Departamentul Automatică și Informatică Industrială
Facultatea Automatică și Calculatoare
Universitatea POLITEHNICA din București

LUCRARE DE DISERTAȚIE

Detectarea și segmentarea zonelor din imagini aeriene utilizând rețele generativ adversale

Coordonator
Prof. Dr. Ing. Dan Popescu

Absolvent
Dimoiu Mihai – Lorin

Anul absolvirii **2020**

Cuprins

1. Introducere

1.1. Obiective

În această lume aflată într-o continuă expansiune, zilnic sunt generate aproximativ 2.5 milioane de teraocteți de date. În acest sens Data Science și Machine Learning pot schimba complet modul de lucru și traiul oamenilor.

Zi de zi sunt capturate volume enorme de imagini aeriene sau din spațiu, iar acest volum este în continuă creștere. Respectivele imagini fac ca interpretarea manuală să devină prohibitivă, prin urmare, trebuie să folosim algoritmi de vedere artificială. Pasul fundamental al mapării automate este alocarea unei clase semantice fiecărui pixel, adică transformarea datelor brute într-o hartă raster (care poate fi prelucrată ulterior după caz, de exemplu, cu ajutorul tehnicilor de vectorizare sau generalizare a hărții). Detectarea automată a punctelor de interes din imaginile aeriene sau din satelit este o mare provocare.

Astfel învățarea supravegheată este cea mai populară tehnică. Pentru a defini clasele specifice sarcinii în analiza statistică este necesară adnotarea către o persoană cu date de antrenare în urma supravegherii.

În cele mai multe cazuri, datele de referință pentru formarea clasificărilor sunt generate manual pentru fiecare caz nou, ceea ce reprezintă un proces care necesită mult timp și este extrem de costisitor. Adnotarea manuală trebuie repetată de fiecare dată când sarcina, locația geografică, caracteristicile senzorului optic sau condițiile de imagistică se schimbă, prin urmare procesul de scalare este slab.

Aceste date sunt colectate de obicei folosind două surse principale:

1. Voluntarii colectează date fie de la fața locului cu trackere GPS, fie digitalizând manual imagini care au fost colectate fie aerian sau din satelit de înaltă rezoluție.
2. Agențiile naționale de cartografie colectează imagini aeriene sau din satelit pentru a le pune la dispoziția unui public mai larg. Abordarea în această lucrare poate fi văzută ca o formă de adnotare de date.
3. Institute de cercetare națională.

Volumul pur de date de învățare poate compensa o precizie mai scăzută daca este folosită o metodă adecvată, robustă de învățare.

Varietatea mare prezentă în seturi de învățare (de exemplu, imagini care acoperă mai mulți kilometrii pătrați) ce ar putea îmbunătăți capacitatea rețelei neuronale în a generaliza imagini noi, nevăzute.

Datele disponibile de antrenare de o calitate înaltă și volumul mare de date suplimentare de antrenare ar putea îmbunătăți rețeaua neuronală.

Rețelele neuronale sunt în prezent cea mai performantă metodă pentru etichetarea semantică a imaginilor de înaltă rezoluție și consider că ele să facă baza studiului în această lucrare.

Odată ce datele de antrenare sunt disponibile, reînvățarea este complet automată pentru diferite tipuri de senzori sau condițiile meteorologice, fără interacțiunea specifică a utilizatorului, cum ar fi preprocesarea imaginilor capturate.

Deoarece evaluările cantitative pentru aceste seturi de date care sunt foarte mari sunt limitate de inexactitatea etichetelor și sunt prezente și în setul de teste, efectuăm și experimente pentru un set de date mai mic.

De asemenea evaluăm capacitățile modelelor în ceea ce privește generalizarea și transferul învățării între locații geografice nevăzute.

Studii similare vor fi efectuate cu vaste imagini proprietare și hărți deținute de autorități și furnizori de sateliți comerciali. În cele din urmă, acesta este un pas mai aproape de viziunea utopică care ne va furniza o serie întreagă de sarcini de mapare care nu mai au nevoie de input-uri de la utilizator, ci pot fi complet automatizate.

1.2.Motivație

Odată cu apariția condițiilor climatice drastice, inundațiile sunt unul dintre cele mai devastatoare pericole naturale, frecvent întâlnite, iar acestea produc repercusiuni grave asupra economiei.

O inundație este o revărsare de apă care scufundă terenurile care sunt de obicei uscate. Inundațiile reprezintă un domeniu de studiu al disciplinei hidrologie și prezintă o preocupare semnificativă în agricultură, inginerie civilă și sănătate publică.

Inundațiile se pot întâmpla pe zonele plane sau joase, când apa este furnizată de ploi sau de zăpadă mai rapid decât se poate infiltra în pământ sau curge. Excesul se acumulează pe loc, uneori până la adâncimi periculoase. Solul de suprafață devine saturat, ceea ce oprește infiltrarea în pământ într-un mod eficient. Infiltrarea este lentă spre neglijabilă prin solul înghețat, beton, pavaj, etc. Inundarea areală începe în zonele plane, precum inundațiile și în depresiunile locale neconectate la un canal de scurgere.

Inundațiile urbane reprezintă inundarea de terenuri sau proprietăți într-un mediu construit, în special în zonele populate, cauzate de precipitații. În zonele urbane, efectele inundațiilor pot fi agravate de străduțele și drumurile pavate, care cresc viteza de curgere a apei. Fluxul de inundații în zonele urbanizate reprezintă un pericol extrem de mare pentru populație cât și pentru infrastructură.

Modul de abordare și a tehnologiilor vor aduce îmbunătățiri activităților ce vizează identificarea, analiza și evaluarea riscurilor pentru fenomenele naturale ce declanșează dezastre așa cum sunt inundații.

1.3.Propuneri noi

Prin această lucrare propun să aduc nou, segmentarea pe imagini color capturate din aer sau din satelit, folosind rețele neuronale generativ adversale și având un set de date inițial foarte mic.

2. Stadiul actual

Prin segmentare semantică se înțelege procesul de conectare a fiecărui pixel dintr-o imagine de input la fiecare pixel din imaginea etichetată. Aceste etichete ar putea include: vegetație, case, mașini, ape, etc. Putem spune că segmentarea semantică este o clasificare a imaginii la nivel de pixeli. Spre exemplu, o imagine care are mai multe case, segmentarea va eticheta toate obiectele ca obiecte imobiliare.

Unele dintre aplicațiile principale ale segmentării sunt: vehiculele autonome, interacțiunea om-calculator, cartografiere, etc. De exemplu, segmentarea semantică este foarte importantă în vehiculele autonome, pentru că este crucial ca modelele să înțeleagă contextul în mediul în care își desfășoară activitatea.

În continuare vom prezenta lucrări actuale din domeniul rețelelor neuronale:

- U-Net: Convolutional Networks for Biomedical Image Segmentation

În procesarea imaginilor pentru biomedicină, este foarte crucial să se obțină o etichetă de clasă pentru fiecare celulă din imagine. Cea mai mare provocare în segmentarea imaginilor din medicină este aceea că imaginile de antrenare nu se găsesc foarte ușor.

Lucrarea [1] se bazează pe o rețea complet conectată. Deoarece sunt puține imagini de antrenare, acest model folosește mărirea datelor aplicând deformări asupra imaginilor de input. Antrenarea pentru acest model se face cu imagini de intrare, imagini segmentate ale acestora și cu un gradient stocastic. Augmentarea datelor este necesară atunci când sunt utilizate foarte puține date de antrenare. Acest model a obținut o performanță IoU de 92%.

- DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs

În această lucrare [2], autorii aduc următoarele contribuții la segmentarea semantică:

1. Convoluții cu filtre eșantionate pentru predicții dense.
2. Acumularea spațială pentru segmentarea obiectelor la scări multiple (ASPP).
3. Îmbunătățirea localizării limitelor obiectelor prin utilizarea DCNN-urilor.

- FastFCN: Rethinking Dilated Convolution in the Backbone for Semantic Segmentation

Lucrarea [3] propune un model de prelevare de probe comun numit Joint Pyramid Upsampling (JPU) pentru a înlocui convoluțiile dilatate care consumă mult timp și memorie.

Această metodă atinge o performanță de IoU de 53.13% pe setul de date Pascal Context și rulează de 3 ori mai rapid.

- Gated-SCNN: Gated Shape CNNs for Semantic Segmentation

Această lucrare este cea mai nouă din blocul de segmentări semantice [4]. Autorii propun o arhitectură CNN în două fluxuri. În această arhitectură, informațiile despre forme sunt procesate pe o ramură separată. Acest flux de forme procesează doar informații legate de margini. Acest lucru este pus în practică prin Gated Convolution Layer (GCL) și supravegherea locală a modelului.

Acest model depășește modelul DeepLab cu 1.5% pe IoU și 4% în testul F1. Modelul a fost evaluat folosind setul de date Cityscapes [5].

- StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation

Lucrarea [6] introduce o arhitectură nouă pentru translatarea de imagine la imagine, numită StarGAN. Această arhitectură utilizează un singur generator și un discriminator. Modelul poate translata imagini din mai multe domenii și să învețe din mai multe seturi de date cu diferite tipuri de informații. Experimentele arată că abordarea sugerată depășește semnificativ modelele de bază în transferul atributelor.

- AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks

Cercetătorii introduc o GAN pentru sintetizarea imaginilor din descrieri de text [7]. Modelul este format din 2 componente: rețeaua generativă pentru a observa subregiuni diferite ale imaginii prin focalizarea pe cuvinte relevante pentru subregiunea corespunzătoare și un model de similitudine multimodală (DAMSM) pentru a calcula similaritatea dintre imaginea generată și descrierea textului.

- High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs

În lucrarea [8], NVIDIA introduce o rețea de tip GAN pentru sintetizarea imaginilor foto-realiste de înaltă rezoluție și anume de 2048x1024 din imagini etichetate semantic. Ei își bazează abordarea antrenarea robustă cu un adversar, precum și pe noi arhitecturi generatoare și discriminatoare pe mai multe nivele. Această nouă abordare depășește semnificativ modelele anterioare în prezicerea segmentării semantice. Mai multe de atât, cercetătorii își extind cadrul pentru a susține manipulări semantice interactive, de exemplu schimbarea categoriei de obiecte sau adăugarea și eliminarea obiectelor sau schimbarea culorii și a texturii obiectelor din imagine.

- Large Scale GAN Training for High Fidelity Natural Image Synthesis

Echipa DeepMind constată că tehnicile actuale sunt suficiente pentru sintetizarea imaginilor de înaltă rezoluție [9]. Această rețea GAN poate genera imagini care arată foarte realist dacă este antrenată la o scară foarte mare. Aceste GAN-uri la scară largă numite și BigGAN-uri, sunt noul stadiu al tehnicii în sinteza imaginilor condiționate. Acest tip de GAN funcționează mult mai bine cu dimensiunea mare a setului de date și cu număr mare de parametrii.

- A Style-Based Generator Architecture for Generative Adversarial Networks

NVIDIA introduce o nouă arhitectură de generator StyleGAN. În această cercetare [10], au abordat problema controlului foarte limitat asupra imaginilor generate cu arhitecturi tradiționale GAN. Un generator în StyleGAN învață să separe diferite aspecte ale imaginii fără nicio supraveghere, ceea ce face posibilă combinarea acestor aspecte în mai multe moduri diferite. De exemplu, putem lua vârsta, lungimea părului, ochelarii de la o persoană și le putem transfera la o altă persoană. Imaginile rezultate depășesc stadiul tehnicii anterioare din punct de vedere al calității și realismului.

- LOGAN: Latent Optimisation for Generative Adversarial Networks

Cercetătorii susțin că optimizarea variabilei latente "z" în timpul antrenării poate îmbunătăți de fapt dinamica modelelor de GAN [11]. Noua metodă de formare pe care o propun, utilizează gradientul natural pentru a optimiza scorul latent la fiecare iterație de antrenament. De fapt, procedura de antrenare rămâne aceeași în timpul propagării. În cel de-al doilea pas al propagării, acest gradient este folosit pentru a optimiza variabila latentă. Noua metodă îmbunătățește rezultatele de la ultima generație ale BigGAN. Potrivit cercetătorilor, optimizarea latentă care produce eșantioane de calitate superioară este mult mai realistă și mai diversă.

3. Metodologie

3.1.Segmentarea imaginilor

Primul lucru pe care o persoană îl face când încearcă să traverseze un drum este de obicei, asigurarea din ambele părți pentru prevenirea unui accident, făcând un bilanț al autovehiculelor de pe drum și va lua o decizie. Creierul este capabil să analizeze, într-o fracțiune de milisecundă, ce fel de vehicul (mașină, autobuz, camion, etc.) vine spre noi. Inteligența artificială poate face acest lucru? Până acum câțiva ani nu se putea din cauza lipsei puterii de calcul mare.

Segmentarea imaginilor semantice este sarcina de a clasifica fiecare pixel într-o imagine dintr-un set predefinit de clase. De obicei, într-o imagine cu diverse entități, vrem să știm ce pixel aparține entității, de exemplu într-o imagine aeriană, putem segmenta pământul, copacii, apa, etc.

Segmentarea semantică este diferită de detectarea obiectelor, deoarece nu prezice niciun pătrat de delimitare în jurul obiectelor. Nu distingem între instanțe diferite ale aceluiași obiect. De exemplu ar putea exista mai multe mașini în scenă și toate ar avea aceeași etichetă.

Pentru a realiza segmentarea semantică, este necesară o mai bună înțelegere a imaginii. Algoritmul ar trebui să dea seama de obiectele prezente și de pixelii care corespund obiectului. Segmentarea semantică este una dintre greutățile pentru înțelegerea completă a scenei.

3.1.1. Segmentare bazată pe regiune

Cea mai populară abordare se bazează pe identificarea obiectelor prezente într-o imagine. Putem împărți sau partiționa imaginea în diferite părți numite segmente. Împărțind imaginea în segmente, putem folosi segmente importante pentru procesul de învățare a conținutului din imagine.

O imagine este o colecție sau un set de pixeli diferiți. Grupăm pixelii care au atribute similare folosind segmentarea imaginii.

Segmentarea imaginii creează o mască de pixeli pentru fiecare clasa din imagine. Această tehnică ne oferă o înțelegere mult mai granulară a claselor (obiectelor) din imagine.

Tehnicile de segmentare a imaginilor au un impact masiv. Ele ne ajută să abordăm această problemă într-o manieră mai granulară și să obținem rezultate mai semnificative.

Există multe alte aplicații în care segmentarea de imagini transformă industriile:

1. Autovehicule autonome.
2. Sisteme de control al traficului.
3. Localizarea obiectelor din imagini aeriene sau din satelit.

O modalitate simplă de a segmenta diferite obiecte ar putea fi utilizarea valorilor de pixeli. Valorile pixelilor vor fi diferite pentru obiecte și fundalul imaginii dacă exista un contrast puternic între ele. În acest caz, putem seta o valoare de prag. Valorile pixelilor care se încadrează sub sau peste acest prag pot fi clasificate în consecință ca obiect sau fundal. Această tehnică este cunoscută sub denumirea de segmente de prag. Segmentarea se poate vedea în Fig. 1:

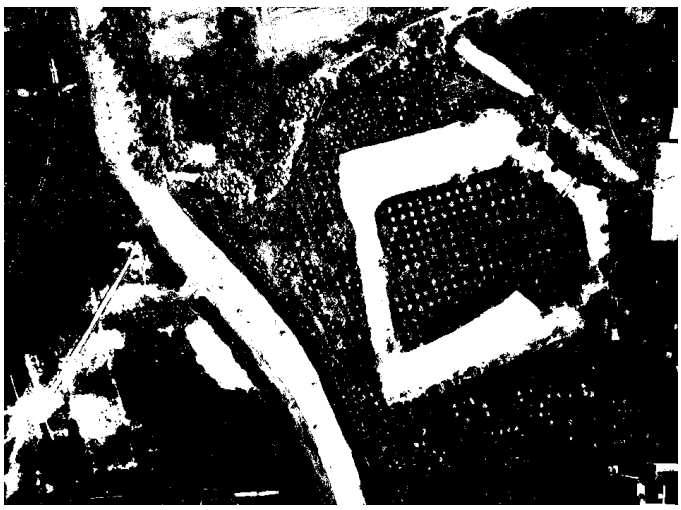


Fig. 1: Rezultatul aplicării segmentarea de prag

Prin împărțirea imaginii în două regiuni (obiect și fundal), definim o singură valoare de prag iar pentru Fig. 1 s-a ales valoarea 127.

Dacă avem mai multe obiecte împreună cu fundalul, trebuie să definim mai multe praguri. Aceste praguri sunt cunoscute ca prag local. Astfel avem Fig. 2:

Fig. 2: Rezultatul aplicării mai multor praguri

Programele folosite pentru realizarea celor două imagini, Fig. 1, respectiv Fig. 2, sunt următoarele:

```python
import cv2

fileName = "../Imagini_Input/Train_Images/DSC05043r.jpg"

black_white = 0
red_green_blue = 1

def readImage(name, colorType):
    img = cv2.imread(name, colorType)
    return img

def writeImage(fileName, file):
    cv2.imwrite(fileName, file)

img = readImage(fileName, black_white)

range_i, range_j = img.shape

for i in range(range_i):
    for j in range(range_j):
        if img[i][j] > 127:
            img[i][j] = 255
        else:
            img[i][j] = 0
writeImage("binary_mask.bmp", img)
```

```python
import cv2

fileName = "../Imagini_Input/Train_Images/DSC05043r.jpg"

black_white = 0
red_green_blue = 1

def readImage(name, colorType):
    img = cv2.imread(name, colorType)
    return img

def writeImage(fileName, file):
    cv2.imwrite(fileName, file)

img = readImage(fileName, black_white)

prag_1 = 50
prag_2 = 100
prag_3 = 150
prag_4 = 200
prag_5 = 255

range_i, range_j = img.shape

for i in range(range_i):
    for j in range(range_j):
        if img[i][j] >= 0 and img[i][j] < prag_1:
            img[i][j] = 25
        if img[i][j] >= prag_1 and img[i][j] < prag_2:
            img[i][j] = 75
        if img[i][j] >= prag_2 and img[i][j] < prag_3:
            img[i][j] = 125
        if img[i][j] >= prag_3 and img[i][j] < prag_4:
            img[i][j] = 175
        if img[i][j] >= prag_4 and img[i][j] < prag_5:
            img[i][j] = 225

writeImage("binary_mask.bmp", img)
```

Fig. 3: Programele pentru generarea Fig. 1 și Fig. 2

În Fig. 2, există 5 segmente diferite după cum se poate observa și în Fig. 3. Se pot seta valori diferite pentru praguri.

Câteva avantaje ale aceste metode sunt:

1. Calculele sunt mai simple.
2. Viteza de operare rapidă.
3. Când obiectul și fundalul au un contrast ridicat, această metodă se comportă foarte bine.

Există însă câteva limitări ale aceste abordări. Când nu avem o diferență semnificativă la nivel de gri sau dacă există o suprapunere a valorilor pixelilor pe scara de gri, va deveni foarte dificil să obținem o segmentare precisă.

3.1.2. Segmentare bazată pe detectarea marginilor

Există întotdeauna o margine între două regiuni învecinate cu valori diferite în scară de gri (valori ale pixelilor). Marginile pot fi luate în considerare ca fiind caracteristici locale pentru o imagine.

Se poate folosi aceste caracteristici pentru a detecta marginile. Acest lucru ajută la detectarea formelor mai multor obiecte prezente într-o imagine dată. Se poate detecta folosind filtre și convoluții.

Un filtru este operatorul Sobel [12]. Este utilizat pentru a detecta marginile. Operatorul Sobel este compus din două matrice, una pentru detectarea marginilor orizontale și cealaltă pentru detectarea marginilor verticale. Observăm acest lucru în Fig. 4:

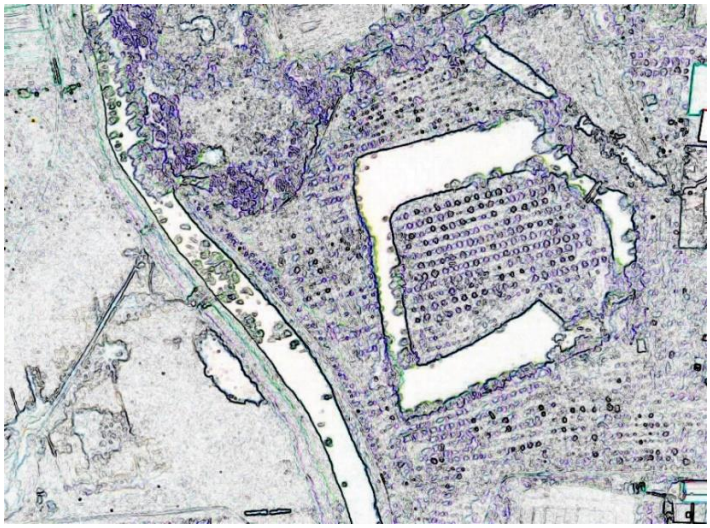


Fig. 4: Segmentarea bazată pe detectarea marginilor

Operatorul Sobel este utilizat în procesarea imaginilor și vedere artificială, în special în algoritmi de detecție a marginilor. Acest filtru este numit după Irwin Sobel și Gary Feldman. Operatorul utilizează două nuclee de 3x3 care sunt puse în convoluție cu imaginea originală pentru

a calcula aproximările derivatelor, unul pentru modificări orizontale și unul pentru modificări verticale. Acest filtru detectează discontinuitățile din imagine.

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \; G_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Ecuația 1: Nucleul vertical și orizontal pentru operatorul Sobel

3.1.3. Segmentarea bazată pe clustering (grupare)

Clustering-ul este sarcina de a împărți datele într-un număr de grupuri, astfel încât datele din același grup (cluster) sunt similare cu alte date din același grup decât cele din alte grupuri. Aceste grupuri sunt cunoscute sub numele de clustere.

Unul dintre algoritmii cei mai folosiți de grupare este K-Means. Parametrul K reprezintă numărul de grupuri.

Algoritmul K-Means funcționează în următorul mod:

1. Se selectează aleator K grupuri inițiale.
2. Se alocă în mod aleator fiecare data oricărui grup.
3. Se calculează centru fiecărui grup.
4. Se calculează distanța tuturor punctelor din centrul fiecărui grup .
5. În funcție de distanță, punctele se realocă în cel mai apropiat grup.
6. Se calculează noul centru al fiecărui grup.
7. Se vor repeta pașii 3, 4, 5, 6, până când centrul grupurilor nu se mai schimbă sau ajungem la un număr maxim de iterații.

În Fig. 5, se poate observa rezultatul aplicării algoritmului K-Means asupra unei imagini cu alegerea a 5 grupuri:

Fig. 5: Aplicarea algoritmului K-Means

3.1.4. Segmentarea folosind rețele neuronale

Rețelele neuronale de tip GAN au arătat un rezultat deosebit în multe sarcini generative pentru a reproduce un conținut nou. Acest model de rețea este inspirat din teoria jocurilor: două modele, unul generator și unul discriminator, care concurează între ele, în același timp făcându-se reciproc mai puternice.

Translatarea dintr-o imagine în altă imagine reprezintă o clasă din probleme de vedere artificială și grafică în care obiectivul este de a învăța maparea între o imagine de intrare și o imagine de ieșire. Acest lucru poate fi aplicat într-o gamă largă, de exemplu, segmentarea unei imagini, îmbunătățirea calității unei imagini, etc.

Generatorul are rolul de a lua o imagine de intrare și de a efectua transformarea pentru a produce imaginea dorită. Un exemplu de intrare ar putea fi o imagine color, iar la ieșire apare o imagine segmentată. Structura generatorului se numește "codificator – decodificator".

Discriminatorul are rolul de a lua două imagini, o imagine de intrare și o imagine necunoscută (rezultată din generator) și decide dacă imagine ieșită a fost produsă de către generator sau nu.

Fig. 6: Imaginea de input și rezultatul segmentării

3.2.Rețele neuronale

Unul dintre motivele pentru care rețelele neuronale au devenit atât de populare este capacitatea lor de a rezolva sarcini complexe fără cunoștințe în domeniu. Asta înseamnă ca aceeași arhitectură poate fi folosită pentru orice altă sarcină de segmentare a imaginii.

Imaginile aeriene sau din satelit reprezintă cazul perfect de utilizare pentru segmentarea imaginilor, deoarece ne oferă o mulțime de aplicații diferite.

Vehiculele autonome sunt un prim caz de utilizare. Siguranța este cel mai important factor atunci când se proiectează un vehicul autonom. Dar siguranța necesită o înțelegere a spațiului

înconjurător. Câteva lucruri pe care un vehicul autonom trebuie să le ia în considerare sunt lățimea drumului, trecerile de pietoni, poziția trotuarelor, etc.

Hărțile actuale nu furnizează aceste informații iar capturarea imaginilor prin diverse locuri este costisitoare. Analizând imagini din satelit sau aerian, putem construi hărți de înaltă rezoluție care în viitor pot ajuta autovehiculele autonome să fie conduse mai în siguranță.

Detectarea inundațiilor este o altă aplicație interesantă. Chiar dacă, imaginile aeriene nu sunt disponibile în timp real, poziția inițială este cunoscută și putem furniza informații esențiale despre pagubele produse de inundații, zona de întindere a inundațiilor, adâncimea lor [13].

Mai mult decât atât, achiziționarea mai multor imagini din zile / săptămâni / luni diferite ne permite să estimăm intensitatea pagubelor din zonele urbane / rurale afectate.

3.2.1. Rețele convoluționale

O rețea neuronală constă într-un număr de straturi convoluționale urmate de straturi dense care produc o probabilitate de apariție.

Putem folosi o idee oarecum similară pentru a clasifica fiecare pixel din imagine. Un pixel poate fi clasificat în una din cele 9 categorii și anume: apă inundație, copaci verzi și vegetație, pământ, case, alee de ciment, drum de asfalt, mașini, drum de pământ, alee de pământ.

Aceste 9 clase (categorii) au următoarele culori:

Fig. 7: Clasele folosite

Fig. 7 s-a realizat utilizând secvența de cod din Fig. 8:

```python
import numpy as np
from matplotlib import pyplot

def create_blank(size = (100, 100, 3), rgb_color=(0, 0, 0)):
    image = np.zeros(size, np.uint8)
    image[:] = rgb_color
    return image

def hex_to_rgb(value):
    value = value.lstrip('#')
    lv = len(value)
    return tuple(int(value[i:i + lv // 3], 16) for i in range(0, lv, lv // 3))

size = (50, 50, 3)

hex_colors = {
    "#be601f": "apa inundatie",
    "#087715": "copaci verzi + vegetatie",
    "#633b08": "pamant",
    "#ff0000": "case",
    "#929292": "alee ciment",
    "#c6c6c6": "asfalt drum",
    "#fff600": "masini",
    "#515151": "drum de pamant",
    "#594a39": "alee de pamant"
    }

color_position = 1
for key, value in hex_colors.items():
    pyplot.subplot(3, 3, color_position)
    pyplot.axis("off")
    pyplot.title(value, y=-0.23)
    pyplot.imshow(create_blank(size, hex_to_rgb(key)))
    color_position += 1

filename = "colors_plot.png"
pyplot.savefig(filename)
```

Fig. 8: Secvența de cod utilizată pentru generarea celor 9 culori utilizate claselor

Așadar pentru a face acest lucru, în mod eficient, dorim să clasificăm fiecare pixel în toate clasele în același timp. Cel mai simplu mod de a realiza acest lucru este de a avea "n" straturi convoluționale, urmate de un strat convolutiv final în care numărul de filtre este egal cu numărul de clase pe care dorim să le prezicem. În cazul nostru 9 filtre deoarece avem 9 clase.

Această arhitectură ar arăta astfel ca în Fig. 9:

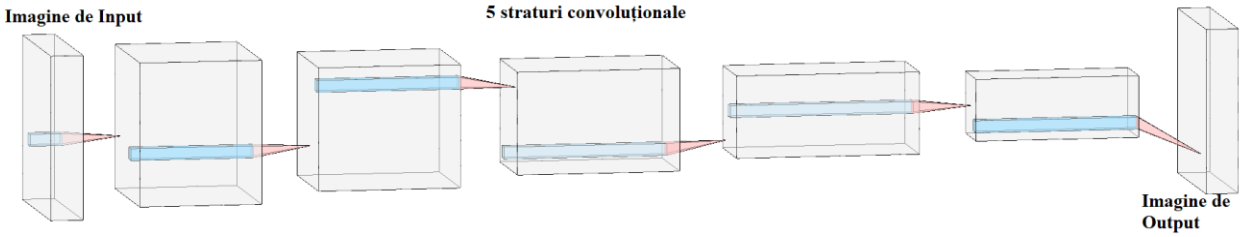


Fig. 9: Arhitectura rețea convoluțională – realizat în [14]

Dezavantajul acestei arhitecturi este că de fiecare dată când avansăm de la un strat la altul, putem să surprindem doar regiunile locale, deoarece filtrele pe care le folosim au dimensiune de 4x4.

Pentru ca rețeaua să înțeleagă obiecte mai complexe, precum, un drum, mașini, case, trebuie să poată combina valorile pixelilor din întreaga imagine. Fără această combinare, o bucată de casă care este obstrucționată de un copac, nu poate fi recunoscută corect de rețea.

Rezolvarea acestei probleme se va face printr-o versiune modificată a unui auto codificator. Un auto codificator reduce dimensiunea caracteristicilor din fiecare strat până când toate informațiile necesare sunt comprimate într-un spațiu mai mic. Acest proces este cunoscut de reducerea dimensionalității. Apoi se inversează procesul pentru a genera o nouă imagine cu dimensiunile imaginii inițiale.

Reducerea dimensionalității este procesul de reducere a numărului de variabile aleatorii care conduc la obținerea unui set de variabile principale. Abordările încearcă să găsească un subset de variabile interne. Cele trei strategii sunt:

1. Strategia de filtrare: câștig de informații.
2. Strategia de înfășurare: căutare ghidată de către acuratețe.
3. Strategia încorporată: adăugare sau eliminare de funcții în timpul construirii modelului.

Acest proces ne oferă două avantaje:

1. Permite rețelei să combine elementele întregii imagini.
2. Reducerea de informații forțează rețeaua să păstreze doar informațiile esențiale. Acest lucru duce la o probabilitate de Over-Fitting mai mică.

După ce am redus dimensiunea de intrare, trebuie să creștem dimensiunea imaginii la dimensiunea inițială. Pentru fiecare strat convoluțional, vom folosi un strat de convoluție transpus pentru a face din nou imaginea. Așadar arhitectura arată în felul următor:

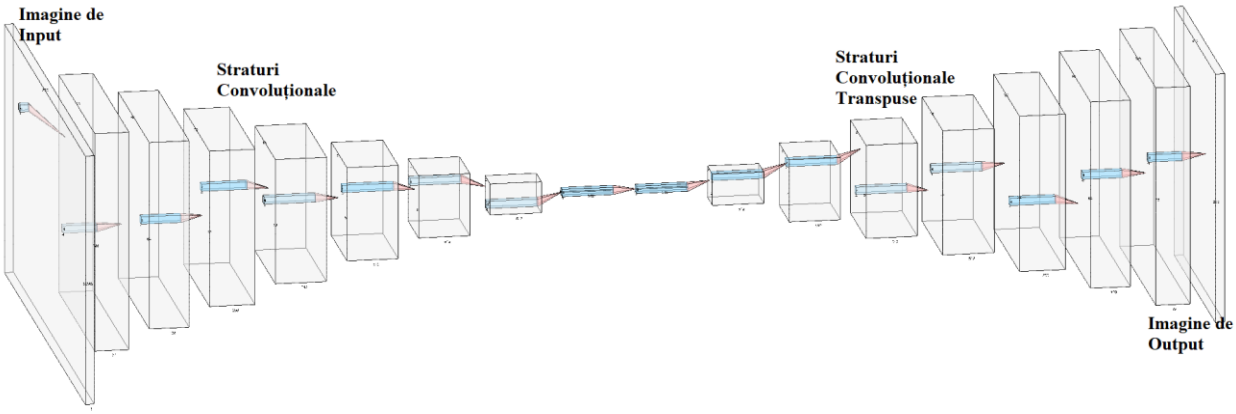


Fig. 10: Arhitectura unui auto codificator – realizat în [14]

Din Fig. 10 se poate vedea că se folosesc 7 straturi convoluționale, pentru a reduce dimensionalitatea, până când imaginea a ajuns comprimată într-o dimensiune de 1x1x512. În timp ce imaginea de input avea 256x256x3 valori (196,608), această comprimare are nevoie doar de 512 de valori. Prin acest mod se forțează rețeaua să găsească o reprezentare mai compactă. Aceeași

rețea s-a folosit și pentru imagini de 512x512x3, ce vor fi comprimate in 2x2x512. Acest lucru duce la un număr total de elemente de 786,432. Acest lucru face să crească efortul de calcul de 4 ori.

Prin folosirea straturilor de convoluție transpusă se restabilește imaginea de output. În comparație cu auto codificatoarele, singura inconvenientă este că imaginea de output sa fie identică cu imaginea de input, ci în schimb putem să introducem o funcție de cost care să impună ieșirii să semene cu imaginea de input.

3.2.2. Convoluția

În Fig. 11, avem o imagine de input de 4x4 la o ieșire de 2x2 folosind un filtru convoluțional de 3x3.

$$A[i,j] = (f * g)[i,j] = \sum_m \sum_n h[m,n] f[i-m, j-k]$$

Ecuația 2: Funcția matematică pentru convoluție

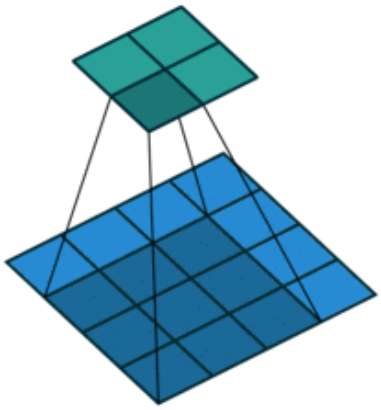


Fig. 11: Filtru convoluțional [15]

Convoluția ține cont de anumiți parametrii precum:

1. Numărul de filtre de ieșire din convoluție.
2. Dimensiunea unui filtru.
3. Pasul de convoluție (stride).
4. Generarea de spațiu în jurul imaginii (padding).

3.2.3. Convoluția transpusă

În Fig. 12, o imagine de input de 4x4 trece la o imagine de output de 16x16, folosind un filtru de convoluție transpus.

Convoluția transpusă ține cont de aceeași parametrii ca și convoluția descrisă mai sus și anume: numărul de filtre de ieșire din convoluție, dimensiunea unui filtru, pasul de convoluție numit stride și generarea de spațiu în jurul imaginii numit padding.

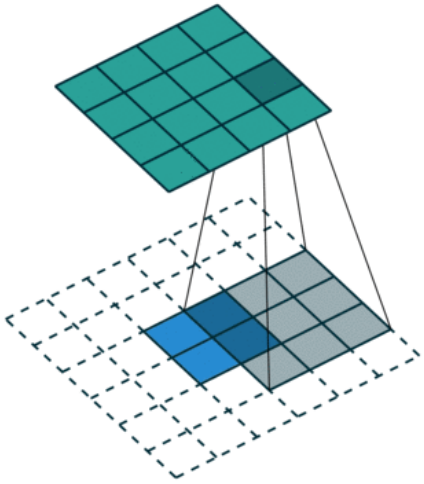


Fig. 12: Convoluție transpusă [15]

Până acum s-a creat un model care poate produce o imagine relativ decentă, dar cu o acuratețe medie. Forțăm rețeaua să comprime toate cunoștințele într-un spațiu mic. Apoi s-a extins imaginea până la dimensiunea inițială, dar deși rețeaua știe că ar trebui să fie o mașină sau casă aflată pe o latură, însă locația nu o cunoaște exact. Fie există posibilitatea mutării a câțiva pixeli a obiectului, fie nu se poate obține un contur complet, deoarece ar putea stoca doar o aproximare a mașinii sau clădirii dar nu și forma perfectă. Soluția este introducerea unei legături între straturi. Dăm voie rețelei să se uite la alte straturi. Acest lucru îi permite rețelei neuronale să realizeze că unele mașini sau case nu sunt perfect aliniate sau că forma unei anumite case nu este perfectă. Se pot schimba ușor mașinile sau adapta conturul caselor.

Arhitectura finală va arata ca în Fig. 13:

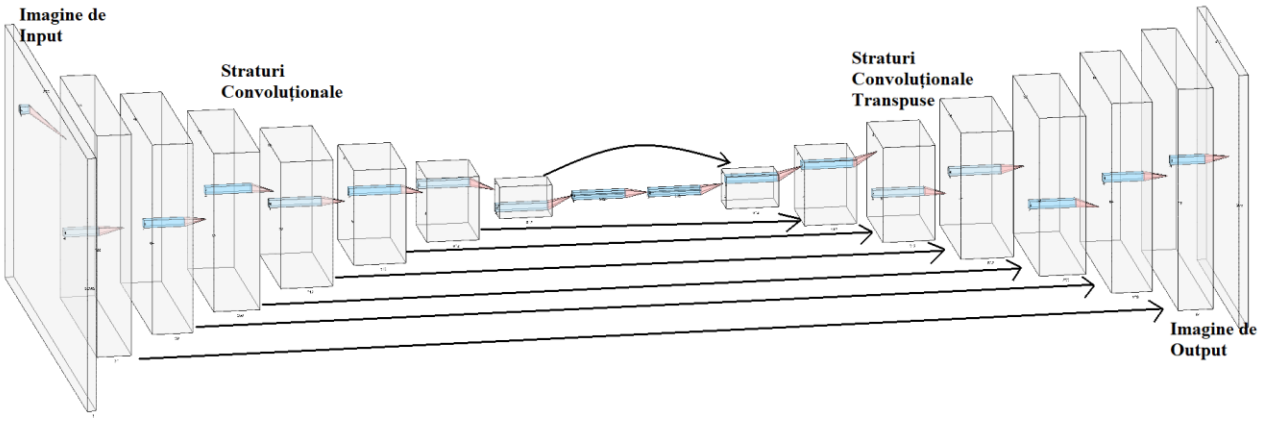


Fig. 13: Auto Codificator cu legături în straturile precedente – realizat în [14]

3.2.4. Funcții de activare

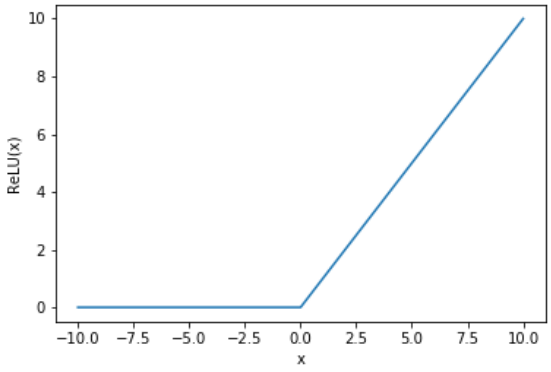
Funcțiile de activare sunt ecuații matematice care determină ieșirea unei rețele neuronale. Funcția este atașată unui neuron din rețea și determină dacă ar trebui să fie activat sau nu, pe baza valorii de intrare a fiecărui neuron dacă aceasta este relevantă pentru predicția modelului. Aceste funcții ajută la normalizarea ieșirii fiecărui neuron într-un interval de (0, 1) sau (-1, 1). Aceste funcții sunt ca o poartă între intrarea neuronului și ieșirea către trece la următorul strat neuronal.

3.2.4.1. ReLU

Un aspect al funcțiilor de activare este că trebuie să fie eficiente din punct de vedere al calculului deoarece ele trebuie să fie calculate pe mii sau chiar milioane de neuroni pentru fiecare eșantion de date. Rețelele neuronale folosesc un algoritm de propagare inversă pentru a antrena modelul. Datorită nevoii de calcul rapid, s-au dezvoltat funcții de activare precum ReLU (Rectified Linear Unit). Ecuația matematică este următoarea:

$$f(x) = f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

Ecuația 3: Funcția ReLU




```python
import matplotlib.pyplot as plt

def rectified(x):
    return max(0.0, x)

x = [i for i in range(-10, 11)]

y = [rectified(i) for i in x]

plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("ReLU(x)")
plt.savefig("plotare_relu.png")
```

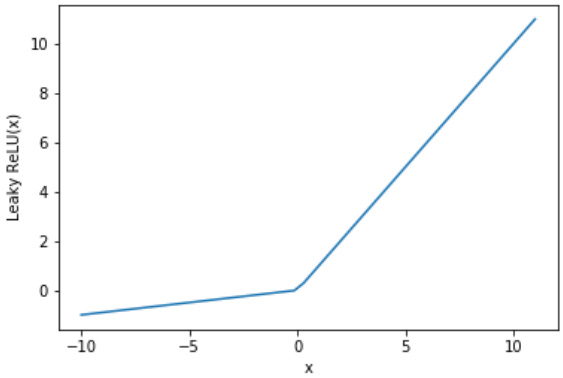
Fig. 14: Funcția de activare ReLU și codul aferent generării

3.2.4.2. Leaky ReLU

Din cauză că unii neuronii rămân inactivi indiferent de intrarea furnizată, performanța este afectată iar acest lucru poate fi corectat folosind ReLU îmbunătățit numit Leaky ReLU. Panta este modificată în zona negativă a lui x, provocând o scurgere, ceea ce duce la o predicție clară. Ecuația următoare demonstrează acest lucru:

$$f(x) = f(x) = \begin{cases} ax, & x < 0 \\ x, & x \geq 0 \end{cases}$$

Ecuația 4: Funcția Leaky ReLU




```python
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-10, 11)

y = np.where(x > 0, x, x * 0.1)

plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("Leaky ReLU(x)")
plt.savefig("plotare_leaky_relu.png")
```

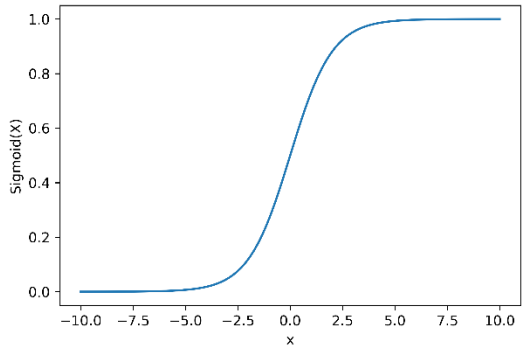
Fig. 15: Funcția de activare Leaky ReLU și codul de generare

3.2.4.3. Sigmoid

Funcțiile sigmoide sunt utilizate în Machine Learning pentru regresia logistică și implementările de bază ale unei rețele neuronale. Acestea nu sunt utilizate pentru rețele neuronale avansate din cauza diverselor dezavantaje precum pierderea informațiilor datorită derivării. Dar aceste funcții sunt simple și ajută la reducerea timpului necesar pentru realizarea unui model.

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Ecuația 5: Funcția Sigmoid




```python
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-10, 10, 100)
y = 1/(1 + np.exp(-x))

plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("Sigmoid(X)")

plt.savefig("sigmoid_function.png", dpi = 600)
plt.close()
```

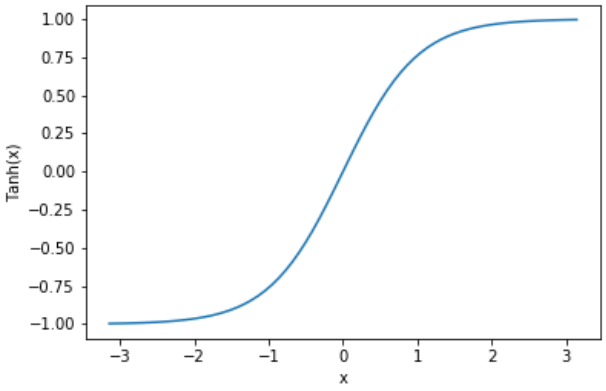
Fig. 16: Funcția de activare sigmoid și codul aferent generării

3.2.4.4. Tangentă hiperbolică

Funcția de tangentă hiperbolică este o altă funcție de activare neliniară între straturile unei rețele neuronale. Împărtășește câteva lucruri în comun cu funcția de activare sigmoidă. Ambele funcții arată similar, dar în timp ce funcția sigmoidă este între intervalul (0, 1), tangenta hiperbolică face mapare între intervalul (-1, 1).

$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{1 + e^{-x}}$$

Ecuația 6: Funcția tangentă hiperbolică




```python
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-np.pi, np.pi, 70)
y = np.tanh(x)

plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("Tanh(x)")
plt.savefig("plotare_tanh.png")
plt.close
```

Fig. 17: Funcția de activare tangentă hiperbolică și codul de generare

3.2.5. Concatenare

Este o metodă prin care se concatenează două intrări de-a lungul unei axe specificate. Intrările trebuie să aibă aceeași formă, cu excepția axei de concatenare.

3.2.6. Abandonarea (Dropout)

Abandonarea funcționează prin eliminarea probabilistică a intrărilor într-un strat, care pot fi variabile de intrare sau activări dintr-un strat anterior. Aceasta are efectul de a simula un număr mai mare de rețele cu o structură de rețea diferită, făcând ca nodurile din rețea să fie în general mai robuste față de intrări. Rata de abandonare poate fi specificată ca probabilitate de a seta fiecare intrare pe 0. Atunci când se sugerează o rată de abandon de 75%, aceasta se va traduce în 25% din inputuri să fie setate pe 0. Abandonarea se folosește doar când se învață modelul de rețea neuronală.

3.2.7. Normalizarea intrărilor

Prin normalizare se înțelege aplicarea unei transformate care menține activarea medie aproape de 0 și deviația standard de activare aproape de 1. Normalizarea se folosește între straturile liniare și non-liniare deoarece normalizează intrarea la funcția de activare, pentru a se centra direct pe funcția de activare. Normalizarea pachetelor diferă, având cu următoarele aspecte cheie:

1. Adăugarea de normalizare la un model determină ca rezultatul unui exemplu să depindă de conținutul tuturor exemplelor dintr-un pachet.
2. Actualizarea ponderilor se bazează pe parcurgerea modelului față de pe rezultatul calculelor gradientului.
3. Când se efectuează inferența unui model care conține normalizarea pachetelor, se dorește de fapt utilizarea statisticilor acumulate și nu a statisticilor pe pachete mai mici.

3.2.8. Inițializarea ponderilor

Inițializările definesc modalitatea de a seta ponderile aleatorii la începutul antrenării unei rețele. Pentru această rețea s-a folosit inițializarea care generează tensori cu o distribuție normală cu o deviație standard de 0.02.

3.3. Explorarea datelor

Pregătirea setului de date este un prim pas în formarea modelului de rețea neuronală pentru segmentare. Imaginile de intrare sunt RGB iar imaginile de output sunt etichetate manual în programul Adobe Photoshop CC2019. Din acest program s-au folosit următoarele funcții pentru a genera manual măști de segmentare:

1. Quick Selection Tool: A fost introdus în versiunea Adobe Photoshop CS3, este similar cu funcția Magic Wand prin faptul că selectează și pixeli în funcție de ton și culoare. Însă Quick Selection Tool este net superior funcției Magic Wand, căutând și texturi similare în imagine, ceea ce o face excelentă la detectarea marginilor obiectelor. Spre deosebire de Magic Wand unde facem clic pe o zonă și sperăm că va selecta zona dorită, Quick Selection Tool funcționează mai mult ca o perie, permițându-ne să selectăm zone pur și simplu "picând" peste ele.
2. Magnetic Lasso Tool: Este unul dintre cele trei instrumente Lasso din Photoshop. Acest instrument poate fi găsit în panoul de instrumente. Spre deosebire de alte instrumente de tip Lasso care nu oferă niciun ajutor și se bazează în totalitate pe propria capacitate de a urmări manual în jurul obiectului, Magnetic Lasso Tool detectează automat marginile, ceea ce înseamnă că aceasta caută în mod activ marginea obiectului în timp ce se deplasează cursorul și se agată de margini ca un magnet.
3. Fill Tool: Acest instrument umple pixelii selectați cu unul din instrumentele de mai sus cu culoarea dorită, uniform. Se poate selecta opacitatea pixelilor iar pentru aceasta se face o combinație între culoarea originala și culoarea dorită. Pentru aceste imagini am folosit opacitate de 100%.

În Fig. 18, am redat secvențe de imagini din setul de antrenare, unde se pot vedea imaginile de bază a setul ce urmează a fi procesat.

Fig. 18: Secvență de date din setul de antrenare

Se alocă fiecărei clase o culoare unică descrisă în Fig. 7. În imaginile de segmentare, valoarea pixelului ar trebui să indice clasa pixelului corespunzător. Desigur, dimensiunea imaginii de intrare și a imaginii de segmentare ar trebui să fie la fel. După ce imaginile de segmentare au fost generate, acestea se pun într-un folder separat de cele de intrare. O astfel de secvență de date este prezentată în Fig. 19:

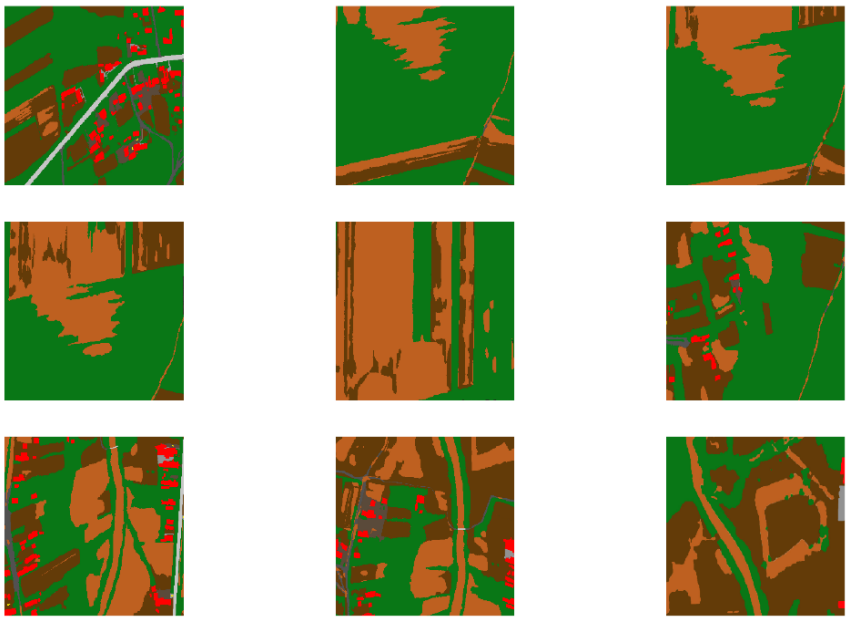


Fig. 19: Secvență de date segmentate din setul de antrenare

Mai jos prezint secvențe de imagini din timpul segmentării manuale efectuate în programul Adobe Photoshop CC2019.

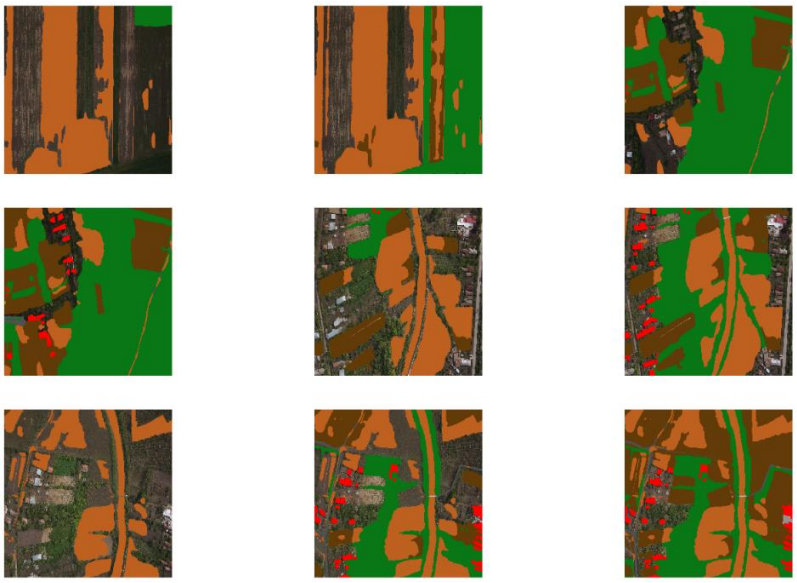


Fig. 20: Secvența de imagini din timpul segmentării manuale

3.3.1. Augmentarea datelor

Deoarece avem un număr mai mic de perechi de antrenare, există posibilitatea ca rezultatele să nu fie bune, deoarece modelul ar putea face Overfitting. Putem crește numărul de inputuri din setul de date aplicând transformări aleatorii. Se pot schimba proprietăți de culoare precum nuanță, saturație, luminozitate, etc. Putem aplica de asemenea transformări de rotire, scalare sau oglindire. Masca imaginii ar trebui să fie transformată la fel. Pentru augmentarea datelor s-a folosit modulul ImgAug [16] din Python. Acest modul ajută la creșterea numărului de imagini pentru antrenarea rețelei. Transformă un set de imagini de intrare într-un set nou, mai mare de imagini modificate.

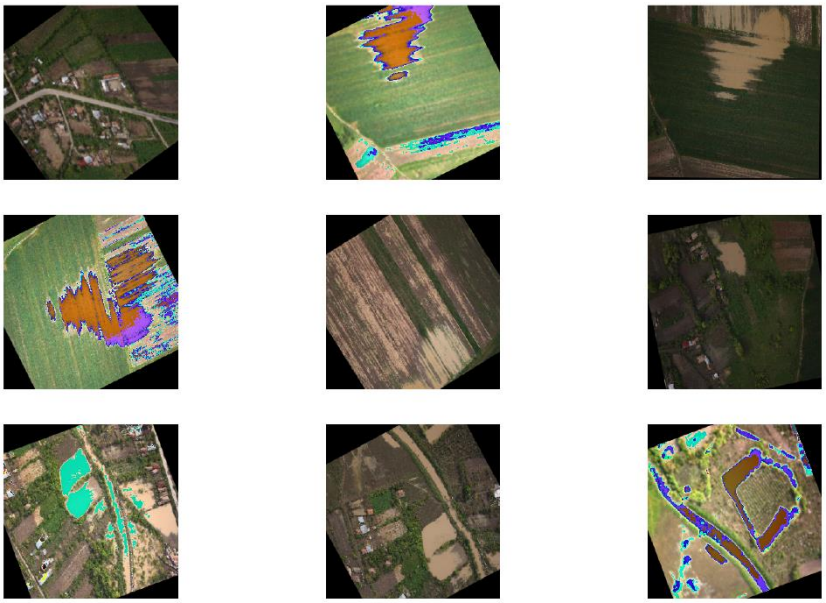


Fig. 21: Augmentarea datelor de antrenare

3.3.2. Alegerea dimensiunii de intrare

Este foarte important o dată cu alegerea arhitecturii modelului să alegerea dimensiunii de intrare al acestuia. Dacă în imagine există un număr mare de clase, dimensiunea de intrare trebuie să fie mai mare. În unele cazuri, dacă dimensiunea de intrare este mare, modelul ar trebui să aibă mai multe straturi. Cu cât variază dimensiunea imaginilor de intrare, de exemplu de la 256x256 la 1024x1024, un model consumă mai multă memorie pe placa video și acest lucru ar însemna o durată mai mare de timp pentru antrenare.

3.4. Funcții de pierdere

Funcțiile de pierdere sunt una dintre componentele cheie în metodele de segmentare semantică a imaginilor. În timpul antrenării modelele de rețele neuronale utilizează funcții simple de pierdere a entropiei încrucișate (Cross Entropy). Pentru obținerea informațiilor granulare dintr-o imagine, atunci trebuie să se folosească funcții de pierdere mai avansate.

3.4.1. Focal Loss

Această funcție de pierdere este o îmbunătățire a criteriului standard de entropie încrucișată. Aceasta se face prin schimbarea formei sale astfel încât se alocă o pondere mai mică pentru clasificate. În cele din urmă, se asigură că nu există un dezechilibru de clase. Entropia este scalată cu factori de scalare pe măsură ce crește încrederea în clasele corecte. Factorul de scalare coboară automat ponderea contribuției unor exemple ușoare în timpul de antrenare și se concentrează pe cele mai grele.

$$FL(p_t) = -\alpha(1-p_t)^{\gamma} log(p_t)$$

Ecuația 7: Formula funcției de pierdere Focal

3.4.2. Entropie încrucișată (Cross Entropy)

Entropia încrucișată este derivată din divergența Kullback-Leibler (KL), care este o măsură a disimilarității între două distribuții. Pentru sarcini comune de antrenare, distribuția datelor este dată de setul de antrenare, deci va fi o constantă. Astfel, minimizarea entropiei încrucișate este echivalentă cu minimizarea divergenței KL.

$$CE(p,q) = \begin{cases} -log(p), & q = 0 \\ -\log(1-p), & altfel \end{cases}$$

Ecuația 8: Funcția de pierdere pentru entropia încrucișată

3.4.3. Dice Loss

Această funcție de pierdere este obținută prin calcularea coeficientului Dice. Funcția cea mai folosită în probleme de segmentare este:

$$DL = 1 - \frac{2\sum_{pixeli} y_{adevarat} y_{prezis}}{\sum_{pixeli} y_{adevarat}^2 + \sum_{pixeli} y_{prezis}^2}$$

Ecuația 9: Funcția de pierdere Dice

3.4.4. Intersecția peste Uniune (IoU)

Această funcție măsoară numărul de pixeli comuni între masca imaginii și imaginea prezisă, care se împarte la numărul total de pixeli prezenți în ambele imagini.

3.4.5. Boundary Loss

Această funcție de pierdere se aplică atunci când avem segmentări extrem de dezechilibrate. Metoda acestei pierderi este cea a distanței pe conturul spațial și nu pe regiuni.

$$BL = \int_{\Omega} \emptyset\, G(p) s_\theta(p) dp$$

Ecuația 10: Funcția de pierdere pentru distanțe pe contur

3.5.Algoritmi de optimizatori

Alegerea unui algoritm de optimizare pentru rețeaua neuronală poate duce la diferențe între rezultate de ordinul minutelor, orelor sau chiar a zilelor. Algoritmul de optimizare Adam este o extensie al algoritmului stocastic de gradient.

Adam este un algoritm de optimizare care poate fi utilizat în locul procedurilor clasice de gradient pentru a actualiza ponderile rețelei iterativ bazate pe datele de antrenare.

Algoritmul a fost prezentat de Diederik Kingma de la OpenAI și Jimmy Ba de la Universitatea din Toronto în lucrarea ICLR din 2015, "Adam: A method for Stochastic Optimization" [17].

Acest algoritm menține o rată de antrenare unică (α) pentru toate actualizările ponderilor, iar rata de antrenare nu se modifică în timpul antrenării. O rată de antrenare este menținută pentru fiecare pondere din rețea și adaptată separat pe măsură ce se desfășoară antrenarea.

Algoritmul calculează rate individuale de antrenare adaptive pentru diferiți parametrii, din estimările primului și celui de-al doilea moment al gradienților.

Adam combină avantajele a 2 tipuri de gradienți și anume:

1. Algoritmul de gradient adaptiv (AdaGrad) menține o rată de antrenare per parametru și îmbunătățește performanța la problemele cu gradienți mai slabi.
2. Propagarea pătrată a mediei rădăcinii (RMSProp) menține rata de antrenare per parametru și sunt adaptate în funcție de media gradienților anteriori pentru ponderi. Algoritmul se descurcă bine la probleme online și non-staționare.

Adam este un algoritm popular în domeniul Machine Learning-ului pentru că se obțin rezultate foarte rapid. Rezultatele empirice demonstrează că Adam funcționează bine în practică și se comportă favorabil și cu alte metode de optimizare stocastică. Adam este adaptat pentru reperele din lucrările de Machine Learning.

3.6.Rețele Generativ Adversale (GAN)

Rețelele Generativ Adversale (GAN) sunt arhitecturi algoritmice care folosesc două rețele neuronale, așezate una împotriva celeilalte pentru a genera noi cazuri sintetice de date care pot păcăli realitatea. Sunt utilizate pe scară largă în generarea de imagini, generare video și generarea vocii.

Studiul acestor GAN-uri este un subiect în vogă în Deep Learning datorită puterii lor. În doar câțiva ani, au trecut de la generarea de numere confuze la crearea de imagini realiste.

GAN-urile au fost introduse într-o lucrare de Ian Goodfellow și alți cercetători de la Universitatea din Montreal, inclusiv Yoshua Bengio, în 2014 [18].

Arhitectura modelului conține mai multe straturi de convoluție, activări neliniare (ReLU), normalizare datelor și reducere (pooling). Straturile inițiale învață conceptele de nivel jos, cum ar fi marginile și culorile, iar pe măsură pe înaintăm în profunzime, se învață concepte de nivel superior, precum obiecte diferite.

La nivele inferioare, neuronii conțin informații pentru o porțiune mică din imagine, pe când neuronii de la nivele superioare conțin informații pentru o porțiune mai mare de imagine. Așadar adăugând mai multe straturi, dimensiunea imaginii continuă să scadă și numărul de canale continuă să crească. Reducerea dimensiunii este făcută de straturile de pooling.

Pentru cazul clasificării de imagini, trebuie să mapăm tensor-ul spațial de la straturile de convoluție la un vector cu lungime fixă. Se folosește un strat complet conectat însă acesta elimină toate informațiile spațiale.

Un tensor este un obiect algebric care descrie o relație multi-liniară între seturi de obiecte algebrice legate de un spațiu vectorial.

Pentru segmentarea semantică, trebuie să reținem informații spațiale, prin urmare nu se folosesc straturi complet conectate.

Un tensor de rezoluție scăzută, care conține informații de nivel înalt, trebuie sa producă rezultate de segmentare de înaltă rezoluție. Se adaugă mai multe straturi de convoluție, cuplate cu straturi de eșantionare (pooling) care cresc dimensiunea tensorului.

Potențialul rețelelor GAN este imens, deoarece pot învăța să imite orice distribuție de date. Adică, GAN-urile pot fi învățate să creeze lumi asemănătoare cu ale noastre în orice domeniu: imagini, muzică, discurs, proză, etc. Sunt roboți care au devenit artiști într-un anumit sens, iar

capacitatea acestora de producție este impresionantă, înfricoșătoare chiar. Dar pot fi folosite și pentru a genera conținut media fals iar tehnologia care stă la baza lor se numește Deepfakes.

Rețelele adversare generative (GAN) sunt clasificate în grupul de modele generative. Asta înseamnă că sunt capabili să producă, adică să genereze date complet noi, „valide”.

Pentru a ilustra, luăm în considerare un exemplu în care dorim să generăm câteva imagini noi pentru învățarea unei rețele de clasificare a imaginilor. Desigur, pentru o astfel de aplicație dorim ca datele noastre de învățare să fie cât mai realiste, probabil destul de similare în stil cu alte seturi de date de învățare a clasificării imaginilor.

Rețelele GAN sunt construite din două rețele opuse separate: Discriminatorul și Generatorul. Discriminatorul este antrenat să clasifice dacă imaginea este una reală sau nu. Pe când Generatorul este antrenat să creeze imagini cu aspect realist atunci când este dat doar un set de imagini zgomotoase ca intrare.

Ponderile rețelei generatorului sunt învățate pe baza funcției de pierdere a Discriminatorului. Astfel, Generatorul este forțat să învețe imaginile pe care le generează. În același timp imaginile arată cât mai real, Discriminatorul devenind mai bun în a discerne imaginile reale de cele artificiale.

Din punct de vedere tehnic, funcția de pierdere a Discriminatorului va fi eroarea de clasificare a imaginilor false față de cele reale, măsurând astfel capacitatea de a distinge imagini reale și generate. Funcția de pierdere a generatorului se va baza pe cât de bine este să păcălească Discriminatorul cu imagini generate – adică eroarea de clasificare a Discriminatorului doar pe imagini false.

Astfel, rețele GAN construiesc o buclă de feedback în care Generatorul ajută la antrenarea Discriminatorului, iar Discriminatorul ajută la antrenarea Generatorului. Ambele rețele ajung mai funcționeze mai bine împreună.

În diagrama de mai jos, Fig. 22, se arată ilustrarea acestui lucru. Se începe prin alegerea aleatoare a unei perechi de imagini din setul de date, imaginea de input și imaginea de referință. Se antrenează rețeaua neuronală generator cu imaginea de input și se compară output-ul rețelei cu imaginea de referință. Apoi imaginea generată este trimisă către rețeaua neuronală discriminator pentru a discerne între imaginea de referință sau cea generată. Se apelează algoritmul de optimizare pe baza erorii dintre cele două imagini, rezultând actualizarea ponderilor fiecărei rețele, pentru a genera imagini mai bune sau a diferenția între imaginile artificiale și cele de referință.

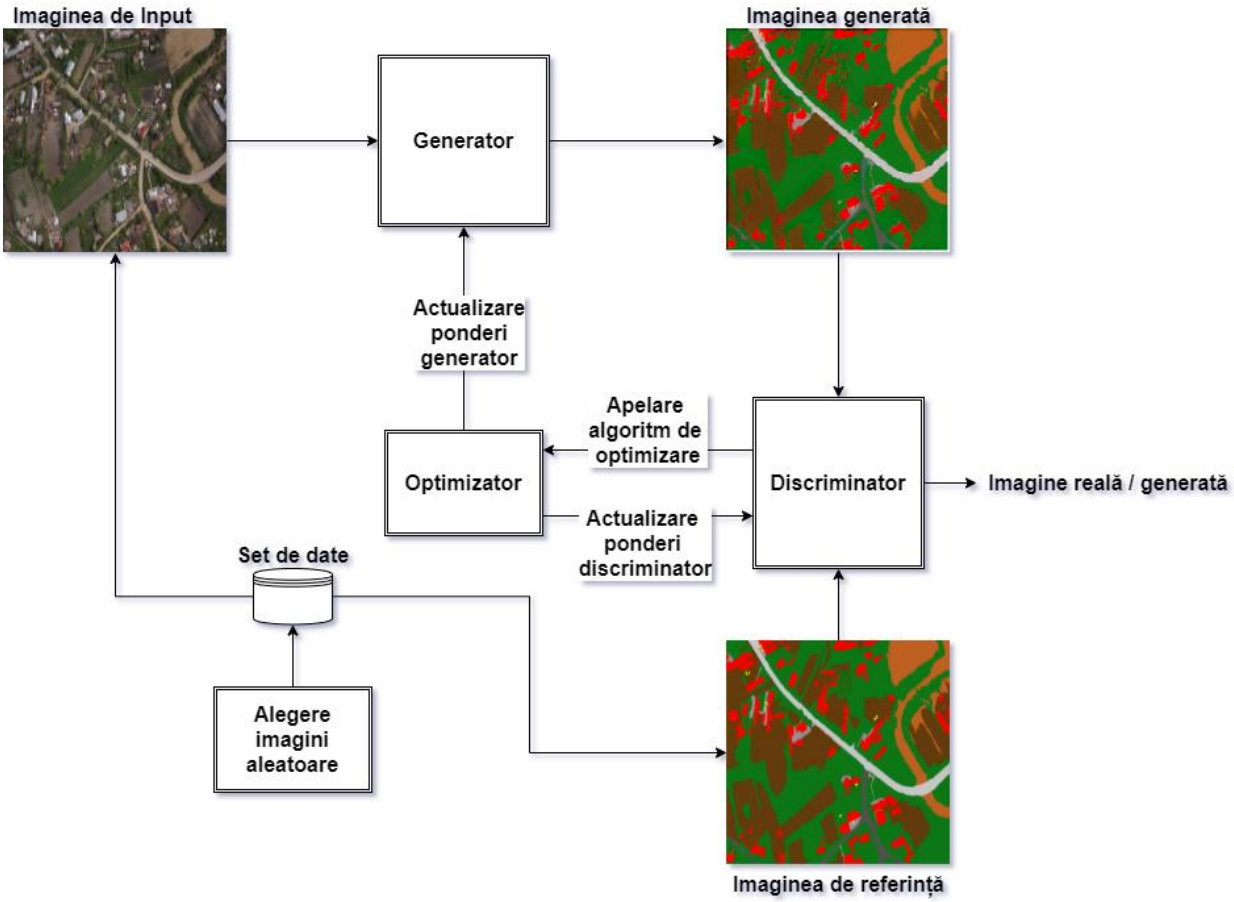


Fig. 22: Arhitectura rețelei GAN – realizat în [19]

3.6.1. Generator – Discriminator

Pentru a înțelege GAN-urile, ar trebui să știm cum funcționează acești algoritmi generativi. Algoritmii discriminatori încearcă să clasifice datele de intrare, adică având în vedere caracteristicile unei instanțe de date, care prezic o etichetă sau o categorie din care fac parte acele date.

Având în vedere toate cuvintele dintr-un email, fiind o instanță de date, un algoritm discriminatoriu ar putea prezice dacă mesajul este spam sau nu. Spamul este una dintre etichete iar cuvintele din email sunt caracteristicile care constituie datele de intrare. Când aceasta problemă este exprimată matematic, eticheta se numește “y”, iar caracteristicile se numesc “x”.

Așadar, algoritmii discriminatori asigură o caracteristică a etichetelor. Scopul lor este dat de această corelație. Un mod de a gândi algoritmii generativi este faptul că fac opusul. În loc să prezică o etichetă dată pentru unele caracteristici, ei încearcă să prezică caracteristici date cu o anumită etichetă.

Pentru a distinge discriminatorul de generator, se pot lua în calcul și următoarele situații:

- Modelele discriminatorii învață granița dintre clase.
- Modelele generative modelează distribuția claselor individuale.

Rețeaua neuronală, numită generator, are proprietatea de a genera noi instanțe de date, în timp ce altă rețea, numită discriminator, are rolul de a evalua autenticitatea datelor. Adică discriminatorul decide dacă fiecare instanță de date pe care o analizează aparține sau nu setului de date de antrenare.

În acest tip de rețele avem o dublă buclă de feedback:

- Discriminatorul se află într-o buclă de feedback cu imaginile reale.
- Generatorul este într-o buclă de feedback cu discriminatorul.

GAN-ul poate fi văzut ca opoziția unui falsificator și a unui polițist într-un joc de prins șoarecele și pisica, unde falsificatorul învață să transmită note false, iar polițistul învață să le detecteze. Ambele sunt dinamice, adică polițistul se pregătește și el, astfel încât fiecare parte vine să antreneze metodele celuilalt într-o escaladare constantă.

Ambele rețele încearcă să optimizeze o funcție obiectivă diferită și opusă sau o funcție de pierdere. Aceasta este în esență un model de actor – critic. Pe măsură ce discriminatorul își schimbă comportamentul, la fel va face și generatorul și invers. Pierderile lor se împing unul împotriva celuilalt.

Poate fi folositor compararea rețelelor generativ adversale cu alte rețele neuronale cum ar fi auto codificatoare (autoencoders), auto codificatoare variaționale (VAE).

Auto codificatoarele codifică datele de intrare ca vectori. Ele creează o reprezentare ascunsă sau comprimată a datelor brute. Sunt utile în reducerea dimensiunii, adică vectorul care servește ca o reprezentare ascunsă comprimă datele brute într-un număr mai mic de dimensiuni importante. Auto codificatoarele pot fi asociate cu un așa numit decodificator, care va permite reconstrucția datelor de intrare pe baza reprezentării sale ascunse, la fel cum procedează o mașină Boltzmann restrânsă.

Auto codificatoarele variaționale sunt un algoritm generativ care adaugă o restricție suplimentară la codificarea datelor de intrare, adică de această dată reprezentările ascunse sunt normalizate. Acest tip de codificatoare sunt capabile să comprime date precum un auto codificator și să sintetizeze date precum un GAN. Oricum, GAN-urile generează date în detalii granulare fine, astfel încât imaginile generate de auto codificatoarele variaționale sunt mai estompate.

Putem grupa algoritmii generativi în următoarele trei tipuri:

- Având o etichetă, se prezic caracteristicile asociate (Naive Bayes).
- Având o reprezentare ascunsă, se prezic caracteristicile asociate (VAE, GAN).
- Având câteva trăsături, se pot prezice și restul de trăsături.

Când se antrenează discriminatorul, trebuie menținute constante valorile generatorului și când se antrenează generatorul, se țin constante valorile discriminatorului. Fiecare ar trebui să se antreneze împotriva unui adversar static.

În timp ce modelul este antrenat pentru segmentarea semantică, codificatorul scoate un tensor care conține informații despre obiecte, precum și forma și dimensiunea acestuia. Decodificatorul preia aceste informații și produce segmentarea.

Dacă facem o stivă de straturi de codificare și decodificare, s-ar putea să pierdem informații la nivelul de jos. Prin urmare, limitele din segmentare produse de codificator ar putea fi inexacte. Pentru compensarea informațiilor pierdute, lăsăm decodificatorul să acceseze caracteristicile de la nivelul de jos produse de straturile de codificare. Acest lucru se realizează prin ignorarea conexiunilor. Ieșirile intermediare ale codificatorului sunt adăugate cu intrările la straturile intermediare ale decodificatorului în pozițiile corespunzătoare (vezi Fig. 13).

Ignorarea conexiunilor din straturile anterioare furnizează informațiile necesare straturilor de decodificare care sunt necesare pentru crearea limitelor precise (a obiectelor din imagine).

Fiecare parte a GAN-ului poate să-l învingă pe celălalt. Dacă discriminatorul este prea bun, acesta va da rezultate atât apropiate de 0 sau de 1 încât generatorul se va chinui pentru a citi gradientul. Dacă generatorul este prea bun, acesta va exploata în mod persistent punctele slabe ale discriminatorului care duc la falsuri negative. Acest lucru poate fi atenuat de ratele de învățare ale rețelelor respective. Cele două rețele neuronale trebuie să aibă un nivel similar de aptitudini.

GAN-urile necesită mult timp pentru a se antrena. Pe un singur GPU, un GAN poate dura de la câteva ore până la câteva săptămâni, iar pe un CPU timpul este exponențial mai mare. În timp ce sunt greu de acordat, GAN-urile au stimulat o mulțime de cercetări și scrieri interesante.

Într-un mediu perfect, generatorul ar capta distribuția generală a datelor de antrenare. Discriminatorul nu ar fi întotdeauna singur dacă intrările sunt reale sa generate.

3.6.2. Arhitectura rețele generator – discriminator

Arhitectura generatorului este prezentată în Fig. 13: Auto Codificator cu legături în straturile precedente – realizat în . Arhitectura discriminatorului este prezentată în următoarea figură:

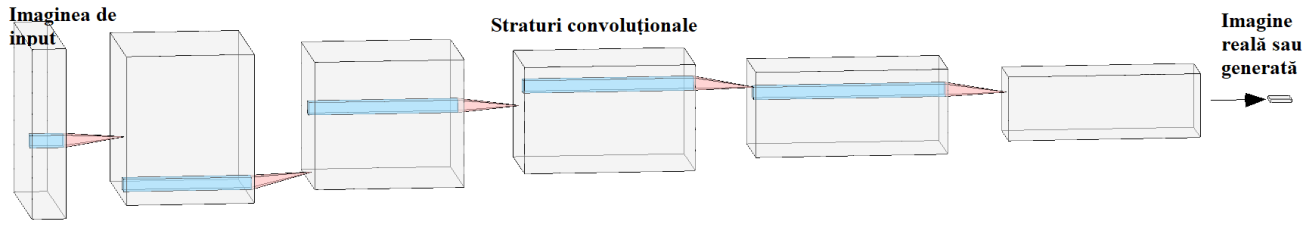


Fig. 23: Arhitectură Discriminator – realizată în [14]

4. Rezultate experimentale

4.1 Tipuri de strategii

Antrenarea sincronă sau asincronă: Acestea sunt două moduri comune de distribuire a antrenării în paralel. Antrenarea distribuită permite extinderea sarcinii de antrenare, astfel încât modelele mai mari să poată fi antrenate eficient într-un ritm mai rapid deoarece necesită mai mult timp pentru a rula.

Se poate extinde antrenarea pe mai multe GPU-uri pe o singură mașină sau pe mai multe mașini dintr-o rețea sau pe TPU-uri din Cloud. Un TPU este o unitate de procesare pentru tensori, un circuit de tip ASIC, dezvoltat de Google special pentru antrenarea rețelelor neuronale.

Pentru a rula modele mari de antrenare, va trebui să le distribuim pe mai multe procesoare, GPU-uri sau mașini. Framework-ul Tensorflow [20] ne poate ajuta să distribuim antrenarea prin împărțirea modelului pe mai multe dispozitive și efectuarea antrenării în paralel.

Există două metode principale de implementare pe care le putem utiliza pentru a distribui modelul de antrenare: paralelism de model sau paralelism de date.

În paralelism de model, modelul este segmentat în diferite părți, astfel încât să îl putem rula în paralel. Se poate rula fiecare parte pe aceleași date în noduri diferite. Această abordare poate reduce nevoia de comunicare între noduri, deoarece nodurile trebuie doar sincronizate între parametrii partajați.

Paralelismul modelului funcționează bine și pe GPU-urile dintr-un singur server care are o magistrală de viteză mare, deoarece restricțiile hardware nu provoacă o limitare pe nod.

Pentru paralelismul de date, datele de antrenare sunt împărțite în mai multe subseturi. Executarea fiecărui subset se face pe același model replicat într-un număr pe un noduri. Trebuie să sincronizăm și parametrii modelului la sfârșitul fiecărui calcul pe subsetul de date pentru a ne asigura că antrenează un model consecvent, deoarece erorile de predicție sunt calculate independent pe fiecare nod în parte. Când rulăm experimente distribuite pe mai multe mașini cu mai multe GPU-uri este foarte important să ne asigurăm ca le folosim la maximum.

4.2 Conectare Cloud Google

Pentru a antrena rețeaua neuronală, ne vom folosi de Cloud-ul pus la dispoziție de Google, numit Google Colaboratory (Colab) [21]. În acest serviciu Cloud avem la dispoziție CPU-uri, GPU-uri și TPU-uri.

Folosind Colab putem:

1. Dezvolta modele pentru Machine Learning folosind cele mai populare framework-uri precum Tensorflow [20], Keras [22] sau OpenCV [23].
2. Folosirea oricăror librării deoarece multe din ele vin preinstalate.
3. Instalarea librăriilor se face foarte ușor cu o simplă comandă de pip.

Un lucru pe care îl face Colab ca să fie cel mai bun este faptul că vine cu o mulțime de librării care facilitează accesul la alte servicii Google, precum mediul de stocare Cloud numit Drive [24], pe care îl vom utiliza în această lucrare. Colab salvează toate fișiere Jupyter Notebook în Google Drive.

Problema apare atunci când trebuie să lucrăm cu un set de date mare, deoarece transferul între Drive și mașina virtuală din Colab poate deveni mare. Deconectarea de la mașina virtuală din Colab duce la pierderea tuturor datelor încărcate în Colab, iar pentru o eventuală reantrenare este nevoie de a face încă o dată transferul.

4.3 Rezultatele antrenării în Cloud

Rețeaua a fost antrenată pentru 30 de epoci, fiecare epocă conține 1000 de iterații. O iterație conține 3 etape: o propagare directă, o calculare a diferențelor între imaginea prezisă și imaginea de input și o rulare a propagării inverse. S-a folosit augmentarea imaginilor de input pentru a obține un set de 410 imagini de antrenare pornind de la un set de 10 imagini de rezoluție 6000x4000.

Augmentarea s-a făcut prin următoarele tehnici:

1. Oglindirea față de axele Ox, Oy, precum și ambele.
2. Rotirea la 90, 180, 270 de grade.
3. Rotirea și oglindirea.
4. Scalarea la 1.5x față de imaginea inițială.
5. Scalarea și oglindirea.
6. Scalarea și rotația.
7. Transformarea de perspectivă, adică se aplică transformări de perspectivă în 4 puncte aleatoare ale imaginii. Fiecare dintre cele 4 puncte este deplasat folosind o distanță aleatoare față de colțul respectiv.
8. Convertirea imaginilor într-o paletă de gri.
9. Schimbarea luminozității imaginii.
10. Aplicarea unui nucleu de încețoșare (blur) și combinarea filtrului cu rotații de imagine.
11. Aplicarea unui nucleu de netezire și combinarea filtrului cu rotații de imagine.
12. Aplicarea unui nucleu de contrast și combinarea cu rotații de imagine.
13. Aplicarea unui nucleu pentru îmbunătățirea detaliilor.
14. Aplicarea unui filtru care suprapune peste imagine simulări ale ceții, ploii și a norilor și de asemenea aplicarea rotațiilor de imagine.

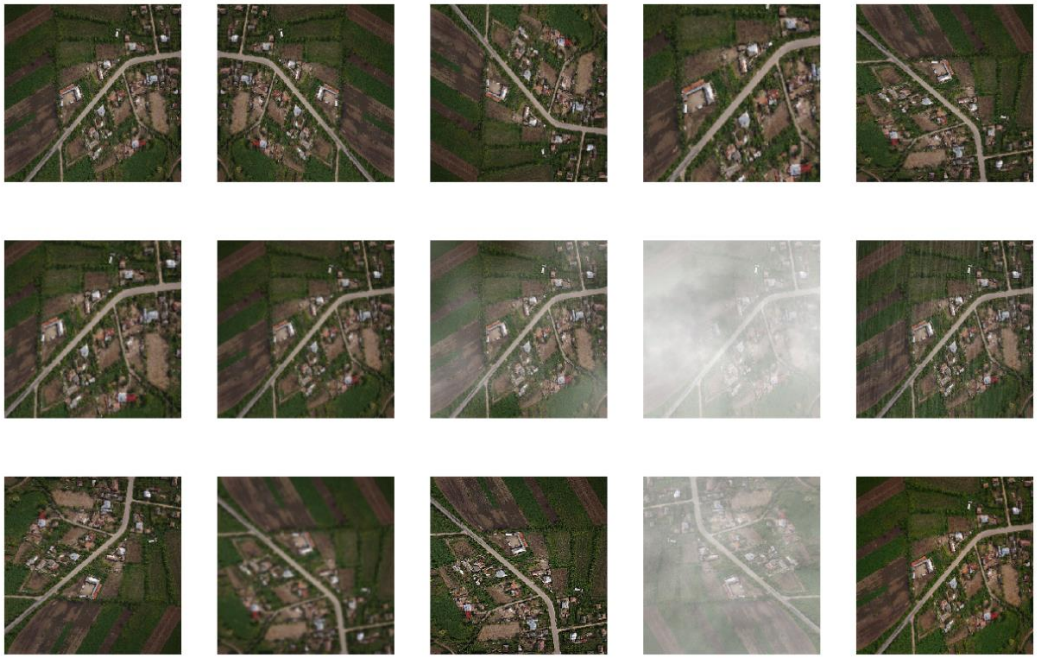


Fig. 24: Aplicarea unor transformări asupra imaginilor de Input

Imaginile de input au fost redimensionate la 256x256 pixeli.

Fig. 25: Rezultatul rețelei neuronale pentru imagini 256x256

Timpul de antrenare pentru antrenarea rețelei neuronale este următorul:

EPOCI	TIMPI DE ANTRENARE (SECUNDE)
3	878.17
6	1,735.03
9	2,592.05
12	3,455.63

15	4,314.78
18	5,167.55
21	6,025.50
24	6,880.39
27	7,734.30
30	8,593.55

Tabelul 1: Timpul de antrenare pentru rețea pe Google Colab având imagini de input de rezoluție 256x256

S-a efectuat antrenarea rețelei prin redimensionarea imaginilor de input si output la dimensiunea de 256x256. Antrenarea s-a efectuat tot pentru 30 de epoci, fiecare epocă având 1000 de iterații.

Fig. 26: Rezultatul antrenării pentru imagini 512x512

Timpul de antrenare pentru antrenarea rețelei neuronale este următorul:

EPOCI	TIMPI DE ANTRENARE (SECUNDE)
3	878.65
6	1,640.14
9	2,452.74
12	3,272.28
15	4,084.27
18	4,895.00
21	5,709.34
24	6,522.73
27	7,334.62
30	8,153.30

Tabelul 2: Timpul de antrenare pentru rețea pe Google Colab având imagini de input de rezoluție 512x512

Aceeași tip de rețea s-a antrenat în Cloud folosind doar un procesor (CPU), pentru o singură epocă având 1000 de iterații. Rezultatele timpilor de rulare se pot vedea în tabelul următor:

ITERAȚII	TIMPI DE ANTRENARE (SECUNDE)
100	321.32
200	634.89
300	945.28
400	1,258.95
500	1,569.16
600	1,883.48
700	2,193.19
800	2,506.74
900	2,816.61
1000	3,130.08

Tabelul 3: Timpul de antrenare pentru rețea folosind un procesor în Cloud-ul Google pentru imagini de input de rezoluție 256x256

Rețeaua a rulat în Cloud și pe un hardware specializat de tip ASIC, numit TPU – Tensor Processing Unit dezvoltat de Google pentru accelerarea antrenării rețelelor neuronale de dimensiuni foarte mari. Acest ASIC este proiectat pentru performanțe maxime și flexibilitate pentru a ajuta dezvoltarea și antrenarea de modele de inteligență artificială. Aceste dispozitive au 16GB de memorie de tip HBM (High-Bandwidth Memory) pentru fiecare core al unui TPU si poate avea până la 2048 de core-uri și 32TB de memorie în total. Puterea de procesare anunțată de Google este de 100 PFlops/s. Rețeaua a rulat timp de 6 epoci, fiecare epocă având 1000 de iterații.

Timpii de antrenare se găsesc în următorul tabel:

EPOCI	TIMPI DE ANTRENARE (SECUNDE)
1	699.41
2	1,391.44
3	2,086.82
4	2,769.10
5	3,454.48
6	4,142.04

Tabelul 4: Timpul de antrenare pentru rețea folosind un hardware specializat TPU pentru imagini de input de rezoluție 256x256

Inversând imaginile de input cu imaginile de output, putem obține o imagine color, similară unei imagini aeriene sau din satelit. Acest lucru ne dă posibilitatea să creăm date artificiale care ne vor ajuta la antrenarea rețelei neuronale pentru segmentare. Această rețea s-a antrenat în Cloud-ul Google pentru 30 de epoci, fiecare epocă având 1000 de iterații.

EPOCI	TIMPI DE ANTRENARE (SECUNDE)
3	341.27
6	667.38
9	992.56
12	1,321.63

15	1,646.84
18	1,972.05
21	2,300.58
24	2,625.81
27	2,950.93
30	3,279.49

Rețeaua a fost antrenată cu 4100 imagini obținute din 10 imagini aeriene, folosind tehnici de augmentare a datelor.

Fig. 27: Rezultatul rețelei antrenate inversând imaginile de input și output

Adăugând un strat nou la fiecare dintre cele 2 rețele, putem crește rezoluția imaginii de la 256x256x3 la 512x512. Rețeaua a fost antrenată pentru 30 de epoci, fiecare epocă având 1000 de iterații.

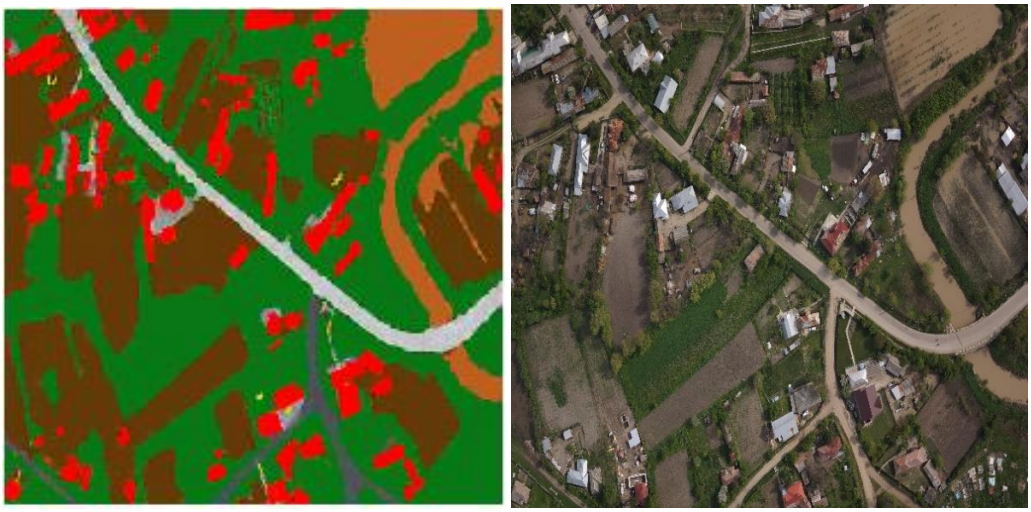


Fig. 28: Rezultatul rețelei după adăugarea unui strat nou

Timpul de antrenare a rețelei în Cloud sunt următorii:

EPOCI	TIMPI DE ANTRENARE (SECUNDE)
3	1,122.33
6	2,245.41
9	3,367.54
12	4,493.02
15	5,610.34
18	6,728.63
21	7,859.50
24	8,981.85
27	10,106.08
30	11,231.82

Tabelul 5: Timpul de antrenare pentru rețeaua neuronală adăugând un strat suplimentar

Modificând grupul de imagini de antrenare pentru o iterație de la o imagine la patru imagini, vom obține următoarele rezultate. Acest model a fost antrenat pentru 20 de epoci, fiecare epocă având 1000 de iterații. Imaginile de input au 256x256x3.

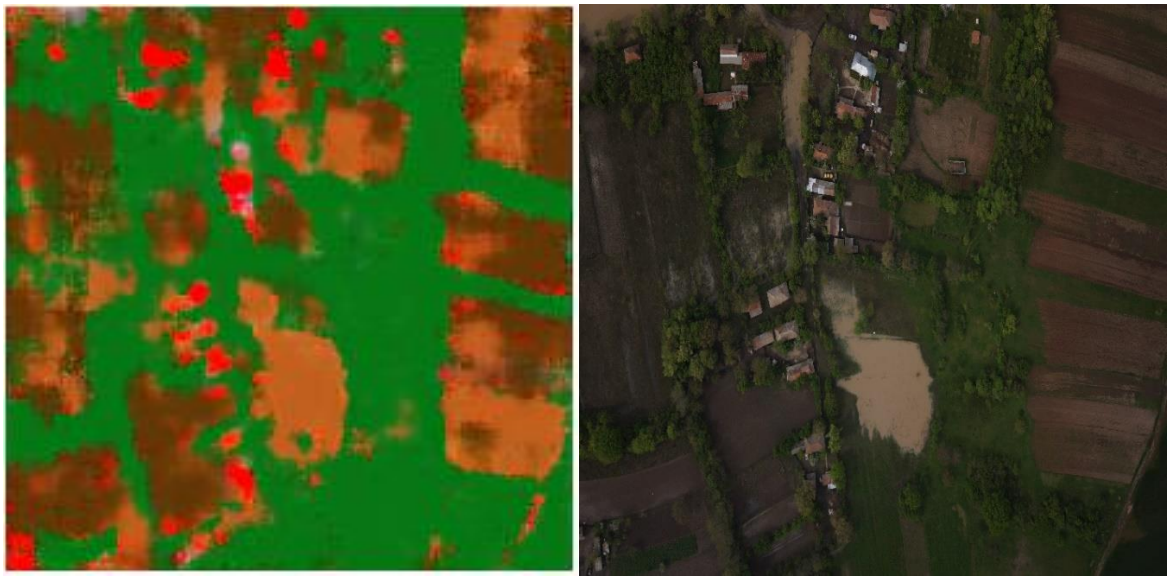


Fig. 29: Rezultatul rețelei neuronale pentru batch-uri de 4 imagini

Acest model are următorul timp de execuție:

EPOCI	TIMPI DE ANTRENARE (SECUNDE)
3	958.76
6	1,899.34
9	2,841.39
12	3,793.48
15	4,729.09
18	5,668.12
20	6,612.20

Tabelul 6: Timp de antrenare pentru grupuri de 4 imagini la input – rezoluție 256x256x3

Asemănător cu modelul anterior, rețeaua a fost antrenată pe grupuri de imagini, de data aceasta grupul conține 16 imagini (batch size).

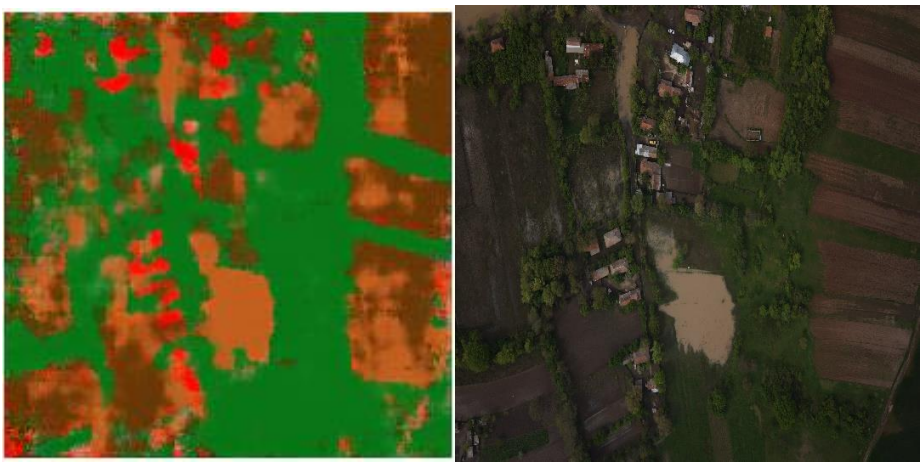


Fig. 30: Rezultatul rețelei pentru batch-uri de 16 imagini

Timpul de antrenare este următorul:

EPOCI	TIMPI DE ANTRENARE (SECUNDE)
3	1,992.06
6	3,990.96
9	6,005.97
12	7,985.64
15	9,931.65
18	12,007.61
20	13,339.44

Tabelul 7: Timpul de antrenare pentru grupuri de 16 imagini la input - rezoluție 256x256x3

Asemănător cu modelele de mai sus, rețeaua a fost antrenată și pentru rezoluții de 512x512x3 cu patru imagini simultane (batch size-ul de 4). Acest model a fost antrenat pentru 15 epoci, fiecare epocă având 1000 de iterații.

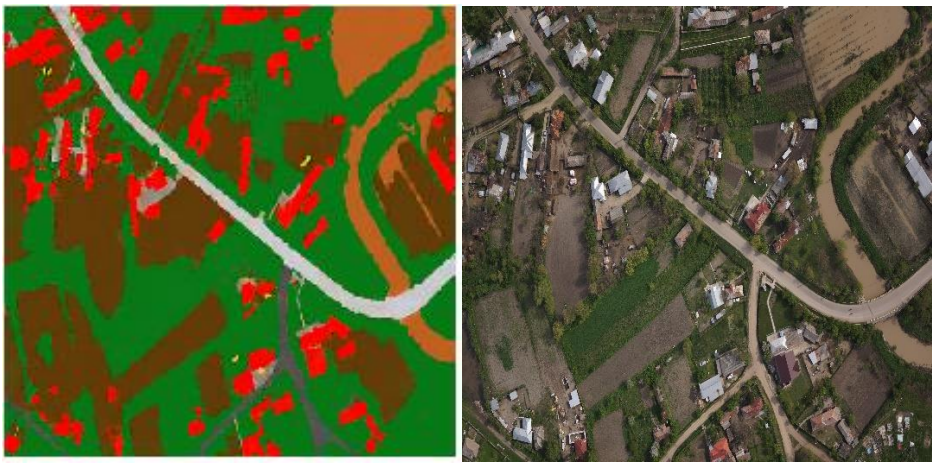


Fig. 31: Rezultatul antrenării cu imagini de 512x512x3 cu batch size de 4

Timpul de antrenare pentru acest model este următorul:

EPOCI	TIMPI DE ANTRENARE (SECUNDE)
3	3,211.84
6	6,305.80
9	9,461.59
12	11,380.44
15	13,244.42

Tabelul 8:Timpul de antrenare pentru grupuri de 4 imagini de input - rezoluție 512x512x3

4.4 Rularea pe mașina locală

S-a efectuat antrenarea și pe mașina locală având un GPU NVIDIA – GTX1070 cu 8GB memorie. Avem prezentat în Fig. 32. Rețeaua a fost antrenată pentru 30 de epoci, fiecare epocă având 1000 de iterații. Primul rând al figurii reprezintă imaginea de antrenare numită și imagine de input, al doilea rând reprezintă imaginea rezultată în urma antrenării iar al treilea rând reprezintă masca imaginii.

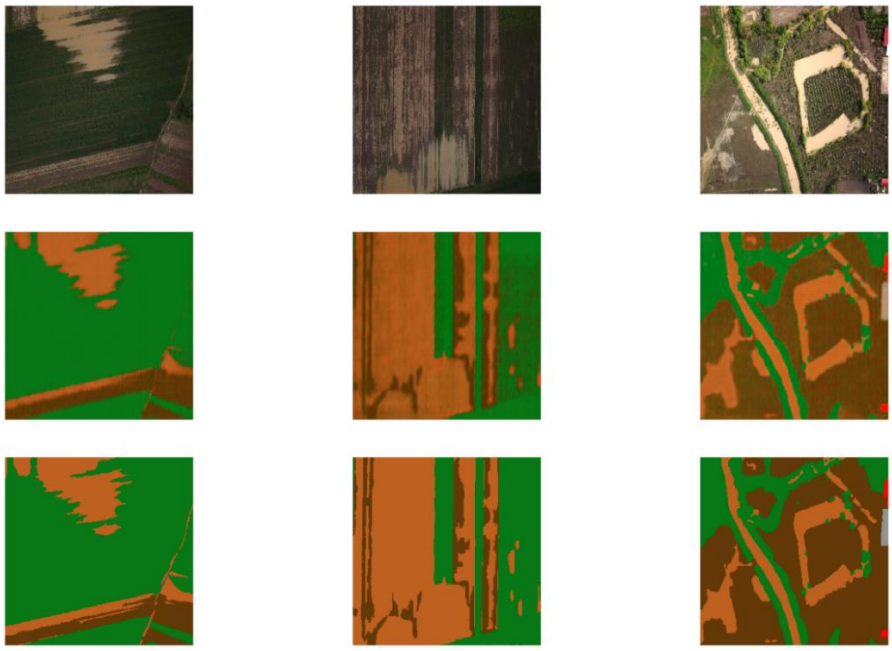


Fig. 32: Rezultate antrenare 30 epoci

Observând Fig. 32, rețeaua poate segmenta bine. Iar timpii de antrenare sunt următorii:

EPOCI	TIMPI DE ANTRENARE (SECUNDE)
3	572.16
6	1,124.65
9	1,676.38
12	2,230.27
15	2,782.40
18	3,334.59

21	3,888.47
24	4,432.65
27	4,979.84
30	5,523.94

Tabelul 9: Timpul de antrenare al rețelei rulat pe mașina locală pentru imagini de input de 256x256

Imaginile de Input au fost împărțite in 9 imagini mai mici și apoi redimensionate la 256x256. S-a antrenat rețeaua cu aceste patch-uri pentru 30 de epoci, fiecare epocă având 1000 de iterații.

Utilizând tehnicile anterioare de augmentare a imaginilor, exemplificate în Fig. 24, am obținut 4100 de imagini de antrenare având rezoluție de 256x256.

Timpii de antrenare sunt următorii pentru rularea pe mașina locală care conține un GPU NVIDIA GTX 1070 – 8GB VRAM, deoarece în Google Colab se alocă doar 12GB RAM iar încărcarea tuturor acestor poze necesita peste 28GB RAM.

EPOCI	TIMPI DE ANTRENARE (SECUNDE)
3	579.77
6	1,154.11
9	1,715.69
12	2,279.34
15	2,843.18
18	3,413.61
21	3,989.80
24	4,559.37
27	5,125.04
30	5,690.11

Tabelul 10: Timpul de antrenare al rețelei având ca input patch-uri de imagini redimensionate la 256x256

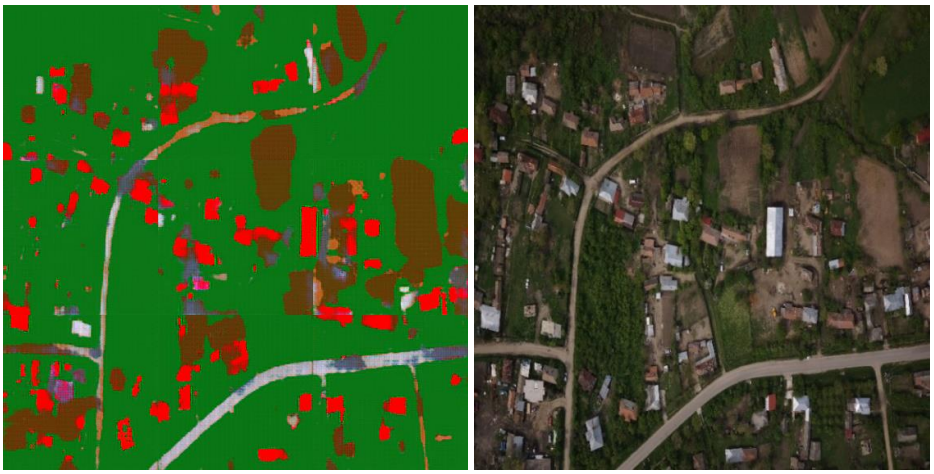


Fig. 33: Rezultatul antrenării pe patch-uri de imagini de 256x256

5. Discuții

Comparând Tabelul 1 și Tabelul 2, putem observa că avem aproximativ același timp de antrenare deoarece este aceeași structură a rețelei neuronale. Astfel se modifică doar dimensiunea imaginii de input, fiind doar necesară o reantrenare a acesteia, cu toate că avem o imagine care conține de 4 ori mai multe elemente.

Spre deosebire de antrenarea pe un CPU, prezentată în Tabelul 3: Timpul de antrenare pentru rețea folosind un procesor în Cloud-ul Google pentru imagini de input de rezoluție 256x256, unde s-a antrenat doar pentru o singură epocă având 1000 de iterații, timpul este de 3130.08 secunde. Se observă clar că acest timp este semnificativ mai mare comparativ cu executarea pe o placă video, prezentată în Tabelul 1, unde pentru același timp s-au executat 12 epoci, având un timp de 3455.63 de secunde. Putem deduce că execuția pe GPU este de aproximativ 12 ori mai rapidă decât pe un CPU. Întrucât Google, pune la dispoziție 4 plăci video și anume NVIDIA K80s, NVIDIA T4s, NVIDIA P4s, NVIDIA P100s, nu putem garanta ce placă video ne este asignată. Așadar, timpii de antrenare pot varia puțin pentru că acest model de rețea neuronală nu este foarte solicitant și poate să se antreneze pe oricare din aceste 4 plăci video.

Rulând pe un hardware specializat TPU, timpul de antrenare pentru 6 epoci este de 4142.04 de secunde. Deoarece acest hardware rulează foarte rapid trebuie să asigurăm un flux al datelor de intrare foarte mare și prin urmare antrenarea pe un TPU este mai lentă decât execuția pe un GPU.

Aceeași rețea s-a rulat și pe mașina locală pentru imagini de input de 256x256, scoțând un timp de antrenare de 5523.94 secunde, precum se vede în Tabelul 8. În comparație cu aceeași rețea antrenată în Cloud, pe mașina locală s-a executat mai repede deoarece în Cloud avem plăci video pre-emptiv, ceea ce înseamnă ca și alți utilizatori pot rula pe aceeași placă cât timp este inactivă și își pregătește datele pentru următoarea iterație.

De asemenea s-a antrenat rețeaua împărțind imaginile de input în 9 sub-imagini și s-au redimensionat la 256x256, ceea ce a dus la un set de date de 4100 de imagini folosind augmentarea imaginilor. Rețeaua se testează folosind un script care împarte imaginea de test în 9 sub-imaginii, se rulează inferența pentru aceste imagini și apoi se reconstruiește imaginea, ceea ce se poate vedea ca rezultat în Fig. 33. După cum se observă antrenarea pe patch-uri de imagini este mai slabă comparativ cu antrenarea rețelei folosind imaginea completă deoarece rețeaua pierde din conștiința pozei (context aware). Aceasta a avut un timp de antrenare pentru 30 de epoci de 5690.11 de secunde.

Întrucât pe Cloud avem disponibili doar 12GB RAM, nu s-a putut efectua antrenarea acesteia și acolo, deoarece imaginile încărcate în memorie depășeau 28GB RAM.

Inversând imaginile de input cu cele de output, putem crea imagini artificiale doar din imagini etichetate. Putem observa Fig. 27.

Dimensiunea modelelor salvate pe disc pentru rețeaua generator, au dimensiunea aproximativă de 212MB, indiferent că folosim poze de input de 256x256x3 sau 512x512x3 sau patch-uri de imagini.

Prin adăugarea unui strat nou în ambele rețele, putem introduce imagini de rezoluție mai mare, astfel crescând rezoluția de la 256x256x3 la 512x512x3, fără să afectăm calitatea imaginii foarte mult redimensionându-le. Acest lucru ne oferă o claritate mai bună a imaginii. Astfel, dimensiunea modelului salvat crește de la 212MB la 851MB, de aproximativ 4 ori mai mare. Ne așteptam să se întâmple mărirea deoarece imaginile au fost de 4 ori mare ca input. Observând Tabelul 5, avem un timp de 11,231.82 secunde pentru 30 de epoci, comparativ cu 8153.30, pentru rețeaua fără un start suplimentar, având aceleași dimensiuni de input și anume 512x512x3. Timpul a crescut de 1.37 ori.

Crescând numărul de imagini (batch size) pentru input la 4 imagini simultane, din Tabelul 6, reiese un timp de antrenare de 6612.20 de secunde pentru a 20 epoci, imaginile având o rezoluție de 256x256x3 și consumând 10.52GB de memorie GPU.

Pentru imagini de 256x256x3 dar modificând batch size-ul la 16 imagini, avem un timp de antrenare de 13,339.44 de secunde pentru antrenarea a 20 de epoci extras din Tabelul 7, fiecare epocă având 1000 de iterații. Memoria GPU consumată este asemănătoare cu antrenarea a 4 imagini simultan, dar timpul de antrenare este de aproximativ de 2 ori mai mare, deoarece efortul de calcul este crește exponențial.

Arhitectura rețelei a fost modificată prin adăugarea unui strat nou, iar acest lucru a permis creșterea rezoluției imaginilor de input de la 256x256x3 la 512x512x3. Acest model de rețea a fost antrenat cu 4 imagini simultan (batch size-ul de 4). Timpul de antrenare este de 13,244.42 de secunde pentru 15 epoci conform Tabelul 8. Astfel s-a putut realiza antrenarea în Cloud pentru că memoria consumată pe GPU este de 11.91GB, ceea ce depășește memoria maximă de pe mașina locală de 8GB. Se aștepta ca acest timp de antrenare să crească de aproximativ 3.3 ori deoarece avem imagini de input cu o rezoluție dublă și antrenăm simultan 4 imagini.

Dimensiunea batch-ului influențează dinamica algoritmului de antrenare și reprezintă un hiperparametru important. Astfel, explorarea dinamicii modelului duce la o performanță mai bună. Gradientul de antrenare este o estimare statistică. Cu cât sunt utilizate mai multe exemple de antrenare, cu atât această estimare va fi mai exactă și ponderile rețelei vor fi ajustate într-un mod mai bun care îmbunătățește performanța.

Alternativ, folosirea a mai puține exemple duce la o estimare mai slabă a gradientului, care depinde de exemplele utilizate la antrenare.

Algoritmii de optimizare care utilizează o singură imagine la un moment dat sunt numite metode stocastice sau online.

O dimensiune a batch-ului de 16 înseamnă că 16 imagini sunt utilizate simultan pentru a estima gradientul înaintea actualizărilor ponderilor.

Dimensiunile batch-ului mai mic sunt preferate din două motive:

1. Cu cât dimensiunea batch-ului este mai mică, se produce un efect de regularizare și eroarea de generalizare este mai mică.
2. Dacă dimensiunea batch-ului este mai mică, poate să fie încărcat tot batch-ul în memoria GPU.

Dimensiunea batch-ului este în general mică, în această lucrare s-au folosit batch-uri de 1, 4 și 16 imagini. Rezultatele prezente confirmă faptul că utilizarea unui batch-ul realizează cea mai bună performanță de antrenare și generalizare pentru un cost computațional dat, pentru o gamă largă de experimente.

Cu toate acestea, dimensiunea batch-ului are impact asupra rapidității de antrenare și a stabilității procesului de antrenare. Batch-urile mici învață repede, dar rezultă un proces de antrenare volatil cu o variație mai mare în acuratețea segmentării. Batch-urile mai mari încetinesc procesul de antrenare, dar etapele finale duc la o convergență mai bună și o stabilitate crescută.

6. Concluzii

În această lucrare s-a adus nou segmentarea de imagini aeriene folosind ca mască imagini color și nu doar alb și negru. În prezent există lucrări care segmentează imaginile aeriene doar pentru drumuri sau doar pentru clădiri, ci nu combinat.

Utilizarea eficientă a datelor este esențială atunci când se utilizează un hardware specializat, deoarece este imposibil să utilizăm un TPU doar dacă putem furniza date suficient de rapid. Seturile de date mici pot fi încărcate în întregime în memorie. Indiferent de formatul de date utilizat, va trebui să folosim fișiere mare de ordinul a 100MB. Acest lucru este foarte important în această utilizare a hardware-ului specializat.

Deși cu performanțe mai bune, există compromisuri în comparație cu executarea unei singur pas (dimensiunea batch-ului egală cu 1). Rularea mai multor pași într-o iterație este mai flexibilă.

GPU-urile și TPU-urile pot reduce radical timpul care este necesar pentru a executa o iterație dintr-o epocă. Atingerea performanțelor maxime necesită un flux de intrare eficient care să furnizeze date pentru următoarea iterație înainte de finalizarea iterației curente.

Pregătirea de date se suprapune execuției unei iterații. În timp ce modelul execută iterația “x”, fluxul de intrare citește datele pentru iterația “x+1”. Prin acest lucru se reduce timpul de iterație maxim antrenamentului și timpul necesar pentru încărcarea datelor.

Într-o aplicație complexă, datele pot fi stocate în servere dedicate, la distanță, precum folosim în această lucrare. Astfel se încărcă datele din mediul de stocare Drive în Cloud-ul de antrenare. Aceste două locații pot fi separate geografic.

Un flux de date care citește informații poate funcționa bine atunci când citește acest lucru la nivelul local, care ar putea deveni un factor de blocare atunci când citește datele de la distanță din cauza diferențelor între stocarea locală și cea la distanță:

1. Time-to-first-byte: Citirea primului octet al unui fișier din stocarea la distanță poate dura mai mult decât citirea locală.
2. Randament de citire: Stocarea la distantă oferă de obicei lățime de bandă mare, dar citirea unui singur fișier poate să nu utilizeze toată capacitatea lățimii de bandă.

În plus după încărcarea în memorie a primilor octeți, poate fi necesară și deserializarea sau decriptarea datelor care și acesta necesită un calcul suplimentar. Acest lucru este prezent indiferent dacă datele sunt stocate local sau la distantă, dar încărcarea în memorie poate fi mai lentă în cazul în care datele se află la distanță.

Deoarece elementele de input sunt independente unele de altele, pre-procesarea poate fi paralelizată pe mai multe core-uri ale unui procesor [25].

Poate unul dintre cei mai importanți pași înaintea proiectării modelelor generative, au fost scrise în lucrarea [26], unde s-au folosit câteva tehnici și anume:

1. Eșantionarea folosind convoluții cu stride de 2x2.
2. Folosirea funcției de activare Leaky ReLU.
3. Folosirea normalizării de date.
4. Inițializarea ponderilor după distribuție gaussiană.
5. Folosirea optimizatorului de gradient Adam
6. Rescalarea imaginilor de input în intervalul [-1; 1].
7. Separarea datelor reale de cele false, netezirea etichetelor (măștilor).
8. Urmărirea funcției de pierdere a discriminatorului dacă se duce rapid spre 0 [27].

Pe viitor se va dori:

1. Rafinarea setul de date.
2. Creșterea treptată a rezoluției imaginilor de input.
3. Adăugarea de clase noi.
4. Analizarea performanțelor datelor.
5. Găsirea și antrenarea unor rețele noi care se vor plia și mai bine pe problema aceasta.
6. Creșterea treptată a dimensiunii de input.
7. Scalarea imaginilor de output, de exemplu pentru imagini de input vom avea 512x512x3 iar ca imagini de output să avem 1024x1024x3.

Preluarea ponderilor rețelei deja învățate sunt utilizarea pentru inițializarea noului model care va fi antrenat pe date similare sau să facă posibilă adăugarea unor noi clase.

Lucrul acesta este cunoscut ca fine-tuning și se folosește pentru:

1. Accelerarea antrenamentului.
2. Deficitul mic al setului de date.

Rețelele GAN când au fost lansate au avut o contribuție importantă și au îmbunătățit semnificativ la performanța segmentării semantice. Pentru a îmbunătății clasificarea zonelor omogene, se poate cerceta un ansamblu de GAN-uri cu abordări diferite. De exemplu: imaginile pot fi ajustate folosind diferite rezoluții ale imaginilor și diferite augmentări pentru intrarea GAN-urilor, ceea ce duce la o segmentare îmbunătățită a obiectelor.

Creșterea datelor adnotate ar putea îmbunătăți considerabil rezultatele. Unele probleme de clasificare ar putea fi rezolvate cu exemple antrenate adecvat. În plus etichetarea completă în detrimentul etichetării parțiale ar îmbunătății detectarea granițelor obiectelor.

7. Bibliografie

[1] P. F. T. B. Olaf Ronneberger, „U-Net: Convolutional Networks for Biomedical Image Segmentation,” Computer Science Department and BIOSS Centre for Biological Signalling Studies, Freiburg, Germany, 2015.

[2] G. P. I. K. K. M. A. L. Y. Liang-Chieh Chen, „DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs,” 2017.

[3] J. Z. K. H. K. L. Y. Y. Huikai Wu, „FastFCN: Rethinking Dilated Convolution in the Backbone for Semantic Segmentation,” 2019.

[4] D. A. V. J. S. F. Towaki Takikawa, „Gated-SCNN: Gated Shape CNNs for Semantic Segmentation,” 2019.

[5] T. C. D. f. S. U. S. Understanding. [Interactiv]. Available: https://www.cityscapes-dataset.com/benchmarks/. [Accesat 2020].

[6] M. C. M. K. J.-W. H. S. K. J. C. Yunjey Choi, „StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation,” 2018.

[7] P. Z. Q. H. H. Z. Z. G. X. H. X. H. Tao Xu, „AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks,” 2017.

[8] M.-Y. L. J.-Y. Z. A. T. J. K. B. C. Ting-Chun Wang, „High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs,” 2018.

[9] J. D. K. S. Andrew Brock, „Large Scale GAN Training for High Fidelity Natural Image Synthesis,” 2019.

[10] S. L. T. A. Tero Karras, „A Style-Based Generator Architecture for Generative Adversarial Networks,” 2019.

[11] J. D. D. B. K. S. T. L. Yan Wu, „2.10. LOGAN: Latent Optimisation for Generative Adversarial Networks,” DeepMind, London, UK, 2019.

[12] P. Dan, Vedere Artificială în Aplicații Industriale, 2006.

[13] S. Cohen, „Methodology for Estimating Floodwater Depths from Remote Sensing Flood Inundation Maps and Topography,” University of Alabama.

[14] A. Lenail. [Interactiv]. Available: http://alexlenail/me/NN-SVG/AlexNet.htm. [Accesat 2020].

[15] Convoluție, Theano, [Interactiv]. Available: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html. [Accesat 2020].

[16] ImgAug, Aleju, [Interactiv]. Available: https://imgaug.readthedocs.io/en/latest/. [Accesat 2020].

[17] J. B. Diederik Kingma, ADAM: A method for Stochastic Optimization, 2015.

[18] I. J. Goodfellow, „Generative Adverserial Networks," 2014.

[19] D. IO. [Interactiv]. Available: https://app.diagrams.net/. [Accesat 2020].

[20] G. Tensorflow. [Interactiv]. Available: https://www.tensorflow.org/. [Accesat 2020].

[21] G. Colaboratory, Google, [Interactiv]. Available: https://colab.research.google.com/. [Accesat 2020].

[22] Keras, Keras, [Interactiv]. Available: https://keras.io/. [Accesat 2020].

[23] OpenCV. [Interactiv]. Available: https://opencv.org/. [Accesat 2020].

[24] Drive, Google, [Interactiv]. Available: https://www.google.com/drive/. [Accesat 2020].

[25] Tensorflow, Google, [Interactiv]. Available: https://www.tensorflow.org/guide/data_performance. [Accesat 2020].

[26] L. M. S. C. Alec Radford, „Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," Facebook AI Research, indico Research, Boston; New York, 2016.

[27] S. Chintala, Facebook AI Research, [Interactiv]. Available: https://github.com/soumith/ganhacks. [Accesat 2020].