



UNIVERSITATEA DIN  
BUCUREȘTI

FACULTATEA DE  
MATEMATICĂ ȘI  
INFORMATICĂ



# LUCRARE DE LICENȚĂ

Absolvent

Tănase Victor-Flavian

Coordonator științific

Prof. Dr. Radu Ionescu

București, iunie 2024



UNIVERSITATEA DIN  
BUCUREȘTI

FACULTATEA DE  
MATEMATICĂ ȘI  
INFORMATICĂ



# DETECȚIA DE CONȚINUT DEEPPFAKE CU AJUTORUL REȚELELOR NEURONALE ADÂNCI

Absolvent

Tănase Victor-Flavian

Coordonator științific

Prof. Dr. Radu Ionescu

București, iunie 2024

## Rezumat

Evoluția capabilităților algoritmilor de inteligență artificială din ultimii ani a reinventat crearea de conținut în sfera digitală. În același timp, progresul în câmpul vederii artificiale a fost văzut de către unele entități ca o oportunitate de a răspândi dezinformare sau de a crea conținut malițios greu de detectat cu ochiul liber.

Ceea ce a început ca cercetare academică realizată de către Justus Thies et al. [11] a dat naștere unui mijloc de fabricare de conținut contrafăcut, capabil să manipuleze imagini și videoclipuri cu un realism nemaivăzut. Acest tip de conținut poate fi folosit în diverse scopuri cu rea-voință precum: șantaj, manipulare politică, înșelătorie, furt de identitate sau creare de conținut neadecvat, de cele mai multe ori aceste atacuri vizând persoane publice.

În scopul combaterii dezinformării, această lucrare are ca obiectiv punerea la dispoziție către public a unui model de inteligență artificială cu o interfață web, ce poate fi ușor de utilizat pentru detectarea sau cel puțin ridicarea suspiciunii asupra imaginilor sau videoclipurilor posibil fabricate.

## **Abstract**

The evolution of the artificial intelligence models in recent years has reinvented the creation of content on social media. In the same time, the progress in Computer Vision has been viewed by others as an opportunity to spread misinformation and create malicious content that can be hardly detected by the naked eye.

What has started as academic research by Justus Thies et al. [11] pioneered a new way to fabricate content, capable to manipulate images and videos with unprecedented realism. This type of content can be used in many harmful ways such as blackmailing, political manipulation, fraudulent schemes, identity theft, and the creation of explicit material. Often, public figures are the primary targets of such attacks.

In order to fight disinformation, this work aims to provide the public with an artificial intelligence model with a web interface, that can be easily used to detect or at least raise suspicion over possibly fabricated images or videos.

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>6</b>
1.1	Descrierea problemei . . . . .	6
1.2	Motivație . . . . .	6
1.3	Contribuție personală . . . . .	6
1.4	Privire de ansamblu și organizarea lucrării . . . . .	6
<b>2</b>	<b>Concepte Teoretice</b>	<b>7</b>
2.1	Rețele neuronale . . . . .	7
2.1.1	O scurtă istorie a rețelelor neuronale . . . . .	7
2.1.2	De la neuronul biologic la cel artificial . . . . .	8
2.1.3	Perceptronul . . . . .	10
2.1.4	Multilayer Perceptron(MLP) . . . . .	12
2.1.5	Backpropagarea . . . . .	13
2.1.6	Algoritmul coborârii pe gradient( <i>Gradient Descent</i> ) . . . . .	14
<b>3</b>	<b>Abordări din literatură</b>	<b>15</b>
3.1	Celeb-DF (Celebrities Deepfake) . . . . .	15
3.2	FaceForensics++ . . . . .	17
<b>4</b>	<b>Tehnologii folosite</b>	<b>19</b>
4.1	Pyhton . . . . .	19
4.1.1	Caracteristici cheie ale limbajului . . . . .	19
4.1.2	Limitări ale limbajului . . . . .	20
4.2	Pytorch . . . . .	22
4.2.1	Torchvision . . . . .	22

4.3	Flask . . . . .	23
<b>5</b>	<b>Abordare propusă, experimente, rezultate</b>	<b>24</b>
5.1	Crearea setului de date . . . . .	24
5.1.1	Celeb-DF . . . . .	25
5.1.2	FaceForensics++ . . . . .	26
5.1.3	Selecția cadrelor . . . . .	26
5.1.4	Extragerea fețelor . . . . .	27
5.2	Selecția arhitecturilor . . . . .	29
5.2.1	ResNet . . . . .	29
5.2.2	EfficientNet . . . . .	30
5.2.3	RegNet . . . . .	31
5.3	Experimente și rezultate . . . . .	32
5.4	Creșterea performanței cu ajutorul Test Time Augumentation(TTA)	32
	<b>Bibliografie</b>	<b>33</b>

# Capitolul 1

## Introducere

### 1.1 Descrierea problemei

### 1.2 Motivație

### 1.3 Contribuție personală

### 1.4 Privire de ansamblu și organizarea lucrării

Acest capitol abordează conceptele fundamentale care stau la baza înțelegerii rețelelor neuronale.

# Capitolul 2

## Concepte Teoretice

### 2.1 Rețele neuronale

Rețelele neuronale stau la baza tuturor algoritmilor moderni de inteligență artificială. Acestea au revoluționat industria învățării automate prin puterea lor de a simula funcțiile cognitive ale creierului uman, fiind des folosite în învățarea de tipare (în engleză pattern recognition), diferite sarcini de clasificare și predicție. Ele pot fi văzute ca funcții complicate care primesc date de intrare sub forma unor valori numerice și returnează valori reprezentative pentru acestea.

#### 2.1.1 O scurtă istorie a rețelelor neuronale

- În lucrarea intitulată „A Logical Calculus of the Ideas Immanent in Nervous Activity” (1943) [6] Warren McCulloch și Walter Pitts au fost primii care au teoretizat o implementare matematică simplificată a neuronului ca o poartă logică binară. Aceștia au demonstrat teoretic faptul că neuronii au capacitatea de a reprezenta orice funcție matematică.
- În 1958, este publicată lucrarea cercetătorului Frank Rosenblatt „The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain” [9], în care acesta introduce conceptul de Perceptron, cel mai simplu model de rețea neuronală.



- Spre finalul anilor 1980, după o lungă perioadă în care studiul rețelelor neuronale a fost neglijat, David E. Rumelhart, Geoffrey Hinton, and Ronald J. Williams (1986) [10] prezintă **backpropagarea**, un algoritm prin care o rețea neuronală putea să învețe eficient, adresând multe dintre problemele ridicate până la acel moment.
- La începutul anilor 2000, tot Geoffrey Hinton, de data aceasta alături de Ruslan Salakhutdinov [5] aduc o soluție pentru problema gradientilor care dispar, prin reducerea dimensionalității datelor.

Aceste lucrări din literatură au ajutat la transformarea unor concepte pur teoretice în unelte cu putere de învățare și de decizie care au ajutat la modelarea industriei moderne și au împins progresul tehnologic către noi limite.

### 2.1.2 De la neuronul biologic la cel artificial

La fel ca multe dintre invențiile revoluționare de-a lungul istoriei, cercetătorii au avut ca sursă de inspirație viețuitoarele. În cazul inteligenței artificiale, aspirația cercetătorilor a fost să relice neuronul biologic.

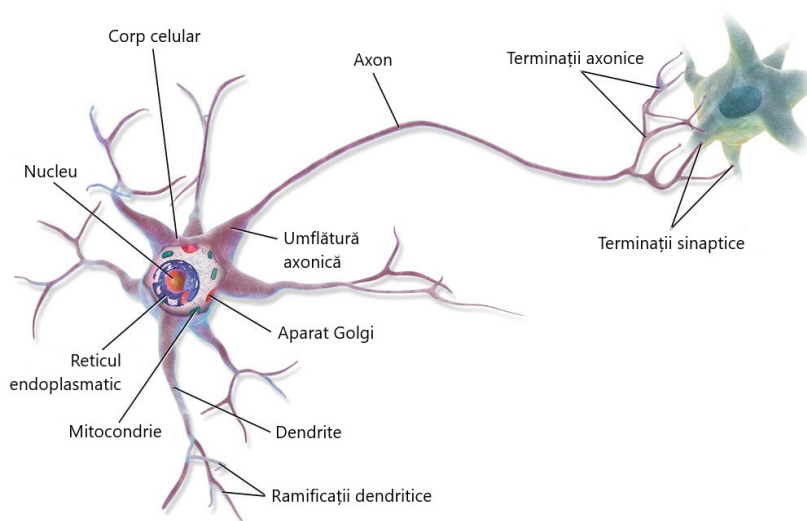


Figura 2.1: Anatomia unui neuron [1]

În zona centrală a neuronului se află corpul celular(soma) care conține nucleul, locul care înmagazinează tot materialul genetic al celulei. Legate de corpul celular

sunt dendritele, care interceptează semnale chimice de la alți neuroni. Atașat de corpul celular, se află o prelungire alungită numită axon, al cărei scop este să transmită semnale electrice către alți neuroni sau către țesuturi. La capătul axonului se găsesc terminațiile acestuia, care sunt legate de dendritele sau de corpul celular ale altui neuron.

În creierul biologic, se găsesc miliarde de neuroni, fiecare având mii de legături cu alți neuroni aceștia fiind dipusi în straturi pentru a crea țeșutul nervos. Cu toate că neuronul în sine, nu funcționează într-un mod complex, ansamblul a miliarde de mecanisme simple într-o rețea imensa are capabilități impresionante.

Plecând de la aceste premise Warren McCulloch și Walter Pitts(1943) [6] au prezentat în lucrarea lor revoluționară o versiune simplificată a neuronului biologic. Aceștia au văzut neuronul artificial ca o pe poartă logică, cu mai multe intrări și o singură ieșire. Un semnal electric era transmis mai departe doar dacă neuronul avea un număr de intrări care trecea de un anumit prag, bazându-se pe principiul de totul sau nimic al neuronilor biologici.

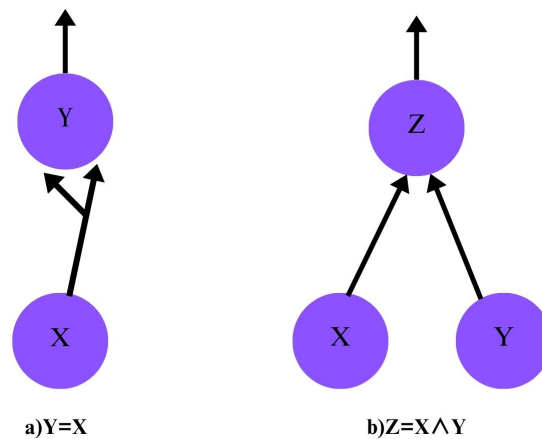


Figura 2.2: Neuroni artificiali calculând operații logice

- a) Neuronul Y nu poate transmite semnalul mai departe doar cu un singur impuls
- b) Neuronul Z are nevoie de ambii neuroni X și Y pentru a trimite un semnal mai departe

### 2.1.3 Perceptronul

În 1958 Frank Rosenblatt [9] introduce în literatură **perceptronul**, cel mai simplu model de rețea neuronală. Acesta duce neuronul artificial propus de către McCulloch și Pitts [6] la un alt nivel. Comparat cu neuronul care transmitea doar semnale binare, cel propus de Rosenblatt este capabil să proceseze și să transmită mai departe valori numerice. Scopul final este de a organiza aceste unități de calcul într-o „rețea” care primește datele de intrare sub formă numerică și returnează o valoare care le reprezintă.

Într-o astfel de rețea fiecare neuron se folosește datele de intrare pentru a produce o valoare de ieșire. Fiecărei valori de intrare îi corespunde o pondere (în engleză: *weight*), un număr real al cărui scop este să controleze contribuția sa la informația transmisă mai departe de către neuron. Aceste valori sunt însumate ponderat, iar la valoarea obținută se adaugă un neuron special, care la rândul lui are o pondere. Acest neuron se numește *bias*, un număr real, notat cu  $b$ , care elimină constângerea ca funcția ce reprezintă datele să treacă prin originea planului sau a hiperplanului. Rezultatul reprezintă o funcție care este dată ca parametru unei funcții liniare numită „funcție de pas” (în engleză: *step function*) pentru a produce valoarea de ieșire a perceptronului.

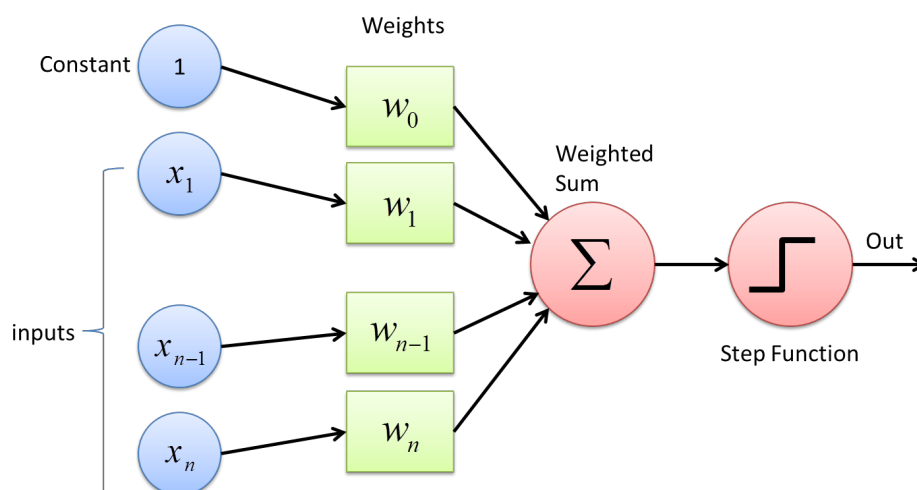


Figura 2.3: Perceptronul [8]

Notății:

- $x = (x_1, x_2, x_3, \dots, x_n)$  datele de intrare
- $w = (w_1, w_2, w_3, \dots, w_n)$  ponderile corespunzătoare
- $b$  = totalitatea neuronilor de *bias*
- $z$  = combinația liniară dintre  $x$  și  $w$
- $z = w_1x_1 + w_2x_2 + \dots + w_nx_n = \sum_{i=1}^n w_ix_i + b = w^Tx + b$
- $step(z) = output\text{-}ul$  perceptronului

O funcție comună folosită drept *step function* este funcția semn definită astfel:

$$\text{sgn}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

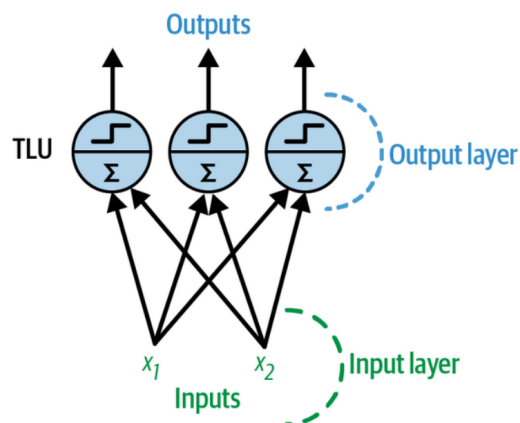


Figura 2.4: Un perceptron capabil să separe date în mai multe clase [3]

În Figura 2.4 este prezentată o arhitectură de perceptron capabilă să distingă între 3 clase. Datele de intrare, în cazul nostru  $x_1$ ,  $x_2$  reprezintă *stratul de intrare*. Fiecare valoare din stratul de intrare este legată de toți neuronii din stratul următor, fiecare legătură având o pondere individuală. Stratul care produce valorile de ieșire se numește *stratul de ieșire* și în exemplul prezentat este format din 3 neuroni, corespunzători fiecărei clasificări posibile.

În ciuda optimismului declanșat de capacitățile remarcabile la acea vreme a perceptronului, în 1969 Marvin Minsky și Seymour Papert [7] au demonstrat limitarea acestui model de a rezolva câteva probleme triviale, precum operația XOR.

### 2.1.4 Multilayer Perceptron(MLP)

Soluția pentru limitările perceptronului a fost unificarea mai multor rețele de tip perceptron într-o rețea mai mare, numită Multilayer Perceptron(MLP). Noua clasificare a straturilor este :

- Stratul de input = stratul ce conține doar valorile de intrare
- Straturile ascunse = straturile prin care se propaga informația. Se situează între stratul de input și cel de output
- Stratul de output = stratul care ne oferă rezultatul trecerii informației prin rețea

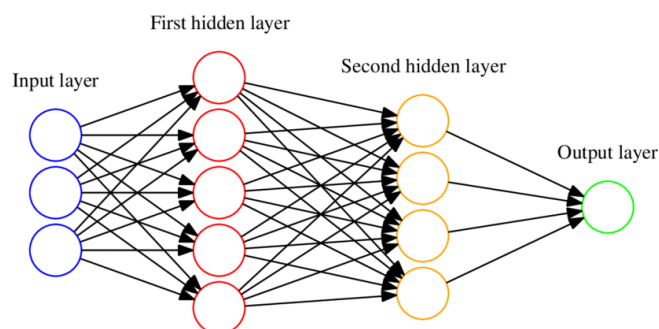


Figura 2.5: Arhitectura de tip MLP(Multilayer Perceptron) [2]

Arhitectura MLP este una de tip **feedforward**, cu alte cuvinte, informația se propagă prin rețea într-o singură direcție, de la stânga la dreapta(dinspre input către output).

### 2.1.5 Backpropagarea

Algoritmul prin care o rețea neuronală învață se numește **backpropagation**. Acesta presupune trecerea prin rețea în repetate rânduri, în doi pași: unul înainte și unul înapoi. După fiecare trecere parametrii(ponderile) sunt actualizați pentru a obține predicții mai bune. Algoritmul poate fi descris astfel:

- Setul de date este spart în blocuri mai mici, cele mai întâlnite dimensiuni fiind de 32, 64 sau 128 de instanțe per bloc
- Fiecare bloc este trecut prin rețea: se calculează output-ul primului strat, care este dat ca intrare pentru urmatorul strat, continuând în același fel până se ajunge la stratul de output. Acesta este pasul forward(*forward pass*).
- Se evaluează performanța predicțiilor cu ajutorul unei funcții numită **funcție de pierdere**. Scopul acestei funcții este să furnizeze un scor care să descrie cât de precise au fost predicțiile rețelei față de rezultatul dorit.
- După calculul erorii, algoritmul calculează contribuția fiecărei ponderi din rețea la eroarea totală, cu ajutorul regulii fundamentale din analiza matematică: regula înlănțuirii(în engleză: *chain rule*).
- Contribuția la eroarea totală a unui parametru se numește **gradient**. Valoarea gradientului practic măsoară cât de mult influențează eroarea totală modificarea acelui parametru.
- În cele din urmă, algoritmul folosește gradientii calculați pentru optimizarea parametrilor cu ajutorul unui algoritm de optimizare.

### 2.1.6 Algoritmul coborârii pe gradient (*Gradient Descent*)

Coborârea pe gradient este un algoritm iterativ de optimizare, folosit des în literatură, al cărui scop este de găsi minimul global al unei funcții. În cadrul rețelelor neuronale acesta este folosit pentru a optimiza parametrii rețelei astfel încat valoarea erorii totale să se apropie la fiecare iterație de minimul funcției de pierdere. După calcularea gradientilor, fiecare pondere se actualizează după formula:

$$w_{new} = w - \eta \frac{\delta L}{\delta w}$$

Unde:

- $w_{new}$  = noua valoare a ponderii
- $\frac{\delta L}{\delta w}$  = gradientul
- $\eta$  = rata de învățare(hiperparametru ce controlează cât de mult se modifică valorile ponderilor)

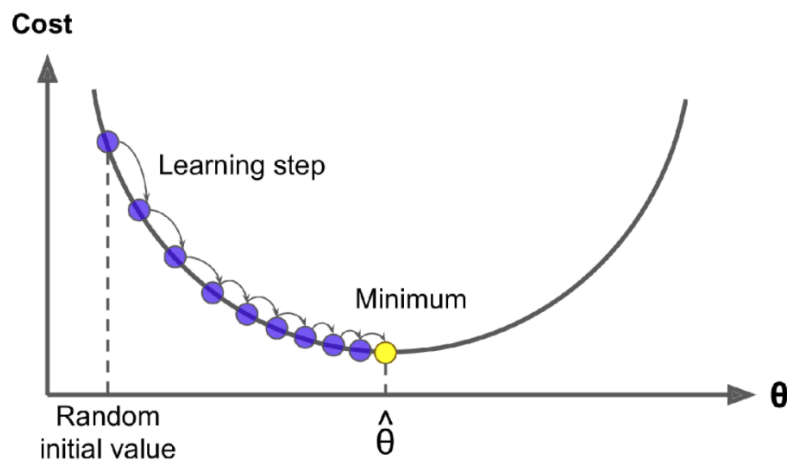


Figura 2.6: Vizualizare a algoritmului de coborârii pe gradient [4]

# Capitolul 3

## Abordări din literatură

Tehnologia de tip deepfake, ajutată de progresul rapid al modelelor adânci, precum Rețelele Generative Adversariale (GAN), au avut un impact semnificativ în sfera digitală prin crearea de conținut sintetic realist. Acest avans în tehnologie reprezintă un adevărat pericol pentru confidențialitatea, securitatea și integritatea informației. Din acest motiv, este nevoie de mecanisme robuste pentru detecție a conținutului nelegitim.

În ultimii ani, diverse lucrări din literatură au abordat această temă folosind inteligență artificială, dar și tehnici tradiționale bazate pe verificarea inconsistențelor din imagini/videoclipuri. Atenția va fi acordată performanțelor pe bazele de date Celeb-DF [li2020celeb] și FaceForensics++ [rössler2019faceforensics], întrucât lucrarea prezentată folosește date din aceste două seturi.

### 3.1 Celeb-DF (Celebrities Deepfake)

Introdusă de Li et al. (2020) [li2020celeb], Celeb-DF este o bază de date vastă care a devenit un nou etalon pentru detecția de conținut fabricat. A fost proiectată cu scopul de a aduce progres în acest câmp de cercetare, abordând mai multe limitări ale seturilor de date anterioare, precum DeepFakeTIMIT [khan2021adversarially] și FaceForensics++ [rössler2019faceforensics], oferind videoclipuri deepfake de înaltă calitate și diversitate. Celeb-DF conține 5639 de videoclipuri fabricate de rezoluție înaltă generate folosind o metodă, ce mi-



nimizează artefactele de compresie și sporește realismul expresiilor și mișcărilor faciale.

Un aspect important al acestei baze de date, este faptul că videoclipurile conțin secvențe cu celebrități, un grup de persoane care are șanse mult mai mari să apară în conținut de tip deepfake.

Pe lângă videoclipurile fabricate, setul de date include și un set divers de videoclipuri originale, oferind astfel un mediu de antrenare propice. Introducerea Celeb-DF a contribuit semnificativ la dezvoltarea tehnicilor de detectare a deepfake-urilor.




Figura 3.1: Captură dintr-un videoclip original cu actorul Brad Pitt



Figura 3.2: Captura dintr-un videoclip DeepFake

Autorii lucrării au testat riguros pe setul de date diferite modele performante în detecția de deepfake-uri, scoțând în evidență robustețea dar și slăbiciunile lor. Dintre acestea, 2 modele s-au remarcat prin performanța lor.

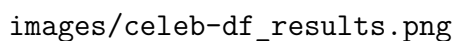
- **XceptionNet**[rössler2019faceforensics][Chollet\_2017\_CVPR]: bazată pe modelul Inception([szegedy2015going]), această arhitectură s-a dovedit a fi foarte puternică pentru task-uri de clasificare. A fost adaptată pentru detecția de deepfake-uri, datorită capabilităților sale de feature extraction. XceptionNet s-a dovedit a se descurca în particular bine la detecția artefactelor introduse în timpul creării deefake-ului, obținând o acuratețe de 65.5%.



images/xception-architecture.png

Figura 3.3: Arhitectura Xception [Chollet\_2017\_CVPR]

- **Face Warping Artifacts**[zhou2018twostream]: Modelul Face Warping Artifacts(FWA)este conceput pentru a detecta artefactele de distorsionare a feței, frecvent întâlnite în videoclipurile deepfake. Aceste artefacte sunt distorsiuni subtile introduse în timpul procesului de manipulare a feței, care pot include structura facială nenaturală sau inconsistențe în textură. Combinat cu alte tehnici de detecție a deepfake-urilor, modelul combinat DSP-FWA(DeepFake Stack Pointer Face Warp Artifacts) a obținut o performanță de 64.6%.



images/celeb-df\_results.png

Figura 3.4: Rezultatele pentru diferite modele [li2020celeb]

## 3.2 FaceForensics++

FaceForensics++ este un set de date și o platformă de benchmark esențială în cercetarea detectării deepfake-urilor și a altor tipuri de conținut deepfake. Dezvoltat de Andreas Rössler și colegii săi, FaceForensics++ a fost conceput pentru a oferi o resursă cuprinzătoare și diversă pentru antrenarea și evaluarea algoritmilor de detectare a conținutului fabricat.

FaceForensics++ include peste 1,000 de videoclipuri de înaltă calitate, preluate din diferite surse și manipulate folosind mai multe tehnici avansate. Aceste videoclipuri sunt împărțite în patru categorii principale ce reprezintă algoritmii folosiți pentru crearea deepfake-urilor:

- **Deepfakes:** Videoclipuri manipulate folosind tehnici de deepfake, care implică substituirea feței unei persoane cu cea a altei persoane folosind rețele generative adversariale (GANs).
- **Face2Face**[11]:
- **FaceSwap:** Tehnica de schimbare a feței între două persoane în videoclipuri.
- **NeuralTextures**[thies2019deferred]: O metodă care utilizează texturi

neurale pentru a asocia expresii faciale sintetice unui model facial.



Figura 3.5: Procesul de creare a unui DeepFake [rössler2019faceforensics]

Setul de date este împărțit în trei subseturi pe baza nivelului de compresie: fără compresie (raw), compresie ușoară (c23) și compresie severă (c40). Acest lucru permite cercetătorilor să testeze robustețea algoritmilor de detecție la diferite niveluri de calitate video.

În lucrarea „Video Face Manipulation Detection Through Ensemble of CNNs”(2020) [bonettini2020video], autorii abordează problema detectării fețelor manipulate

în secvențele video, folosind setul FaceForensics++. Metoda propusă utilizează modelul EfficientNetB4[**tan2019efficientnet**], îmbunătățit prin mecanisme de atenție [**vaswani2017attention**]. Antrenarea modelelor a fost făcută pe seturile de date FaceForensics++ și DeepFake Detection Challenge(o bază de date ce conține peste 119.000 de secvențe din video-uri create pentru o competiție organizată de [Kaggle](#)).



Figura 3.6: EfficientNetB4 cu atenție [**bonettini2020video**]

Acuratețea modelului a fost evaluată folosind metrica AUC(Area Under Cur-

ve), care indică ce capacitate are modelul în a distinge între clasele pozitive și negative.

Autorii au antrenat diverse versiuni ale arhitecturii EfficientNet-B4, pe care în final le-au unit într-un ansamblu de rețele, care a obținut scorul AUC de 0.9444.

# Capitolul 4

## Tehnologii folosite

Lucrarea constă în dezvoltarea unui model de inteligență artificială și punerea la dispoziție a unei interfețe web prin care utilizatorii îl pot folosi cu ușurință. Pentru crearea datasetului și a diverselor funcții ajutătoare ale aplicației s-a folosit limbajul **Python**, în timp ce pentru antrenarea, evaluarea și salvarea modelului s-a utilizat framework-ul **PyTorch**. Interfața web este dezvoltată folosind framework-ul **Flask**, ce permite crearea paginilor web în limbajul Python. În următoarele subcapitole vor fi descrise pe scurt, fiecare dintre aceste tehnologii.

### 4.1 Python

**Python** este un limbaj de programare de nivel înalt, interpretat și dinamic, cunoscut pentru sintaxa sa simplă, care permite dezvoltatorilor să scrie cod eficient și ușor de întreținut. Creat de Guido van Rossum [**van1guido**] și lansat pentru prima dată în 1991, Python a devenit unul dintre cele mai populare limbaje de programare datorită versatilității și comunității sale mari, care a creat o mulțime de biblioteci și framework-uri utile.

#### 4.1.1 Caracteristici cheie ale limbajului

O caracteristică cheie a acestui limbaj este portabilitatea sa. Codul sursă scris în limbajul Python poate fi rulat pe o varietate de sisteme de operare, inclusiv Windows, macOS, Linux, și chiar și pe dispozitive mobile. Acest lucru se datorează

în mare măsură faptului că Python este un limbaj interpretat, care nu are nevoie de compilare înainte de a fi rulat.

Python folosește un program numit interpretor, care citește codul Python și îl traduce în instrucțiuni pe care calculatorul le poate înțelege. Interpretorul Python este disponibil pentru o gamă largă de sisteme de operare, ceea ce înseamnă că programele Python scrise pe un sistem de operare pot fi rulate pe orice alt sistem de operare fără modificări semnificative.

De asemenea, Python are o mulțime de biblioteci și cadre de lucru care sunt, de asemenea, portabile, ceea ce înseamnă că pot fi folosite pe orice sistem de operare pe care rulează Python. Acest lucru face ca Python să fie o alegere populară pentru dezvoltarea de aplicații multi-platform.

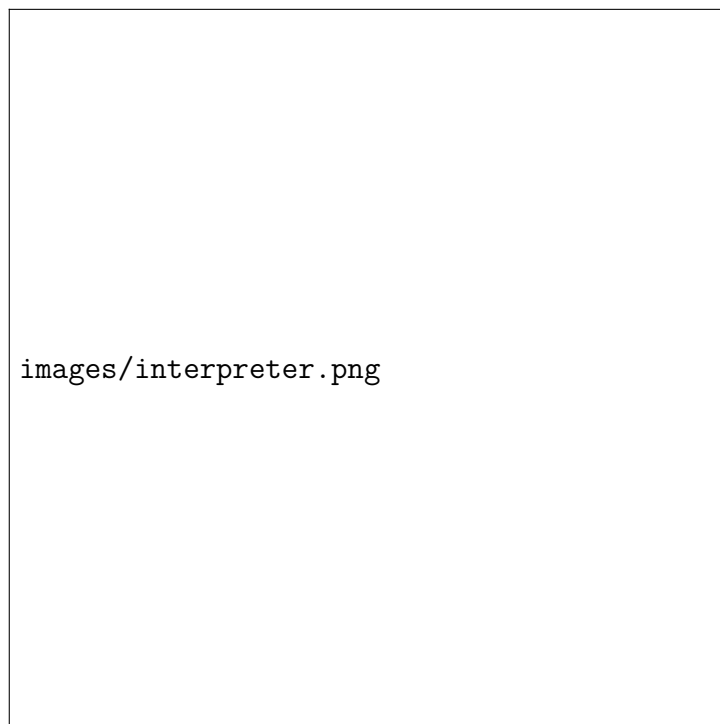


Figura 4.1: Interpretorul de Python[**interpreter**]

Python este un limbaj de programare multiparadigmă, adică poate fi folosit pentru diverse stiluri de programare, ceea ce oferă developer-ilor flexibilitate pentru preferințele personale.

Unul dintre cele mai populare paradigme de programare în Python este progra-



marea orientată pe obiecte (OOP), care permite crearea obiecte care conțin atât date, cât și funcții care lucrează cu aceste date. OOP este o alegere populară pentru dezvoltarea de aplicații complexe, unde este utilă organizarea codului într-un stil modular și reutilizarea acestuia.

O altă paradigmă populară în Python este programarea funcțională, care se concentrează pe scrierea de funcții care sunt independente una de cealaltă. Programarea funcțională poate fi utilă în situații în care este importantă corectitudinea și ușurința de testare a codului.

În plus față de acestea, Python mai suportă programarea procedurală, programarea concurrentă, și chiar programarea logică.

#### 4.1.2 Limitări ale limbajului

Deși Python este un limbaj de programare extrem de popular și versatil, nu este lipsit de limitări. Cele mai semnificative dintre acestea sunt:

##### **Performanța**

Python folosește un interpretor, ceea ce înseamnă că, în general, rulează mai lent decât limbajele compilate precum C sau C++ . Aceasta se datorează faptului că Python este un limbaj dinamic, iar interpretarea codului la rulare adaugă un overhead semnificativ.

Python utilizează un mecanism numit **GIL(Global Interpreter Lock)** pentru a gestiona execuția thread-urilor. GIL permite doar unui singur thread să execute cod Python la un moment dat, ceea ce limitează performanța în aplicațiile multi-threaded și afectează negativ utilizarea eficientă a procesoarelor multi-core.

De asemenea, este lent în sarcinile de procesare numerică intensivă. Această slăbiciune poate fi diminuată prin utilizarea bibliotecilor optimizate precum NumPy, care implementează calculele mult mai eficiente.

##### **Consumul de Memorie**

Python consumă mai multă memorie comparativ cu limbajele mai puțin abstracte, cum ar fi C și C++. Obiectele Python sunt mai voluminoase din cauza metadatelor asociate, ceea ce poate conduce la o utilizare ineficientă a memoriei.

Gestionarea memoriei se face automat prin utilizarea unui „garbage collector” pentru gestionarea automată a memoriei, care poate introduce un overhead suplimentar și poate duce la probleme de performanță în aplicații cu cerințe mari de memorie.

În plus, flexibilitatea oferită de structurile de date dinamice, cum ar fi listele și dicționarele, vine cu un cost suplimentar în termeni de consum de memorie.

## Distribuirea aplicațiilor

Distribuirea aplicațiilor Python poate fi mai complexă comparativ cu limbajele compilate. Deși există instrumente precum PyInstaller sau cx-freeze care pot crea executabile pentru aplicațiile scrise în Python, acestea nu sunt la fel de simple și directe ca procedeele folosite pentru alte limbaje de programare.

Aplicațiile Python adesea depind de un număr mare de biblioteci externe, care trebuie gestionate și distribuite împreună cu aplicația. O soluție pentru această problemă este Docker, ce poate rezolva problema gestionării și distribuirii bibliotecilor externe pentru aplicațiile Python prin containerizarea întregului mediu de rulare al aplicației. Docker permite pachetelor și dependențelor necesare să fie incluse într-un container ușor de distribuit, asigurând consistența și funcționalitatea aplicației indiferent de mediul în care este rulată.

O altă provocare în aplicațiile dezvoltate în python este compatibilitatea bibliotecilor. De multe ori există probleme legate de compatibilitatea între versiuni diferite ale limbajului și ale bibliotecilor, ceea ce poate complica procesul de distribuire a aplicațiilor.

## 4.2 Pytorch

[PyTorch](#) este o bibliotecă open-source de învățare automată dezvoltată de Facebook’s AI Research lab (FAIR). Lansată în 2016, PyTorch a câștigat rapid popularitate datorită flexibilității și ușurinței sale de utilizare, devenind unul dintre principalele instrumente pentru cercetare și dezvoltare în domeniul inteligenței artificiale.

Ceea ce face framework-urile de învățare automată diferite de altele este fap-

tul că facilitează utilizarea plăcilor grafice pentru comutații paralele mult mai eficiente.

Unul dintre principalele avantaje ale PyTorch este manipularea dinamică a grafurilor de calcul, ceea ce permite cercetătorilor să ajusteze arhitecturile de rețele neurale în timp real, fără a necesita recompilare. Această caracteristică este esențială pentru experimentare și prototipare rapidă, accelerând munca în proiectele de inteligență artificială. Datorită acestor calități, PyTorch a devenit extrem de popular în mediile academice și industriale, fiind folosit pe larg pentru cercetare și dezvoltarea aplicațiilor comerciale de IA.

### 4.2.1 Torchvision

[Torchvision](#) este un pachet din ecosistemul PyTorch care oferă acces la seturi de date populare, modele pre-antrenate și transformări pentru imagini, fiind special conceput pentru vederea artificială.

Acesta facilitează implementarea rapidă a algoritmilor de computer vision, reducând semnificativ timpul necesar pentru preprocesarea datelor și pentru implementarea de noi experimente. Pachetul include un set divers de instrumente pentru detectarea obiectelor, segmentarea semantică și clasificarea imaginilor, împreună cu funcționalități extinse pentru îmbunătățirea și augmentarea imaginilor.

## 4.3 Flask

[Flask](#) este un micro-framework web pentru Python, cunoscut pentru simplitatea sa și pentru capacitatea de a construi rapid aplicații web. A fost creat de Armin Ronacher și este bazat pe biblioteca Werkzeug WSGI și pe Jinja2 pentru template-uri. Flask oferă un nucleu minimal care poate fi extins cu ajutorul altor biblioteci, ceea ce îl face extrem de flexibil și adaptabil la diverse cerințe de dezvoltare web.

Unul dintre principalele avantaje ale Flask este ușurința cu care poate fi învățat și implementat, făcându-l ideal pentru proiecte de dimensiuni mici și mijlocii, precum și pentru prototipuri rapide ale aplicațiilor web. Flask vine cu un server de dezvoltare și un debugger încorporate, facilitând testarea și depanarea aplicațiilor

în timpul dezvoltării.

De asemenea, suportă extensii care adaugă funcționalități precum, autentificare sau manipularea sesiunilor, permițând dezvoltatorilor să creeze aplicații complexe cu efort minim.

# Capitolul 5

## Abordare propusă, experimente, rezultate

Pentru ca un model de inteligență artificială să învețe eficient trăsăturile necesare unei sarcini de clasificare, acesta necesită un volum mare de date de antrenare de calitate. Primul pas în dezvoltarea modelului a fost crearea unui set de date coerent, care să poată fi ulterior utilizat pentru antrenarea și testarea diverselor arhitecturi. Aceste arhitecturi sunt selectate în general pe baza performanțelor anterioare din literatură. Pentru fiecare arhitectură, s-a evaluat performanța diferitelor combinații de hiperparametri folosind metrici de performanță specifice. În urma acestor testări, a fost creată o interfață web pentru modelul cu cele mai bune rezultate, facilitând astfel utilizarea sa de către oricine.

### 5.1 Crearea setului de date

Setul de date este compus din 24668 de imagini ce conțin fețe din bazele de date **Celeb-DF**[li2020celeb] și **FaceForensics++** [rössler2019faceforensics]. Imaginile reprezintă cadre selectate aleator din zona de mijloc a videoclipurilor pentru a asigura prezența unei fețe în prim-plan. Acestea au fost inițial împărțite în 2 categorii: imagini pozitive(clasa imaginilor reale) și imagini negative(clasa imaginilor fabricate). Apoi, pentru antrenarea și testarea optimă a modelului distribuția datelor a fost următoarea :

- 80% date de antrenare
- 15% date de testare
- 5% date de validare(folosite pentru testare în timpul antrenării)

În plus, pentru a asigura o distribuție optimă a datelor, fiecare subset conține aproximativ 50% imagini negative și 50% imagini pozitive.

O problemă posibilă a acestei abordări este o performanță crescută artificial în cazul în care imagini diferite din același videoclip ar apărea în setul de antrenare, respectiv cel de testare și validare. Împărțirea finală a setului garantează faptul că imaginile din același videoclip se află toate în același subset de date.



Figura 5.1: Un director cu date extrase din videoclipuri

### 5.1.1 Celeb-DF

Baza de date Celeb-DF conține 5639 de videoclipuri de tip deepfake și 590 de videoclipuri reale colectate de pe platforma Youtube. Videoclipurile sunt în principal interviuri cu 59 de celebrități cu o distribuție variată a vârstei, sexului și etniei. Videoclipurile deepfake sunt obținute prin schimbarea de fețelor celor 59 de actori între ei.

La crearea datasetului pentru modelul din lucrare au fost selectate câte 2 cadre din cele 5639 de videoclipuri deepfake și câte 10 cadre din cele 590 de videoclipuri reale.

### 5.1.2 FaceForensics++

FaceForensics++ este un set de date amplu, conceput cu scopul a facilita antrenarea modelelor de inteligență artificială pentru detectarea conținutului fabricat. Include 1000 de secvențe video originale, care au fost manipulate folosind patru metode automate de manipulare facială: Deepfakes, Face2Face, FaceSwap și NeuralTextures. Aceste videoclipuri sunt preluate din 977 videoclipuri de pe platforma YouTube, asigurând că fiecare conține o imagine frontală a feței, permițând astfel o manipulare realistă. Videoclipurile sunt disponibile comprimate în format h264 cu factori diferiți de compresie: raw, c23 și c40.

Pentru setul de date din această lucrare au fost alese videoclipurile cu compresie de tip c23. Ca și contribuție la imaginile negative au fost folosite câte 100 de materiale video generate cu Deepfakes, Face2Face, FaceSwap, respectiv NeuralTextures, din care s-au extras aleator câte două cadre conținând fețe, în timp ce din videoclipurile reale s-au extras câte 10 cadre din X videoclipuri.

### 5.1.3 Selecția cadrelor

Procesarea videoclipurilor pentru extragerea imaginilor a fost realizată cu ajutorul librăriilor [open-cv](#) și [shutil](#). Clasa `cv2.VideoCapture` din OpenCV permite capturarea video prin inițializarea unui obiect de captură, citirea cadrelor și afișarea acestora. Obiectul de captură folosește un cap de citire în interiorul videoclipului, care este folosit pentru a trece de la un cadru la altul.

Mecanismul prin care sunt alese cadrele este unul cu elemente aleatoare și poate fi descris astfel: pentru fiecare videoclip se determină numărul de cadre, iar acest număr de cadre este împărțit într-un număr arbitrar intervale egale; selecția cadrelor se face aleator din intervalul format de reuniunea intervalelor de mijloc. După o selecție, cadrul este eliminat din mulțimea de extragere pentru a nu putea fi ales de două ori.

```

# HOG based face detector
face_detector = dlib.get_frontal_face_detector()

# Create a list of starting points in the video to sample from
frame_bins = np.linspace(1, num_frames, num_bins, dtype=int)

for _ in range(sample_size):

    # Select only a middle section from the video
    candidate_frame_set = list(range(frame_bins[2], frame_bins[-2] + 1))

    # Randomly select the frame to sample
    # Pop it from the list so it can't be sampled multiple times
    start_frame = random.choice(candidate_frame_set)
    candidate_frame_set.remove(start_frame)

    # Set the starting point at the selected sampled frame
    reader.set(cv2.CAP_PROP_POS_FRAMES, start_frame)

```

Figura 5.2: Algoritmul de selecție al cadrelor

#### 5.1.4 Extragerea fețelor

Pentru a concentra toată puterea de învățare a modelului asupra înfățișării protagonistului, cadrele în care apar actorii nu sunt suficiente. De aceea, a fost necesară extragerea fețelor actorilor principali, lucru ce s-a realizat cu ajutorul unei funcții din modulul dlib, mai exact `get_frontal_face_detector()`. Funcția folosește un model de inteligență artificială pre-antrenat, detecția fețelor fiind realizată cu ajutorul histogramelor de gradienti(HOG) introduse în literatură de către Navneet Dalal și Bill Trigg în 2005[dalal2005histograms].

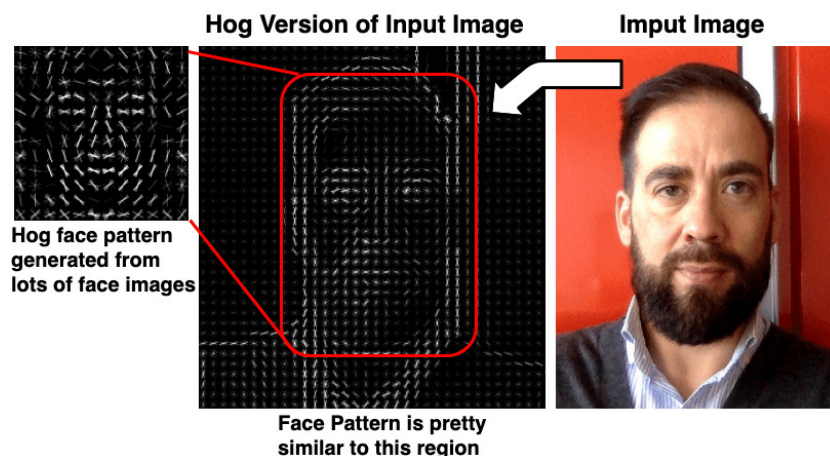


Figura 5.3: Histograme de gradienti(HOG)[variz\_\_2024]



Histogramele de gradienti funcționează doar pentru imagini cu un singur canal de culoare, adică în tonuri de gri. Fiecare cadru extras din videoclipuri este convertit corespunzător, iar mai apoi este folosit ca parametru pentru funcția ce folosește modelul.

```
# Detect faces
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
detected_faces = face_detector(gray, 1)
```

Figura 5.4: Descriptorii HOG sunt folosiți pe imagini gri

Detectorul facial folosește imaginea în tonuri de gri pentru a determina coordonatele patrulaterului ce încadrează fața detectată. Acesta returnează valorile pixelilor ce descriu fața din imagine, care mai apoi sunt folosite pentru a decupa imaginea principală și a extrage fața.



Figura 5.5: Imaginea originală



Figura 5.6: Fața decupată din imagine

## 5.2 Selecția arhitecturilor

Având la dispoziție setul de date creat, următorul pas a fost selectarea unui model de inteligență artificială care se pliază pe scopul aplicației finale și în același timp să aibă un număr de parametri proporțional cu puterea de calcul a sistemului folosit.

Când vine vorba de sarcini ce includ utilizarea de imagini, arhitecturile convoluționale și, mai nou, cele de tip vision transformer, sunt opțiunile cele mai performante. Pentru ca timpul de antrenare și testare să fie fezabil, pentru această aplicație decizia a fost ca modelul ales să fie o rețea neuronală convoluțională.

Un alt aspect important al antrenării modelelor de inteligență artificială este utilizarea de modele pre-antrenate pe alte seturi de date ca punct de plecare pentru învățarea unui set de date. Acest proces se numește **transfer learning**. Este des întâlnit în literatură deoarece s-a dovedit a fi foarte eficient și utilizarea acestui procedeu este facilitată de disponibilitatea în framework-urile de învățare automată a mai multor modele pre-antrenate des folosite.

### 5.2.1 ResNet

ResNet este o arhitectură convoluțională introdusă de către Kaiming He et al. în anul 2016[**he2016deep**] și a reprezentat o revelație în vederea artificială, câștigând detașat competiția ImageNet.

ResNet este considerată o arhitectură puternică datorită mai multor caracteristici inovatoare și avantaje pe care le oferă. Una dintre principalele inovații aduse de ResNet este utilizarea blocurilor reziduale. Aceste blocuri permit antrenarea rețelelor neuronale foarte adânci fără a suferi de problema gradientilor care dispar (vanishing gradients), o problemă comună în rețelele adânci care face ca gradientii să devină foarte mici, îngreunând astfel antrenarea eficientă.

Blocurile reziduale folosesc o conexiune directă (skip connection) între straturi, ceea ce permite ca informația să fie transmisă direct peste mai multe straturi. Acest lucru ajută modelul să păstreze informațiile esențiale și să învețe mai eficient caracteristicile relevante ale datelor.

ResNet este, de asemenea, flexibilă și scalabilă. Arhitectura sa permite con-

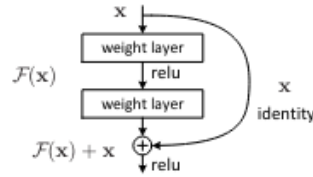


Figura 5.7: Bloc rezidual[he2016deep]

struirea unor rețele foarte adânci, cum ar fi ResNet-50, ResNet-101 sau chiar mai adânci, fără a compromite performanța. Această adâncime suplimentară permite modelului să capteze și să învețe caracteristici extrem de complexe și detaliate din datele de antrenament.

Pentru setul de date disponibil, arhitectura aleasă a fost ResNet-50, un model cu 25M(25 de milioane) de parametri, având și un model pre-antrenat pe setul de date ImageNet, disponibil în [PyTorch](#).

### 5.2.2 EfficientNet

Modelul EfficientNet a fost propus de către Mingxing Tan și Quoc V. Le în anul 2019 [tan2019efficientnet] printr-o lucrare care avea ca scop oferirea unei analize și al unui mod eficient de a scala rețelele neuronale convoluționale. Aceasta se realizează prin ajustarea simultană a dimensiunilor de adâncime, lățime și rezoluție a rețelei într-un mod coordonat și echilibrat. Spre deosebire de metodele tradiționale care măresc doar una dintre aceste dimensiuni, abordarea EfficientNet optimizează utilizarea resurselor și îmbunătățește performanța generală.

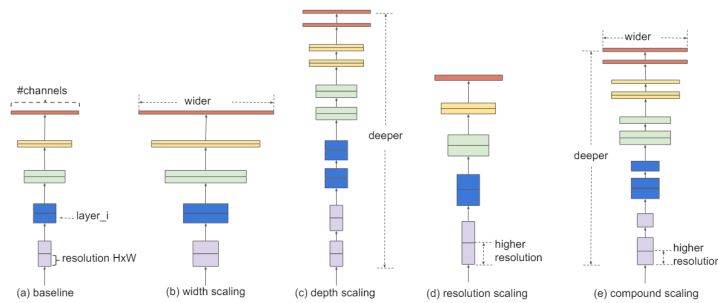


Figura 5.8: EfficientNet folosește o scalare eficientă a rețelei[tan2019efficientnet]

Arhitectura este, de asemenea, foarte eficientă din punct de vedere computațional. Datorită designului său optimizat EfficientNet este capabilă de performanță folosind mai puține resurse computaționale față de alte modele mai adânci. Acest lucru face ca EfficientNet să fie o alegere excelentă pentru aplicații unde resursele de calcul sunt limitate.

### 5.2.3 RegNet

RegNet este o arhitectură dezvoltată cu scopul de a fi antrenată într-un mod eficient și scalabil. Modelul a fost introdus de către departamentul de inteligență artificială de la Facebook(FAIR)[radosavovic2020designing] și se remarcă prin simplitatea și performanțele sale. RegNet reușește să aibă rezultate competitive cu alte arhitecturi, fiind mult mai simplă de implementat și înțeles.

Arhitectura este compusă din mai multe etape, fiecare conținând o serie de blocuri. Această abordare pe etape permite rețelei să crească treptat în adâncime și complexitate. În cadrul acestor etape, RegNet folosește blocuri de tip bottleneck, similare cu cele utilizate în arhitecturile ResNet, care ajută la menținerea eficienței computaționale pe măsură ce adâncimea rețelei crește.

O caracteristică ce scoate în evidență arhitectura este utilizarea spațiilor de design, introduse ca noutate în lucrare, un concept care permite explorarea și optimizarea diferitelor configurații de rețea. Prin utilizarea de diverși parametri, cum ar fi numărul de canale (lățimea) și numărul de straturi (adâncimea), cei care proiectează rețeaua pot identifica cele mai eficiente modele fără să mai treacă prin încercări repetate.

În ceea ce privește performanța, modelele RegNet au rezultate competitive pe setul de date ImageNet, capabile să concureze cu arhitecturi mai complexe. În plus, design-ul RegNet asigură eficiență computațională, ceea ce îl face folositor pentru dezvoltarea de modele capabile cu resurse limitate.

### **5.3 Experimente și rezultate**

### **5.4 Creșterea performanței cu ajutorul Test Time Augumentation(TTA)**

# Bibliografie

- [1] Bruce Blaus, *Multipolar neuron*, <https://en.wikipedia.org/wiki/Neuron>, Accesat: 7 Mai 2024, 2013.
- [2] Matěj Fanta, „Rules extraction from deep neural networks Master thesis”, Teză de doct., Aug. 2019, DOI: [10.13140/RG.2.2.22319.28324](https://doi.org/10.13140/RG.2.2.22319.28324).
- [3] Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, ed. de Nicole Taché și Michele Cronin, O'Reilly Media, 2022.
- [4] *Gradient Descent*, <https://pantelis.github.io/cs634/docs/common/lectures/optimization/sgd/>, Accesat: 9 Mai 2024.
- [5] Geoffrey E Hinton și Ruslan R Salakhutdinov, „Reducing the dimensionality of data with neural networks”, în *science* 313.5786 (2006), pp. 504–507.
- [6] Warren S McCulloch și Walter Pitts, „A logical calculus of the ideas immanent in nervous activity”, în *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.
- [7] Marvin Minsky și Seymour Papert, „An introduction to computational geometry”, în *Cambridge tiass.*, HIT 479.480 (1969), p. 104.
- [8] *Perceptron Definition*, <https://images.deepai.org/glossary-terms/perceptron-6168423.jpg>, Accesat: 8 Mai 2024.
- [9] Frank Rosenblatt, „The perceptron: a probabilistic model for information storage and organization in the brain.”, în *Psychological review* 65.6 (1958), p. 386.

- [10] David E Rumelhart, Geoffrey E Hinton și Ronald J Williams, „Learning representations by back-propagating errors”, în *nature* 323.6088 (1986), pp. 533–536.
- [11] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt și Matthias Nießner, „Face2face: Real-time face capture and reenactment of rgb videos”, în *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2387–2395.