



UNIVERSITATEA DIN BUCUREŞTI
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INTELIGENȚĂ ARTIFICIALĂ

Îmbunătățirea unui sistem de recomandare

LUCRARE DE DISERTAȚIE

COORDONATOR ȘTIINȚIFIC
CONF. DR. BOGDAN ALEXE

ABSOLVENT
ADRIAN ISPAS

BUCUREŞTI, ROMÂNIA

IUNIE 2019

Abstract

Abstractul în limba română.

Abstract

Abstractul în limba engleză.

Cuprins

Listă de figuri	7
Listă de tabele	9
Listă fragmente de cod	10
1 Introducere	12
1.1 Motivație	12
1.2 Obiective propuse	13
1.3 Structura lucrării	13
2 Fundamente teoretice	16
2.1 Sisteme de recomandare	16
2.1.1 Notiuni generale	16
2.1.2 Strategii de recomandare	16
2.1.3 Funcții de eroare	18
2.2 Optimizarea cu metoda gradientului descendent	24
2.2.1 Notiuni generale	24
2.2.2 Variante ale metodei	24
2.2.3 Algoritmi de optimizare	25
2.3 Modelul LightFM	26
2.4 Rețele neurale convoluționale	28
2.4.1 Notiuni generale	28
2.4.2 VGG	30
2.4.3 InceptionV3	31
2.4.4 ResNet	32

2.4.5	NASNet	35
2.5	Clustere	38
2.5.1	Notiuni generale	38
2.5.2	K-nearest neighbors	39
2.5.3	Metrici de evaluare a clusterelor	40
3	Descrierea soluției	44
3.1	Implementarea modelului de recomandare	44
3.1.1	Inițializarea	44
3.1.2	Antrenarea	46
3.1.3	Evaluarea	48
3.2	Clasterizarea posterelor	49
3.3	Construcția bazei de date	50
3.4	Optimizarea parametrilor modelului	52
4	Evaluarea experimentală	57
4.1	Bază de date filme	57
4.2	Bază de date postere	58
4.3	Rezultate clusterizare postere	60
4.3.1	Sanity check	60
4.3.2	Rezultate generale	66
4.4	Rezultate sistem de recomandare	68
Bibliografie		76

Listă de figuri

2.1	Filtrarea coloborativă	17
2.2	Filtrarea bazată pe conținut	18
2.3	Matricea de interacțiuni	19
2.4	Setul de antrenare	20
2.5	Procedura de învățarea BPR	20
2.6	Online WARP Loss Optimization	22
2.7	Algoritmul k-os pentru alegerea unui element pozitiv	23
2.8	Algoritmii k-os AUC și WARP	24
2.9	Exemplu rețea convoluțională	29
2.10	Exemplu de filtru aplicat peste input	29
2.11	Exemplu de pooling	30
2.12	Configurații VGG	31
2.13	Factorizarea în filtre convoluționale mici	32
2.14	Factorizarea spațială în convoluții asimetrice	32
2.15	Clasificator auxiliar	33
2.16	Reducerea eficientă a dimensiunii	33
2.17	Arhitectura InceptionV3	34
2.18	Învățarea reziduală	34
2.19	Rețeaua ResNet	36
2.20	NAS	37
2.21	Celule normale și de reducere	37
2.22	Exemplu clustere	38
2.23	Exemplu 1 de clasificare cu kNN	41
2.24	Exemplu 2 de clasificare cu kNN	41
3.1	Structura setului de date pentru utilizatori	51

3.2 Structura setului de date pentru filme	52
4.1 Exemple de postere	59
4.2 Postere input sanity check	60
4.3 Silhouette score pentru clasterele din sanity check	61
4.4 Rezultate sanity check pentru rețeaua VGG16	61
4.5 Rezultate sanity check pentru rețeaua VGG19	62
4.6 Rezultate sanity check pentru rețeaua InceptionV3	63
4.7 Rezultate sanity check pentru rețeaua ResNet50	64
4.8 Rezultate sanity check pentru rețeaua NASNet	65
4.9 Scorul silhouette pe setul de date de postere	66
4.10 Claster cu postere predominant negre	67
4.11 Claster cu postere predominant din filme de animație	67
4.12 Claster cu postere combinate	68
4.13 Comparație acuratețea modelului cu parametrii optimi pe tipuri de metadate	69
4.14 Comparație precizie@k a modelului cu parametrii optimi pe tipuri de me-	
tadate	71
4.15 Comparație acuratețea modelului cu parametrii optimi pe tipuri de rețele	
preamtrenate	72
4.16 Comparație acuratețea modelului cu parametrii optimi pe tipuri de rețele	
preamtrenate și cu metadatele de genuri	72
4.17 Comparație precizia@k a modelului cu parametrii optimi pe tipuri de rețele	
preamtrenate	73
4.18 Comparație precizia@k a modelului cu parametrii optimi pe tipuri de rețele	
preamtrenate și cu metadatele de genuri	73

Listă de tabele

3.1	Parametrii optimizați pentru modelul de recomandare pe tipuri de featureuri	55
3.2	Parametrii optimizați pentru modelul de recomandare pe tipuri de featureuri și modele de rețele preantrenate	55

Listings

3.1	Definirea modelului de recomandare	44
3.2	Instanțierea unui model	46
3.3	Funcția de inițializare a bazei de date	47
3.4	Antrenarea modelului	47
3.5	Acuratețea unui model	48
3.6	Precizia@k a unui model	49
3.7	Spațiul parametrilor de optimizat	52
3.8	Funcție obiectiv de minimizat	53
4.1	Construcția setul de date cu postere	59

Capitolul 1

Introducere

1.1 Motivație

Volumul de date crește semnificativ de la an la an astfel până în 2020 se estimează că pentru fiecare persoană de pe planetă vor fi creați în fiecare secundă 1.7 MB de date, ceea ce înseamnă peste 13 milioane de GB creați în fiecare secundă în lume. În 2018 în fiecare minut se vizionau peste 97 de mii de ore de conținut pe Netflix. Peste 4.3 milioane de videoclipuri erau vizionate pe Youtube. Pe Spotify se ascultau 750 de mii de melodii, iar Amazon pregătea peste o mie de pachete [1].

În România, Netflix pune la dispoziție 575 de filme și 208 seriale. În Regatul Unit sunt disponibile 2425 de filme și 542 de seriale, iar în Statele Unite Ale Americii sunt disponibile 2942 de filme și 629 de seriale [2]. Amazon oferă cumpărătorilor o gamă cu un total de peste 119 milioane de produse, dintre care 44.2 milioane de cărți, 10.1 milioane de electronice sau 4.5 milioane de produse realizate manual [3].

Cu cât volumul de date pus la dispoziție de o platformă este mai mare cu atât este mai mare și necesitatea unui sistem de recomandare care să vină în ajutorul utilizatorului final pentru a explora mai ușor gama de produse oferită de respectiva platformă. De asemenea, acel sistem de recomandare se vrea a fi îmbunătățit astfel încât să ofere fiecărui utilizator o experiență cât mai personalizată prin care să recomande, în cazul platformelor de streaming video, conținut relevant pentru a fi consumat de utilizatorul final, sau în cazul platformelor de ecommerce, produse pe care utilizatorul ar fi dispus să le cumpere.

În majoritatea cazurilor sistemele de recomandare se bazează pe metadatele utilizatorilor, precum: regiunea, vîrstă, genul, ce alte produse a accesat sau cumpărat și metadatele

produselor: categoria din care face parte, ratingul acestuia. La acestea se pot adauga și alte informații precum: ce alte produse a apreciat un alt user cu un profil asemanător.

1.2 Obiective propuse

În majoritatea cazurilor primul contact pe care îl avem cu un clip de pe Youtube, cu un film sau serial de pe Netflix sau un produs de pe Amazon este contactul vizual cu imaginea de prezentare a acelui produs.

Astfel, prezenta lucrare de disertație are drept obiectiv principal introducerea în sistemul de recomandare de informații vizuale extrase din imaginile de prezentare ale produselor. Informatiile vizuale sunt reprezentate de clusterele create peste imaginile asociate produselor. Fiecare produs are o imagine de prezentare, iar fiecare imagine are un cluster căruia îi aparține din intervalul $[1, N]$ unde N este corelat cu numărul de categorii de produse din baza de date pe care se execută optimizarea. N poate fi ales și pe baza altor rationamente.

Scopul final al acestei abordări fiind acela de a observa evoluția metricilor de evaluare, în cazul nostru acuratețea și precizia@k, atunci când informația vizuală este introdusă într-un sistem de recomandare, fiind singura informație prezentă exceptând matricea de interacțiuni, dar și cum se comportă un sistem de recomandare când primește această informație împreună cu alte informații, spre exemplu categoria unui articol.

Acuratețea, în acest context, este definită ca fiind probabilitatea ca un exemplu pozitiv ales în mod aleator să fie clasat mai sus în recomandări decât un exemplu negativ ales în mod aleator. Precizia@k este definită de numărul de exemple pozitive aflate în primele k recomandări.

1.3 Structura lucrării

Prezenta lucrare de disertație începe prin detalierea fundamentelor teoretice în capitolul II unde sunt prezentate teoriile ce stau la baza realizării acestei lucrări. În deputul capitolului definim noțiunile generale despre sistemele de recomandare, tipuri de sisteme de recomandare. Odată definite noțiunile de bază, prezentăm diversele strategii de recomandare, cu punctele lor forte și mai puțin forte, folosite în implementarea un astfel

de sistem. Discuția despre sistemele de recomandare se închide prin definirea funcțiilor de eroare folosite pentru optimizare intr-un sistem.

De completat ...

Capitolul 2

Fundamente teoretice

2.1 Sisteme de recomandare

2.1.1 Noțiuni generale

Sistemele de recomandare au scopul de oferi sugestii cât mai relevante de articole utilizatorilor unei platforme pe baza unor strategii. Un sistem de recomandare poate folosi una sau mai multe strategii de recomandare după cum vom vedea în continuare. În cazul în care se folosesc cel puțin două strategii, sistemul de recomandare devine un sistem de recomandare hibrid. Prin folosirea mai multor strategii se urmărește ca fiecare strategie să vină în completarea celorlalte strategii cu avantajele sale. De cele mai multe ori, în implementarea unui sistem de recomandare, se folosește tehnica de filtrare coloborativă împreună cu o altă strategie de recomandare [4].

2.1.2 Strategii de recomandare

Filtrarea coloborativă

Filtrarea coloborativă se bazează pe faptul că utilizatorii care au în prezent preferințe similare vor avea și în viitor preferințe destul de similare. Această abordare folosește ratingurile pe care le dă utilizatorii sau oricare altă formă de a da un feedback, îmi place/nu îmi place, pentru a identifica preferințele comune dintre grupurile de utilizatori. Odată identificate preferințele se generează recomandări pe baza similarităților dintre utilizatori.

Dezavantajul acestei strategii apare în momentul în care în sistem intră un nou utili-

zator. Datorită faptului că utilizatorul este nou, sistemul nu are un istoric al preferințelor lui, iar în consecință nu îl poate asigna unui grup de utilizatori pe baza preferințelor [4].

COLLABORATIVE FILTERING

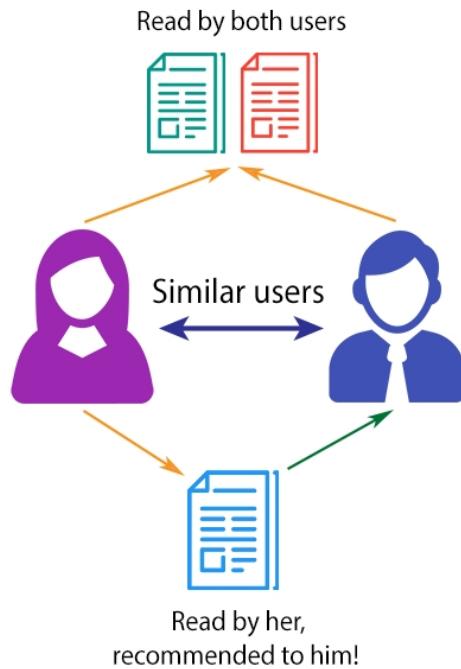


Figura 2.1: Filtrarea coloborativă. Imagine preluată din [5].

Filtrarea bazată pe conținut

Filtrarea bazată pe conținut pleacă de la premisa că utilizatorii cărora le-au plăcut articole definite de anumite caracteristici în trecut, vor aprecia aceleași tip de articole și în viitor. Această abordare folosește caracteristicile articolelor pentru a le compara cu profilul utilizatorilor și a oferi recomandări. Calitatea recomandărilor rezultate folosind această strategie este influențată de setul de caracteristici ales pentru articole. Similar cu filtrarea coloborativă, filtrarea bazată pe conținut prezintă dezavantaje în momentul în care în sistem intră un nou utilizator fără istoric [4].

Filtrarea demografică

Filtrarea demografică folosește atribute precum vârsta, genul, educația, etc. pentru a identifica categoriile de utilizatori. Nu prezintă dezavantaje atunci când apar noi utilizatori în sistem și nu se folosește de ratinguri, sau alt sistem de feedback, pentru a face recomandări.

CONTENT-BASED FILTERING

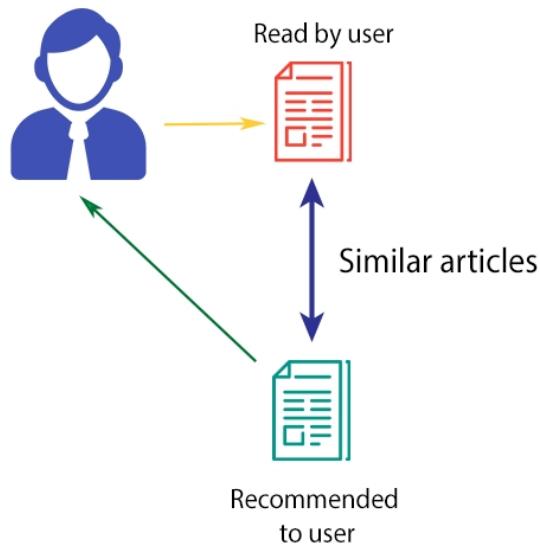


Figura 2.2: Filtrarea bazată pe conținut. Imagine preluată din [5].

Dezavantajul este reprezentat de faptul că procesul de colectare al datelor demografice poate fi îngreunat de legislație, fapt ce reprezintă o limitare a acestei metode [4].

Filtrarea bazată pe cunoștințe

Filtrarea bazată pe cunoștințe folosește cunoștințele despre utilizatori și articole pentru a spune ce articole îndeplinesc cerințele utilizatorilor și generează recomandări în consecință. Filtrare bazată pe cunoștințe are la bază constrângerile și este capabilă să recomande chiar și articole complexe care nu sunt cumpărate atât de des, precum mașini sau case [4].

2.1.3 Funcții de eroare

BPR: Bayesian Personalised Ranking

Este o metodă ce se bazează pe feedback implicit (click-uri, ratinguri, achiziții, vizualizări). Există multe metode ce se bazează pe acest feedback implicit, precum matrix factorization (MF), k-nearest neighbors (kNN), însă acestea nu sunt optimizate pentru ranguri. Metoda de învățare este bazată pe gradientul descendent și este recomandată

atunci când se dorește optimizarea acurateții.

Definim în continuare U ca fiind mulțimea de utilizatori și I ca fiind mulțimea de articole. Feedback-ul implicit este reprezentat de mulțimea $S \subseteq U \times I$. De asemenea, definim $I_u^+ := i \in I : (u, i) \in S$ și $U_i^+ := u \in U : (u, i) \in S$.

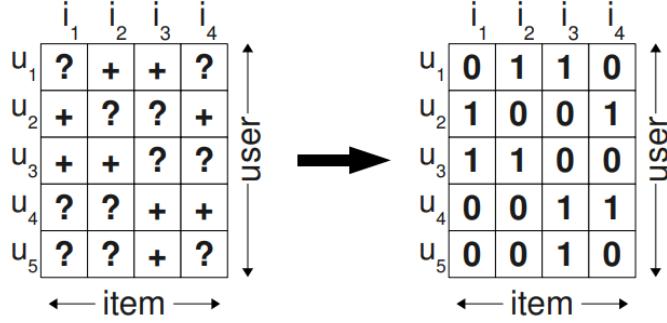


Figura 2.3: Matricea de interacțiuni, mulțimea S . Imagine preluată din [9].

O abordare uzuală pentru recomandarea de articole este să fie estimat scorul \hat{x}_{ui} care să reflecte preferința utilizatorului u pentru articolul i . Apoi fiecare articol primește un rang după sortarea scorurilor.

Setul de antrenare (vezi figura 2.4) este definit de mulțimea $D_S := \{(u, i, j) | i \in I_u^+ \wedge j \in I \setminus I_u^+\}$ unde (u, i, j) înseamnă că utilizatorul u preferă articolul i în detrimentul articolului j .

Criteriul de optimizare pentru rangurile personalizate este definit după cum urmează:

$$BPR - OPT := \sum_{(u, i, j) \in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta \|\Theta\|^2 \quad (2.1)$$

unde σ este funcția sigmoid, $\sigma(x) := \frac{1}{1+e^{-x}}$, Θ reprezintă vectorul parametru al modelului care definește interacțiunea dintre utilizatorul u , articolul i și articolul j , iar λ_Θ reprezintă parametrii de regularizare.

Cu aceste definiții putem defini și procedura de învățare a BPR după cum urmează în figura 2.5:

WARP: Weighted Approximate-Rank

Această metodă își are originile în procesarea imaginilor și anume pentru un set de reprezentări ale unor imagini $x \in R^d$ și pentru un set de reprezentări ale unor adnotări

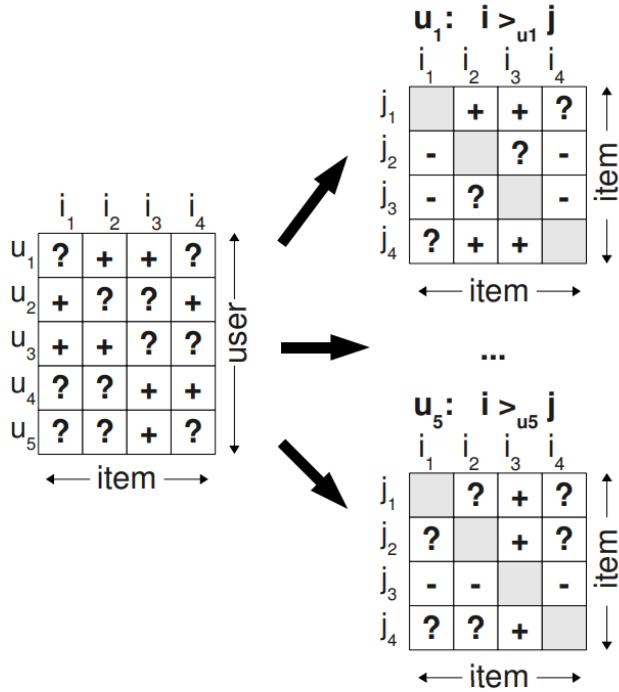


Figura 2.4: Setul de antrenare. + reprezintă articolele i pe care utilizatorul le preferă în locul articolelor j , – utilizatorul preferă articolele j în loc de i , iar ? reprezintă lipsa informației despre acea interacțiune. Imagine preluată din [9].

```

1: procedure LEARNBPR( $D_S, \Theta$ )
2:   initialize  $\Theta$ 
3:   repeat
4:     draw  $(u, i, j)$  from  $D_S$ 
5:      $\Theta \leftarrow \Theta + \alpha \left( \frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\Theta \cdot \Theta \right)$ 
6:   until convergence
7:   return  $\hat{\Theta}$ 
8: end procedure

```

Figura 2.5: Optimizarea modelului bazată metoda gradientului descendente cu parametrul de învățare α și regularizarea λ_Θ . Imagine preluată din [9].

$i \in \Upsilon = \{1, \dots, Y\}$ - inidici intr-un dicționar cu posibile adnotări, metoda învăță să mapeze imagini din spațiul reprezentărilor într-un spațiu comun R^D

$$\Phi_I(x) : R^d \rightarrow R^D \quad (2.2)$$

în același timp învățând și mapări pentru adnotări în același spațiu

$$\Phi_W(i) : 1, \dots, Y \rightarrow R^D \quad (2.3)$$

Scopul principal fiind acela de a oferi ranguri posibilelor adnotări pentru o imagine dată astfel încât cel mai mare rang să descrie cel mai bine conținutul semnatic al imaginii.

Modelul folosit este definit în continuare:

$$f_i(x) = \Phi_W(i)^T \Phi_I(x) \quad (2.4)$$

Metoda învață să producă ranguri optimizate pentru primele adnotări din listă, ceea ce înseamnă că optimizează precizia@k.

În ceea ce privește funcția de eroare definim: $f(x) \in R^Y$ ce produce un scor pentru fiecare etichetă și unde $f_i(x)$ este valoarea etichetei i . Definim funcția de eroare pentru ranguri ca fiind:

$$err(f(x), y) = L(rank_y(f(x))) \quad (2.5)$$

unde $rank_y(f(x))$ este rangul etichetei corecte date de $f(x)$:

$$rank_y(f(x)) = \sum_{i \neq y} I(f_i(x) \geq f_y(x)) \quad (2.6)$$

unde I este funcția indicator, iar $L(\cdot)$ transformă rangul în penalizare

$$L(k) = \sum_{j=1}^k \alpha_j, \quad cu \quad \alpha_1 \geq \alpha_2 \geq \dots \geq 0. \quad (2.7)$$

$L(\cdot)$ poate lua diferite forme în funcție de ce se dorește a optimiza: $\alpha_j = \frac{1}{Y-1}$ optimizează rangul mediu, $\alpha_j = 1$ și $\alpha_{j>1} = 0$ optimizează proporția de ranguri corecte aflate în top, iar valorile mari ale lui α optimizează primele k în lista de ranguri[8].

Cu definițiile prezentate mai sus putem descrie algoritmul acestei metode după cum urmează.

k-OS WARP

Abordarea k-OS WARP a fost studiată în viață reală pe două sisteme, Google Music și YouTube unde au fost obținute îmbunătățiri ale metricilor de evaluare, iar în cazul YouTube a dus la creșterea numărului de clickuri și la creșterea duratei de vizionare.

Fie D o mulțime de articole pentru un utilizator pentru care trebuie să se facă ranguri astfel încât cele mai relevante articole să fie în top. Fie U o mulțime de antrenare de utilizatori cu o mulțime de ranguri cunoscute. Considerăm interacțiunile pozitive ca fiind

Algorithm 1 Online WARP Loss Optimization

Input: labeled data (x_i, y_i) , $y_i \in \{1, \dots, Y\}$.

repeat

- Pick a random labeled example (x_i, y_i)
- Let $f_{y_i}(x_i) = \Phi_W(y_i)^\top \Phi_I(x_i)$
- Set $N = 0$.
- repeat**

 - Pick a random annotation $\bar{y} \in \{1, \dots, Y\} \setminus \{y_i\}$.
 - Let $f_{\bar{y}}(x_i) = \Phi_W(\bar{y})^\top \Phi_I(x_i)$
 - $N = N + 1$.
 - until** $f_{\bar{y}}(x_i) > f_{y_i}(x_i) - 1$ or $N \geq Y - 1$
 - if** $f_{\bar{y}}(x_i) > f_{y_i}(x_i) - 1$ **then**

 - Make a gradient step to minimize:
 $L(\lfloor \frac{Y-1}{N} \rfloor) |1 - f_y(x_i) + f_{\bar{y}}(x_i)|_+$
 - Project weights to enforce constraints (2)-(3).

 - end if**
 - until** validation error does not improve.

Figura 2.6: Online WARP Loss Optimization. Imagine preluată din [8].

date de articole pe care un utilizator le-a cumpărat, vizualizat, plăcut. Toate celelalte interacțiuni sunt considerate ca având ratinguri necunoscute. Definim D_u ca reprezentând articolele pozitive pentru utilizatorul u . Modelul factorizat este următorul:

$$f_d(u) = \frac{1}{|D_u|} \sum_{i \in D_u} V_i^T V_d \quad (2.8)$$

unde V este o matrice de dimensiune $m \times |D|$, câte un vector pentru fiecare articol conținând parametrii ce trebuie învățați. Definim în continuare $f(u)$ ca fiind vectorul tuturor scorurilor articolelor $1, \dots, |D|$ pentru un utilizator u . Pentru a învăța f , putem considera funcția de minimizare a obiectivului ca fiind:

$$\sum_{u=1}^{|U|} L(f(u), D_u) \quad (2.9)$$

unde L este funcția de eroare care măsoară discrepanța dintre ratingurile cunoscute D_u și predictiile făcute pentru utilizatorul u . În acest sens putem defini două funcții de eroare: eroare AUC și eroare WARP.

$$L_{AUC}(f(u), D_u) = \sum_{d \in D_u} \sum_{\bar{d} \in D \setminus D_u} \max(0, 1 - f_d(u) + f_{\bar{d}}(u)) \quad (2.10)$$

Această funcție este optimizată prin metoda gradientului descendente, se selectează un utilizator, un articol pozitiv și un articol negativ aleator și se face un pas al gradientului. Însă această funcție nu optimizează foarte bine rangurile elementelor din top. O funcție

care face acest lucru mai bine este:

$$L_{WARP}(f(u), D_u) = \sum_{d \in D_u} \Phi(rank_d(f(u))) \quad (2.11)$$

unde $\Phi(n)$ convertește rangul elementului pozitiv d într-o pondere unde rangul este definit după cum urmează:

$$rank_{du} = \sum_{\bar{d} \notin D_u} I(f_d(u) \geq 1 + f_{\bar{d}}(u)) \quad (2.12)$$

unde I este funcția indicator.

Pentru a generaliza funcțiile de mai sus se propune funcția de eroare numită *k-Order Statistic (k-OS)* după cum urmează: pentru un utilizator dat, u , fie o vectorul care indică ordinea elementelor pozitive în lista de ranguri $f_{D_{U_{o_1}}}(u) > f_{D_{U_{o_2}}}(u) > \dots > f_{D_{U_{o|s|}}}(u)$.

$$L_{K-OS}(f(u), D_u) = \frac{1}{Z} \sum_{i=1}^{|D_u|} P\left(\frac{i}{|D_u|}\right) \Phi(rank_{D_{U_{o_i}}}(f(u))) \quad (2.13)$$

unde $Z = \sum_i P\left(\frac{i}{|D_u|}\right)$ normalizează ponderile introduse de P . $P\left(\frac{j}{100}\right)$ este ponderea asignată fracțiunii j a elementelor pozitiv ordonate.

Cu aceste noțiuni definite putem defini algoritmul pentru funcțiile de eroare AUC și WARP în contextul K-os după cum urmează în figura 2.8.

Algorithm 1 K-os algorithm for picking a positive item.

We are given a probability distribution P of drawing the i^{th} position in a list of size K . This defines the choice of loss function.

Pick a user u at random from the training set.

Pick $i = 1, \dots, K$ positive items $d_i \in \mathcal{D}_u$.

Compute $f_{d_i}(u)$ for each i .

Sort the scores by descending order, let $o(j)$ be the index into d that is in position j in the list.

Pick a position $k \in 1, \dots, K$ using the distribution P .

Perform a learning step using the positive item $d_{o(k)}$.

Figura 2.7: Algoritmul k-os pentru alegerea unui element pozitiv. Imagine preluată din [19].

Algorithm 2 K-os WARP loss

```
Initialize model parameters (mean 0, std. deviation  $\frac{1}{\sqrt{m}}$ ).
repeat
    Pick a positive item  $d$  using Algorithm 1.
    Set  $N = 0$ .
repeat
    Pick a random item  $\bar{d} \in \mathcal{D} \setminus \mathcal{D}_u$ .
     $N = N + 1$ .
until  $f_{\bar{d}}(u) > f_d(u) - 1$  or  $N \geq |\mathcal{D} \setminus \mathcal{D}_u|$ 
if  $f_{\bar{d}}(y) > f_d(u) - 1$  then
    Make a gradient step to minimize:
     $\Phi(\frac{|\mathcal{D} \setminus \mathcal{D}_u|}{N}) \max(0, 1 + f_{\bar{d}}(u) - f_d(u))$ .
    Project weights to enforce constraints, e.g. if  $\|V_i\| > C$  then set  $V_i \leftarrow (CV_i)/\|V_i\|$ .
end if
until validation error does not improve.
```

(a) K-os WARP

Algorithm 3 K-os AUC loss

```
Initialize model parameters (mean 0, std. deviation  $\frac{1}{\sqrt{m}}$ ).
repeat
    Pick a positive item  $d$  using Algorithm 1.
    Pick a random item  $\bar{d} \in \mathcal{D} \setminus \mathcal{D}_u$ .
if  $f_{\bar{d}}(u) > f_d(u) - 1$  then
    Make a gradient step to minimize:
     $\max(0, 1 + f_{\bar{d}}(u) - f_d(u))$ .
    Project weights to enforce constraints, e.g. if  $\|V_i\| > C$  then set  $V_i \leftarrow (CV_i)/\|V_i\|$ .
end if
until validation error does not improve.
```

(b) K-os AUC

Figura 2.8: Algoritmii k-os AUC și WARP. Imagine preluată din [19].

2.2 Optimizarea cu metoda gradientului descendente

2.2.1 Noțiuni generale

Metoda gradientului descendente este unul dintre cei mai populari algoritmi folosiți pentru optimizarea rețelelor neurale. Metoda gradientului descendente presupune optimizarea funcției obiectiv, fie $J(\theta)$ - parametrizată cu parametrii modelului $\theta \in R^d$, actualizând parametrii în direcția opusă gradientului funcției obiectiv $\nabla_{\theta} J(\theta)$. Rata de învățare η determină dimensiunea pașilor pe care metoda îi face pentru a ajunge la minimul local [20].

2.2.2 Variante ale metodei

Batch gradient descent

Batch gradient descent calculează gradientul funcției de cost pentru întreg setul de antrenare:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta) \quad (2.14)$$

Pentru a face o actualizare trebuie să calculăm gradienții pentru întreg setul, astfel gradientul descendente poate fi foarte încet. De asemenea nu se poate actualiza live un model cu exemple noi.

Batch gradient descent converge garantat către un minim global pentru suprafețele convexe sau către un minim local pentru suprafețele nonconvexe.

Stochastic gradient descent

Spre deosebire de metoda prezentată anterior, stochastic gradient descent realizează o actualizare pentru fiecare exemplu de antrenare $x^{(i)}$ și pentru fiecare etichetă y^i :

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^i) \quad (2.15)$$

Această abordare realizează calcule redundante pentru un set de antrenare mare în sensul că recalculează gradienți pentru exemple similare. Această redundanță poate fi eliminată printr-o singură actualizare la un anumit moment, ceea ce o face și mai rapidă. Metoda prezintă fluctuații puternice. Fluctuațiile pot ajuta metoda să ajungă la un minim local mai bun însă aceste fluctuații pot convergență către un minim exact. Din punct de vedere al convergenței această metodă converge către un minim global pentru suprafetele convexe sau către un minim local pentru suprafetele nonconvexe.

Mini-batch gradient descent

Mini-batch gradient descent poate fi considerată ca o îmbinare a celor mai bune părți din cele două metode prezentate mai sus în sensul că se realizează o actualizare pentru fiecare mini-batch de n exemple de antrenare:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \quad (2.16)$$

În această abordare pot fi remarcate: a) poate conduce către o convergență mai stabilă prin reducerea varianței parametrilor actualizați; b) poate folosi foarte optimizat optimizarea matricelor.

Cele mai comune valori alese pentru dimensiunea mini-batchurilor sunt cuprinse între 50 și 256, însă aceste valori pot varia foarte mult de la aplicație la aplicație.

2.2.3 Algoritmi de optimizare

Adagrad

Adagrad este un algoritm utilizat pentru optimizarea gradientului care își adaptează rata de învățarea la parametrii. Realizează actualizări mari pentru parametrii rari și actualizări mici pentru parametrii frecvenți, ceea ce îl face potrivit pentru seturi de date sparse.

Adagrad utilizează o rată de învățare diferită pentru fiecare parametru θ_i la fiecare pas t . Vom seta $g_{t,i}$ ca fiind gradientul funcției obiectiv cu parametrul θ_i la pasul t :

$$g_{t,i} = \nabla_{\theta_t} J(\theta_t, i) \quad (2.17)$$

Algoritmul modifică rata generală de învățare la fiecare pas t pentru fiecare parametru θ_i bazându-se pe gradientii precedenți calculați pentru θ_i :

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \quad (2.18)$$

unde $G_t \in R^{d \times d}$ este o matrice diagonală unde fiecare element de pe diagonala, (i, i) , reprezintă suma pătratelor gradientilor, ϵ este o variabilă cu valoare mică, de obicei $1e-8$ folosită pentru a evita împărțirea la 0.

Unul dintre principalele avantaje pe care îl aduce algoritmul de optimizare Adagrad este reprezentat de faptul că elimină necesitatea de a tună manual rata de învățare, care de cele mai multe ori este setată la 0.01 și de prea puține ori modificate.

2.3 Modelul LightFM

Modelul LightFM este un model hibrid de matrix factorisation în care utilizatorii și articolele sunt reprezentate sub formă de combinații liniare de factorilor latenți a caracteristicilor. Modelul fiind hibrid, utilizează două strategii de învățare și anume învățarea coloborativă și filtrarea bazată pe conținut.

Cerințele de la care a fost dezvoltată structura modelului LightFM sunt [18]:

1. Modelul trebuie să învețe reprezentările utilizatorilor și articolele din datele de interacțiune: această cerință este realizată prin utilizarea reprezentărilor latente. De exemplu, dacă două articole, fie X și Y sunt apreciate de aceeași utilizator, atunci reprezentările celor două articole, X și Y , vor fi apropiate. Pe de altă parte, dacă articolele X și Y nu sunt apreciate de aceeași utilizator, atunci reprezentările celor două articole vor fi îndepărtate. Astfel, dacă reprezentările celor două articole X și Y sunt similare putem recomanda cu un grad ridicat de încredere articolul Y unui utilizator dacă acel utilizator a interacționat cu articolul X ;
2. Modelul trebuie să poată face recomandări pentru articole și utilizatori noi: această cerință este indeplinită cu ajutorul prin reprezentarea articolelor și utilizatorilor sub

formă de combinații liniare a caracteristicilor. Se aplică această abordare deoarece caracteristicile unui articol sau utilizator sunt cunoscute în momentul în care intră în sistem (de cele mai multe ori). De exemplu, Un film științifico-fantastic cu Leonardo DiCaprio poate fi reprezentat ca sumă a reprezentării genului științifico-fantastic și a reprezentării actorului Leonardo DiCaprio. Unu utilizator masculin din România poate fi reprezentat ca sumă a reprezentării utilizatorilor masculini și reprezentării țării România.

Din punct de vedere formal modelul LightFM este definit după cum urmează în continuare [18]. Fie U mulțimea de utilizatori, I mulțimea de articole, F^U mulțimea caracteristicilor utilizatorilor, F^I mulțimea caracteristicilor articolelor. Fiecare utilizator interacționează cu un număr de elemente prin interacțiuni pozitive sau prin interacțiuni negative. Mulțimea tuturor interacțiunilor user - articol este definită ca $(u, i) \in U \times I$, unde în această reuniune sunt incluse atât interacțiunile pozitive cât și interacțiunile negative.

Utilizatorii și articolele sunt complet descrise de caracteristicile lor. Fiecare user u este descris de un set de caracteristici $f_u \subset F^U$. Similar și pentru fiecare articol i , este descris de un set de caracteristici $f_i \subset F^I$. Caracteristicile sunt cunoscute dinainte și sunt reprezentate de metadatelor utilizatorilor sau articolelor.

Modelul este parametrizat d-dimensional pentru caracteristicile encodeate ale utilizatorilor și articolelor și anume e_f^U și e_f^I pentru fiecare caracteristică f . Fiecare feature i se mai adaugă un bias, b_f^U pentru caracteristicile utilizatorilor și b_f^I pentru caracteristicile articolelor.

Reprezentarea latentă a unui utilizator u este dată de suma vectorilor latenți de caracteristici:

$$q_u = \sum_{j \in f_u} e_j^U \quad (2.19)$$

Similar și pentru articolul i :

$$p_i = \sum_{j \in f_i} e_j^I \quad (2.20)$$

Biasul pentru utilizatorul u este dat de suma biasurilor caracteristicilor: caracteristici:

$$b_u = \sum_{j \in f_u} b_j^U \quad (2.21)$$

Similar și pentru articolul i :

$$b_i = \sum_{j \in f_i} b_j^I \quad (2.22)$$

Predictia modelului pentru utilizatorul u și articolul i este data de produsul dintre reprezentarea utilizatorului și reprezentarea articolului, ajustată cu biasurile pentru utilizator și articol:

$$\hat{r}_{ui} = f(q_u \cdot p_i + b_u + b_i) \quad (2.23)$$

unde f poate fi reprezentat de multe tipuri de funcții, însă în acest model f este setat ca fiind funcția sigmoid:

$$f(x) = \frac{1}{1 + \exp(-x)} \quad (2.24)$$

Obiectivul de optimizare al modelului constă în maximizarea probabilității următoare:

$$L(e^U, e^I, b^U, b^I) = \prod_{(u,i) \in S^+} \hat{r}_{ui} \times \prod_{(u,i) \in S^-} (1 - \hat{r}_{ui}) \quad (2.25)$$

2.4 Rețele neurale convoluționale

2.4.1 Noțiuni generale

Rețelele neurale convoluționale sunt rețelele formate din neuroni ce învață ponderi (w) și biasuri (b). Scopul rețelei convoluționale este de a primi o imagine la input și de a scoate la output un scor pentru fiecare clasă ce corespunde imaginii.

Spre exemplu, la input se dă o imagine cu un autovehicul, iar rețea convoluțională poate spune că în imagine este o mașină în proporție de 80%, un camion în proporție de 10%, un avion în proporție de 6%, o barcă în proporție de 3% sau un cal în proporție de 1%.

Rețelele convoluționale sunt compuse dintr-o secvență de straturi ce poate fi împărțită în trei tipuri principale [7]:

1. Stratul convoluțional este stratul de bază într-o rețea. Parametrii acestui strat sunt reprezentați de filtre învățabile, unde fiecare filtru reprezintă o mică bucată din imaginea de input. De exemplu, un filtru pentru acest strat poate avea dimensiunea de $5 \times 5 \times 3$, dimensiune ce reprezintă faptul că se iau 5 pixeli pe lățime, 5 pixeli

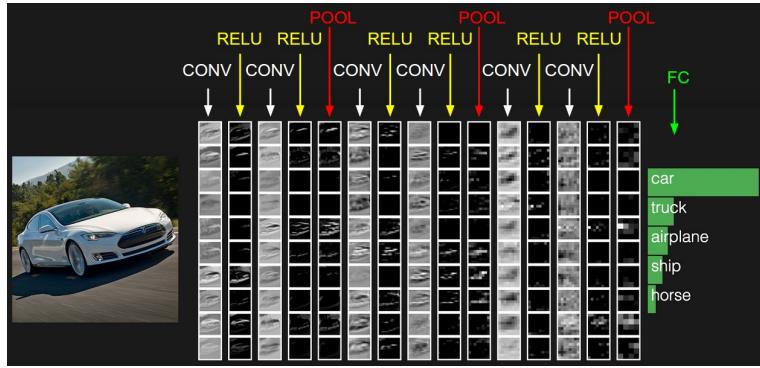


Figura 2.9: Exemplu de rețea conoluțională care primește la input o imagine și produce la output o listă de clase ce pot descrie imaginea de input. Imagine preluată din [7].

pe înălțime și o adâncime de 3 pixeli, unde adâncimea reprezintă canalele RGB. În continuare se glisează fiecare filtru peste input și se compune produsul dintre filtre și input la fiecare poziție. În urma acestei operații se produce un vector de activare 2-dimensional care reprezintă răspunsul filtrului la fiecare poziție. Altfel spus, rețeaua va învăța filtre care se activează atunci când sunt prezente anumite tipuri de caracteristici, precum culoarea sau orientarea (vezi figura 2.8).

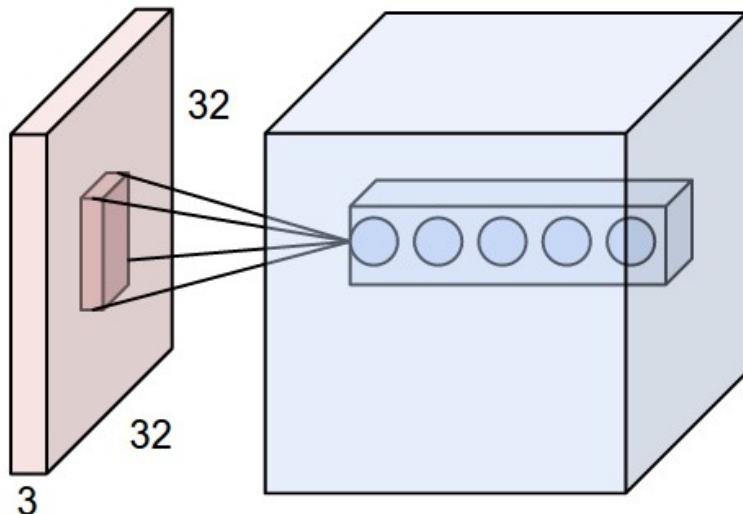
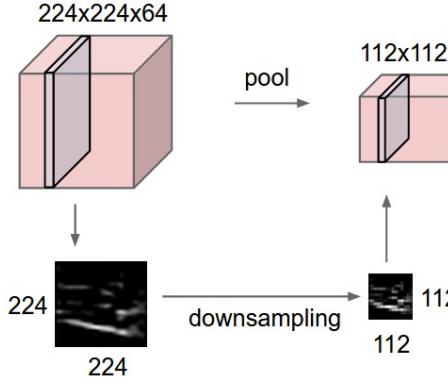
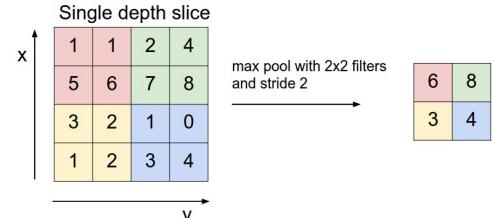


Figura 2.10: Exemplu de filtru aplicat peste input într-un strat conoluțional. Imagine preluată din [7].

2. Stratul de pooling reprezintă o practică des folosită între mai multe straturi conoluționale succesive. Această operație reduce numărul de parametrii (dimensiunea modelului), computațiile din rețea și controlează overfittingul. Se execută independent pe fiecare nivel al adâncimii unui input și pastrează valoarea maximă a acelei



(a) Reducerea dimensiunii.



(b) Filtrul de 2×2 aplicat ce păstrează valoarea maximă.

Figura 2.11: Exemplu de pooling. Imagine preluată din [7].

zone (de cele mai multe ori). Rezultatul este o zonă de caracteristicii mai mică dar care păstrează cea mai relevantă statistică (vezi figura 2.9).

3. Fully-Connected Layer este stratul în care caracteristicile sunt vectorizate pentru a putea fi folosite.

2.4.2 VGG

VGG este o arhitectură clasică de rețea cu filtre convoluționale foarte mici, de dimensiune 3×3 și care poate avea un număr de straturi de ponderi de 16 - 19.

În ceea ce privește arhitectura (vezi figura 2.10), inputul în rețea convoluțională este de dimensiune fixă și anume 224×224 imagine RGB. Mai departe, imaginea este trecută printr-un set de straturi convoluționale unde sunt utilizate filtre de dimensiune mică, 3×3 - fiind cea mai mică dimensiune ce poate captura noțiunile de stânga/dreapta, sus/jos sau centru. Într-una dintre configurații se utilizează un filtru convoluțional de dimensiune 1×1 . Pasul în straturile convoluționale este fixat la 1 pixel.

Poolingul este compus din cinci straturi de max-pooling care urmează după unele straturi convoluționale. Max-poolingul este calculat cu ferestre de 2×2 pixel și cu pas de 2 pixeli.

Odată trecută imaginea prin straturile convoluționale și cele de pooling ajunge în trei straturi fully-connected. Primele două straturi au câte 4096 de canale fiecare, iar al treilea are 1000 de canale. Canalele celui de-al treilea strat sunt asociate claselor, fiecare canal reprezentă o clasă.

Ultimul strat din rețea este un strat soft-max [10].

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figura 2.12: Configurații ale rețelei VGG. Imagine preluată din [10].

2.4.3 InceptionV3

Prima arhitectură de Inception a apărut sub numele de GoogLeNet. O a doua versiune de Inception a fost definită prin introducerea de batch-uri normalizează. Iar mai apoi, versiunea a treia în care a fost adăugate ideea de factorizare.

Factorizarea în filtre conoluționale mici (vezi figura 2.11) presupune înlocuirea stratului cu filtru de dimensiune 5×5 cu două straturi de dimensiune 3×3 astfel reducânduse dimensiunea de la $5 \times 5 = 25$ la $3 \times 3 + 3 \times 3 = 18$.

Factorizarea spațială în convoluții asimetrice (vezi figura 2.12) presupune înlocuirea stratului cu filtru de dimensiune 3×3 cu două straturi de dimensiune 3×1 și 1×3 astfel reducânduse dimensiunea de la $3 \times 3 = 9$ la $3 \times 1 + 1 \times 3 = 6$.

Clasificatorul auxiliar (vezi figura 2.13) este utilizat în InceptionV3 ca regulizator și este poziționat în partea superioară a utimelor 17×17 straturi. Batch-urile normalizează sunt de asemenea folosite în clasificatorul auxiliar.

Reducerea eficientă a dimensiunii (vezi figura 2.14) se face prin utilizarea a două

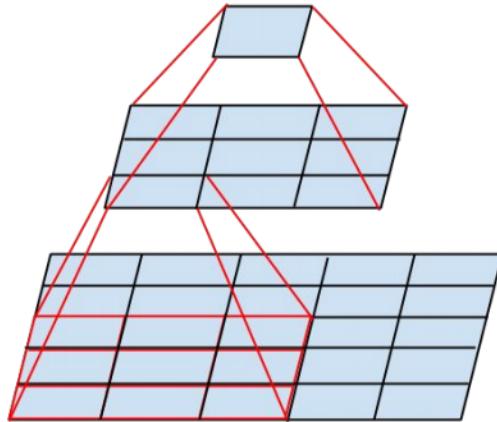


Figura 2.13: Factorizarea în filtre convoluționale mici. Filtrul de dimensiune 5×5 înlocuit cu două de dimensiune 3×3 . Imagine preluată din [14].

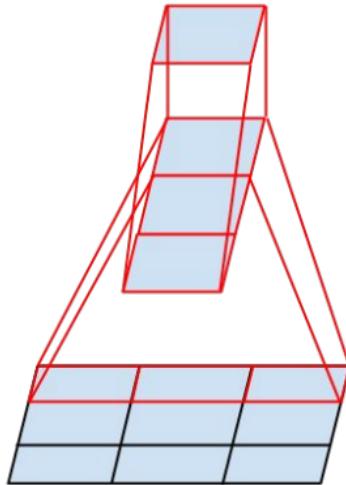


Figura 2.14: Factorizarea în filtre convoluționale mici. Filtrul de dimensiune 3×3 înlocuit cu două de dimensiune 3×1 și 1×3 . Imagine preluată din [14].

blocuri paralele, fie P și C. Primul dintre acestea, P, fiind un strat de activare de pooling (media sau maxim pooling). Ambele straturi au filtre cu pas 2 care sunt concatenate.

Arhitectura completă este prezentată în figura 2.15.

2.4.4 ResNet

Fie $H(x)$ maparea de bază unde x reprezintă inputul. Funcția reziduală poate fi aproximată cu $F(x) := H(x) - x$, maparea de bază fiind $F(x) + x$.

Învățarea reziduală se aplică la câteva grupuri de straturi. Putem defini un bloc de

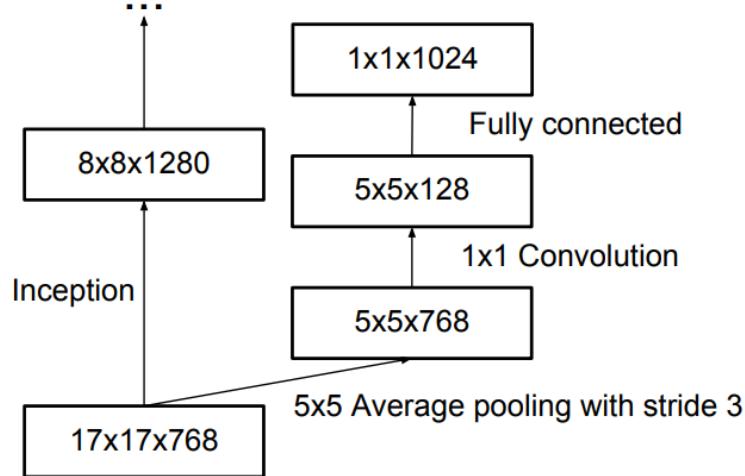


Figura 2.15: Clasificatorul auxiliar. Imagine preluată din [14].

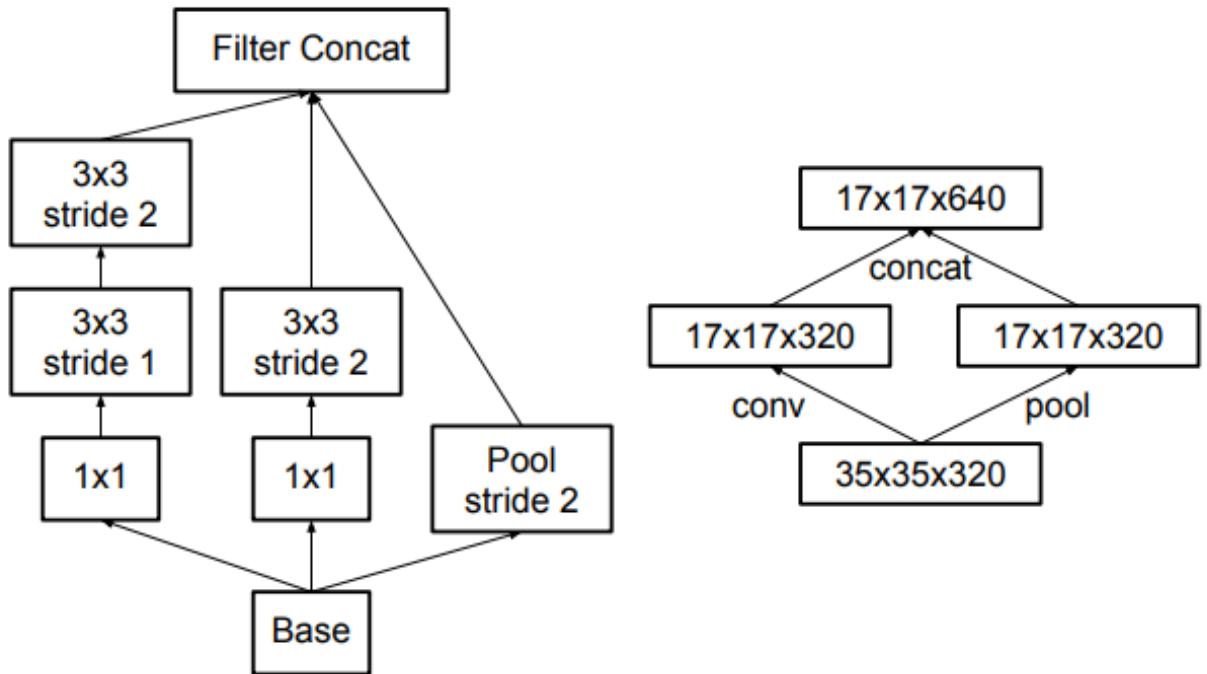


Figura 2.16: Modulul care reduce dimensiunea. Diagrama din dreapta reprezintă aceeași soluție însă din perspectiva dimensiunii rețelei. Imagine preluată din [14].

straturi ca fiind

$$y = F(x, \{W_i\}) + x \quad (2.26)$$

unde x și y reprezintă inputul și outputul straturilor considerate. Funcția $F(x, \{W_i\})$ reprezintă maparea reziduală ce trebuie învățată.

Dimensiunea lui x și F din ecuația de mai sus trebuie să fie egale. Redefinim ecuația

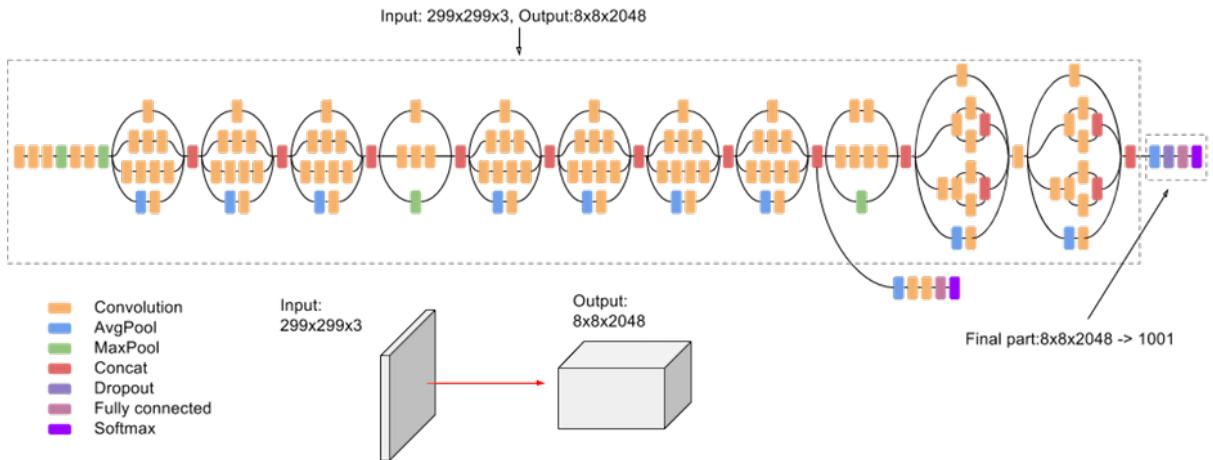


Figura 2.17: Arhitectura rețelei InceptionV3. Imagine preluată din [14].

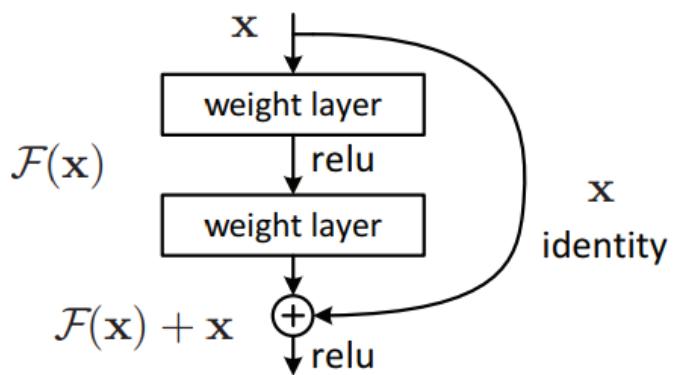


Figura 2.18: Învățarea reziduală. Imagine preluată din [11].

după cum urmează

$$y = F(x, \{W_i\}) + W_s x \quad (2.27)$$

unde W_s este o proiecție liniară a scurtăturilor conexiunilor pentru ca dimensiunile să se potrivească.

ResNet (vezi figura 2.17) pleacă de la o rețea simplă. Rețeaua simplă fiind inspirată de rețeaua VGG. Straturile convoluționale au în general filtre de dimensiune 3×3 și se bazează de două reguli de design: - pentru outputuri cu același număr de caracteristici, straturile vor avea același număr de filtre; - dacă numărul de caracteristici este injumătatit, numărul de filtre este dublat astfel încât să fie păstrată complexitatea de timp pe strat.

Poolingul se realizează după straturile convoluționale cu un pas de 2 pixeli. Rețeaua se termină cu un strat de pooling mediu și un strat fully-connected softmax cu 1000 de canale.

Bazată pe rețea descrisă mai sus, rețea reziduală presupune inserția unor scurtături. Scurtăturile identice (vezi formula 2.8) pot fi direct utilizate când inputul și outputul au aceeași dimensiune. Când dimensiunea crește considerăm două opțiuni: - scurtătura calculează în continuare maparea identității. Această opțiune nu introduce parametrii noi; - proiecția scurtătului din formula 2.9 este utilizată pentru a potrivi dimensiunile. În ambele situații când se folosesc scurtăturile pentru a sări peste două straturi sunt calculate cu un pas de 2 pixeli.

2.4.5 NASNet

NASNet este o arhitectură de rețea bazată pe tehnica de căutare Neural Architecture Search (NAS, figura 2.18). NAS presupune un controler cu o rețea neurală recurrentă care conține mai multe rețele copii cu arhitecturi diferite. Rețelele copii sunt antrenate să convergă pentru a obține o anumită precizie pe un set de antrenare. Rezultatele sunt utilizate pentru a actualiza controlerul ceea ce înseamnă că acest controler va genera arhitecturi mai bune în timp.

Plusul principal pe care îl aduce rețea NASnet este reprezentat de proiectarea unui nou spațiu de căutare astfel încât cea mai bună arhitectură pe setul de date CIFAR-10 poate scala către rezoluții ale imaginilor cât mai mari într-un interval definit. Astfel, acest spațiu poartă numele de *NASNet search space*. În abordarea NASNet, arhitecturile rețelelor convoluționale sunt manual predeterminate, fiind compuse din celule convoluționale repetitive de multe ori unde, fiecare celulă convoluțională are aceeași arhitectură dar ponderi diferite.

Pentru a construi mai ușor arhitecturi scalabile pentru imagini de orice dimensiune este nevoie de două tipuri de celule convoluționale pentru a îndeplini două funcții principale: - celule convoluționale care returnează o hartă de caracteristici cu aceeași dimensiune. Acest tip de celule se numește *Celulă Normal*; - celule convoluționale care returnează o hartă de caracteristici cu înălțimea și lungimea hărții divizată cu un factor doi. Acest tip de celule se numește *Celulă de reducere* (vezi figura 2.19).

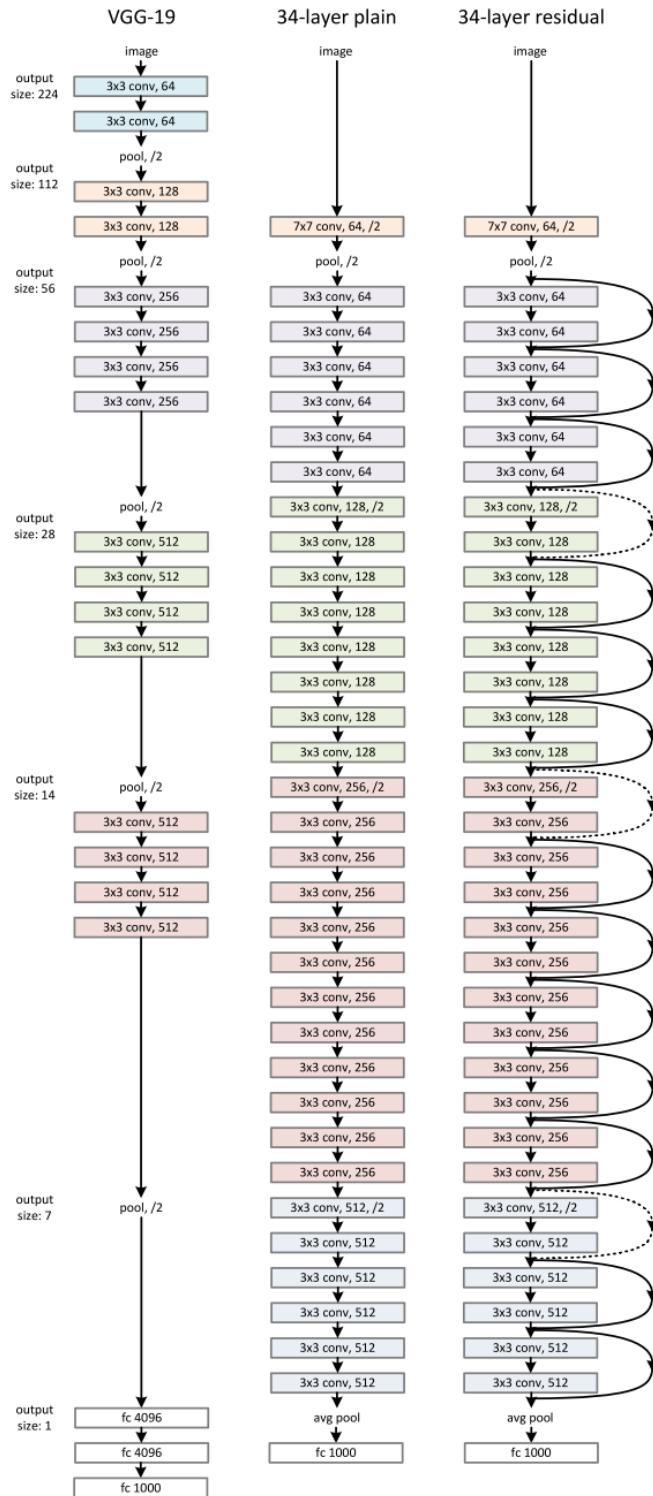


Figura 2.19: Prima rețea (stânga) este o rețea VGG19. A doua rețea (centru) este o rețea simplă cu 34 de straturi. A treia rețea (dreapta) este o rețea reziduală cu 34 de straturi. Imagine preluată din [11].

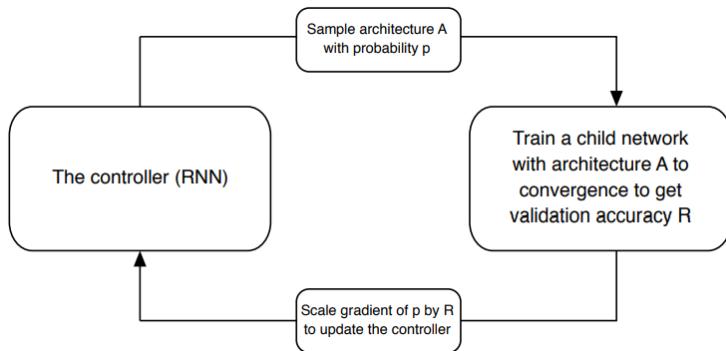


Figura 2.20: Privire de ansamblu asupra unei Neural Architecture Search. Imagine preluată din [13].

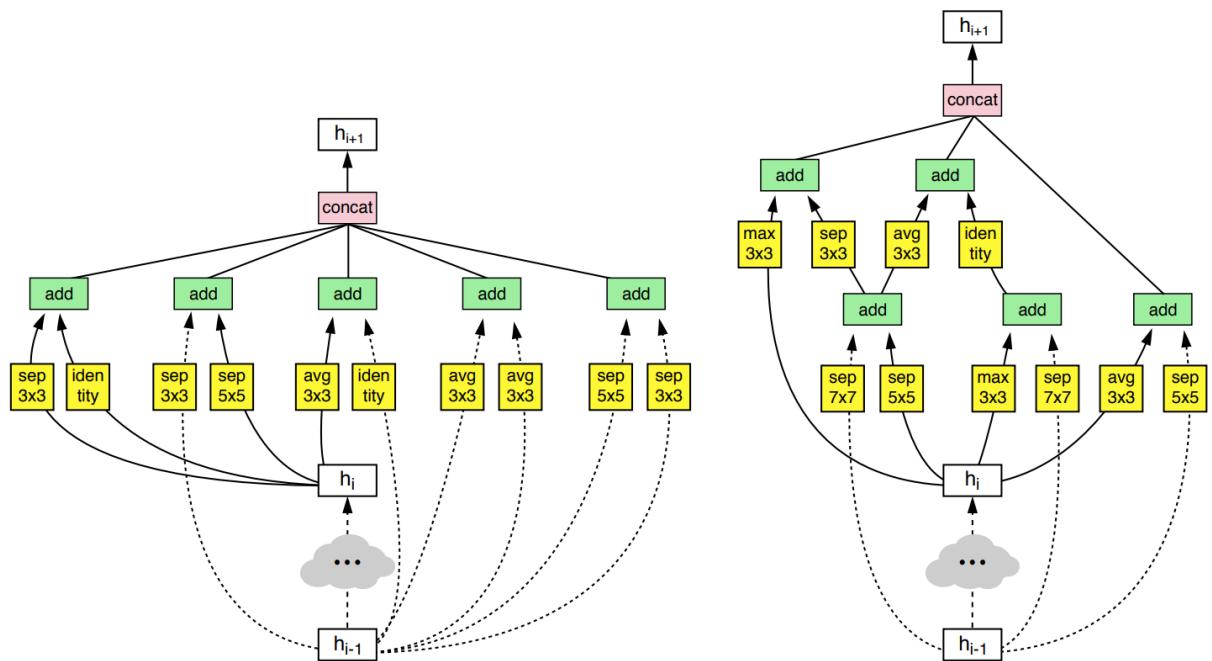


Figura 2.21: Celule normale (dreapta). Celule de reducere (stânga). Imagine preluată din [13].

2.5 Clustere

2.5.1 Noțiuni generale

Clusterizarea este un proces de grupare a unor articole în sensul în care articolele din același cluster sunt foarte similare între ele din punct de vedere al caracteristicilor. Această metodă este des utilizată în data mining, analiza datelor, machine learning, recunoașterea tipelor sau regăsirea informației. Există mai multe tipuri de algoritmi de clusterizare, însă, ideea de bază este aceași: clusterele sunt grupuri cu distanțe foarte mici între membrii grupului (fie distanța euclidiană, de exemplu).

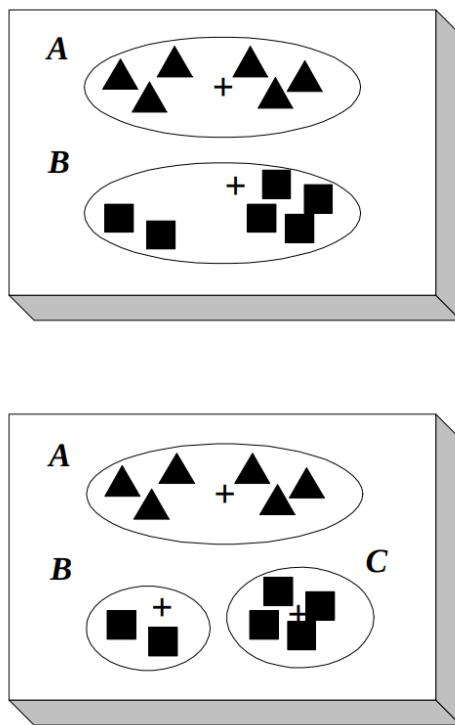


Figura 2.22: Exemplu de două clustere (sus). Exemplu de trei clustere (jos). Imagine preluată din [15].

Procesul de clustering poate fi împărțit în etape după cum urmează [15]:

1. Colectarea datelor: alegerea articolelor pentru care se va aplica clusterizarea;
2. Screening-ul inițial: presupune extragerea caracteristicilor relevante pentru fiecare articol din dataset;
3. Reprezentarea: presupune pregătirea datelor pentru a putea fi folosite de către algoritmul de clusterizare, tot aici alegânduse și măsura de similaritate;

4. Tendință de grupare: se verifică dacă datele au o tendință naturală de grupare; poate fi sărită pentru baze de date mari;
5. Strategia de clusterizare: se alege algoritmul de clusterizare și parametrii inițiali;
6. Validarea: se evaluatează manual/vizual sau prin alte metode definite rezultatele obținute în urma clusterizării;
7. Interpretarea: în această se compară rezultatele pe mai multe clustere, combinații de clustere și se trag concluziile.

2.5.2 K-nearest neighbors

KNN reprezintă un model de clasificare simplu și eficient în multe cazuri. Pentru ca un articol t să fie clasificat sunt căutați cei mai propriați k vecini formând regiunea lui t . Cei mai apropiati veci sunt căutați cu o măsură de similaritate, de obicei distanța euclidiană sau similaritatea cosinus. Votul majoritar din acea regiune este folosit pentru a decide clasificarea lui t . k -ul este valoarea de care depinde destul de mult rata de succes a clasificării, cea mai simplă metodă de a alege un k optim fiind reprezentată de rularea algoritmului pentru mai multe valori ale lui și observarea evoluției rezultelor.

Fie D o colecție de n clase cunoscute $\{d_1, d_2, \dots, d_n\}$. $Sim(d_i)$ - similaritatea celui mai îndepărtat punct din regiunea locală, $N(d_i)$ - numărul de puncte din interiorul unei regiuni locale. Algoritmul de construcție al modelului este definit după cum urmează [16]:

1. selectăm o măsură de similaritate și creem o matrice de similaritate peste baza de date de antrenare;
2. setăm toate datele cu eticheta neclasificate;
3. pentru fiecare intrare cu eticheta de neclasificat căutăm cea mai mare regiune locală care acoperă cel mai mare număr de vecini cu aceeași categorie.
4. căutăm intrarea d_i cu cea mai mare regiune N_i printre toate regiunile locale și creem o reprezentare $\langle Cls(d_i), Sim(d_i), Num(d_i), Rep(d_i) \rangle$ în modelul M pentru a reprezenta toate intrările acoperite de regiunea N_i și setăm etichete pentru toate aceste intrări;

5. repetăm pași 3 și 4 până când toate intrările din baza de date de antrenare au fost clasificate;
6. modelul M este format din toate reprezentările setate în procesul de învățatate descris mai sus.

Algoritmul de clasificare este definit după cum urmează:

1. pentru ca o nouă intrare d_t să fie clasificată, calculăm similaritatea ei cu toate celealte reprezentări din model;
2. dacă d_t este acoperit doar de o reprezentare $\langle Cls(d_j), Sim(d_j), Num(d_j), Rep(d_j) \rangle$ în sensul că distanța de la d_t la d_j este mai mică decât $Sim(d_j)$, d_t este astfel clasificat ca făcând parte din clasa lui d_j ;
3. dacă d_t este acoperit de două sau mai multe clase, clasificăm d_t ca făcând parte din reprezentarea cu cea mai mare valoare a $Num(d_j)$, adică regiunea care acoperă cel mai mare număr de intrări din baza de date de antrenare;
4. dacă nu există nicio reprezentare în modelul M care să acopere d_t , clasificăm d_t cu o clasă nouă.

Un exemplu vizual de execuție a algoritmului de kNN, parcurs pas cu pas, este prezentat în figura 2.21. Exemplu conține 36 de intrări din 2 clase marcate prin pătrat și cerc. Datele de test sunt reprezentate prin tringhiuri.

Un al doilea exemplu de clasificare este prezentat în figura 2.22:

2.5.3 Metrici de evaluare a clusterelor

Coeficientul silhouette

Coeficient (vezi formula 2.10) este folosit pentru a evalua clusterele în învățarea nesupervizată. Este calculat utilizând distanța euclidiană medie intra-clustere (a) și distanța medie către cel mai apropiat cluster (b) pentru fiecare intrare, adică distanța dintre o intrare și cel mai apropiat cluster din care nu face parte. Numărul de etichete trebuie să respecte constrângerea $2 \leq nr_{etichete} \leq nr_{etichete} - 1$. Valorile returnate de acest coeficient sunt cuprinse în intervalul $[-1, 1]$. Valorile apropiate de 0 indică clustere care se suprapun, valorile negative în general indică că există intrări asignate în clusterul greșit, iar

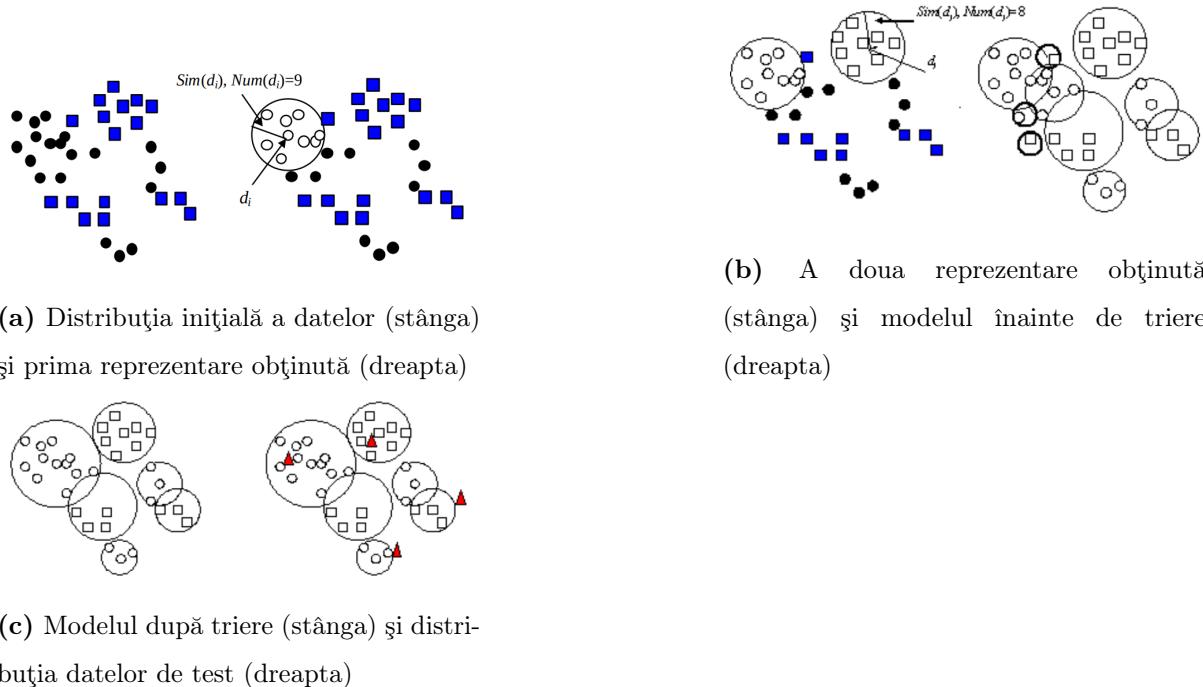


Figura 2.23: Exemplu 1 de clasificare cu kNN. Imagine preluată din [16].

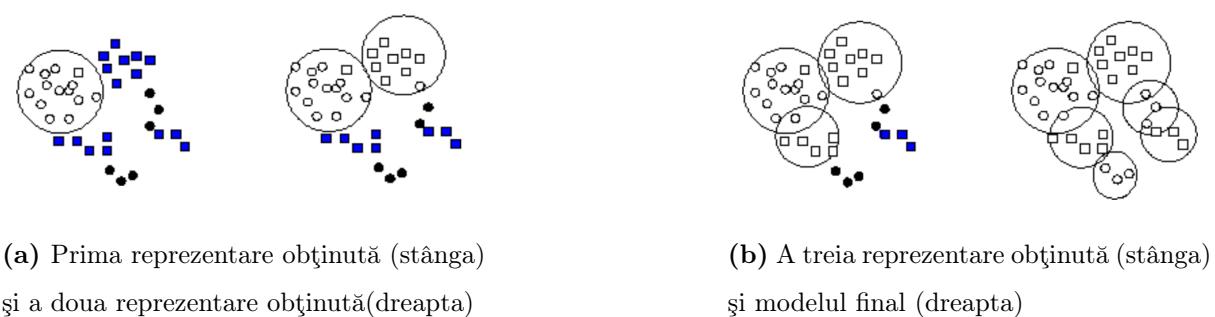


Figura 2.24: Exemplu 2 de clasificare cu kNN. Imagine preluată din [16].

valorile apropiate de 1 indică o separație bună intre clustere [17].

$$\frac{b - a}{\max(a, b)} \quad (2.28)$$

Capitolul 3

Descrierea soluției

3.1 Implementarea modelului de recomandare

3.1.1 Inițializarea

În implementarea prezentei aplicații a fost folosit framework-ul de construcție a sistemelor de recomandare *LightFM* [21] peste care am construit un wrapper. *LightFM* conține o implementare în Python a unor algoritmi de recomandare atât pentru feedback implicit cât și pentru feedback explicit. Metadatele articolelor și utilizatorilor pot fi incorporate intr-un algoritm tradițional de matrix factorization. Reprezentarea fiecărui articol și utilizator este o sumă a reprezentărilor latente ale caracteristicilor, ceea ce îi permite să generalizeze recomandările și către articole sau utilizatori noi.

În continuare este prezentată inițializarea modelului de recomandare și explicația parametrilor folosiți atât cei definiți ca fiind maleabili cât și ceilalți parametrii puși la dispoziție de framework dar asupra căror nu s-a intervenit în prezența implementare:

```
1 from lightfm import LightFM
2
3 __all__ = [ 'KingRec' ]
4
5
6 class KingRec( object ):
7     def __init__( self , no_components=50, loss='warp' , learning_rate=0.05 ,
8                  alpha=0.02, scale=0.07):
9         self .no_components = no_components
10        self .loss = loss
```

```

10     self.learning_rate = learning_rate
11     self.item_alpha = alpha
12     self.user_alpha = alpha * scale
13
14     self.model = LightFM(no_components=no_components,
15                           learning_rate=learning_rate, loss=loss,
16                           item_alpha=self.item_alpha,
17                           user_alpha=self.user_alpha,
18                           random_state=2019)

```

Listing 3.1: Definirea modelului de recomandare

Parametrii maleabili:

- **no_components**: numărul de componente reprezintă dimensiunea encodării latente a caracteristicilor utilizatorilor și articolelor. Spre exemplu, numărul de caracteristici pe fiecare user este de 1000 și avem 100 de utilizatori, ceea ce înseamnă o matrice de dimensiune 100×1000 . Cu numărul de componente setat la 50, cum este prezentat în fragmentul de cod de mai sus, înseamnă ca numărul caracteristicilor pe fiecare user va fi redus la 50 de unde rezulta noua matrice a encodărilor latente a caracteristicilor de dimensiune 100×50 .
- **learning_rate**: reprezintă rata de învățare de start pe care o folosește sistemul de recomandare. Această rată se poate schimba pe parcursul rulării ca urmare a faptului că s-a folosit algoritmul de optimizare Adagrad. Mai multe detalii despre acest algoritmi au fost prezentate în capitolul 2.2.3.
- **loss** funcția de eroare care poate fi una dintre următoarele funcții implementate: logistic, bpr, warp, warp-kos. Mai multe detalii despre funcțiile de eroare au fost prezentate în capitolul 2.1.3.
- **item_alpha** penalizarea L2 norm a caracteristicilor articolelor. Este asignată direct din parametrul *alpha*.
- **user_alpha** similar cu *item_alpha* și este rezultatul înmulțirii dintre parametrul *alpha* și *scale*.
- **random_state**: definește starea din care pleacă modelul de recomandare și este folosită de obicei pentru o reproducere mai ușoară a acelorași rezultate de la o rulare

la alta.

Alți parametrii asupra carora nu s-a intervenit în prezenta implementare:

- **k**: utilizat atunci când se folosește algoritmul k-OS și reprezintă al k -lea exemplu pozitiv ce va fi selectat din n exemple pozitive pentru fiecare utilizator. Valoarea predefinită este setată la 5.
- **n**: utilizat atunci când se folosește algoritmul k-OS și reprezintă numărul maxim de exemple pozitive pentru fiecare actualizare. Valoarea predefinită este setată la 10.
- **learning_schedule**: reprezintă algoritmul de optimizare și poate fi unul dintre adagrad sau adadelta. În experimentele realizate în această lucrare s-a folosit valoarea predefinită din model și anume adagrad.
- **max_sampled**: reprezintă numărul maxim de exemple negative folosite în antrenarea cu WARP. Valoarea predefinită este setată la 10.

Odată definit modelul de recomandare putem crea o instanță a acestuia cu parametrii doriti după cum urmează:

```
1 from kingrec import KingRec
2
3 _, learning_rate, no_components, alpha, scaling = load_params(optimized_for
4     ='auc_clusters')
5 king_rec = KingRec(no_components=no_components, learning_rate=learning_rate
6     , alpha=alpha, scale=scaling, loss='warp')
```

Listing 3.2: Instantierea unui model

Această instanță va fi folosită în continuare în antrenarea și evaluarea modelului. Funcția *load_params* încarcă parametrii optimi pentru pentru o anumită configurație. Mai multe detalii despre această optimizare a parametrilor este prezentată în capitolul 3.4.

3.1.2 Antrenarea

Cu o instanță a unui model creată putem antrena acel model pe un set de antrenare. Pentru a încărca un set de antrenare putem folosi funcția *init_movielens* care încarcă datasetul pus la dispoziție de grouplens [22]. Mai multe detalii despre structura setului de date se pot găsi în capitolul 4.1.

```
1 def init_movielens(path, min_rating=0.0, k=3, item_features=None, cluster_n  
=18, model='vgg19')
```

Listing 3.3: Funcția de inițializare a bazei de date

unde:

- **path:** este calea către folderul cu fișierele din setul de date;
- **min_rating:** este ratingul minim pentru care vor fi construite interacțiunile dintre utilizatori și articole. Spre exemplu, dacă setăm un rating minim de 3.5, mulțimea interacțiunilor pozitive va fi definită de acele interacțiuni pentru care utilizatorii au acordat cel puțin un rating de 3.5, acestea fiind considerate interacțiuni pozitive. Restul interacțiunilor sunt considerate negative;
- **k:** este un parametru folosit doar pentru statisticii, în contextul de față prezintă câți utilizatori au minim k interacțiuni în setul de date. Valoarea k este folosită în metrica de evaluare $precizie@k$ după cum este prezentat în capitolul 3.1.1;
- **item_features:** poate lua una sau mai multe valori din mulțimea '*genres*', '*clusters*' și descrie tipurile de metadate ale articolelor care se doresc să fie prezente în setul de date. Desigur, acest parametru poate fi omis, astfel nefiind adăugate metadate suplimentare pe lângă interacțiunile dintre utilizatori și articole.
- **cluster_n:** parametru folosit doar atunci când în *item_features* este prezentă valoarea *clusters* și descrie în câte clastere trebuie să fie împărțite posterele filmelor. Clasterele sunt generate în prealabil și se găsesc în path-ul menționat;
- **model:** specifică modelul cu care au fost create posterele. Poate fi unul dintre următoarele: vgg19, inception_v3, resnet50.

Funcția *init_movielens* returnează atât setul de date de antrenare cât și pe cel de testare. De asemenea, dacă este specificat prin parametrii returnează și metadatele filmelor.

Cu setul de date încarcat, putem face antrenarea cu funcția *fit* sau *fit_partial*. Diferența principală dintre cele două funcții este reprezentată de faptul că *fit_partial* reia antrenarea din starea curentă a modelului pe când *fit* începe dintr-o stare nouă.

```
1 model = king_rec.model
```

```
2 model.fit(interactions=train, item_features=item_features, epochs=epochs,
    verbose=True, num_threads=threads)
```

Listing 3.4: Antrenarea modelului

unde:

- **interactions**: primește interacțiunile definite dintre utilizatori și filme;
- **item_features**: reprezintă metadatele filmelor, dacă este cazul;
- **epochs**: reprezintă numărul de epoci pentru care vrem să facem antrenarea
- **verbose**: dacă este setat pe *True* afișează progresul antrenării;
- **num_threads**: numărul de threaduri pe care să fie executată antrenarea.

3.1.3 Evaluarea

Pe un model antrenat vom evalua două metriki pe setul de date de testare și anume *acuratețea* și *precizia@k*. Prima dintre ele, *acuratețea* este definită ca fiind probabilitatea ca un exemplu pozitiv ales în mod aleator să fie clasat mai sus în recomandări decât un exemplu negativ ales în mod aleator. Cea de-a doua, *precizia@k* este definită de numărul de exemple pozitive aflate în primele *k* recomandări.

Acuratețea poate fi calculată cu funcția *auc_score*:

```
1 test_auc = auc_score(model, test_interactions, item_features=item_features,
    num_threads=threads).mean()
```

Listing 3.5: Acuratețea unui model

unde:

- **model**: este modelul antrenat anterior;
- **test_interactions**: reprezintă interacțiunile de test create cu funcția de inițializare a bazei de date;
- **item_features**: reprezintă metadatele filmelor create cu funcția de inițializare a bazei de date;
- **num_threads**: numărul de threaduri pe care se va executa evaluarea modelului.

Rezultatul produs este reprezentat de un scor de acuratețe pentru fiecare utilizator din setul de antrenare, iar rezultatul final al modelului este reprezentat de *mean*-ul tuturor rezultatelor de acuratețe per utilizator.

Precizia@ k poate fi calculată cu funcția *precision_at_k*:

```
1 test_precision = precision_at_k(model, test_interactions, item_features=
    item_features, k=k, num_threads=threads).mean()
```

Listing 3.6: Precizia@ k a unui model

unde:

- **model, test_interactions, item_features și num_threads:** descrise ca mai sus;
- **k:** numărul de recomandări peste care se calculează precizia pentru un utilizator.

Rezultatul produs este reprezentat de un scor de precizie@ k pentru fiecare utilizator din setul de antrenare, iar rezultatul final al modelului este reprezentat de *mean*-ul tuturor rezultatelor de precizie per utilizator.

3.2 Clasterizarea posterelor

Procesul de clusterizare al posterelor presupune transformarea posterelor filmelor din imagini în clastere asociate fiecărui poster. Numărul total de clustere poate varia de la 2 până la 20 în cadrul acestei aplicații. De cele mai multe ori s-au folosit 18 clustere pentru a fi mapate 1 la 1 cu genurile filmelor.

Acest proces poate fi împărțit în două etape:

1. Extragerea caracteristicilor din postere: în extragerea caracteristicilor din postere se folosesc mai multe rețele preantrenate printre care VGG16, VGG19, InceptionV3, ResNet50 sau NASNet. Mai multe detalii despre rețelele preantrenate pot fi găsite în capitolul 2.4. Folosim fiecare dintre rețelele menționate anterior până la stratul de clasificare, adică fără top-ul rețelelor, pentru a ne extrage doar caracteristicile din imagini. Imaginele în toate cele cinci rețele preantrenate intră cu dimensiunea fixă de $224 \times 224 \times 3$, inclusiv în rețea NASNet, chiar dacă aceasta poate accepta și dimensiuni mai mari. Fiecare film are câte un set de postere asociate, după cum

este prezentat în capitolul 4.2. Din aceste seturi de postere alegem pentru fiecare film doar primul poster, poster pe care il trecem prin fiecare dintre rețelele menționate mai sus. Odată trecut prin fiecare rețea, salvăm caracteristicile aferente fiecărei rețele într-un fișier specific. Aceste fișiere sunt formate din id-ul filmului și caracteristicile extrase de rețeaua specifică din primul poster al filmului. Funcția care se ocupă de acest proces este *extract_images_features()* din modulul *extract_features.py*.

2. Creeare clusterelor pe baza caracteristicilor posterelor: odată create fișierele cu caracteristici pentru fiecare rețea putem rula funcția *explore_clusters()* din modulul *create_clusters.py*. În acest moment fiecare fișier de caracteristici va fi parcurs, iar pentru fiecare dintre aceste fișiere vom crea între 2 și 20 de clastere. La final, toate aceste clastere vor fi salvate într-un fișier asociat rețelei preantrenate. Spre exemplu, rulam funcția de creeare clastere pentru rețeaua preantrenată vgg19. Funcția va citi din fișierul asociate rețelei caracteristicile posterelor pentru fiecare film. Cu aceste caracteristici va genera 2, 4, 6, ..., 20 de clastere. La final toate clasterele sunt salvate într-un fișier specific rețelei sub forma: id film, id poster, cluster 2 (unde va fi trecut în care claster dintre cele două face parte posterul curent), cluster 4 (similar cu cluster 2), ..., cluster 20 (similar cu celelalte).

3.3 Construcția bazei de date

Pentru construcția setului de date a fost folosită clasa *Dataset* din framework-ul LightFM. Prima etapă în construirea de citirea ratingurilor date de utilizatori filmelor. Odată citite ratingurile aplicăm filtrarea de minim rating, dacă este cazul. Ratingurile astfel rămase compun matricea de interacțiune. Se face un split de 80% date de antrenare și 20% date de testare pe aceste interacțiuni.

Pe parte de caracteristici, se folosesc prestabilitătătatea pentru utilizatori și pentru filme matrici de identitate. Spre exemplu, dacă avem 500 de utilizatori și 1000 de filme vom avea matricea de caracteristici a utilizatorilor de dimensiune 500×500 având valoarea 1 pe diagonală, iar matricea de caracteristici a filmelor de dimensiune 1000×1000 , având de asemenea valoarea 1 pe diagonală.

În continuare, în funcție de parametrii setați în funcția *init_movielens* se încarcă și

celealte caracteristici specifice filmelor. Dacă avem setate genurile pentru a fi încărcate în matricea de caracteristici atunci pentru fiecare film vom citi din fișierul specific genurilor din setul de date genurile acestora. Un film poate avea unul sau mai multe genuri. Pentru a introduce aceste caracteristici în matricea de caracteristici vom aplica *one hot encoding* ceea ce presupune că pentru o mulțime de genuri de dimensiune n vom codifica mulțimea genurilor asociate unui film într-un vector de prezență de dimensiune n . Ceea ce înseamnă că dacă avem o mulțime de șapte genuri, fie ele genul 1, genul 2, ..., genul 7. Dacă un film are genul 3 și genul 7 atunci vectorul lui de caracteristici asociate genurilor va fi următorul $[0, 0, 1, 0, 0, 0, 1]$.

De asemenea funcția *init_movielens* poate inițializa și clasterele aferente fiecărui film. Procesul este similar cu cel definit mai sus pentru transferul genurilor filmelor în matricea de caracteristici.

Cu cele de mai sus definite propune un exemplu ce înglobează descrierea de mai sus: Fie un set de date cu 500 de utilizatori și 1000 de filme. Numărul total de genuri este de 18. Numărul total de clastere este corelat cu numărul total de genuri, ceea ce înseamnă că vom avea un total de 18 clastere.

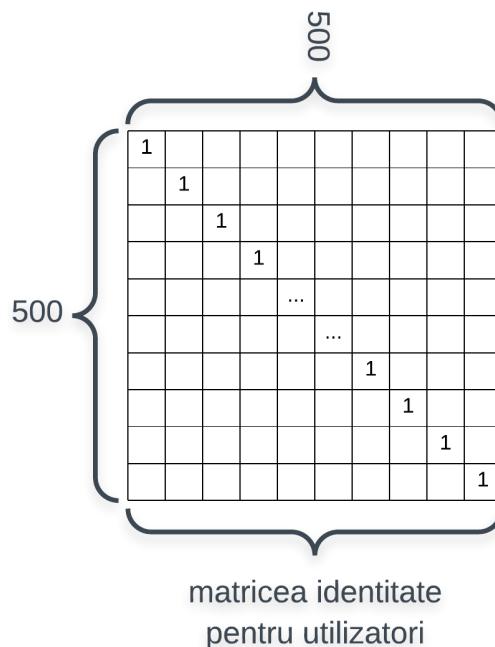


Figura 3.1: Structura setului de date pentru utilizatori.

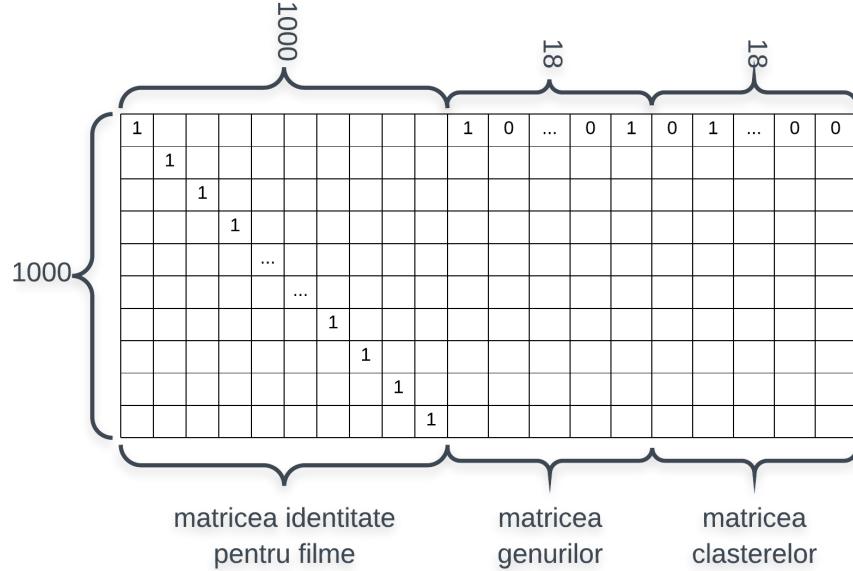


Figura 3.2: Structura setului de date pentru filme.

3.4 Optimizarea parametrilor modelului

De multe ori în algoritmi de machine learning parametrii aleşi precum rata de învățare sau numărul de epoci pot influența semnificativ rezultatul final. Astfel, pentru a reduce acest deficit există două variante: prima dintre ele presupune alegerea manuală a acestor parametrii prin diverse încercări; o a doua metodă presupune folosirea unor scripturi/librării specializate în minimizarea unor funcții definite fapt ce automatizează procesul de găsire a parametrilor optimi.

O astfel de librărie este folosită în prezența lucrare de disertație și anume librăria skopt [22]. Skopt este o librărie simplă și eficientă care minimizează funcții. Aceasta primește ca input o funcție de minimizat și un spațiu. Spațiul este definit ca fiind mulțimea parametrilor de optimizat și intervalele în care aceștia iau valori, spre exemplu:

```

1 space = [(1, 250), # epochs
2             (10 ** -4, 1.0, 'log-uniform'), # learning_rate
3             (20, 200), # no_components
4             (10 ** -6, 10 ** -1, 'log-uniform'), # item_alpha
5             (0.001, 1., 'log-uniform'), # user_scaling
6             (1, 5), # k for k-OS training
7         ]

```

Listing 3.7: Spațiul parametrilor de optimizat

Un exemplu de funcție obiectiv de optimizare este următorul:

```
1 def objective(params):
2     epochs, learning_rate, no_components, item_alpha, scale = params # '
3     k_os,
4
5
6     user_alpha = item_alpha * scale
7
8     model = LightFM(loss=loss, random_state=2019, learning_rate=
9         learning_rate,
10            no_components=no_components, user_alpha=user_alpha,
11            item_alpha=item_alpha)
12     model.fit(train, item_features=item_features, epochs=epochs,
13            num_threads=threads, verbose=True)
14
15
16     patks = function_to_optimize(model, test, item_features=item_features,
17            num_threads=threads)
18     mapatk = np.mean(patks)
19
20     # Make negative because we want to minimize objective
21     out = -mapatk
22
23
24     # Handle some weird numerical shit going on
25     if np.abs(out + 1) < 0.01 or out < -1.0:
26         return 0.0
27     else:
28         return out
```

Listing 3.8: Funcție obiectiv de minimizat

unde `function_to_optimize` poate fi una dintre funcțiile de precizie@k sau de acuratețe. De menționat că în cazul acestor două funcții o valoare mai bună înseamnă o valoare mai mare vom face outputul funcției obiectiv negativ astfel încât minimizând funcția obiectiv de fapt să aveam o valoare mai bună pentru funcția de precizie și acuratețe.

Scriptul de a fost rulat pentru a optimiza parametrii modelului în următoarele situații:

1. **fără metadate**
2. **cu genuri**
3. **cu clastere pe rețea vgg19, 18 clastere**

- 4. cu genuri și clastere pe rețeaua vgg19, 18 clastere**
- 5. cu clastere pe toate celelalte rețele preantrenat, 18 clastere**
- 6. cu genuri și clastere pe toate celelalte rețele preantrenate, 18 clastere**

Rezultatele obținute se pot vedea în tabelele 3.1 și 3.2. Analizând cele două tabele putem trage următoarele concluzii, pe baza tabelului 3.1:

1. funcția de eroare *warp* obține cele mai bune rezultate atât pe precizie@k cât și pe acuratețe fie că alegem să folosim metadatele în orice combinație a lor, fie că alegem să nu le folosim, fapt pentru care în experimentele efectuate și relatate în tabelul 3.2 am ales să folosim doar această funcție de eroare;
2. introducerea de metadate este clar o îmbunătățire față de varianta fără aceste metadate;
3. există o corelație între genuri și clastere pe postere, rezultatele pe acuratețe și precizie fiind destul de apropiate;
4. modelul de recomandare cu metadatele de clustere converge mai repede către un rezultat optim decât cu metadatele de genuri, atât pe precizie cât și pe acuratețe;
5. folosite împreună, genurile + clasterele, se obține o îmbunătățire a sistemului de recomandare de aproximativ 1% atât pe acuratețe cât și pe precizie față de modelul fără aceste metadate.

Analizând tabelul 3.2 observăm următoarele:

1. cea mai bună precizie și acuratețe pentru un model care folosește doar clastere a fost obținută cu clasterele create prin rețeaua resnet50;
2. cea mai bună precizie atunci când se folosesc atât genurile cât și clasterele a fost obținută cu clasterele create prin rețeaua vgg19;
3. cea mai bună acuratețe atunci când se folosesc atât genurile cât și clasterele a fost obținută cu clasterele create prin rețeaua inception_v3.

În capitolul următor va utiliza acești parametrii optimi găsiți pentru a studia evoluția performanței modelului în timp, studiu ce ne va permite să tragem concluziile finale.

Tabela 3.1: Parametrii optimizați pentru modelul de recomandare pe tipuri de featureuri

Features	Loss	Optimize	Optimal params						Results
			epochs	learning rate	no components	item alpha	scaling	k os	
None	warp	precision_at_k	141	0.043040683676705736	21	0.00541554967720231	0.014726505321746962		0.0920
None	warp	auc_score	93	0.013125743984880447	169	2.6154143367150727e-06	0.04382333041868763		0.9309
None	warp-kos	precision_at_k	131	0.016193013939983108	131	0.014891088630376074	0.064172162850665	3	0.0915
None	warp-kos	auc_score	136	0.025315151875417254	136	0.025315151875417254	0.0014438337247755933	5	0.9123
None	bpr	precision_at_k	145	0.011882573141627583	145	0.011882573141627583	0.008731133377250924		0.0818
None	bpr	auc_score	100	0.38336028927731636	22	0.38336028927731636	0.6705805738529935		0.8738
genres	warp	precision_at_k	136	0.075490395178898	82	0.007065549151367718	0.00799962475267643		0.0990
genres	warp	auc_score	133	0.026238747910509397	193	0.0027085249085071626	0.07322973067589604		0.9384
genres	warp-kos	precision_at_k	106	0.04588316930944897	200	0.005855900490702136	0.09739540959401453	5	0.0968
genres	warp-kos	auc_score	128	0.031396765253117284	103	5.6689548595143295e-06	0.2992760477740958	5	0.9184
genres	bpr	precision_at_k	4	0.3988094699004826	174	0.00020130127273975477	0.9668511270812562		0.0793
genres	bpr	auc_score	113	0.3787098755163822	20	1.412418076659026e-06	0.8846058572960187		0.8697
clusters	warp	precision_at_k	63	0.05647434188275842	98	0.0031993742820159436	0.0933642796909375		0.0938
clusters	warp	auc_score	42	0.0570326091236193	68	0.0029503539747277366	0.02563602355611453		0.9338
clusters	warp-kos	precision_at_k	111	0.12149792200676351	30	0.005138574720440468	0.22386245632097518	3	0.0900
clusters	warp-kos	auc_score	106	0.02060268158807219	153	0.0002768009203471932	0.01729102049278139	5	0.9139
clusters	bpr	precision_at_k	112	0.02783417000783745	53	0.043059513850700865	0.04509016538546181		0.0794
clusters	bpr	auc_score	76	0.39695046755245167	20	3.559358324483847e-05	0.749186059016229		0.8656
genres, clusters	warp	precision_at_k	96	0.1703221223672566	22	0.004206346506337412	0.041303781930858034		0.0980
genres, clusters	warp	auc_score	120	0.027730397776550147	189	0.0011133373244076297	0.4922360335772573		0.9406
genres, clusters	warp-kos	precision_at_k	83	0.07486946768773611	190	0.007918526926383375	0.012439949030585647	5	0.0916
genres, clusters	warp-kos	auc_score	149	0.037438422223599	98	6.392983080540728e-05	0.6204979332067604	5	0.9205
genres, clusters	bpr	precision_at_k	19	0.0012968105572226996	140	9.939007330655304e-05	0.0011379548833006527		0.0597
genres, clusters	bpr	auc_score	98	0.3429430411358865	21	8.687526249607698e-06	0.7296865286380925		0.8681

Tabela 3.2: Parametrii optimizați pentru modelul de recomandare pe tipuri de featureuri și modele de rețele preantrenate

Features	Loss	Optimize	Optimal params					Model	Results
			epochs	learning rate	no components	item alpha	scaling		
clusters	warp	precision_at_k	232	0.07171978672352887	42	0.006517845577815826	0.016142300018137722	vgg19	0.0935
clusters	warp	auc_score	89	0.018841927704689492	139	0.0008662511914237855	0.2864763834214625	vgg19	0.9325
genres, clusters	warp	precision_at_k	218	0.12470857345083873	73	0.005478316990150038	0.04637764141484815	vgg19	0.0995
genres, clusters	warp	auc_score	236	0.031860755009764305	139	0.0010930770083784052	0.8362665749306415	vgg19	0.9413
clusters	warp	precision_at_k	119	0.00852211930222011	192	7.276515301192984e-05	0.027052254503857717	inception_v3	0.0836
clusters	warp	auc_score	232	0.02981041359364386	84	0.004287524090264805	0.040501994149651166	inception_v3	0.9327
genres, clusters	warp	precision_at_k	245	0.028963892665938032	43	0.0006238083410955659	0.36579038826022736	inception_v3	0.0905
genres, clusters	warp	auc_score	250	0.019411170816577752	136	0.000832333176050233	0.4767783602102349	inception_v3	0.9425
clusters	warp	precision_at_k	88	0.07492160698420884	21	0.004634987385145838	0.028198967823831238	resnet50	0.0953
clusters	warp	auc_score	198	0.016780379637566917	169	0.0012939223653296507	0.6692069103186539	resnet50	0.9342
genres, clusters	warp	precision_at_k	224	0.04214027912721876	186	0.008676073688466915	0.0024915458462563605	resnet50	0.0970
genres, clusters	warp	auc_score	211	0.09767064566975311	48	0.003428832598553235	0.11239835090728653	resnet50	0.9397

Capitolul 4

Evaluarea experimentală

4.1 Bază de date filme

În cadrul prezentei diserații a fost folosită ca bază de date pentru evaluarea experimentală baza oferita de grouplens [22]. Aceștia pun la dispoziție mai multe seturi de date printre care:

1. **MovieLens 20M Dataset**: conține 20 milioane de ratinguri oferite de 138 000 de utilizatori peste 27 000 de filme;
2. **MovieLens 10M Dataset**: conține 10 milioane de ratinguri oferite de 72 000 de utilizatori peste 10 000 de filme;
3. **MovieLens 1M Dataset**: conține 1 milion de ratinguri oferite de 6 000 de utilizatori peste 4 000 de filme.

În evaluarea curentă a fost folosită cea mai recentă versiune de bază de date, actualizată la 9/2018, de dimensiune mică și anume *ml-latest-small*. Această bază de date conține 100 000 de ratinguri și 3 600 de taguri aplicate peste 9 000 de filme de 600 de utilizatori. Această bază de date este un subset dintr-o bază de date cu 27 de milioane de ratinguri și 1 milion de taguri, aplicate peste 58 000 de filme de 280 000 de utilizatori.

Fiecare utilizator din *ml-latest-small* a acordat ratinguri pentru cel puțin 20 de filme. Baza de date nu conține informații demografice, fiecare utilizator fiind reprezentat doar de un id.

Structura fișierelor din baza de date:

- **ratings.csv**: fiecare linie din acest fișier reprezintă un rating dat de un utilizator unui film, fișierul având formatul $userId, movieId, rating, timestamp$, unde $timestamp$ reprezintă momentul de timp când a fost acordat ratingul. Fiecare rating este cuprins între 0 și 5, iar între ratinguri este un pas de 0.5;
- **movies.csv**: conține metadate despre filme precum titlul și genul. Formatul fișierului este următorul: $movieId, title, genres$. Titlul conține și anul de apariție a filmului sub forma *Titlu (an)*. Genul poate fi unul sau mai multe dintre următoarele: *Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery Romance, Sci-Fi, Thriller, War Western, (no genres listed)*.
- **links.csv**: conține referințele către site-urile movielens, imdb și themoviedb sub următorul format: $movieId, imdbId, tmdbId$.
- **tags.csv**: fiecare linie conține un tag aplicat de un utilizator unui film, fișierul având următorul format $userId, movieId, tag, timestamp$. Fiecare tag este reprezentat de un cuvânt sau de o scurtă frază.

4.2 Bază de date postere

În ceea ce privește baza de date pentru postere, aceasta a fost construită în cadrul dezvoltării prezentei lucrări cu ajutorul a două elemente: fișierul de *links* din setul de date *ml-latest-small* și cu API-ul pus la dispoziție de către platforma **themoviedb.org** [24].

Procesul de construcție a acestui set de postere pentru filme poate fi rezumat pe pași după cum urmează:

1. **Pasul 1**: extragerea id-urilor filmelor aferente platformei themoviedb.org;
2. **Pasul 2**: pentru fiecare film se face un request către API pentru a lua lista de postere pentru filmul curent;
3. **Pasul 3**: având adresele posterelor pentru filmul curent, se descarcă și se salvează posterile la dimensiunea originală într-un folder specific filmului.

Din punct de vedere al implementării putem menționa funcțiile:



Figura 4.1: Exemplu de postere pentru filmul Paddington 2.

```

1 def get_tmdb_posters(tmdb_api_key, max_movie_posters=10):
2     # pasul 1
3     tmdb_movies_id = get_tmdb_ids()
4     # pasul 2 - 3
5     download_images(tmdb_api_key, tmdb_movies_id, max_movie_posters)
6
7 # unde download_images are urmatoarea definitie:
8 def download_images(tmdb_api_key, tmdb_movies_id, max_movie_posters=10):

```

Listing 4.1: Construcția setul de date cu postere

unde:

- **tmdb_api_key** este api key-ul unic asociat unui cont pe platforma themoviedb.org care a solicitat acces la API-ul platformei;
- **max_movie_posters** definește care este numărul maxim de postere care va fi descărcat pentru un film. Numărul de postere descărcate poate fi mai mic în cazul în care nu sunt cel puțin **max_movie_posters**.

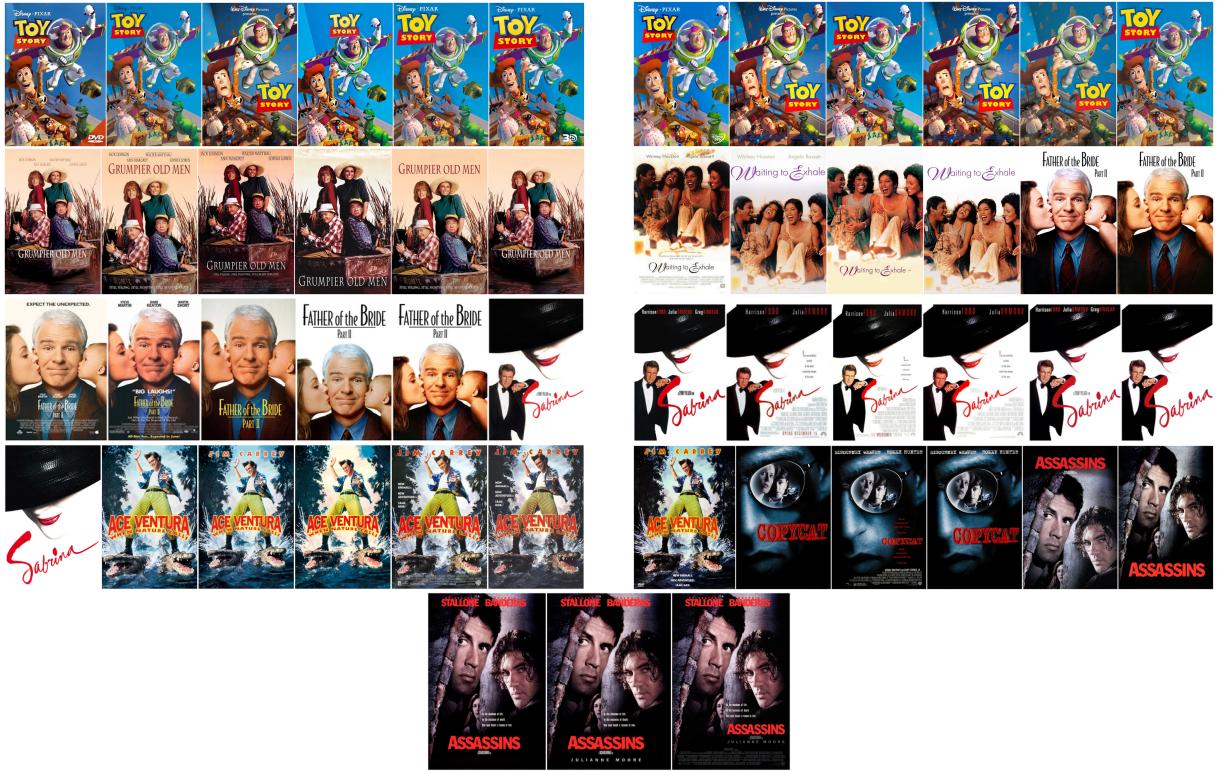


Figura 4.2: Posterele de input pentru sanity check.

4.3 Rezultate clusterizare postere

4.3.1 Sanity check

Pentru testarea validității clasterelor un test de sanity check în care s-au luat opt filme și s-a încercat clasificarea lor în șapte clastere folosind caracteristicile extrase din rețelele preantrenate VGG16, VGG19, InceptionV3, ResNet50 și NASNet, clasterele create cu kNN iar evaluarea este făcută atât vizual cât și prin urmărirea evoluției metricii silhouette de la 2 la 7 clastere. Posterele de input pentru sanity check sunt prezentate în figura 4.2.

Din punct de vedere al metricii de evaluare silhouette rezultatele se prezintă după cum urmează în figura 4.3. După cum se observă rețelele InceptionV3 și NASNet pare a ieși mereu mai bine decât celelalte, cu un avantaj al rețelei InceptionV3. Această observație fiind valabilă pentru toate clasterele de la 2 la 7. Toate celelalte rețele sunt destul de apropiate în rezultate în special începând cu clasterul 3 și mai puțin la clasterul 2. Toate celelalte rețele, cu excepția ResNet50 au fluctuații. ResNet50 este singura care păstrează în mod continuu trendul ascendent de la clasterul 2 până la clasterul 7.

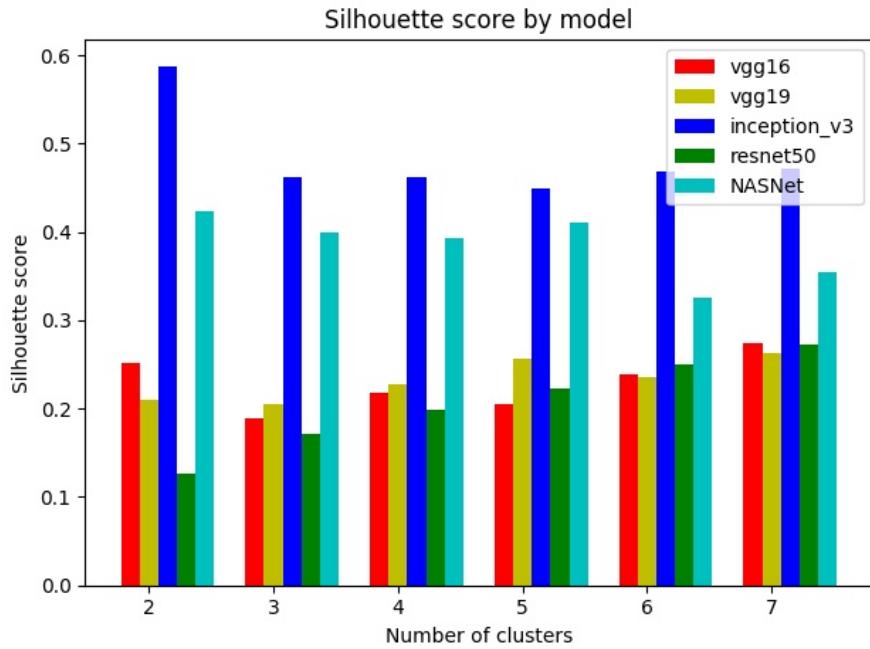


Figura 4.3: Silhouette score pentru clasterele din sanity check.



Figura 4.4: Rezultate sanity check pentru rețeaua VGG16

Din punct de vedere vizual, rezultatatele clasterelor sunt prezentate în figurile 4.4 - 4.8.



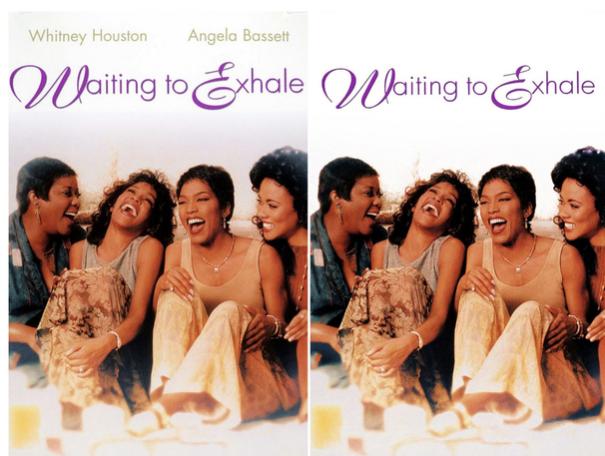
(a) claster 1



(b) claster 2



(c) claster 3



(d) claster 4



(e) claster 5

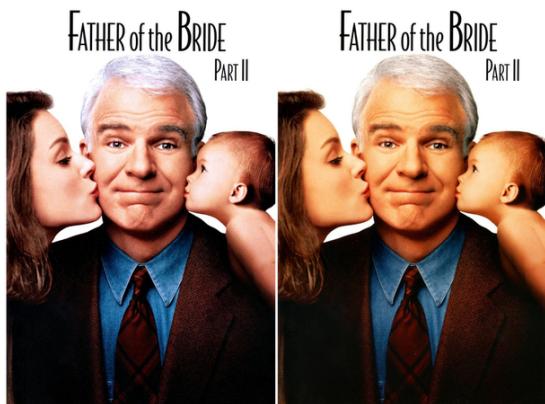


(f) claster 6



(g) claster 7

Figura 4.5: Rezultate sanity check pentru rețeaua VGG19



(a) claster 1



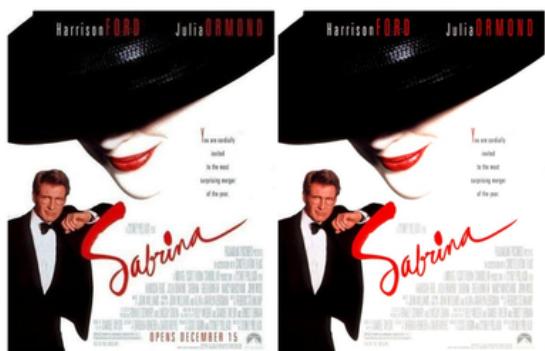
(b) claster 2



(c) claster 3



(d) claster 4



(e) claster 5



(f) claster 6



(g) claster 7

Figura 4.6: Rezultate sanity check pentru rețeaua InceptionV3



(a) claster 1



(b) claster 2



(c) claster 3



(d) claster 4



(e) claster 5



(f) claster 6



(g) claster 7

Figura 4.7: Rezultate sanity check pentru rețeaua ResNet50

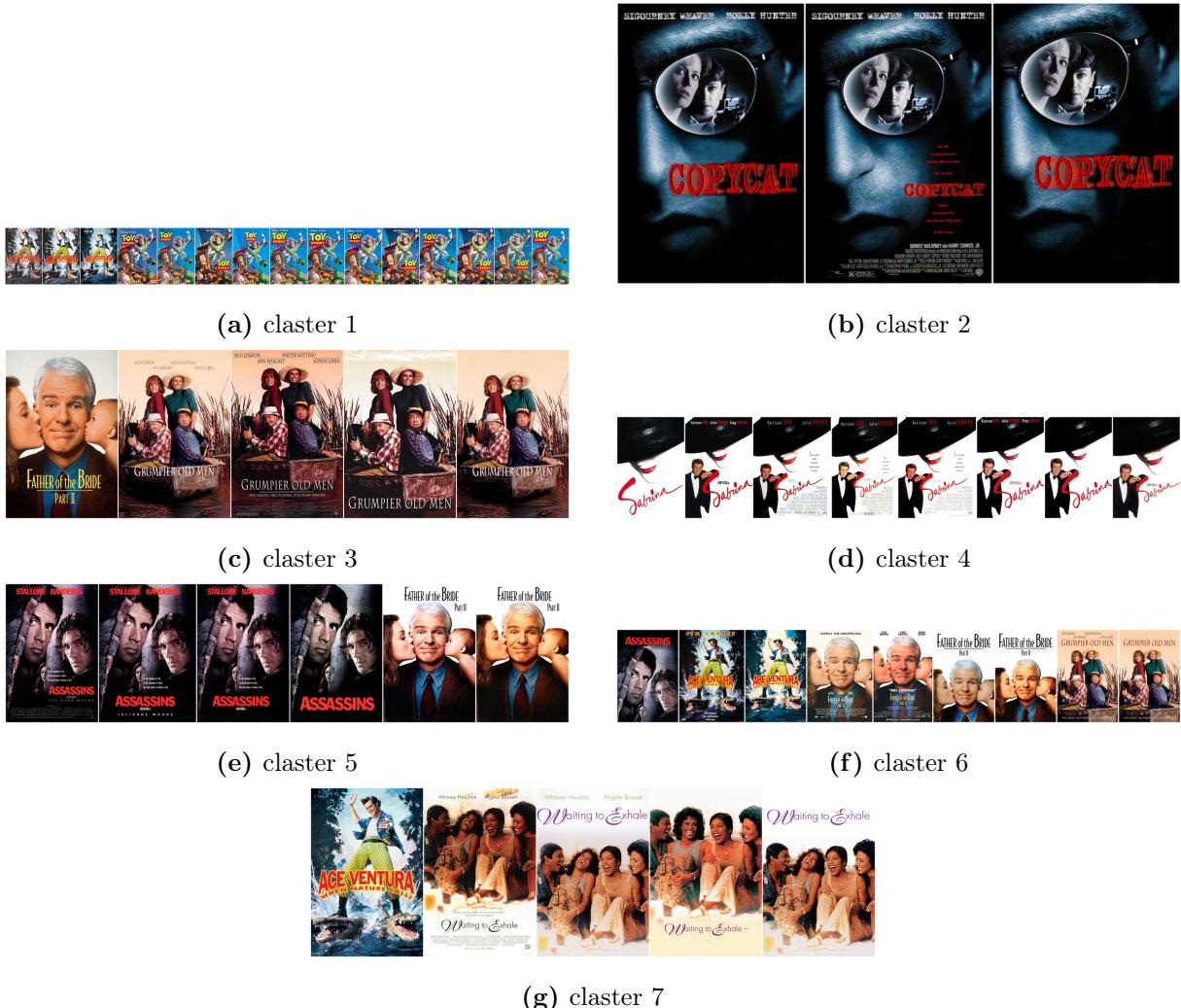


Figura 4.8: Rezultate sanity check pentru rețeaua NASNet

După cum se observă, din punct de vedere vizual doar rețeaua ResNet50 reușește să treacă acest sanity check clasificând săse filme în șase claster diferite, iar două filme în același cluster. Din punct de vedere vizual posterele celor două filme clasificate în același cluster sunt destul de similare având în prim plan mai multe persoane, predominant începând cu partea de jos a posterelor, iar în partea de sus fiind conținut alb, în general, cu text.

Observăm că deși InceptionV3 și NASNet aveau scoruri mai mari pe coeficientul silhouette, rețeaua ResNet50 are rezultate vizuale mai bune pe sanity check. De remarcat corelația dintre faptul că rețeaua ResNet50 este singura a cărui coeficient silhouette creștea de la un claster la altul, ceea ce înseamnă că acele claster devineau din ce în ce mai bune pe măsură ce ne apropiam de numărul de claster din realitate.

4.3.2 Rezultate generale

Pentru evaluarea clasterelor pe toate pe toate posterele filmelor, aproximativ 9 000, au fost eliminate din evaluare rețelele VGG16 și NASNet în baza rezultatelor prezentate în tabelul 3.1, în cazul rețelei VGG16 și a costului de memorie și timp de execuție din timpul experimentelor, în cazul rețelei NASNet. Se observă că rețeaua InceptionV3 are

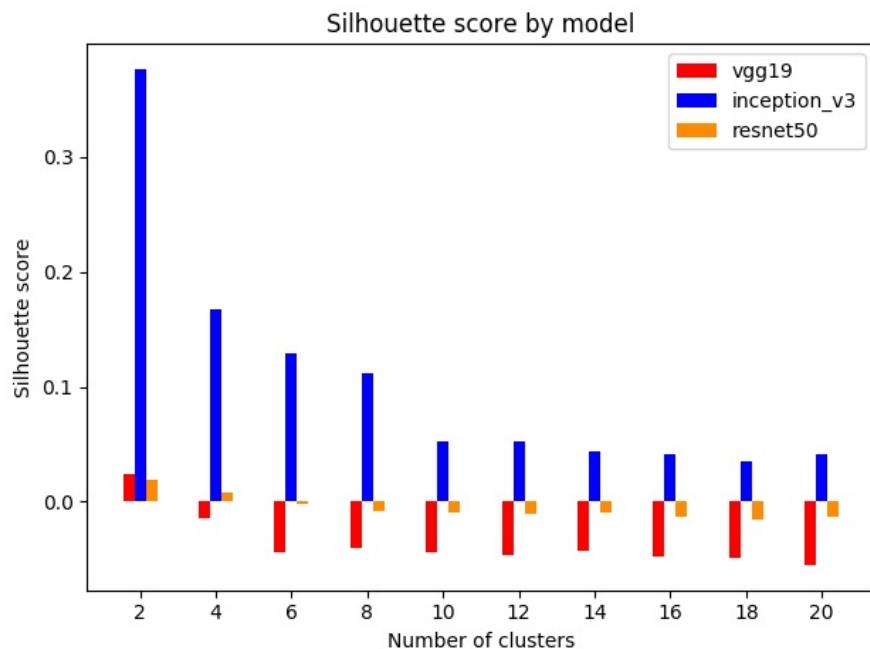


Figura 4.9: Scorul silhouette pe setul de date de postere.

scorurile pentru toate clasterele de la 2 la 20 pozitive. Rețelele ResNet50 și VGG19 sunt

negative, însă apropiate de 0 cu un avantaj de scoruri mai bune pentru rețeaua ResNet50.

În figurile 4.10 - 4.12 sunt prezentate rezultatele vizuale din trei claster create cu rețeaua preantrenată ResNet50.

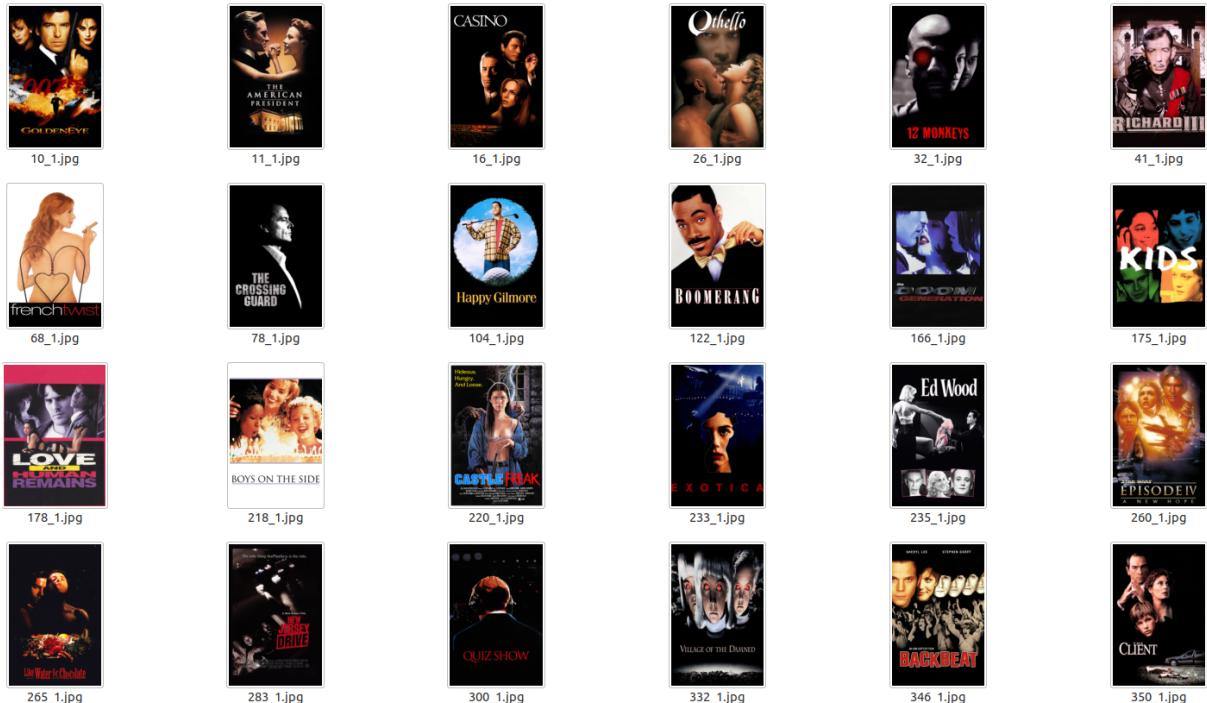


Figura 4.10: Claster cu postere predominant negre.

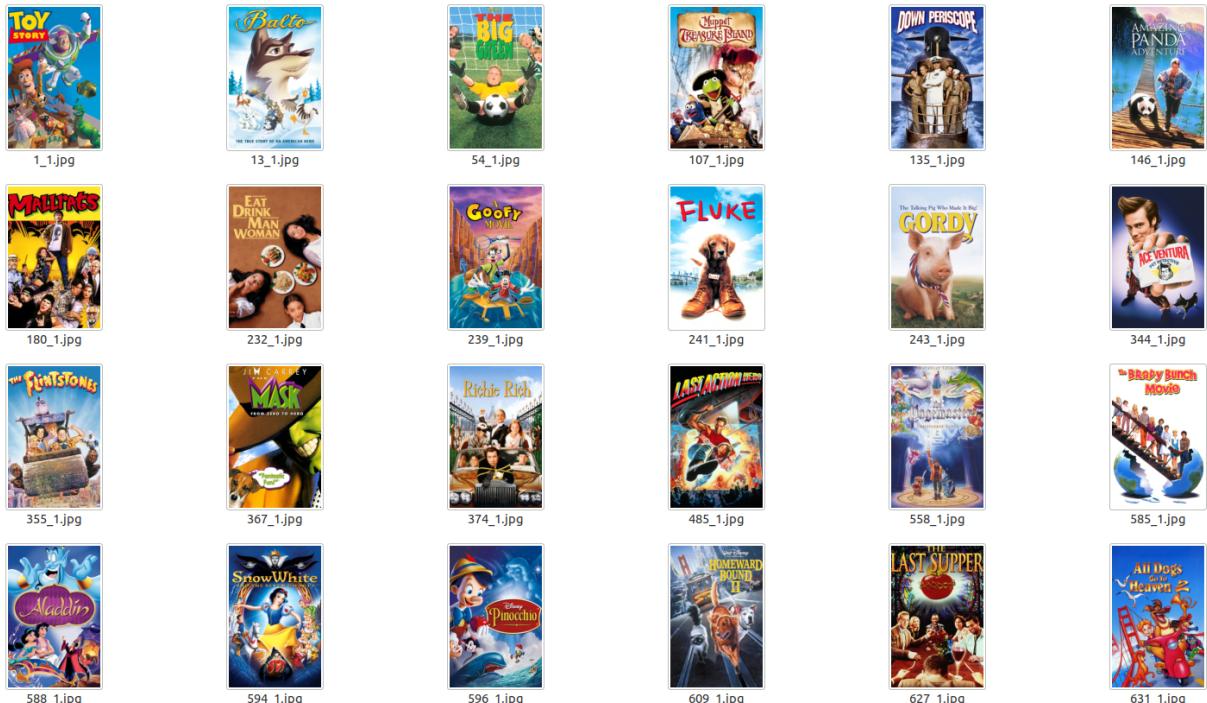


Figura 4.11: Claster cu postere predominant din filme de animație.

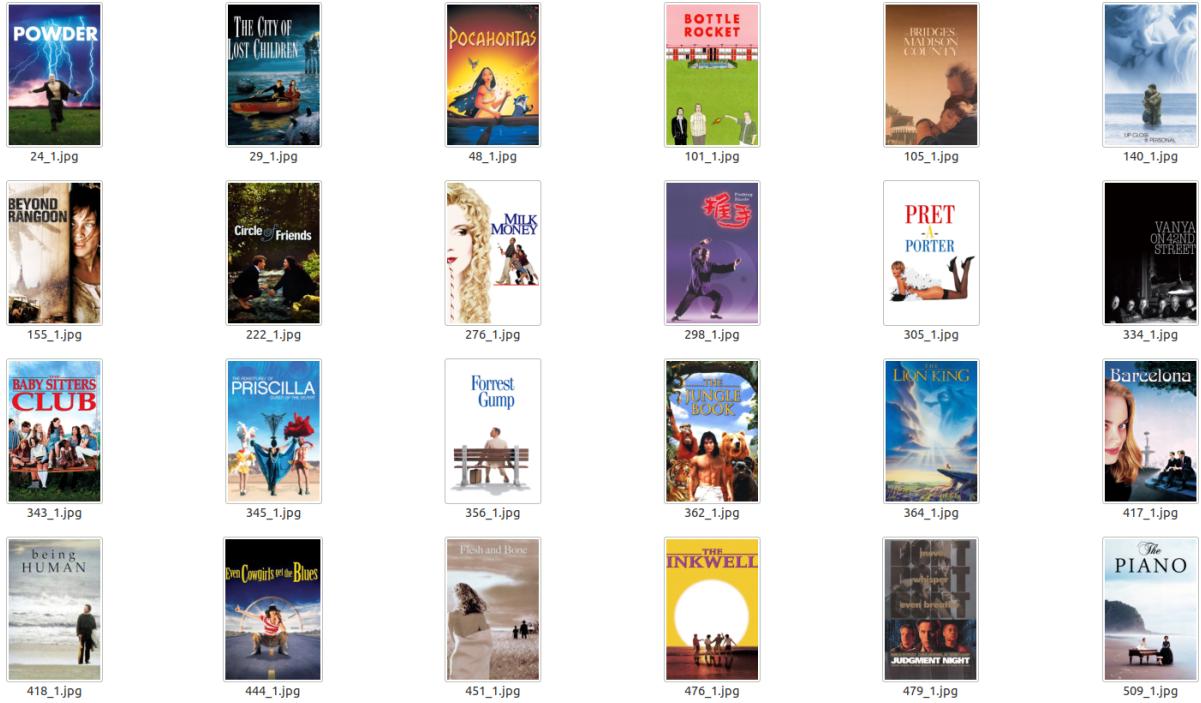


Figura 4.12: Claster cu postere combinate.

4.4 Rezultate sistem de recomandare

În continuare vom studia rezultatele obținute de metricile de evaluare **precizie@k** și **acuratețe** pe timp de antrenare în epoci, pe metadate folosite în antrenarea modelului și pe tipuri de rețele preantrenate cu care au fost generate clasterele posterelor. Toate mențiunile către parametrii optimi pentru o anumite configurație se referă la parametrii prezentați în tabelele 3.1 și 3.2.

Înainte de a prezenta rezultatele metricilor trebuie menționate două aspecte importante legate de configurația setului de date utilizat:

1. ratingul minim pentru interacțiuni a fost setat la 3.5, ceea ce înseamnă că s-au folosit ca interacțiuni pozitive doar acele interacțiuni care au cel puțin acest rating;
2. rezultatul maxim al metricii de precizie@k este direct influențat de numărul interacțiuni dintre utilizatori și filme. Spre exemplu, dacă un utilizator are un număr total de interacțiuni de 5, iar metrica noastră de precizie are k-ul setat la 10, atunci rezultatul maxim pentru precizie@5 va fi 0.5, deoarece doar maxim 50% deoarece nu există mai multe de 5 interacțiuni pozitive pentru acel utilizator.

În cazul împărțirii utilizate numărul de interacțiuni se prezintă după cum urmează:

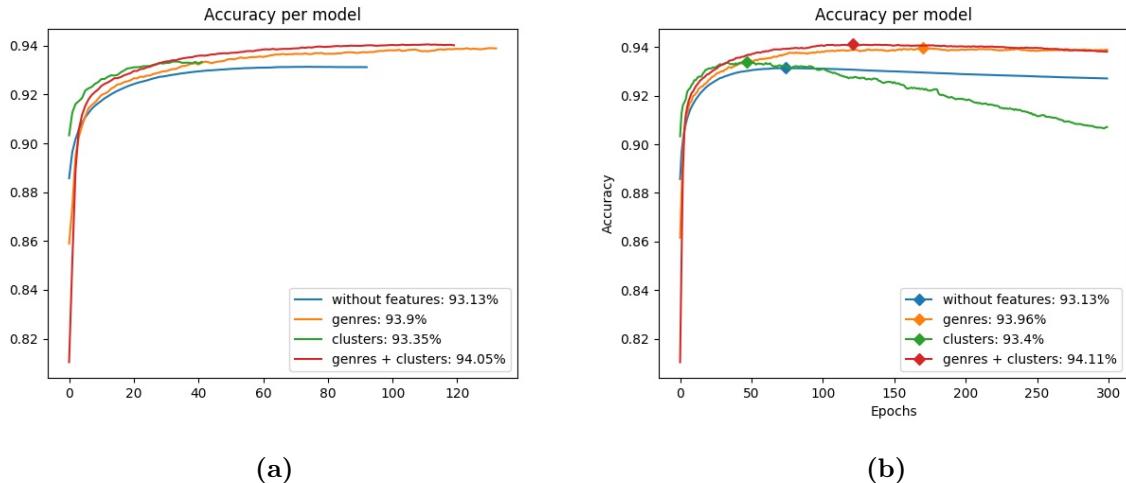


Figura 4.13: Comparație acuratețea modelului cu parametrii optimi pe tipuri de metadate (a) și cu parametrii optimi pe tipuri de metadate cu număr setat de epoci (b).

- per tot setul de date, numărul de interacțiuni pe utilizator: minim: 1; mediu: 101; maxim: 1459; $\langle k=10 \rangle$: 20/609
- per setul de date de train, numărul de interacțiuni pe utilizator: minim: 1; mediu: 81; maxim: 1165; $\langle k=10 \rangle$: 38/609
- per setul de date de test, numărul de interacțiuni pe utilizator: minim: 1; mediu: 20; maxim: 294; $\langle k=10 \rangle$: 296/598

Rezultatele metricii de acuratețe sunt prezentate în figura 4.10 (a). Aceste rezultate sun obținute utilizând parametrii optimi din tabelul 3.1. Reamintim că toate clasterele folosite pentru obținerea parametrilor optimi din acest tabel au fost create folosind rețeaua VGG19. Principalele concluzii pe care e putem trage pe baza acestui grafic sunt următoarele:

- modelul fără metadate are un start mai bun decât decât modelul cu genuri sau modelul cu genuri și cu clastere, însă acesta nu urcă prea mult pe durata perioadei de antrenare, astfel ajungând să aibă cel mai slab rezultat dintre toate cele patru variante de model testate;
- modelul cu metadatele de genuri are un start destul de slab, fiind mai bun doar de modelul în care se folosesc atât genurile cât și clasterile, însă, până la final, acest model obține un rezultat bun și apropiat de rezultatul modelului cu genuri și

clastere. O altă mențiune importantă ar fi că acest model are cea mai mare durată de antrenare față de toate celelalte modele;

- modelul cu metadatele de clastere are cel mai bun start și în același timp cea mai scurtă perioadă de antrenare pentru a obține cel mai optim rezultat dintre toate modelele testate. Acuratețea obținută de acest model este peste acurateșea obținută de modelul fără metadate dar sub cea obținută de modelul cu genuri însă destul de apropiată de aceasta;
- cel mai bun rezultat din acest experiment este obținut de modelul în care se folosesc atât metadatele de genuri cât și cele de clastere. Modelul are cel mai slab start însă cel mai bun rezultat final. Observăm că modul în care este obținut rezultatul este o combinație a avantajelor modelelor cu metadate de genuri și clastere. Pe de-o parte, acuratețea crește semnificativ față de modelul fără metadate și este apropiată de acuratețea modelului cu genuri. Pe de alta parte timpul de necesar obținerii aceluia rezultat optim este mai scăzut decât la modelul cu genuri, timp influențat de metadatele de clastere.

Concluzia principală a acestui experiment este că un model ce conține genuri și clastere are o performanță de până la 1% în plus față de modelul fără metadate și mai mare față de oricare alt model, iar timpul necesar pentru obținerea acestei performanțe este mai mic în comparație cu cel mai apropiat model din punct de vedere al metricii evaluate.

Pentru a avea o imagine mai completă de cum se pot comporta aceste modele în timp am eliminat parametrul optimi găsit pe numărul de epoci și l-am setat pentru fiecare model la un număr de 300 de epoci. Iar după cum se observă în figura 4.13 (b) toate modelele după punctul optim în care ating performanța maximă încep să scadă. Modelele fără metadate, cu metadatele de genuri și cu metadatele de clastere au o scădere destul de lină în timp, pe când scăderea este mai acelărată în cazul modelului ce folosește metadatele de clastere față de celelalte modele. Acest al doilea grafic conturează și mai bine o concluzie pe care o putem trage despre modelul cu clastere și anume: acesta atinge performanța maximă foarte repede din punct de vedere al timpului de execuție în comparație cu oricare al model testat în cadrul acestei lucrări, iar din acel moment în care a fost obținută performanța maximă nu mai este recomandată prelungirea procesului de antrenare scăderea care va urma după punctul maxim va fi foarte acelărată.

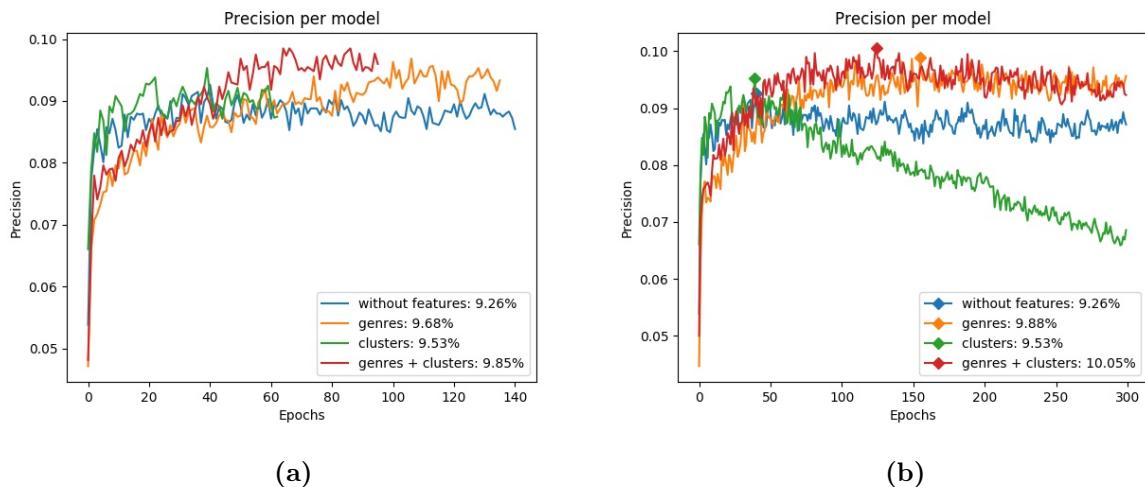


Figura 4.14: Comparație precizie@k a modelului cu parametrii optimi pe tipuri de metadate (a) și cu parametrii optimi pe tipuri de metadate cu număr setat de epoci (b), unde $k = 10$.

Rezultatele modelelor pe metrica precizie@k cu $k = 10$ sunt prezentate în figura 4.14 (a). Un lucru care se poate remarcă este faptul că, la fel ca în cazul metricii de acuratețe, se păstrează atât ordinea modelelor în rezultatele de precizie cât și în ordinea modelelor în timp de antrenare, în sensul celui mai bun rezultat, precizie mare, timp de antrenare mic. Aceeași corespondență între specificațiile rezultatelor obținute de metrii se regăsește și în situația în care mărim perioada de antrenare pentru a vedea o imagine de ansamblu după cum se poate observa în figura 4.14 (b).

În continuare vom prezenta rezultatele metricilor obținute prin creearea clasterelor cu rețelele preantrenate VGG19, InceptionV3 și ResNet50, atât clasterele singure cât și în combinație cu metadatele de genuri. După cum se poate observa în figura 4.15 rețeaua VGG19 își menține timpul de antrenare scăzut plus celelalte specificații descrise în experimentele anterioare. Rețeaua ResNet50 are cel mai bun rezultat al acurateții, iar ca timp de antrenare este situată între VGG19 și InceptionV3. În cele din urmă, rețeaua InceptionV3 are un rezultat mai bun decât VGG19 însă timpul de antrenare necesar este cel mai mare față de celelalte rețele.

Dacă sunt introduse și metadatele de genuri în sistemul de recomandare (vezi figura 4.16) ordinea rețelelor după performanța lor se schimbă astfel: rețeaua InceptionV3 devine cea mai performantă, însă își menține timpul de antrenare ridicat; rețeaua VGG19 devine a doua ca performanță însă are o creștere semnificativă a timpului de antrenare necesar;

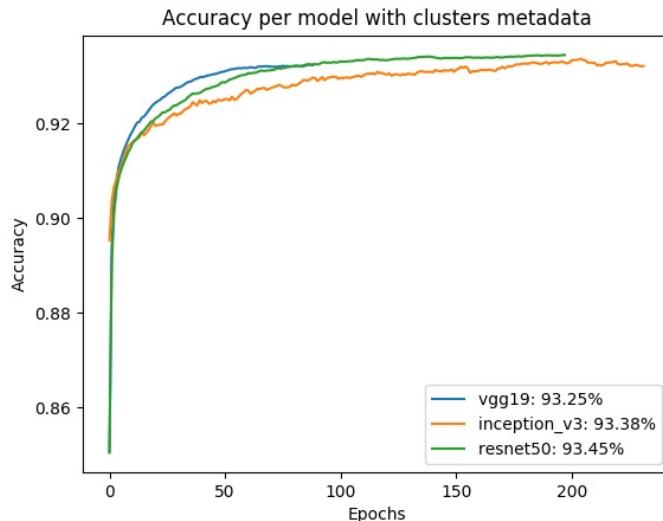


Figura 4.15: Comparație acuratețea modelului cu parametrii optimi pe tipuri de rețele preantrenate.

rețeaua ResNet50 este depășită de celelalte rețele ca performanță a metricii însă timpul de antrenare rămâne în aceași zonă.

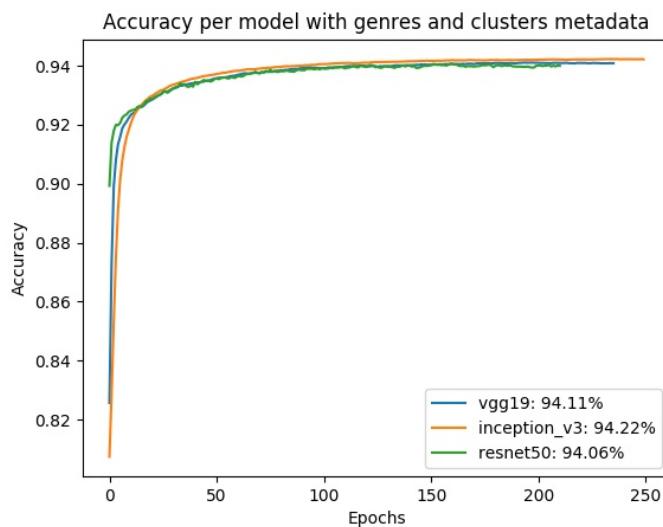


Figura 4.16: Comparație acuratețea modelului cu parametrii optimi pe tipuri de rețele preantrenate și cu metadatele de genuri.

În ceea ce privește metrica de precizie doar pe clastere (vezi figura 4.17) putem menționa următoarele: rețeaua VGG19 necesită cel mai mare timp de antrenare însă are cel mai bun rezultat al metricii de evaluare; rețeaua ResNet50 este situată între VGG19 și InceptionV3 ca performanță a preciziei și cea mai bună ca timp de antrenare; rețeaua

Inception V3 are cea mai slabă performanță pe precizie dintre cele trei rețele dar ca timp de antrenare este situată între VGG19 și ResNet50.

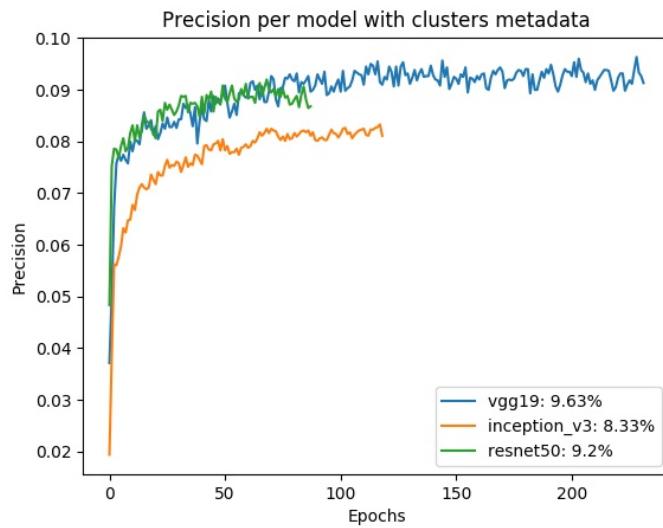


Figura 4.17: Comparație precizia@k a modelului cu parametrii optimi pe tipuri de rețele preantrenate.

Odată introduse și metadatele de genuri (vezi figura 4.18), toate rețelele tind să aibă aceeași perioadă de antrenare; rețeaua InceptionV3 are cea mai slabă performanță; rețeaua ResNet50 devine cea mai performantă, VGG19 trecând pe locul secund ca performanță.

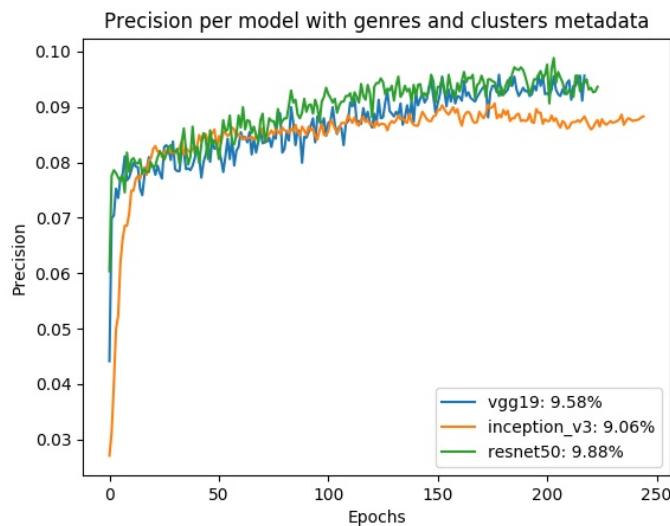


Figura 4.18: Comparație precizia@k a modelului cu parametrii optimi pe tipuri de rețele preantrenate și cu metadatele de genuri.

Concluzii

Concluziile disertației ...

Bibliografie

[1] Data never sleeps 6.0

<https://www.domo.com/learn/data-never-sleeps-6>

[2] Netflix International: What movies and TV shows can I watch, and where can I watch them?

<https://www.finder.com/global-netflix-library-totals>

[3] How Many Products Does Amazon Sell? – April 2019

<https://www.scrapehero.com/number-of-products-on-amazon-april-2019/>

[4] Erion Çano and Maurizio Morisio. *Hybrid Recommender Systems: A Systematic Literature Review*. CoRR abs/1901.03888, 2019

[5] An Overview of Recommendation Systems

<http://datameetsmedia.com/an-overview-of-recommendation-systems/>

[6] LightFM 1.15 - documentation

<http://lyst.github.io/lightfm/docs/lightfm.html>

[7] CS231n: Convolutional Neural Networks for Visual Recognition

<http://cs231n.stanford.edu/2018/syllabus.html>

[8] Jason Weston, Samy Bengio and Nicolas Usunier. *Wsabie: Scaling up to large vocabulary image annotation*. IJCAI. Vol. 11, p2764-2770, 2011.

[9] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner and Lars Schmidt-Thieme. *BPR: Bayesian personalized ranking from implicit feedback*. CoRR abs/1205.2618, 2012.

[10] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. CoRR abs/1409.1556, 2014.

- [11] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. *Deep Residual Learning for Image Recognition*. CoRR abs/1512.03385, 2015.
- [12] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens and Zbigniew Wojna. *Rethinking the Inception Architecture for Computer Vision*. CoRR abs/1512.00567, 2015.
- [13] Barret Zoph, Vijay Vasudevan, Jonathon Shlens and Quoc V. Le. *Learning Transferable Architectures for Scalable Image Recognition*. CoRR abs/1707.07012, 2017.
- [14] Advanced Guide to Inception v3 on Cloud TPU
<https://cloud.google.com/tpu/docs/inception-v3-advanced>
- [15] Periklis Andritsos. *Data Clustering Techniques*. University of Toronto, Dep. of Computer Science, 2002.
- [16] Gongde Guo, Hui Wang, David Bell, Yixin Bi, Kieran Greer. *KNN Model-Based Approach in Classification*. Lecture Notes in Computer Science, vol 2888. Springer, Berlin, Heidelberg, 2003.
- [17] Sklearn metrics - silhouette score
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html
- [18] Maciej Kula *Metadata Embeddings for User and Item Cold-start Recommendations*. CoRR abs/1507.08439, 2015.
- [19] Jason Weston, Hector Yee, Ron J. Weiss. *Learning to Rank Recommendations with the k-Order Statistic Loss*. Proceedings of the 7th ACM conference on Recommender systems. ACM, 2013
- [20] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. CoRR abs/1609.04747, 2016
- [21] Welcome to LightFM's documentation!
<http://lyst.github.io/lightfm/docs/home.html>

- [22] F. Maxwell Harper and Joseph A. Konstan. *The MovieLens Datasets: History and Context*. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19.
- [23] skopt API documentation
<https://scikit-optimize.github.io/>
- [24] API Overview - The Movie Database (TMDb)
<https://www.themoviedb.org/documentation/api>