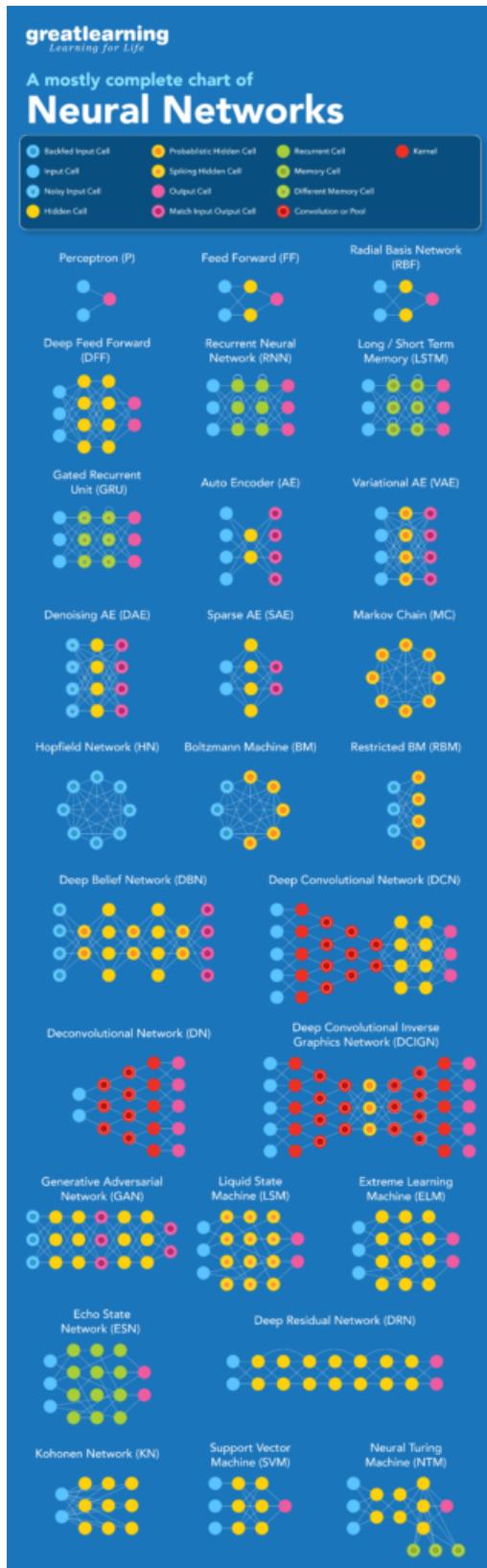


Figura 1.5: Diverse tipuri de ANN[NNM]

## 3.3 Rețele neuronale convolutionale

### 3.3.1 Ce este o RNC?

O rețea neuronală convoluțională (ConvNet/CNN) este un algoritm de învățare profundă care poate prelua o imagine de intrare, poate atribui importanță (greutăți și bias-uri învățate) diferitelor aspecte/obiecte din imagine și poate să le diferențieze unul de celălalt. Preprocesarea necesară într-un ConvNet este mult mai mică în comparație cu alți algoritmi de clasificare. În timp ce în metodele primitive, filtrele sunt proiectate manual, cu suficient antrenament, ConvNet-urile au capacitatea de a învăța aceste filtre/caracteristici. [RNC]

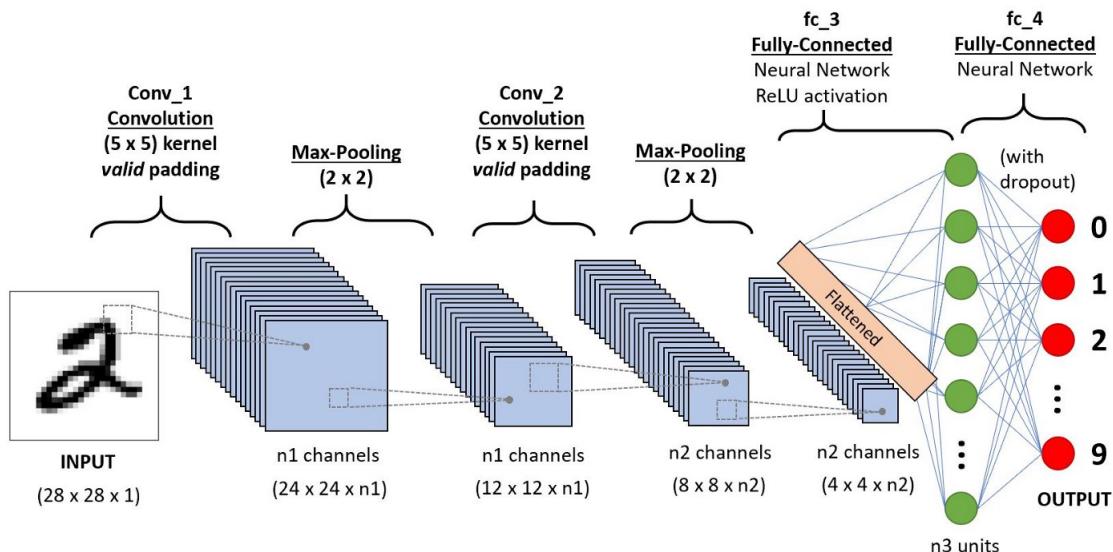


Figura 1.6: Secvență de RNC pentru a clasifica litere scrise de mâna[RNC]

### 3.3.2 Stratul convolutional

Elementul implicat în efectuarea operației de conoluție în prima parte a unui strat convolutional se numește Kernel/Filter. The Kernel shifts n times(n - size of Kernel), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering. Filtrul se deplasează spre dreapta cu o anumită valoare de pas până analizează lățimea completă. Mergând mai departe, sare în jos la începutul (stânga) al imaginii cu aceeași valoare de pas și repetă procesul până când întreaga imagine este străbătută.

În cazul imaginilor cu mai multe canale (de exemplu, RGB), Kernel-ul are aceeași adâncime ca cea a imaginii de intrare. Înmulțirea matricei este efectuată între  $K_n$  și stivă  $\hat{I} = ([I_1, I_2]; [I_2, I_3]; [I_3, I_4])$  și toate rezultatele sunt însumate cu bias-ul pentru a ne oferi o ieșire de caracteristică de canal de adâncime 1.

Obiectivul operațiunii de convoluție este extragerea caracteristicilor de nivel înalt, cum ar fi marginile, din imaginea de intrare. ConvNet-urile nu trebuie să fie limitate la un singur strat convoluțional. În mod convențional, primul ConvLayer este responsabil pentru captarea caracteristicilor de nivel scăzut, cum ar fi marginile, culoarea, orientarea gradientului etc. Cu straturi adăugate, arhitectură se adaptează și la caracteristicile de nivel înalt, oferindu-ne o rețea care are o înțelegere sănătoasă a imaginii din setul de date, similar cu cum am face noi.

Există două tipuri de rezultate ale operației - unul în care caracteristica convolută este redusă în dimensionalitate în comparație cu intrarea, iar celălalt în care dimensionalitatea este fie crescută, fie rămâne aceeași. Acest lucru se face prin aplicarea Valid Padding în cazul primei, sau Same Padding în cazul celui din urmă. [RNC]

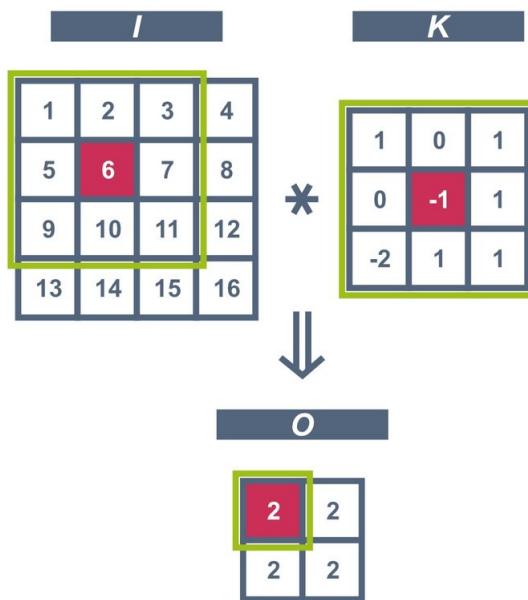


Figura 1.7: Exemplu convoluție[EC2]

### 3.3.3 Stratul de agregare(pooling layer)

Similar cu Stratul Convoluțional, Stratul Pooling este responsabil pentru reducerea dimensiunii spațiale a Caracteristicii Convoluționale. Această este pentru a reduce puterea de calcul necesară procesării datelor prin reducerea dimensionalității. Mai mult, este util pentru extragerea caracteristicilor dominante care sunt invariante rotационale și pozitionale, menținând astfel procesul de antrenare eficientă a modelului.

Există două tipuri de Pooling: Pooling maxim și Pooling mediu. Max Pooling returnează valoarea maximă din portiunea imaginii acoperită de Kernel. Pe de altă

parte, Average Pooling returnează media tuturor valorilor din portiunea imaginii acoperită de Kernel.

Max Pooling funcționează și ca un suprimator de zgomot. Îndepărtează complet activările zgomotoase și efectuează, de asemenea, eliminarea zgomotului împreună cu reducerea dimensionalității. Pe de altă parte, Average Pooling realizează pur și simplu reducerea dimensionalității ca mecanism de suprimare a zgomotului. Prin urmare, putem spune că Max Pooling are performanțe mult mai bune decât Average Pooling. [RNC]

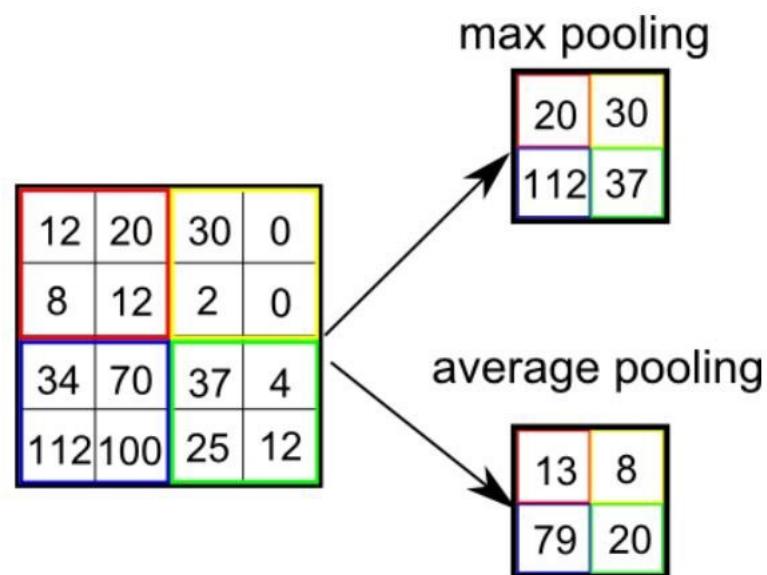
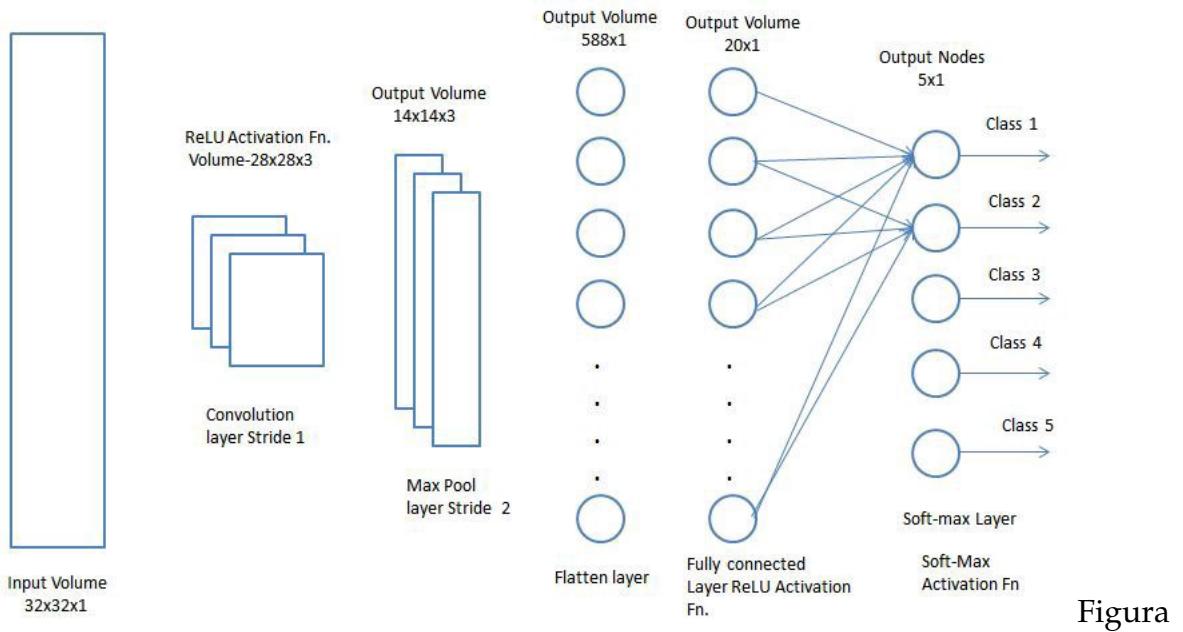


Figura 1.8: Max pooling și average pooling[RNC]

### 3.3.4 Stratul complet conectat

Adăugarea unui strat complet conectat este o modalitate (de obicei) ieftină de a învăța combinații neliniare ale caracteristicilor de nivel înalt, aşa cum sunt reprezentate de rezultatul stratului convolutional. Stratul Fully-Connected învăță o funcție posibil neliniară în acel spațiu. [RNC]



Figura

1.9: Stratul complet conectat[RNC]

### 3.3.5 Stratul de abandon(dropout layer)

Stratul de abandon este o mască care anulează contribuția unor neuroni la următorul strat și îl lasă nemodificat pe toti ceilalți. Putem aplica un strat Dropout vectorului de intrare, caz în care anulează unele dintre caracteristicile sale; dar îl putem aplica și pe un strat ascuns, caz în care anulează niște neuroni ascunși.

Straturile de abandon sunt importante în antrenamentul CNN-urilor, deoarece previn supraadaptarea datelor de antrenament. Dacă acestea nu sunt prezente, primul lot de mostre de antrenament influențează învățarea într-o manieră disproportional de mare. Acest lucru, la rândul său, ar împiedica învățarea caracteristicilor care apar numai în mostrele sau loturile ulterioare: [CNN]

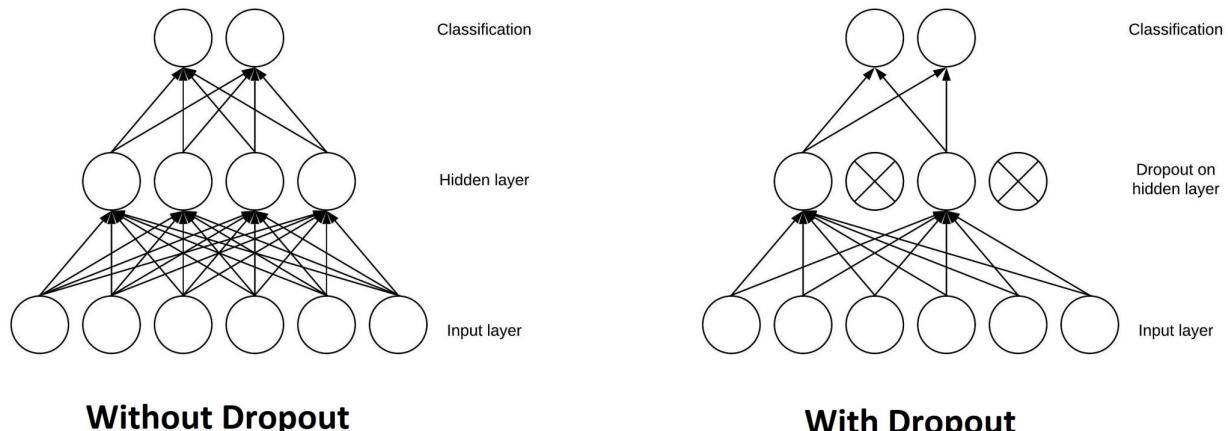


Figura 3.0[CNN]

Să presupunem că arătăm zece imagini cu un cerc, succesiv, unui CNN în timpul antrenamentului. CNN nu va află că liniile drepte există; în consecință, va fi destul de confuz dacă mai târziu îi vom arăta o imagine a unui pătrat. Putem preveni aceste cazuri adăugând straturi Dropout la arhitectură rețelei, pentru a preveni supraadaptarea.

## 3.4 Mask R-CNN

### 3.4.1 Segmentarea instanțelor

Ce este segmentarea instanțelor și prin ce diferă de segmentarea semantică?

Segmentarea semantică detectează toate obiectele prezente într-o imagine la nivel de pixel. Afisează regiuni cu clase sau obiecte diferite

Segmentarea semantică grupează pixelii într-un mod semnificativ din punct de vedere semantic. Pixelii care aparțin unei persoane, drum, clădire, gard, bicicletă, mașini sau copaci sunt grupați separat.



Figura 3.1: Segmentare semantică[Kha]

Segmentarea instanțelor este identificarea fiecărei instanțe de obiect pentru fiecare obiect cunoscut dintr-o imagine.

Segmentarea instanțelor atribuie o etichetă fiecărui pixel al imaginii. Este folosit pentru sarcini precum numărarea numărului de obiecte



Figura 3.2: Segmentarea instanțelor[Kha]

Segmentarea instanțelor necesită

- Detectarea obiectelor aplicată tuturor obiectelor dintr-o imagine. Aici scopul este de a clasifica obiecte individuale și de a localiza fiecare instanță de obiect folosind o casetă de delimitare
- Segmentarea fiecărei instanțe. Aici scopul este de a clasifica fiecare pixel într-un set fix de categorii fără a diferenția instanțele de obiect

### 3.4.2 Cum funcționează un Mask R-CNN?

Modelul Mask R-CNN este împărțit în două părți

- Reteaua de propuneri regionale (Region proposal network) pentru a propune casete de delimitare a obiectelor candidate.
- Clasificator binar de mască pentru a genera mască pentru fiecare clasă

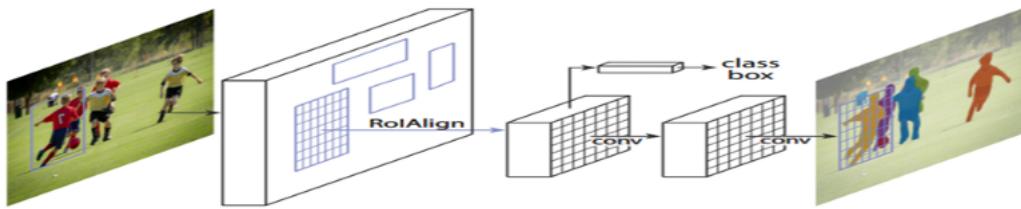


Figura 3.3: Mask R-CNN[Kha]

- Imaginea este rulată prin CNN pentru a genera hărțile caracteristicilor.
- Region Proposal Network (RPN) utilizează un CNN pentru a genera regiunile multiple de interes (RoI) folosind un clasificator binar ușor. Face acest lucru folosind 9 casete de ancore peste imagine.
- Clasificatorul returnează scorurile obiect/fără obiect. Suprimarea non-max este aplicată ancorelor cu scor de obiectitate ridicat
- Rețeaua RoI Align scoate mai multe casete de delimitare, mai degrabă decât una singură definită și le deformează într-o dimensiune fixă.
- Caracteristicile deformate sunt apoi introduse în straturi complet conectate pentru a face clasificarea folosind softmax, iar predictia cutiei de delimitare este rafinată în continuare folosind modelul de regresie
- Caracteristicile deformate sunt, de asemenea, introduse în clasificatorul Mask, care constă din două CNN-uri pentru a scoate o mască binară pentru fiecare RoI. Mask Classifier permite rețelei să genereze măști pentru fiecare clasă fără competiție între clase

## Mask RCNN

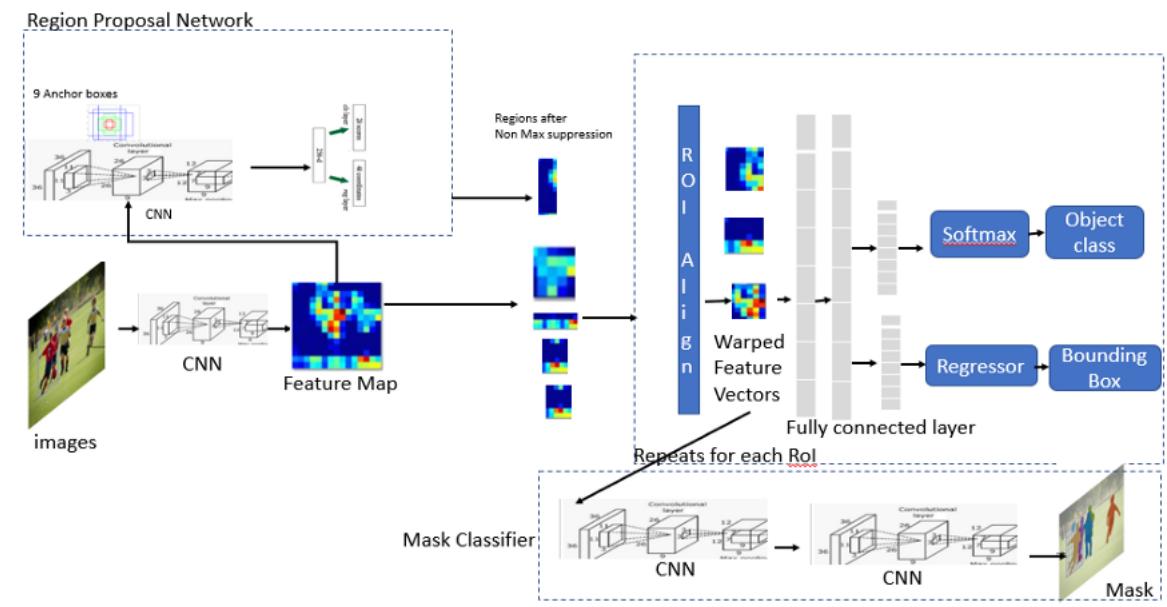


Figura 3.4: Mask R-CNN[Kha]

# Capitolul 4

## Tehnologiile din aplicație

### 4.1 Tehnologi client

#### 4.1.1 HTML

HTML (HyperText Markup Language) este codul care este folosit pentru a structura o pagină web și conținutul acesteia. De exemplu, conținutul ar putea fi structurat într-un set de paragrafe, o listă de puncte cu marcatori sau folosind imagini și tabele de date.

##### Ce este HTML?

HTML este un limbaj de marcare care definește structura conținutului. HTML constă dintr-o serie de elemente, pe care le utilizați pentru a încadra sau încheia diferite părți ale conținutului pentru a face că acesta să apară într-un anumit fel sau să acționeze într-un anumit fel. Etichetele care includ un cuvânt sau o imagine pot crea un hyperlink către altă parte, pot pune cuvinte în cursive, pot face fontul mai mare sau mai mic și aşa mai departe. [JTM]

#### 4.1.2 CSS

Cascading Style Sheets (CSS) este un style sheet language folosit pentru a descrie prezentarea unui document scris într-un limbaj de marcare precum HTML. CSS este o tehnologie de bază a World Wide Web, alături de HTML și JavaScript.

CSS este conceput pentru a permite separarea prezentării și a conținutului, inclusiv aspectul, culorile și fonturile. Această separare poate îmbunătăți accesibilitatea conținutului; oferă mai multă flexibilitate și control în specificarea caracteristicilor de prezentare; permiteți mai multe pagini web să partajeze formatarea prin specificarea CSS-ului relevant într-un fișier .css separat, ceea ce reduce complexitatea și repetarea conținutului structural; și permiteți că fișierul .css să fie stocat în ca-

che pentru a îmbunătăti viteză de încărcare a paginii între paginile care partajează fișierul și formatarea acestuia.

Separarea formatării și a conținutului face, de asemenea, posibilă prezentarea aceliasi pagini de marcă în stiluri diferite pentru diferite metode de randare, cum ar fi pe ecran, în tipărire, prin voce (prin browser sau cititor de ecran bazat pe vorbire) și dispozitive tactile pe baza Braille. CSS are și reguli pentru formatare alternativă dacă conținutul este accesat pe un dispozitiv mobil. [CSS]

### 4.1.3 JavaScript

JavaScript este un limbaj de programare care este una dintre tehnologiile de bază ale World Wide Web, alături de HTML și CSS. Peste 97% dintre site-uri web folosesc JavaScript pe partea client pentru comportamentul paginii web, încorporând adesea biblioteci terțe. Toate browserele web majore au un motor JavaScript dedicat pentru a execută codul pe dispozitivele utilizatorilor.

JavaScript este un limbaj la nivel înalt, adesea compilat la timp, care se conformează standardului ECMAScript. Are tastare dinamică, orientare pe obiecte bazată pe prototip și funcții de primă clasă. Este multi-paradigmă, care acceptă stiluri de programare bazate pe evenimente, funcționale și imperative. Are interfețe de programare a aplicațiilor (API) pentru lucrul cu text, date, expresii regulate, structuri de date standard și Modelul obiectului documentului (DOM).

Standardul ECMAScript nu include nicio intrare/iesire (I/O), cum ar fi facilități de rețea, stocare sau grafică. În practică, browserul web sau alt sistem de rulare furnizează API-uri JavaScript pentru I/O.

Motoarele JavaScript au fost utilizate inițial numai în browserele web, dar acum sunt componente de bază ale unor servere și ale unei varietăți de aplicații. Cel mai popular sistem de rulare pentru această utilizare este Node.js.

Deși Java și JavaScript sunt similare că nume, sintaxă și bibliotecile standard respective, cele două limbi sunt distințe și diferă foarte mult în design. [JS2]

### 4.1.4 jQuery

jQuery este un software gratuit, cu sursă deschisă, care utilizează licență permisivă MIT. Din mai 2019, jQuery este folosit de 73% dintre cele mai populare 10 milioane de site-uri web. Analiză web indică faptul că este cea mai răspândită bibliotecă JavaScript cu o marjă mare, având de cel puțin 3 până la 4 ori mai multă utilizare decât orice altă bibliotecă JavaScript.

Sintaxa jQuery este o bibliotecă JavaScript concepută pentru a facilita navigarea într-un document, selectarea elementelor DOM, crearea de animații, gestionarea evenimentelor și dezvoltarea aplicațiilor Ajax. jQuery oferă, de asemenea,

capabilități pentru dezvoltatori de a crea pluginuri pe partea superioară a bibliotecii JavaScript. Acest lucru le permite dezvoltatorilor să creeze abstracții pentru interacțiune și animație la nivel scăzut, efecte avansate și widget-uri la nivel înalt, cu teme. Abordarea modulară a bibliotecii jQuery permite crearea de pagini web dinamice și de aplicații web puternice. [JQ2]

## 4.2 Tehnologi Server

### 4.2.1 Python

#### Introducere

Limbajul Python este incredibil de ușor de folosit și de învățat pentru noii începători și noii veniți. Limbajul python este unul dintre cele mai accesibile limbaje de programare disponibile deoarece are sintaxă simplificată și nu complicată, ceea ce pune mai mult accent pe limbajul natural. Datorită ușurinței sale de învățare și utilizare, codurile Python pot fi scrise și executate cu ușurință mult mai rapid decât alte limbaje de programare.

Când Guido van Rossum crea python în anii 1980, s-a asigurat că îl proiectează pentru a fi un limbaj de uz general. Unul dintre principalele motive pentru popularitatea lui python ar fi simplitatea să în sintaxă, astfel încât să poată fi citit și înțeles cu ușurință chiar și de către dezvoltatorii amatori.

De asemenea, se poate experimenta rapid prin schimbarea codului de bază al lui python, deoarece este un limbaj interpretat, ceea ce îl face și mai popular printre toate tipurile de dezvoltatori.

#### Sute de librari și framework-uri

Datorită sponsorizării sale corporative și a comunității mari de susținere a lui python, python are biblioteci excelente pe care le puteți folosi pentru a selecta și a economisi timp și efort în ciclul inițial de dezvoltare. Există, de asemenea, o multime de servicii media în cloud care oferă suport pe mai multe platforme prin instrumente asemănătoare bibliotecii, care pot fi extrem de benefice.

Bibliotecile cu un accent specific sunt, de asemenea, disponibile, cum ar fi nltk pentru procesarea limbajului natural sau scikit-learn pentru aplicațiile de învățare automată.

Există multe cadre și biblioteci sunt disponibile pentru limbajul Python, cum ar fi:

- matplotlib pentru trasarea diagramelor și graficelor

- SciPy pentru aplicații de inginerie, știință și matematică
- BeautifulSoup pentru analiză HTML și XML
- NumPy pentru calcul științific
- Django pentru dezvoltare web pe partea de server

### **Big dată, Machine Learning și Cloud Computing**

Cloud Computing, Machine Learning și Big Dată sunt unele dintre cele mai populare tendințe din lumea informatică în acest moment, ceea ce ajută multe organizații să-și transforme și să-și îmbunătățească procesele și fluxurile de lucru.

Limbajul Python este al doilea cel mai popular instrument folosit după limbajul R pentru știință datelor și analiză. Multe multe sarcini de lucru de procesare a datelor din organizație sunt alimentate numai de limbajul Python. Cea mai mare parte a cercetării și dezvoltării se desfășoară în limbajul Python, datorită numeroaselor aplicații, inclusiv ușurința de a analiză și organiza datele utilizabile.

Nu numai asta, ci și sute de biblioteci Python sunt folosite în mii de proiecte de învățare automată în fiecare zi, cum ar fi TensorFlow pentru rețele neuronale și OpenCV pentru viziune computerizată etc.

#### **4.2.2 Flask**

##### **Ce este Flask?**

Flask este un cadru de aplicație web scris în Python. A fost dezvoltat de Armin Ronacher, care a condus o echipă de pasionați internaționali de Python numită Pocco. Flask se bazează pe setul de instrumente Werkzeug WSGI și pe motorul de şabloane Jinja2. Ambele sunt proiecte Pocco.

##### **De ce este Flask un framework web bun?**

Spre deosebire de framework-ul Django, Flask este foarte Pythonic. În plus, este foarte explicit, ceea ce crește lizibilitatea.

#### **4.2.3 OpenCV**

OpenCV (Open Source Computer Vision Library) este o bibliotecă de software open source pentru viziune pe computer și învățare automată. OpenCV a fost construit pentru a oferi o infrastructură comună pentru aplicațiile de viziune computerizată și pentru a accelera utilizarea percepției mașinii în produsele comerciale. Fiind un

produs cu licență BSD, OpenCV facilitează utilizarea și modificarea codului pentru companii.

Biblioteca are peste 2500 de algoritmi optimizați, care include un set cuprinzător de algoritmi de viziune computerizată și de învățare automată atât clasici, cât și de ultimă generație. Acești algoritmi pot fi folosiți pentru a detecta și recunoaște fețe, identifica obiecte, clasifică acțiunile umane în videoclipuri, urmăriți mișările camelei, urmăriți obiectele în mișcare, extrageți modele 3D de obiecte, produceti nori de puncte 3D din camere stereo, uniți imagini împreună pentru a produce o rezoluție înaltă. Imaginea unei scene întregi, găsiți imagini similare dintr-o bază de date de imagini, eliminați ochii roșii din imaginile realizate cu ajutorul blițului, urmăriți mișările ochilor, recunoașteți peisajul și stabiliți markeri pentru a-l suprapune cu realitatea augmentată etc. OpenCV are peste 47 de mii de utilizatori comunitate și numărul estimat de descărcări depășește 18 milioane. Biblioteca este utilizată pe scară largă în companii, grupuri de cercetare și de către organismele guvernamentale.

Alături de companii bine stabilite precum Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota care folosesc biblioteca, există multe startup-uri precum Applied Minds, VideoSurf și Zeitera, care folosesc pe scară largă OpenCV. Utilizările deschise de OpenCV cuprind de la împreunarea imaginilor streetview, detectarea intruziunilor în videoclipurile de supraveghere în Israel, monitorizarea echipamentelor miniere în China, ajutarea robotilor să navigheze și să ridice obiecte de la Willow Garage, detectarea accidentelor de înecare a piscinei în Europa, rularea artei interactive în Spania și New York, verificând pistele pentru reziduuri în Turcia, inspectând etichetele produselor din fabrici din întreaga lume până la detectarea rapidă a feței în Japonia. [OCVa]

### **Algoritmi pentru restaurarea imaginilor**

OpenCV implementează 2 algoritmi de restaurare a imaginilor:

- Primul algoritm se bazează pe lucrarea \*\* „An Image Inpainting Technique Based on the Fast Marching Method”\*\* de Alexandru Telea în 2004. Se bazează pe Fast Marching Method. Luati în considerare o regiune din imagine care urmează să fie pictată. Algoritmul începe de la granița acestei regiuni și merge în interiorul regiunii umplând treptat totul în graniță. Este nevoie de un mic cartier în jurul pixelului din cartier pentru a fi vopsit. Acest pixel este înlocuit cu suma ponderată normalizată a tuturor pixelilor cunoscuți din vecinătate. Alegera greutăților este o chestiune importantă. Mai multă pondere este acordată acelor pixeli care se află în apropierea punctului, aproape de normalul graniței și celor aflați pe contururile graniței. Odată ce un pixel este vopsit, acesta

se mută la următorul cel mai apropiat pixel folosind metodă Fast Marching. FMM se asigură că acei pixeli din apropierea pixelilor cunoscuți sunt pictați mai întâi, astfel încât să funcționeze că o operație euristică manuală. [OCVb]

- Al doilea algoritm se bazează pe lucrarea \*\* „Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting”\*\* de Bertalmio, Marcelo, Andrea L. Bertozzi și Guillermo Sapiro în 2001. Acest algoritm se bazează pe dinamică fluidelor și utilizează ecuații cu diferențe partiale. Principiul de bază este eurizitic. Mai întâi se deplasează de-a lungul marginilor de la regiuni cunoscute la regiuni necunoscute (deoarece marginile sunt menite să fie continue). Continuă izofotele (liniile care unesc puncte cu aceeași intensitate, la fel cum contururile unesc puncte cu aceeași altitudine) în timp ce potrivește vectorii de gradient la limita regiunii de pictură. Pentru această se folosesc unele metode din dinamică fluidelor. Odată ce sunt obținute, culoarea este umplută pentru a reduce variația minimă în acea zonă. [OCVb]

#### 4.2.4 Numpy

NumPy este pachetul fundamental pentru calculul științific în Python. Este o bibliotecă Python care oferă un obiect matrice multidimensional, diverse obiecte derivate (cum ar fi matrice și matrice mascate) și o gamă largă de rutine pentru operații rapide pe matrice, inclusiv matematice, logice, manipulare a formelor, sortare, selecțare, I/O, transformate Fourier discrete, algebră liniară de bază, operații statistice de bază, simulare aleatorie și multe altele.

La baza pachetului NumPy se află obiectul ndarray. Această încapsulează rețele n-dimensionale de tipuri de date omogene, multe operații fiind efectuate în cod compilat pentru performanță. Există câteva diferențe importante între tablourile NumPy și secvențele standard Python:

- Matricele NumPy au o dimensiune fixă la creare, spre deosebire de listele Python (care pot crește dinamic). Schimbarea dimensiunii unui ndarray va crea o nouă matrice și va șterge originalul.
- Elementele dintr-o matrice NumPy trebuie să fie toate de același tip de date și, prin urmare, vor avea aceeași dimensiune în memorie. Excepție: se pot avea matrice de obiecte (Python, inclusiv NumPy), permitând astfel matrice de elemente de dimensiuni diferite.
- Matricele NumPy facilitează operațiuni matematice avansate și alte tipuri de operații pe un număr mare de date. De obicei, astfel de operațiuni sunt executate mai eficient și cu mai puțin cod decât este posibil folosind secvențele încorporate din Python.

- O multitudine din ce în ce mai mare de pachete științifice și matematice bazate pe Python utilizează matrice NumPy; deși acestea acceptă de obicei intrarea în secvența Python, ele convertesc o astfel de intrare în matrice NumPy înainte de procesare și adesea scot matrice NumPy. Cu alte cuvinte, pentru a utiliza eficient o mare parte (poate chiar cea mai mare parte) din software-ul științific/matematic bazat pe Python de astăzi, este insuficient să știi cum să folosești tipurile de secvențe încorporate din Python - trebuie să știi și cum să folosești tablourile NumPy.

#### De ce este NumPy rapid(că timp de execuție)?

Vectorizarea descrie absența oricărei bucle explicite, indexări etc., în cod - aceste lucruri au loc, desigur, doar „în spatele scenei” în codul C optimizat, pre-compilat. Codul vectorizat are multe avantaje, printre care se numără:

- codul vectorizat este mai concis și mai ușor de citit
- mai puține linii de cod înseamnă, în general, mai puține erori
- codul seamănă mai mult cu notația matematică standard (făcând mai ușoară, de obicei, codificarea corectă a constructelor matematice)
- vectorizarea are că rezultat mai mult cod „Pythonic”. Fără vectorizare, codul nostru ar fi plin de bucle ineficiente și greu de citit.

Broadcasting este termenul folosit pentru a descrie comportamentul implicit element cu element al operațiunilor; în general, în NumPy toate operațiile, nu doar operațiile aritmetice, ci și cele logice, biți, funcționale etc., se comportă în acest mod implicit element cu element, adică difuzează. Mai mult, în exemplul de mai sus, a și b ar putea fi rețele multidimensionale de aceeași formă, sau un scalar și o matrice, sau chiar două rețele cu forme diferite, cu condiția că matricea mai mică să fie „expandabilă” la formă celei mai mari. În acest fel încât difuzarea rezultată să fie lipsită de ambiguitate. [NPY]

#### 4.2.5 Proiect Mask R-CNN

Proiectul Mask\_RCNN [Abd17] este open source și disponibil pe GitHub sub licența MIT, care permite oricui să utilizeze, să modifice sau să distribuie codul gratuit.

Contribuția acestui proiect este suportul modelului de detectare a obiectelor Mask R-CNN în TensorFlow ≥ 1.0 prin construirea tuturor strukturilor în modelul Mask R-CNN și oferind un API simplu pentru a-l antrena și testa.

Modelul Mask R-CNN prezice eticheta clasei, casetă de delimitare și masca pentru obiectele dintr-o imagine. Iată un exemplu a ceea ce modelul ar putea detecta.

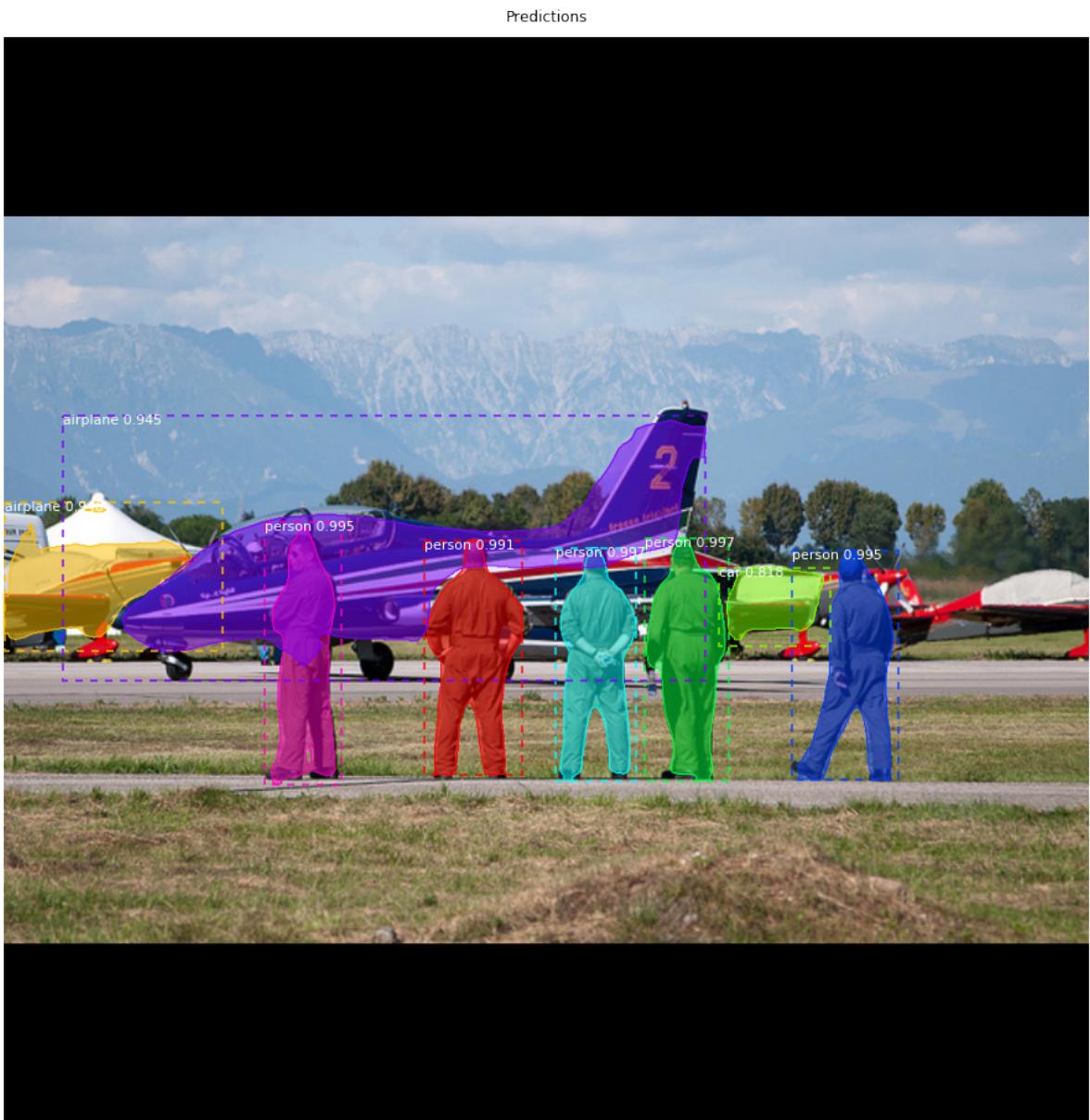


Figura 4.1[Abd17]

În cadrul acestui proiect voi folosi un model preantrenat pe MS COCO.

Cu ajutorul acestui Mask R-CNN pot facilita aplicarea măștilor peste obiectele pe care este antrenat. De exemplu, în cazul în care utilizatorul ar dori ștergerea unei persoane, folosind acest proiect i se va genera masca.

# Capitolul 5

## Aplicatia

### 5.1 Arhitectura aplicatie

Figura 5.1 reprezintă diagrama de arhitectură a sistemului. Fiind o aplicație web, urmează modelul client-server pentru comunicarea prin internet. Aplicația front-end este clientul și Backend este server-ul.

Comunicarea între server și client se face la unele operați cu http request, iar la altele cu un socket.

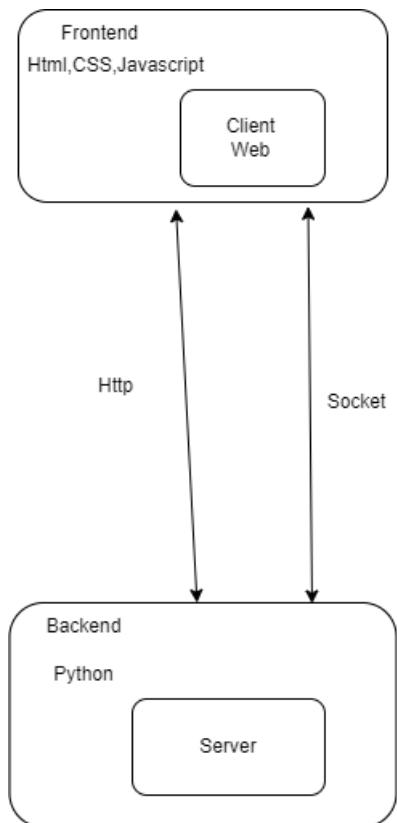


Figura 5.1: Diagrama cazurilor de utilizare

## 5.2 Diagrama cazurilor de utilizare

In Figura 5.2 putem vedea cazurile de utilizare pe care le are un utilizator la pornirea aplicatie si interactiunile dintre ele si server.

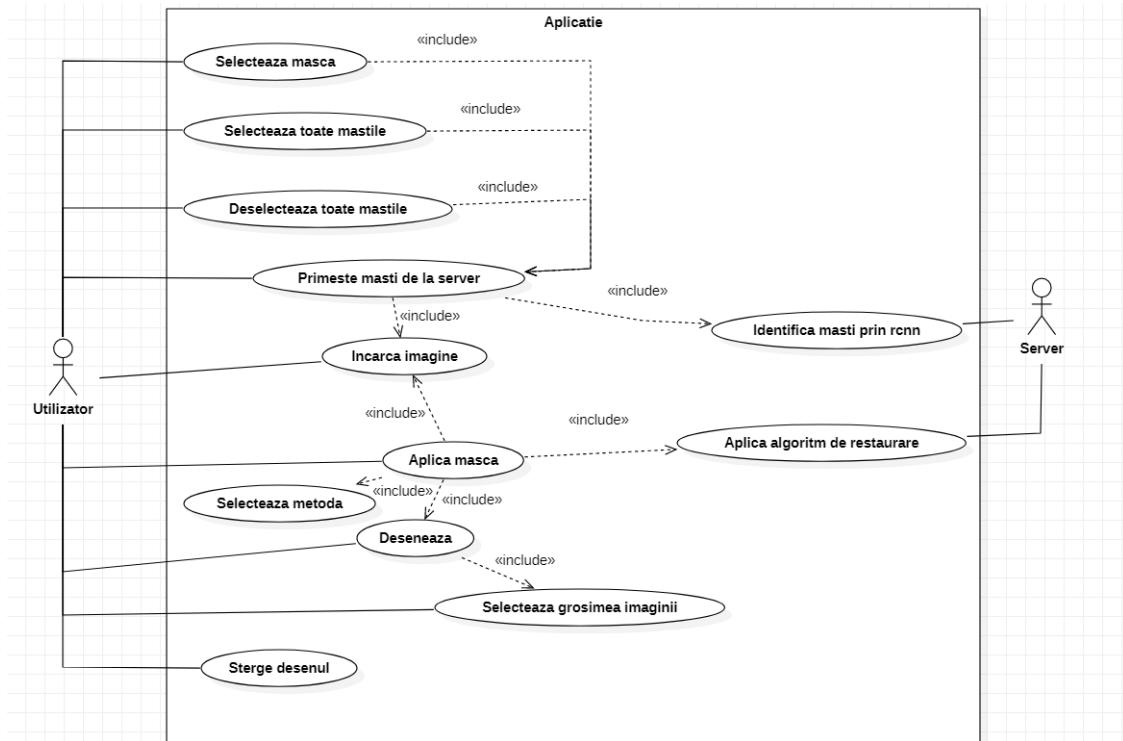


Figura 5.2: Diagrama cazurilor de utilizare

## 5.3 Manualul utilizatorului

- butonul **Upload image** este folosit pentru a încarcă din memoria locală o imagine pe care dorim să aplicăm algoritmul de umplere; dacă se alege încărcarea unei imagini măștile și imaginea umplută vor fi șterse.
- butonul **clear** este utilizat pentru a șterge masca actual desenată; odată cu stergerea măștii desenate se va șterge și imaginea rezultată după umplere
- butonul **get Masks** generează toate măștile la obiectele pe care ai-ul de pe server este antrenat; ele vor apărea în chenarul din dreapta;
- în urmă apăsării unei **casetă de bifat** din dreptul unei măști generate de ai, aceea masca va fi colorată peste imaginea inițială și va fi folosită în algoritmul de umplere
- butonul **Inpaint** va genera imaginea umplută peste măștile specifice

- **bara de glisare** determină grosimea cu care se desenează masca
- butonul **Select All** selectează toate măștile generate de ai
- butonul **Unselect All** deselectează toate măștile generate de ai
- dropdown-ul din chenarul din dreapta permite selectarea metodei folosite pentru umplere



Figura 5.3: Exemplu de folosire al aplicației

# Capitolul 6

## Concluzii și eventuale îmbunătățiri

Odată cu apariția calculatoarelor personale puternice și a dispozitivelor mobile automatizarea ușurat viețile oamenilor. Umplerea de imagini este una din artele vechi de prelucrarea a imaginilor care și în zi de azi se face manual pentru maximă de siguranță. În acest sens am încercat să creăm un algoritm cât mai bun pentru a automatiză acest proces.

Diferența principală dintre algoritmul propus de umplere și alte algoritme este că folosește 2 tipuri de umplere (difuzie și regiuni similare) în funcție de dimensiunea lui  $\Omega$ .

Eventuale imbunatarii la acest algoritm ar fi să antrenăm și să folosim un GAN(Generative Adversarial Networks, sunt foarte folosite pentru generare de imagini). Deasemenea interfață aplicației ar putea fi îmbunătățită cu un aspect mai complex și un design mai prietenos utilizatorilor.

# Bibliografie

- [Abd17] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. [https://github.com/matterport/Mask\\_RCNN](https://github.com/matterport/Mask_RCNN), 2017.
- [ACT04] P. Pérez A. Criminisi and K. Toyama. Region filling and object removal by exemplar-based image inpainting, 2004. pp. 1200–1212.
- [ANN] Artificial neural network - basic concepts. opened 17/04/2022.
- [CNN] Cnn care foloseste dropout layer. <https://www.baeldung.com/cs/ml-relu-dropout-layers>. opened 22/05/2022.
- [CSS] Css. <https://en.wikipedia.org/wiki/CSS>. opened 23/05/2022.
- [EC2] Exemplu conoluție. [https://www.researchgate.net/figure/A-2D-convolution-example\\_fig2\\_334416432](https://www.researchgate.net/figure/A-2D-convolution-example_fig2_334416432). opened 22/05/2022.
- [EL99] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling, 1999. pp. 1033–1038.
- [FS99] Yoav Freund and Robert E. Schapire. *Large margin classification using the perceptron algorithm*. Cornell Aeronautical Laboratory, 1999.
- [JQ2] Jquery. <https://en.wikipedia.org/wiki/JQuery>. opened 23/05/2022.
- [JS2] Javascript. <https://en.wikipedia.org/wiki/JavaScript>. opened 23/05/2022.
- [JTM] Html. <https://devdocs.io/html/>. opened 23/05/2022.
- [Kha] Renu Khandelwal. Computer vision: Instance segmentation with mask r-cnn. <https://towardsdatascience.com/computer-vision-instance-segmentation-with-mask-r-cnn-79835021>. opened 28/05/2022.

- [Loh] H. Lohninger. Structure of measured data. [http://www.statistics4u.com/fundstat\\_eng/cc\\_data\\_structure.html](http://www.statistics4u.com/fundstat_eng/cc_data_structure.html). Online: 17/04/2022.
- [MBB00] V. Caselles M. Bertalmio, G. Sapiro and C. Ballester. "image inpainting", 2000. pp. 417–424.
- [MMHS09] K. A. Moustafa M. M. Hadhoud and S. Z. Shenoda. Digital images inpainting using modified convolution based method, 2009.
- [MMOC01] R. McKenna M. M. Oliviera, B. Bowen and Y. S. Chang. Fast digital image inpainting, 2001. pp. 261–266.
- [NNI] Neural networks. <https://www.ibm.com/cloud/learn/neural-networks>. opened 21/05/2022.
- [NNM] Neural networks. <https://www.mygreatlearning.com/blog/types-of-neural-networks/>. opened 21/05/2022.
- [NPY] Numpy. <https://numpy.org/doc/stable/user/whatisnumpy.html>. opened 23/05/2022.
- [OCVa] Opencv. <https://opencv.org/about/>. opened 23/05/2022.
- [OCVb] Opencv algoritm de restaurare a imaginilor. [https://docs.opencv.org/4.x/df/d3d/tutorial\\_py\\_inpainting.html](https://docs.opencv.org/4.x/df/d3d/tutorial_py_inpainting.html). opened 23/05/2022.
- [Olt00] V. Olteanu, A. și Lupu. Neurofiziologia sistemelor senzitivo-senzoriale, 2000.
- [RNC] Retea neuronala convolutionala. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-theory-and-practicality-e4d096135e47>. opened 21/05/2022.
- [Roo19] Matthew Roos. Deep learning neurons versus biological neurons. <https://towardsdatascience.com/deep-learning-versus-biological-neurons-floating-point-numbers-2019>.
- [Ros57] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

- [RY19] Pranaw Kumar Nabhan Yousef Rammah Yousef, Ayush Agarwal. Object removal using a new exemplar-based, image inpainting algorithm and seam carving. <https://www.ijrte.org/wp-content/uploads/papers/v8i4/D9093118419.pdf>, 9 2019.
- [VB14] Raluca Vreja and Remus Brad. Image inpainting methods evaluation and improvement. <https://www.hindawi.com/journals/tswj/2014/937845/>, 2014.