

UNIVERSITATEA POLITEHNICĂ DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL DE CALCULATOARE



PROIECT DE DIPLOMĂ

Descrierea automată a imaginilor în limba Română prin
intermediul Învățării Automate

Andrei-Vlad Man

Coordonator științific:

S.L. dr. ing. Dumitru-Clementin Cercel

BUCUREȘTI

2024

CUPRINS

1	Introducere	1
1.1	Context	1
1.2	Problema	1
1.3	Obiective	1
1.4	Soluția propusă	2
1.5	Rezultatele obținute	2
1.6	Structura lucrării	2
2	Lucrări Asemănătoare și State-of-the-art	3
3	Tehnologii folosite	6
3.1	Rețelele Neurale Convoluționale	6
3.1.1	ResNet50	7
3.1.2	ResNet50V2	9
3.1.3	InceptionV3	11
3.1.4	InceptionResNetV2	13
3.1.5	EfficientNet	14
3.2	Rețelele Neuronale Recurente	15
3.2.1	LSTM	15
3.2.2	GRU	15
3.3	Mecanisme de atenție	17
3.3.1	Atenție de tip Bahdanau	17
3.3.2	Atenție de tip Luong	18
3.4	Transformers	19
3.4.1	Vision Transformer - ViT	21

3.5	Învățare prin transfer de cunoștințe	21
3.6	Tokens și Embeddings	22
4	Experimentele propuse	24
4.1	Setul de date Flickr30k	24
4.1.1	Traducere și corectare	24
4.1.2	Vocabular și variante	24
4.2	Arhitectura modelelor propuse	26
4.2.1	CNN-Encoder RNN-Decoder	26
4.2.2	Vision Transformer-Encoder RNN-Decoder	28
4.2.3	CNN-Transformer-Encoder Transformer-Decoder	28
4.3	Tehnici pentru generarea descrierilor	29
4.3.1	Greedy Search	29
4.3.2	Beam Search	30
4.4	Evaluarea modelelor	31
4.4.1	Bleu	31
4.4.2	Rouge	32
4.4.3	Meteor	33
4.4.4	Ter	34
4.4.5	Cider	35
5	Rezultate	37
5.1	Rezultate cantitative	37
5.1.1	Rezultate obținute utilizând arhitectura CNN-Encoder RNN-Decoder	37
5.1.2	Rezultate obținute utilizând arhitectura ViT-Encoder RNN-Decoder .	40
5.1.3	Rezultate obținute utilizând Embeddings preantrenate	41
5.1.4	Rezultate obținute utilizând arhitectura Transformer-Encoder Transformer-Decoder	42
5.1.5	Rezultate obținute utilizând tehnica Beam Search pentru generarea descrierilor	43

5.2	Rezultate calitative	44
6	Concluzii și Direcții Viitoare	47

SINOPSIS

În această lucrare propun un set de experimente în cadrul domeniului de Descriere Automată a imaginilor prin intermediul Învățării Automate, pornind de la un set de date realizat în colaborare cu doi studenți, "Flickr30k" tradus automat și corectat manual. Arhitecturile folosite includ Rețele Convoluționale, Rețele Recurente, Transformeri și Transformatori Vizuali. Rezultatele obținute nu sunt comparabile cu state-of-the-art însă unul dintre modelele propuse generează descrieri decente pentru început, care sunt reprezentative pentru imagini.

ABSTRACT

In this work I propose a set of experiments on the Image Captioning for Romanian Language task, starting with an automatically translated and humanly revised "Flickr30k" dataset, on which I worked among two other students. The architectures used for these experiments include Convolutional Networks, Recurrent Networks and both Visual and Text Transformers. The results are not comparable to other state-of-the-art works, but one of the models proposed is generating pretty decent and accurate captions.

MULȚUMIRI

Vreau să aduc mulțumiri studenților Alexandru-Gabriel Rait și George-Andrei Dima care m-au ajutat cu corectarea setului de date. De asemenea, vreau să aduc mulțumiri și coordonatorului S.L. dr. ing. Dumitru-Clementin Cercel pentru implicarea activă a acestuia, pentru punerea în contact cu colegii menționați anterior și pentru îndrumările cu privire la implementare și experimente.

1 INTRODUCERE

1.1 Context

"Descrierea automată a imaginilor" (în engleză: "Image Captioning"), este procesul de a prezice o descriere a unei imagini date. Acest obiectiv se află la intersecția a două domenii ale Inteligenței Artificiale, acelea fiind "Viziunea Calculatorului" (în engleză: "Computer Vision") și "Procesarea Limbajului Natural" (în engleză: "Natural Language Processing"). Aceasta este o problemă complexă comparativ cu clasificarea sau recunoașterea de imagini, descrierea generată trebuie să identifice multiple obiecte și modul în care acestea interacționează, precum și caracteristicile acestora; în același timp folosind un limbaj natural pentru a transpune coerent informația.

1.2 Problema

O utilitate principală a acestei probleme este de a ajuta indivizii cu deficiențe vizuale să poată interpreta conținutul web, și nu numai, în format vizual. Mai departe, problema poate fi extinsă și pentru a beneficia sistemele în timp real, care capturează informația în format video.

1.3 Obiective

Majoritatea cercetării se realizează pe seturi de date în limba engleză, deci descrierile acestor algoritmi vor fi în limba engleză. Studiul pe limba Română al acestui obiectiv este restrâns spre inexistent, însă acesta este foarte important pentru a obține atât informația rapid cât și a putea fi integrată ulterior în comunitatea locală.

Această lucrare are ca scop explorarea a mai multor tipuri de modele și arhitecturi, experimentele fiind realizate pe un set foarte utilizat de date care a fost tradus în limba română. Rezultatele "state-of-the-art" (din engleză, rezultate cu cea mai mare performanță) nu sunt un scop al lucrării, obiectivul principal fiind determinarea celor mai bune arhitecturi pe setul de date propus. Această lucrare are potențialul să devină o rampă de lansare pentru mai multe experimente viitoare, ținându-se cont de ce modele au performat mai bine.

1.4 Soluția propusă

Pentru aceste experimente, vom folosi tehnologii mai simple, ușor de implementat și antrenat față de cele utilizate în modelele state-of-the-art, respectiv arhitecturi tip Encoder-Decoder formate din Rețele Convolaționale și Rețele Recurente, Transformeri sau combinații hibride. De asemenea, este propus și utilizat un set de date tradus automat și ulterior corectat manual de mine alături de alți doi studenți. Modelele state-of-the-art necesită atât cunoștințe avansate în domeniu cât și o cantitate mare de date și resurse (hardware sau financiare), motiv pentru care vom încerca doar abordări realiste pentru început.

1.5 Rezultatele obținute

Rezultatele obținute sunt, conform așteptărilor, sub cele state-of-the-art pentru task-ul în limba Engleză. Pentru limba Română, nu există în schimb un termen de comparație. Totuși, unul dintre modele oferă performanțe bune, fiind capabil să genereze descrieri coerente și care au legătură cu imaginea primită.

1.6 Structura lucrării

Lucrarea este structurată pe 6 capitole.

În capitolul 1, respectiv cel de față, este prezentată problema, motivația și câteva detalii referitoare la obiective, implementare și rezultate.

În capitolul 2 se va discuta puțin despre lucrările state-of-the-art și alte lucrări relevante în realizarea experimentelor propuse.

În capitolul 3 vor fi detaliate tehnologiile folosite, respectiv CNN-uri, RNN-uri, Mecanisme de Atenție și Transformatori și cum pot fi utilizate.

În capitolul 4 va fi prezentat setul de date propus, alături de arhitecturile rezultate, tehnicile de generare ale descrierilor și metrici pentru evaluare.

Capitolul 5 conține rezultatele experimentelor și respectiv comentarea acestora.

În capitolul 6 este formulată o concluzie asupra studiului și planurile pe viitor referitoare la acest subiect.

2 LUCRĂRI ASEMĂNĂTOARE ȘI STATE-OF-THE-ART

Putem să considerăm că limba Română este un "low-resource language", adică o limbă care are un conținut online mult mai redus în comparație cu altele. Aceasta este o statistică observată și de către W3Tech.com [49], prin intermediul unor chestionare tip "survey". Studiul menționat plasează limba Română la mijlocul clasamentului, cu un procent de 0.5% din total. Clasamentul este dominat de limba Engleză, cu 52.2%, Spaniolă 5.5%, Germană 4.7% și limba Japoneză cu 4.3%. Pentru scopul acestei lucrări nu a existat până acum un set de date care să conțină imagini și descrieri în limba Română.

Astfel, pentru realizarea unor experimente incipiente, am fost inspirat din lucrări ce abordează același obiectiv în alte limbi "low-resource" cum ar fi limba Nepaleză [38], limba Hindi [29, 30, 31], limba Vietnameză [33] și limba Asameză [9]. Aceste lucrări folosesc atât arhitecturi cu Rețele Neurale Convolaționale alături de Rețele Neurale Recurente, cât și arhitecturi ce se apropie de metodologiile state-of-the-art cu Transformeri. Seturile de date propuse de aceștia sunt traduse automat, iar apoi corectate manual.

Aceste lucrări vor fi utilizate și pentru implementarea experimentelor, respectiv arhitectura CNN-Transformer [38], și arhitecturile CNN-RNN [29, 30, 31, 33, 29] adoptând tehnologii precum RNN bidirecțional și mecanisme de atenție, sau transformeri vizuali.

Cele mai multe studii, respectiv lucrările tip state-of-the-art, se bazează pe setul de date MSCOCO[25]. Acesta este o colecție de dimensiuni foarte mari de imagini și descrieri (164K în momentul publicării însă numărul acestora a crescut prin seturi în plus de testare/validare), publicat de Microsoft și constă în poze cu obiecte comune într-un anumit context. Deși propus pentru detecția obiectelor, acest set de date poate fi utilizat pentru o multitudine de task-uri din domeniul Computer Vision și Natural Language Processing.

Modelele cele mai performante folosesc chiar și colecții mai mari de date pentru a fi preantrenate, iar ulterior cunoștințele lor se transferă pe seturile de date de tip benchmark. Acest proces este unul foarte costisitor în schimb, necesitând foarte multe resurse hardware, respectiv financiare. O să observăm mai departe cum modelele state-of-the-art sunt deseori finanțate de corporații și companii private care beneficiază de bugete extreme.

Modelele state-of-the-art folosesc tehnologii de ultimă oră, cum ar fi transformeri generativi, transformeri multimodali, sau Modele lingvistice de dimensiune mare (LLM sau Large Language Model).

Modelul mPLUG[22] este în prezent cel mai performant din cadrul acestui task. Acesta este un model încrucișat ("cross-modal" din engleză), o arhitectură tip transformer cu intrare multiplă, de diferite tipuri (text și vizual) și conexiuni tip scurtătură. Este intitulat ca fiind un

model "foundation", în sensul în care este preantrenat pe o cantitate foarte mare de date și informații, cu cunoștințe ce pot fi transferate ulterior către multe alte task-uri. Acest model este propus de câțiva cercetători din compania "Alibaba Group", cel mai mare site de comerț din regiunea Chinei. Prin intermediul acesteia, reușesc să antreneze modelul pe un număr de 14 milioane de perechi de imagini și descrieri.

Similar este și modelul OFA[51] care introduce o arhitectură simplificată și unificată pentru modele încrucișate. Spre deosebire de concurenții săi, este antrenat pe mai puține date și într-un mod mai eficient, atingând performanțe similare. La rândul său, acest studiu este sprijinit de grupul "Alibaba", iar modelul este antrenat pe 20 de milioane de imagini.

Modelul GIT[50] folosește un encoder pentru text și unul pentru imagini, iar peste acestea sunt folosiți transformeri generativi și modele lingvistice. Acest studiu este realizat de câțiva cercetători de la "Microsoft", modelul fiind preantrenat pe aproximativ 800 de milioane de perechi imagine-descriere.

Modelul BLIP-2[23] folosește encodori vizuali și LLM preantrenate, fără a le mai antrena ulterior, iar conexiunea acestora este realizată de un transformer antrenat în două etape. Prima oară învață cu ajutorul encoder-ului vizual, iar a doua doar cu ajutorul componentei generative tip LLM. Astfel, are mult mai puțini parametri de antrenat, fapt ce rezultă într-o complexitate scăzută și un timp de antrenare redus, chiar și pe seturi de date largi. Acest model este realizat cu ajutorul companiei "Salesforce", fiind utilizată o colecție de 129 de milioane de imagini.

Arhitectura ExpansionNetv2[17] introduce un mecanism de "Expansiune" care procesează intrările fără a ține cont de lungimea secvenței, fiind capabil să învețe mai eficient în comparație cu metodologiile clasice ce folosesc mecanisme de atenție. Acesta este antrenat doar pe setul de date MSCOCO. În esență, acesta folosește arhitectura Transformer împreună cu blocuri de expansiune.

Modelul LEMON[18] este rezultatul unui studiu privind scalarea modelelor preantrenate atât lingvistic cât și vizual, fiind experimentate dimensiuni între 13 și 675 de milioane de parametri. Experimentul este realizat pe o colecție de 200 de milioane de perechi de imagini-descrieri și este de asemenea finanțat de către "Microsoft". Acesta folosește un extractor de caracteristici ale imaginilor și transformerul care poate fi scalat. Rezultatele prezentate arată că această scalare are un impact pozitiv asupra performanței, inclusiv în cazul transferului de cunoștințe.

Modelul GRIT[34] este o rețea în totalitate de tip transformer, cu mențiunea că folosește concepte de grilă și regiune în antrenarea și prezicerea descrierilor. Astfel, sunt folosiți doi transformeri vizuali în paralel pentru extragerea caracteristicilor. Cel de tip grilă are rolul să învețe relațiile și contextul dintre obiecte, iar responsabil de regiune se va ocupa de recunoașterea obiectelor. Ulterior, informațiile sunt combinate și trecute prin decodor, generând o descriere calitativă din punct de vedere al obiectelor, numărul acestora și contextul lor.

Prompt Tuning[53] este un studiu care urmărește transferarea de cunoștințe ale modelelor

Tabela 1:
Modele state-of-the-art și scorurile acestora pe diverse metrice folosite drept benchmark

Model	Bleu-4	Cider	Meteor
mPLUG	46.5%	155.1%	32.0%
OFA	44.9%	154.9%	32.5%
GIT	44.1%	151.1%	32.2%
BLIP-2 ViT-G	43.7%	145.8%	-
ExpansionNet v2	42.7%	143.7%	30.6%
LEMON	42.6%	145.5%	31.4%
GRIT	42.4%	144.2%	30.6%
Prompt Tuning	41.81%	141.4%	31.51%

multimodale generative prin intermediul "prompting-ului" (interogarea unui model să realizeze un anumit task). Spre deosebire de finetuning sau învățarea prin transfer, această modalitate necesită reantrenarea a semnificativ mai puțini parametri, de exemplu doar 1%, și duce la rezultate similare.

3 TEHNOLOGII FOLOSITE

3.1 Rețelele Neurale Convoluționale

Rețelele Neurale Convoluționale sau CNN, așa cum sunt cunoscute în ziua de azi, au fost propuse de către LeCun et al.[21] în anul 1989, fiind folosite pentru a clasifica cifre scrise de mână, utilizând un set de date al Serviciului Poștal al Statelor Unite ale Americii. Această abordare s-a dovedit a fi una foarte bună din punct de vedere al performanței, cu o eroare de clasificare de doar 0.14% pe setul de antrenare și doar 5.0% pe setul de testare. Acesta observă și că rețeaua converge foarte rapid, și arată că "backpropagation" poate fi folosit și pentru procese dificile, care necesită mult timp.

O rețea convoluțională are câțiva parametri: număr de filtre C , dimensiunea filtrului, de formă pătrată, $K \times K$, stride sau pasul de deplasare, și padding-ul filtrului P . Să zicem că rețeaua primește ca intrare o imagine de forma (W, H, D) . Ieșirea convoluției va avea următoarea formă, respectând propriii parametri:

$$W' = \frac{W - K + 2 * P}{S} + 1 \quad (1)$$

$$H' = \frac{H - K + 2 * P}{S} + 1 \quad (2)$$

$$D' = C \quad (3)$$

iar stratul va avea de învățat $(K \times K \times D) \times C + C$ parametrii. Pentru straturile de tip "pooling" /unire avem ca parametrii dimensiunea K și stride S , și următoarele formule pentru forma ieșirii, cu mențiunea că nu are parametrii de învățat:

$$W_p = \frac{W - K}{S} + 1 \quad (4)$$

$$H_p = \frac{H - K}{S} + 1 \quad (5)$$

$$D_p = D \quad (6)$$

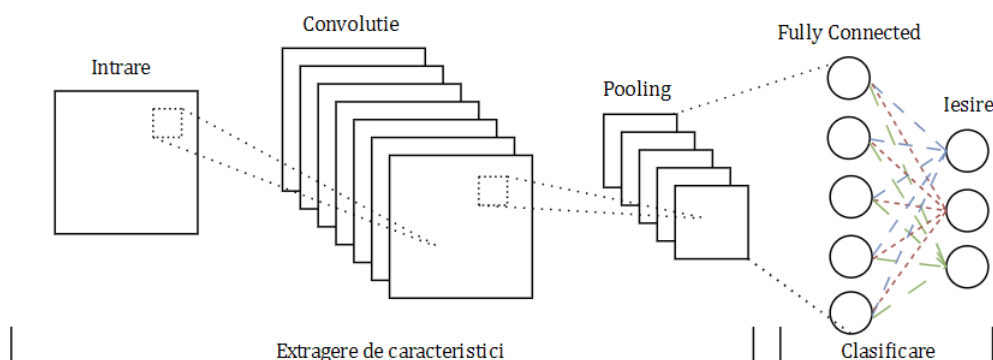


Figura 1: Re'ea simplă CNN.

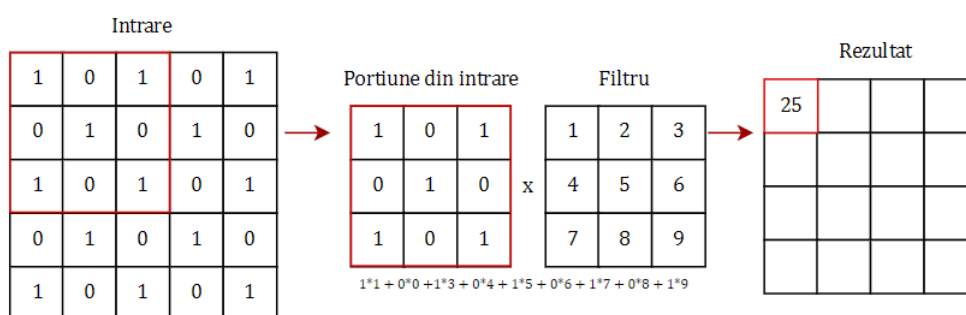


Figura 2: Metoda de calcul pentru o re'ea CNN.

Odată cu avansul tehnologic al plăcilor grafice și posibilitatea de accelerare a calculului folosind tehnologii precum programarea "CUDA", au devenit fezabile și Rețelele Convoluționale adânci, care pot fi antrenate pe mai multe date și să recunoască mai multe clase de obiecte. Unul dintre modelele care urmează să impună standarde în acest domeniu este AlexNet, dezvoltat de Alex Krizhevsky et al.[19], care are 5 straturi de convoluții și 3 straturi complet conectate ("fully connected" în engleză). Modelul depășește cu mult modelele state-of-the-art din acel moment, devenind ulterior și o sursă de inspirație din punct de vedere al Rețelelor Convoluționale Adânci. Deși task-ul de "Image Captioning" este diferit de clasificarea imaginilor, putem folosi CNN preantrenate pentru a extrage caracteristicile unor imagini, folosind doar partea de convoluții și pooling a acestora (fig. 1).

Ulterior vor fi prezentate în rezumat Rețelele Convoluționale folosite în aceste experimente.

3.1.1 ResNet50

Rețelele Neurale Reziduale Adânci ("Deep Residual Networks" sau "ResNet" în engleză) au fost introduse de către o echipă de cercetători de la Microsoft (Kaiming He et al.) în anul 2015 [14]. Creșterea numărului de straturi reprezenta o problemă datorită impedimentului notoriu: dispariția sau explozia gradientului ("Vanishing/ Exploding Gradient Problem"

în engleză), ceea ce împiedică convergența rețelelor. Rezolvarea acestei probleme a fost prin inițializare normalizată și straturi intermediare de normalizare, fapt care face posibilă convergența rețelelor cu zeci de straturi prin coborârea gradientului stocastic ("Stochastic Gradient Descent" în engleză). Însă, odată cu convergența acestor rețele adânci, apare o altă problemă: acuratețea devine saturată și se degradează încet, iar straturile adiționale duc la creșterea erorii [14].

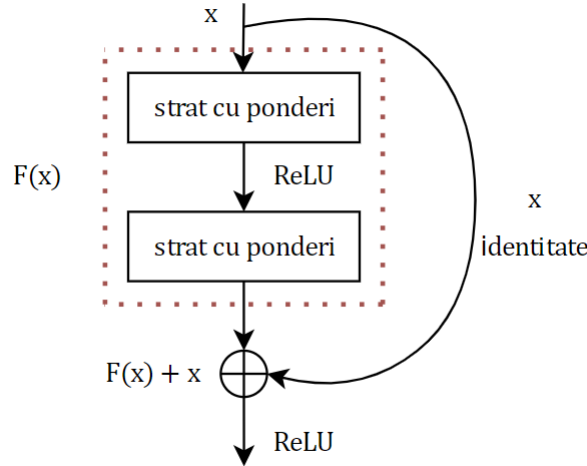


Figura 3: Bloc de construcție al unei rețele reziduale. [14]

Echipa propune astfel o metodă pentru "Învățare prin Rețele Reziduale Adânci". Pentru un strat al unei rețele neuronale, avem maparea unei intrări-ieșiri ca fiind $H(x)$. Straturile neliniare vor avea ca scop să mapeze:

$$F(x) = H(x) - x \quad (7)$$

Maparea originală se obține prin:

$$F(x) + x = H(x) \quad (8)$$

Ipoteza cercetătorilor este că optimizarea mapării reziduale este mai simplă decât optimizarea mapării originale, aceasta fiind nedefinită. Ca exemplu, dacă vrem să mapăm o funcție identică, este mai ușor să optimizăm rezidualul spre 0 decât să îl aproximăm la valoarea sa doar prin straturile neliniare. [14]

Vom considera un bloc al rețelei fiind definit prin următoarea formulă, unde x și y reprezintă intrarea și ieșirea blocului iar funcția $F(x, W_i)$ este rezidualul ce trebuie învățat:

$$y = F(x, W_i) + x \quad (9)$$

Pentru fig. 3, cu două straturi, vom avea următoarea formulă, unde σ este ReLU:

$$F = W_2 * \sigma * (W_1 * x) \quad (10)$$

Astfel, prin conexiunea tip scurtătură ("shortcut" în engleză), nu sunt introduși nici parametrii în plus și nici complexitate computațională. [14]

Blocurile reziduale pot fi construite prin conexiuni scurtătură (Fig. 3), care au maparea identitate și sunt adunate la ieșirile straturilor neliniare. Astfel, rețeaua poate fi în continuare antrenată prin SGD și backpropagation. [14]. Rezultatele lor arată că aceste rețele reziduale sunt foarte ușor de optimizat iar adăugarea straturilor duce la creșterea acurateței.

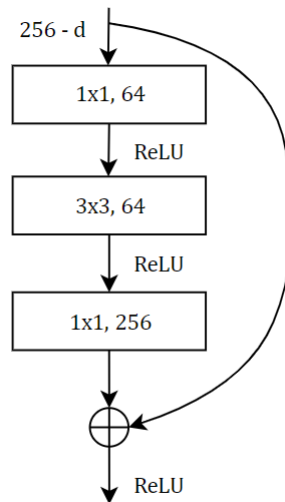


Figura 4: Bloc bottleneck de construcție al rețelei ResNet50. [14]

Modelul ResNet50, alături și de cele mai adânci, folosesc o arhitectură bottleneck, având ca scop să reducă timpul de antrenare. Astfel, fiecare bloc rezidual va folosi 3 straturi în loc de 2 (Fig. 4), fiind convoluții de 1x1, 3x3, 1x1, unde cele de 1x1 au rolul de a micșora, iar apoi restaurează dimensiunea, iar stratul de 3x3 este bottleneck-ul deoarece are dimensiunile de intrare și ieșire mai mici. Astfel, beneficiază la rândul lor de performanță sporită, dar nu sunt la fel de economice [14]. Rezultatul cercetării a fost unul pozitiv, modelele ResNet având o eroare de doar 3.57% pe setul de date ImageNet, fapt care le-a adus premiul I în concursul de clasificare ILSVRC 2015, implementând ulterior și modele care au câștigat locurile 1 în task-uri de detectare, localizare și segmentare în competiția ILSVRC & COCO 2015. [14]

3.1.2 ResNet50V2

Aceeași echipă revine un an mai târziu, în 2016, cu noi descoperiri. Calculul unui strat al rețelelor ResNet (numit acum Unitate Reziduală) poate fi rescris astfel [15]:

$$y_l = h(x_l) + F(x_l, W_l) \quad (11)$$

$$x_{l+1} = f(y_l) \quad (12)$$

H este considerată mapare identică ($h(x) = x$). Dacă luăm și f ca fiind mapare identică, ajungem la următoarea ecuație:

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i, W_i) \quad (13)$$

Această ecuație conduce și la o proprietate utilă a backpropagation-ului:

$$\frac{\partial \epsilon}{\partial x_l} = \frac{\partial \epsilon}{\partial x_L} * \frac{\partial x_l}{\partial x_L} = \frac{\partial \epsilon}{\partial x_L} * \left(1 + \frac{\partial}{\partial x_l} * \sum_{i=l}^{L-1} F(x_i, W_i)\right) \quad (14)$$

Gradientul $\frac{\partial \epsilon}{\partial x_l}$ poate fi decompus în doi termeni: $\frac{\partial \epsilon}{\partial x_L}$, care propagă gradientul în mod direct fără a afecta straturile rețelei, și un termen aditiv care se propagă prin straturi. Termenul aditiv asigură că gradientul este improbabil să fie mereu 0, deoarece în general acest termen nu poate fi egal cu -1 într-un mini-exemplu. (cf. He et al., 2016)

Arhitectura nouă propusă și testată de Kaiming He și colaboratorii săi constă în alterarea conexiunilor tip scurtătură. Un exemplu ar fi introducerea parametrului δ , care reprezintă o scalare:

$$y_l = \delta_l * h(x_l) + F(x_l, W_l) \quad (15)$$

Ecuațiile propuse anterior se transformă astfel, unde notația F absoarbe acum scalarii în funcțiile reziduale [15]:

$$x_L = \left(\prod_{i=l}^{L-1} \delta_i\right) * x_l + \sum_{i=l}^{L-1} F(x_i, W_i) \quad (16)$$

$$\frac{\partial \epsilon}{\partial x_l} = \frac{\partial \epsilon}{\partial x_L} * \left(\left(\prod_{i=l}^{L-1} \delta_i\right) + \frac{\partial}{\partial x_l} * \sum_{i=l}^{L-1} F(x_i, W_i)\right) \quad (17)$$

În acest exemplu, am folosit pentru simplificare o scalare. În practică, au fost folosite diferite transformări complexe.

Rezultatul acestor experimente nu a fost unul pozitiv. Aceste manipulări multiplicative ale scurtăturilor pot afecta în mod negativ performanțele modelelor.

Experimentul a dus mai departe la alterarea funcției de activare. Activarea (ecuația 18) va influența ambele ramuri ale următoarei unități reziduale (ecuația 19). Astfel, se propune o activare g asimetrică care va influența doar termenul F al ecuației, ajungând la soluția din ecuația 20:

$$x_{l+1} = f(y_l) \quad (18)$$

$$y_{l+1} = f(y_l) + F(f(y_l), W_{l+1}) \quad (19)$$

$$x_{l+1} = x_l + F(g(y_l), W_l) \quad (20)$$

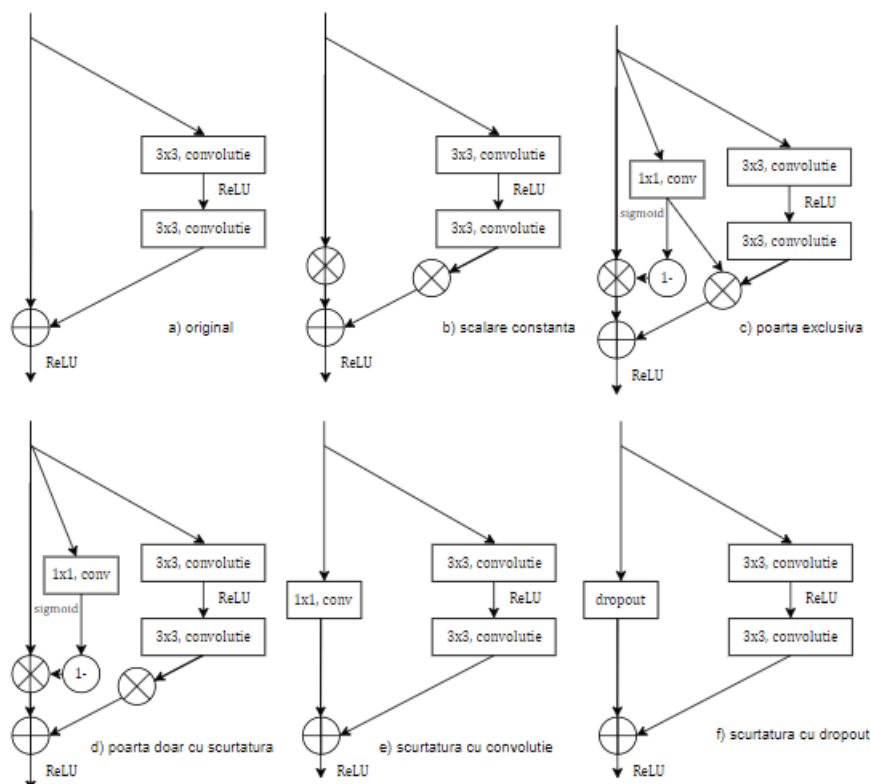


Figura 5: Blocuri de construcție utilizate în experimente. [15]

Această îmbunătățire reduce overfit-ul și aduce familia de modele ResNetV2 în clasament pe primele locuri. Deși inițial au fost propuse doar modele cu peste 100 de straturi, această arhitectură a fost adoptată și de ResNet50V2.

3.1.3 InceptionV3

Proiectul Inception, pornit de cercetătorii de la Google Szegedy et al. începe în 2014 [40], având ca scop dezvoltarea unor modele de categorie ușoară, care să fie utilizabile în lumea reală, nu doar pe super calculatoare. Rezultatul lor, GoogLeNet, are de 12x mai puțini parametri decât arhitectura câștigătoare din anii precedenți, beneficiind în același timp și de mai multă acuratețe.

Liderul de la acea vreme, modelul propus de Girshick et al. [12], Rețele Convoluționale bazate pe Regiuni, folosea filtre pentru culoarea și consistența pixelilor, urmate de un clasificator CNN pentru detectarea obiectelor.

Modul principal în care puteai îmbunătăți acuratețea unei rețele era să îi crești dimensiunea, însă acest lucru atrage overfit-ul și o complexitate computațională sporită. Modul cel mai bun de a evita aceste probleme este reducerea straturilor complet conectate și introducerea straturilor slab conectate, inclusiv în interiorul convoluțiilor.

Această abordare a fost continuată de cei de la Google, aducând câteva îmbunătățiri substanțiale. Plecând de la premiza lui Arora et al. [2], pentru a obține invarianță în traducere, arhitectura trebuie construită pe baza unor blocuri. Astfel, trebuie găsită o construcție optimă și apoi repetată spațial.

Un strat va fi responsabil să analizeze statisticile stratului anterior, iar apoi să le organizeze în mânunchiuri cu corelație mare. Mai multe mânunchiuri pot fi acoperite de o convoluție 1x1 în următorul strat. De asemenea, pot exista și regiuni mai puțin dense, ce vor fi acoperite de convoluții mai mari (3x3 sau 5x5), care vor predomina în adâncimea rețelei. Acest aspect însă poate duce la o explozie computațională rapidă, deci ar trebui evitate cât mai mult.

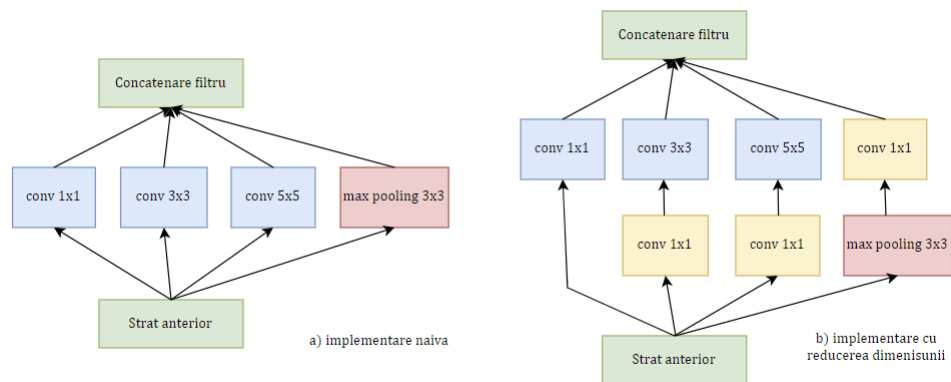


Figura 6: Blocuri de construcție ale rețelei Inception. [40]

În acest sens, sunt adăugate convoluții de 1x1 înainte de convoluțiile 3x3 sau 5x5 pentru a reduce dimensiunea calculelor, iar acestea sunt compresate și agregate în masă (în straturile de max pooling). Un alt beneficiu al acestei strategii este și introducerea activării ReLU între convoluțiile respective. Astfel, rețelele Inception sunt construite pe baza blocurilor din figura 6.

Un studiu ulterior, realizat tot de Szegedy et al. în 2016, sugerează că aceste convoluții 5x5 reprezintă încă un bottleneck, așa că se propune înlocuirea lor cu mini-rețele [41]. O convoluție 5x5 va fi înlocuită astfel de 2 convoluții 3x3 suprapuse. Experimente ulterioare au arătat că pot fi înlocuite și convoluțiile 3x3 cu două convoluții în paralel de 1x3 și 3x1. Pentru modelul InceptionV3 a fost aleasă în schimb prima variantă iar convoluțiile 3x3 vor avea stride de 1 sau 2. O altă optimizare constă în regularizare în momentul antrenării, numită Regularizare prin Netezirea Etichetei ("Label Smoothing Regularization" sau "LSR" în engleză). Scopul acestei regularizări este de a împiedica modelul să fie extrem de încrezător în predicție, reducând logaritmii cei mai mari.

Considerăm $p(k)$ fiind probabilitatea unei etichete prezise de rețea și $q(k)$ fiind adevărul de bază. Pentru simplificarea exemplului, considerăm $q(k)$ fiind egal cu 1, dacă k este clasa corectă, altfel 0. Minimizarea funcției de loss l (ecuația 21) este echivalentă cu maximizarea log-likelihood-ului etichetei (ecuația 22), care poate fi 1 sau 0. Acest fapt duce la o încredere

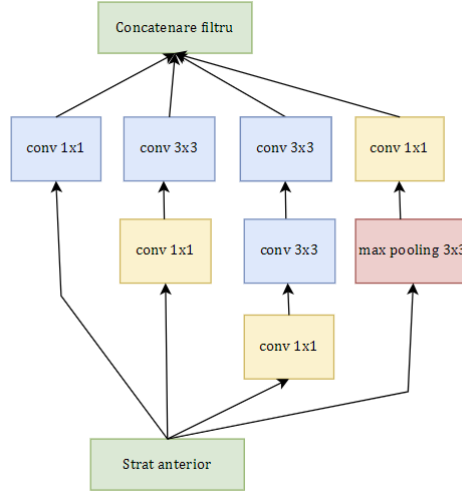


Figura 7: Bloc de construcție al rețelei InceptionV3. [41]

sporită în prezicerea etichetei. Pentru o prezicere mai diversă, implementăm q' , reducând astfel cel mai probabil răspuns (ecuația 23). Funcția de loss devine o pereche de două funcții de loss (ecuația 24), cea din urmă având rolul de a penaliza predicția referitor la distribuția uniformă a etichetelor. [41]

$$l = - \sum_{k=1}^K (\log(p_k)) * q(k) \quad (21)$$

$$q(k) = \delta_{k,y} \quad (22)$$

$$q'(k) = (1 - \epsilon) * \delta_{k,y} + \frac{\epsilon}{K} \quad (23)$$

$$H(q', p) = - \sum_{k=1}^K (\log(p_k)) * q'(k) = (1 - \epsilon) * H(q, p) + \epsilon * H(u, p) \quad (24)$$

Rezultatele acestor modificări obțin o acuratețe top-1 de 17.2% și top-5 de 3.58% [41], devenind state-of-the-art.

3.1.4 InceptionResNetV2

Echipa revine în 2017 pentru a testa noile descoperiri, respectiv Rețelele Neuronale Reziduale. Szegedy et al. își propun să adopte în arhitectura lor noul concept care este în topul performanțelor și permite scalarea rețelelor prin straturi [39].

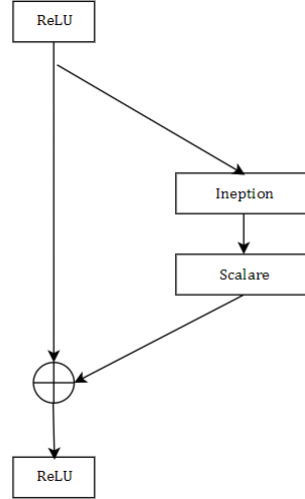


Figura 8: Bloc de construcție al rețelei InceptionResNetV2. [39]

Arhitectura propusă încorporează conceptul de bloc rezidual, care conține un bloc clasic Inception. Rezultatele sunt o îmbunătățire de 0.3% pentru top-1 și 0.15% pentru top-5.

3.1.5 EfficientNet

Mingxing Tan et al. își propun în 2019 să dezvolte o serie de modele compatibile cu dispozitivele mobile, numite MnasNet. În acest sens, mergând pe Rețele Neuronale Convoluționale clasice, aplică un concept numit Căutare Neurală. Căutarea are ca scop să maximizeze acuratețea modelului, dar în același timp cu o constrângere de timp T pentru inferență. [42]

$$\max_m ACC(m) \quad (25)$$

$$LAT(m) \leq T \quad (26)$$

Pentru simplitatea problemei, putem să rescriem ecuația rezultând într-o problemă care se rezumă la găsirea unei soluții Pareto optimale (nu mai există o altă soluție mai bună). Regula de alegerea ale lui α și β este ca soluțiile Pareto optimale să aibă recompense similare chiar și cu trade-off-uri diferite de acuratețe și latență. [42].

$$\max_m ACC(m) * \left[\frac{LAT(m)}{T} \right]^w \quad (27)$$

$$w = \begin{cases} \alpha & \text{daca } LAT(m) \leq T \\ \beta & \text{altfel} \end{cases} \quad (28)$$

Căutarea are în scop, de fapt, găsirea parametrilor potriviți pentru straturile de convoluție, un tuplu (H, W, M, N) , unde (H, W, M) este dimensiunea de intrare, (H, W, N) dimensiunea de ieșire, (H, W) este rezoluția de intrare și M, N dimensiunea filtrelor pentru fiecare strat. [42]

Mingxing Tan revine ulterior la această idee, propunând o soluție de scalare uniformă a modelelor, plecând de la arhitectura de bază, creând astfel familia de modele EfficientNet [43]. Aceste modele se bazează pe ideea că dacă vrem să mărim efortul computațional al unei rețele cu 2^N , atunci trebuie să scalăm adâncimea cu α^N , lățimea cu β^N și dimensiunea intrării cu γ^N , unde α, β, γ sunt obținuți printr-un grid search pe modelul de bază [43].

Modelele rezultate au acuratețe foarte apropiată de modelele state-of-the-art, însă necesită de 8.4 ori mai puțini parametri și de 16 ori mai puține FLOPS (unitate de măsură privind numărul de operații cu numere reale pe secundă) [43].

3.2 Rețelele Neuronale Recurente

3.2.1 LSTM

Rețelele Neuronale Recurente (RNN) își folosesc conexiunile de feedback pentru a stoca informații sub forma unor activări. Dificultatea acestora apare în schimb în efortul computațional sau reprezentarea slabă a cunoștințelor datorită dispariției informației de-a lungul procesului. Hochreiter et. al introduc în 1997 conceptul de "Memorie de scurtă durată pe termen lung" ("Long Short-Term Memory" sau prescurtat "LSTM" în engleză). Această memorie poate ține informația relevantă și pentru 1000 de pași de timp, fără a fi afectată de zgomot sau intrări eronate. [16]

Problema principală a rețelor recurente era faptul că eroarea produsă de o intrare devenea ori foarte mare ori foarte mică, afectând astfel actualizarea informației din interior. LSTM implementează conceptul de eroare constantă, ce se transmite prin celule, și rămâne în același timp relevantă pentru actualizarea informației.

3.2.2 GRU

GRU (Gated Recurrent Unit) este introdusă în 2014 de către Kyunghyun Cho et. al [6], alături de arhitectura RNN Encoder-Decoder. Această unitate recurentă este inspirată de LSTM, însă este mai simplă de implementat și mai eficientă în calcule.

Poarta de reset, r_j , este calculată ca în ecuația 29, unde sigma este funcția logistică sigmoid, iar notația $[X]_j$ reprezintă al j-lea element din vectorul X . x și h_{t-1} reprezintă intrarea și starea anterioară. W_r și U_r sunt ponderile/matricile care vor fi învățate. Similar funcționează și poarta de actualizare z (ecuația 30). [6]

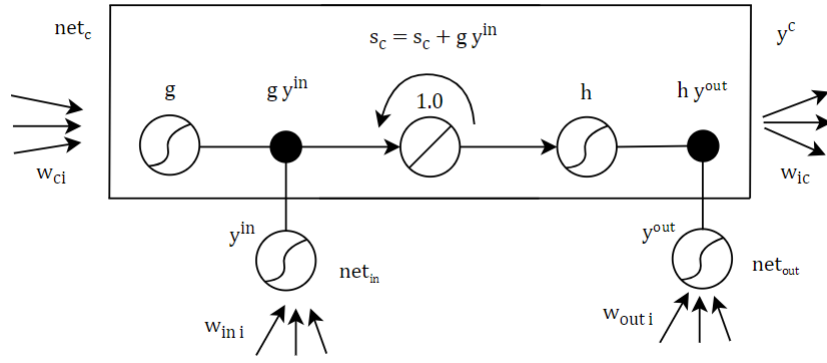


Figura 9: Reprezentare grafică a unei celule LSTM c și unitățile poartă in și out . Conexiunea recurentă cu pondere 1.0 indică feedback cu un delay de o unitate de timp. Este baza caruselului de eroare constantă (CEC). [16]

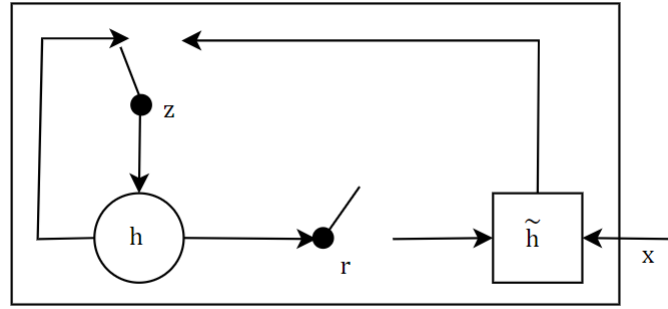


Figura 10: Reprezentare grafică a unei celule GRU. Poarta z selectează dacă starea ascunsă h să fie actualizată cu h' . Poarta r decide dacă starea anterioară să fie ignorată. [6]

$$r_j = \sigma * ([W_r x]_j + [U_r h_{t-1}]_j) \quad (29)$$

$$z_j = \sigma * ([W_z x]_j + [U_z h_{t-1}]_j) \quad (30)$$

Activarea porții este calculată ca în ecuațiile 31 și 32.

$$h_j^t = z_j h_j^{t-1} + (1 - z_j) h h_j^t \quad (31)$$

$$h h_j^t = \theta([W x]_j + [U(r \cdot h_{t-1})]_j) \quad (32)$$

Când poarta de reset este aproape de 0, starea ascunsă este forțată să ignore starea anterioară și să se reseteze cu intrarea curentă. Pe scurt, aceasta poate abandona orice informație considerată irelevantă mai târziu, având o reprezentare mai compactă. Poarta de actualizare

controlează câtă informație poate fi primită din unitatea precedentă, similar cu modul de acțiune al LSTM. Unitățile care capturează dependențe pe un interval scurt de timp se vor reseta mai des, iar cele care țin dependențe lungi se vor actualiza mai des. [6]

3.3 Mecanisme de atenție

3.3.1 Atenție de tip Bahdanau

În arhitectura Encoder-Decoder propusă de Cho et. al [6], encoderul primește o secvență de intrare, o secvență de vectori $x = (x_1, x_2, \dots, x_n)$, într-un vector. Abordarea cea mai comună este de a folosi un RNN astfel:

$$h_t = f(x_t, h_{t-1}) \quad (33)$$

$$c = q(\{h_1, \dots, h_{T_x}\}) \quad (34)$$

unde h_t este starea ascunsă la momentul t , iar c este un vector generat din secvențele ascunse, f și q fiind funcții nonliniare.

Decoderul prezice următorul cuvânt y_t , primind vectorul de context c și restul cuvintelor $\{y_1, \dots, y_{t-1}\}$:

$$p(y) = \prod_{t=1}^T p(y_t | \{y_1, \dots, y_{t-1}\}, c) \quad (35)$$

$$p(y_t | \{y_1, \dots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c) \quad (36)$$

unde g este o funcție nonliniară, iar s_t este starea ascunsă a RNN-ului.

Bahdanau et. al [3] propun un mecanism de aliniere folosind un RNN bidirecțional ca encoder, iar decoderul efectuează o căutare prin secvență în timpul predicției:

$$p(y_i | \{y_1, \dots, y_{i-1}\}, x) = g(y_{i-1}, s_i, c_i) \quad (37)$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \quad (38)$$

unde s_i este starea ascunsă la momentul i al RNN-ului. Astfel, spre deosebire de propunerea anterioară, predicția y_i se realizează pe baza unui vector c_i distinct. Acest vector de context depinde de o secvență de adnotări $\{h_1, \dots, h_{T_x}\}$ pe care encoderul le mapează din secvența

de intrare. Fiecare adnotare conține informație legată de întreaga secvență, cu o atenție deosebită pe elementele învecinate:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \quad (39)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \quad (40)$$

$$e_{ij} = a(s_{i-1}, h_j) \quad (41)$$

cu a fiind o rețea feedforward antrenată împreună cu restul componentelor. În esență, acesta este un mecanism de atenție, unde decoderul decide asupra căror cuvinte să-și bazeze predicția. Informația poate fi distribuită pe toată secvența, iar decoderul alege cu atenție ce va folosi din respectiva secvență. [3]

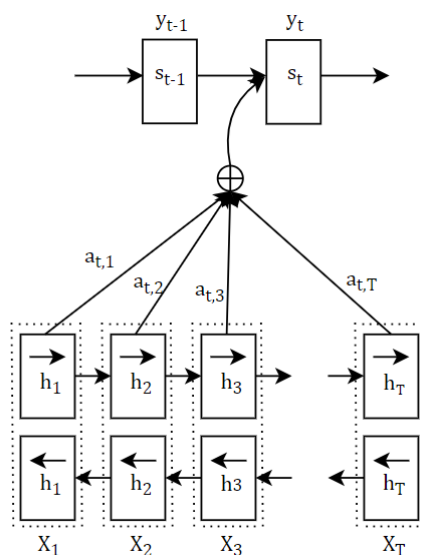


Figura 11: Ilustrare grafică a modului în care se generează y_T , dată fiind secvența (x_1, \dots, x_T) [3]

Experimentele au arătat că aceasta este o arhitectură mai performantă decât cea clasică fără mecanism de atenție, este mai robustă în fața lungimii propozițiilor și asigură un aliniament corect. Experimentele au fost realizate pe traducerea din engleză în franceză și s-au dovedit a fi foarte corecte. [3]

3.3.2 Atenție de tip Luong

Modelul de atenție propus anterior de Bahdanau et. al [3] este numit ulterior atenție globală, deoarece ia în considerare toate stările ascunse ale encoderului când vine vorba de vectorul

de context c_t . Luong et. al [26] propun un alt mecanism de atenție, atenția locală. Problema atenției globale este efortul computațional, dar și impracticabilitatea când vine vorba de secvențe mai lungi de text. Atenția locală are scopul să se axeze pe un subset restrâns de cuvinte.

Putem rescrie ecuația anterioară a modelului de atenție Bahdanau, unde h_t este starea ascunsă a predicției iar h_s starea ascunsă a sursei:

$$a_t(s) = \text{align}(h_t, h_s) = \frac{\exp(\text{score}(h_t, h_s))}{\sum_{s'} \exp(\text{score}(h_t, h_{s'}))} \quad (42)$$

$$\text{score}(h_t, h_s) = h_t^T h_s \quad (43)$$

Aliniamentul predictiv este dat de următoarea ecuație:

$$p_t = S \cdot \sigma(v_p^T \tanh(W_p h_t)) \quad (44)$$

cu S fiind lungimea propoziției sursei, iar W_p și v_p sunt parametrii modelului care vor fi învățați. p_t este un rezultat în intervalul $[0, S]$ (sigmoid). Pentru a favoriza aliniamentul specific în zona p_t , adăugăm o distribuție gaussiană centrată în p_t , rezultând ecuația:

$$a_t(s) = \text{align}(h_t, h_s) \cdot \exp\left(-\frac{(s - p_t)^2}{2\sigma^2}\right) \quad (45)$$

Astfel este calculată predicția prin intermediul mecanismului de atenție locală Luong, folosind o funcție de produs scalar. Modelul Luong performează mai bine pe secvențe mai lungi și cuvinte rare. [26]

3.4 Transformers

Conceptul de "Transformer" a fost introdus în 2017 de către Vaswani et al. [46] ca fiind o rețea simplă, bazată în întregime pe mecanisme de atenție. Cele mai multe modele care se ocupă cu secvențe sunt bazate pe rețele convoluționale sau rețele recurente complexe și mari. Modelele cele mai performante conectează encoderul și decoderul prin mecanisme de atenție. Transformerul se bazează exclusiv pe mecanisme de atenție și straturi feedforward, fiind mult mai eficient și în special paralelizabil, deci mai rapid. Experimentele inițiale în domeniul traducerii automate au depășit cu ușurință modelele anterioare, obținând statutul de state-of-the-art. Această arhitectură se poate generaliza ușor și pentru alte obiective, având rezultate bune chiar și pe seturi mici de date.

Atât intrarea cât și ieșirea transformer-ului sunt trecute prin embeddings poziționale.

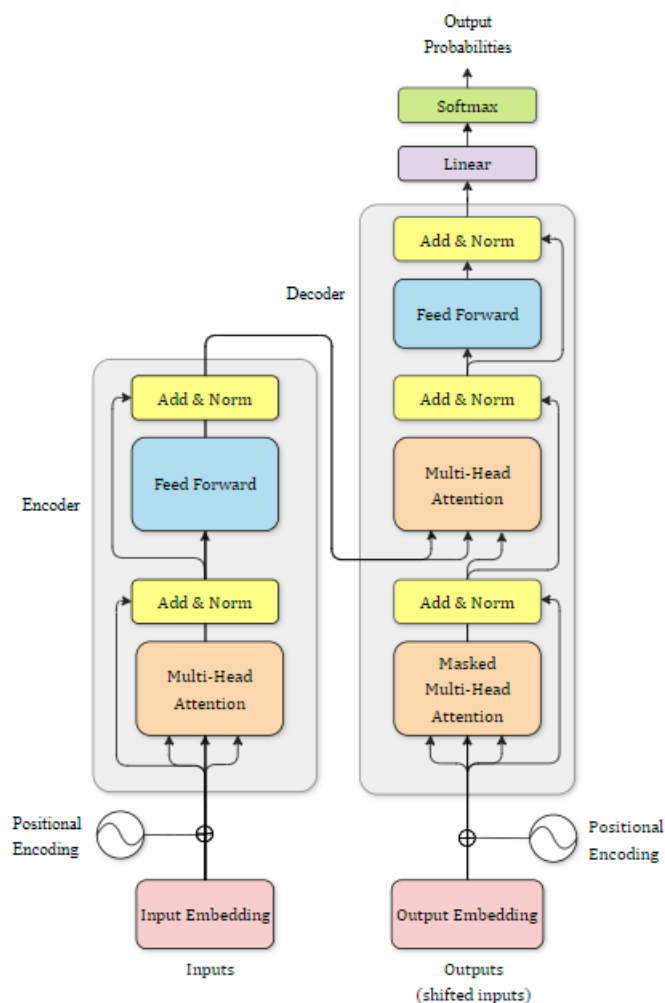


Figura 12: Arhitectura Transformer [46]

Encoderul transformerului este format din 2 sub-straturi: un mecanism de atenție cu capete multiple ("Multi-head Self-Attention") și un strat FCN (Fully Connected Network), iar fiecare dintre aceste straturi are o conexiune reziduală în jurul său, care se adaugă la ieșire și se normalizează.

Decoderul este format din 3 sub-straturi: un mecanism de atenție "Masked Multi-head" pentru secvența de ieșire, un alt mecanism de atenție "Multi-head" care leagă primul mecanism de encoder, iar apoi un strat complet conectat. Similar cu encoderul, fiecare strat are o conexiune reziduală și este adunat și normalizat.

Mecanismul de atenție cu multiple capete este format de fapt din mai multe componente de tip atenție multiplicativă scalată ("Scaled Dot-Product Attention"). Acest submecanism primește interogări și chei de dimensiune d_k și valori de dimensiune d_v . Calculăm produsul scalar al interogărilor și cheilor, împărțim cu $\sqrt{d_k}$, iar apoi prin SoftMax obținem ponderile pentru valorile asociate. Acest proces este asociat următoarei formule:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V \quad (46)$$

Ideea din spatele scalării cu $\sqrt{d_k}$ este faptul că pentru dimensiuni mari de intrare, produsul scalar crește foarte mult, iar ieșirea SoftMax produce gradienti prea mici. Astfel, pentru a controla magnitudinea, acesta este scalat (micșorat).

$$Multihead(Q, K, V) = Concat(head_1, head_2, \dots, head_n) \cdot W^O \quad (47)$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \quad (48)$$

unde proiecțiile W_i^Q , W_i^K , W_i^V sunt matrici de parametri.

Pentru că modelul nu folosește nici convoluții, nici rețele recurente, trebuie să introducem un mecanism care să țină cont și de secvențe în sine. Astfel, este introdus conceptul de embedding pozițional ("Positional Embedding"). Pe lângă embeddings normale pentru text, adăugăm și codificări poziționale de aceeași dimensiune, care sunt apoi însumate. Au fost testate și embeddings pre-antrenate, însă acestea au produs rezultate aproape identice [46].

3.4.1 Vision Transformer - ViT

Odată cu transformatorii bazati pe text devenind un standard în domeniul "Natural Language Processing", arhitectura s-a extins și în domeniul "Computer Vision" prin cercetările lui Dosovitskiy et al. [11] în 2020. Antrenat pe un set foarte mare de imagini, acesta poate fi adaptat pe seturi de date propuse pentru benchmark, obținând rezultate mai bune decât rețelele convoluționale și necesitând mai puține resurse pentru antrenare.

O imagine este împărțită în pătrate de dimensiuni egale. Imaginea are o structură 3D, (H, W, C) , iar transformer-ul acceptă embeddings de dimensiuni 2D. Astfel, trebuie să transformăm imaginea într-o secvență $(N, P \times P \times C)$ unde P este dimensiunea pătratului și N numărul de pătrate rezultate. Fiecare pătrat este embeddizat și i se aplică codificare pozițională, proces similar cu arhitectura transformer originală. În forma lor finală, aceste embeddări sunt trimise ulterior transformer-ului [11].

3.5 Învățare prin transfer de cunoștințe

Învățarea prin transfer (sau "Transfer Learning" în engleză) este un proces prin care utilizăm informația preexistentă în modele antrenate pentru a îmbunătăți rezultatele pe alte seturi de date sau task-uri înrudite[35]. În mod concret, în exemplul nostru, folosim CNN sau ViT deja antrenate pe task-uri de clasificare și le reutilizăm pentru a genera informații vizuale în

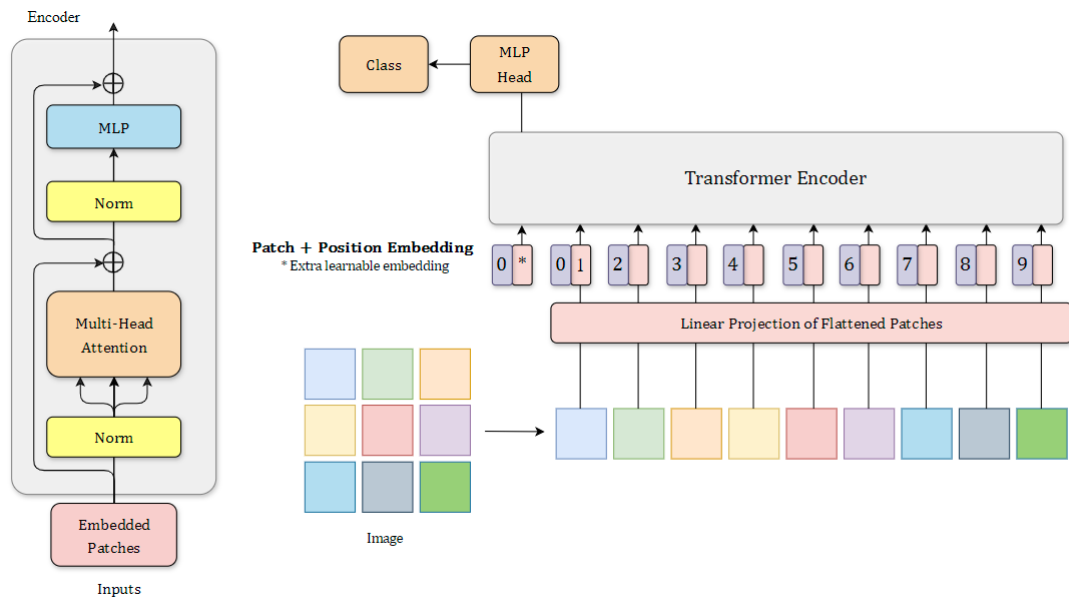


Figura 13: Arhitectura Vision Transformer și modul în care aceasta funcționează [11]

modelele noastre. Similar, putem folosi și transformeri bazate pe text (de exemplu, BERT) pentru a genera embeddings pentru descrieri. Avantajele acestei proceduri includ reducerea costului computațional (mai puțin timp și resurse necesare pentru antrenare), o varietate mai largă de cunoștințe asupra setului de date (de obicei, aceste rețele sunt antrenate pe seturi de date foarte mari) și o creștere a acurateții (rețelele sunt antrenate îndelungat până la un punct optim).

3.6 Tokens și Embeddings

Embeddings sunt reprezentări ale cuvintelor sau fragmentelor de cuvinte menite să fie consumate de către modelele de tip Machine Learning. În esență, embeddings ajută modelele să stabilească și să găsească relații între cuvinte, fiind deseori reprezentate printr-un vector n -dimensional. De exemplu, prin acest proces, cuvinte cu sens similar, cum ar fi nume de țări sau capitale, pot fi reprezentate alături într-un spațiu vectorial de dimensiune mare. Însă, primul pas în acest proces este tokenizarea. Pentru a putea asimila informația din text, trebuie să transformăm reprezentarea unei propoziții într-o secvență de numere ce poate fi ulterior procesată. Acest proces, depinzând de fiecare algoritm, poate însemna pur și simplu separarea cuvintelor întregi și asocierea acestora cu diferite ID-uri, numere; sau posibilitatea spargerii cuvintelor în sub-cuvinte.

Algoritmul Word2Vec propus de Mikolov et al. [28] folosește un model lingvistic probabilistic tip FFN (Feedforward Neural Network), în care intrarea este proiectată printr-un strat de proiecție, iar apoi printr-un strat de output. Vocabularul acestui model este construit printr-un arbore binar Huffman, producând coduri scurte pentru cele mai comune cuvinte. În mod

specific, modelul CBOW sau skip-gram încearcă să maximizeze clasificarea cuvintelor în raport cu restul cuvintelor din propoziție. Astfel, se alege un cuvânt și cele din vecinătate sunt prezise cu ajutorul algoritmului menționat mai sus.

În experimentele noastre, putem folosi embeddings pre-antrenate pe corpusul limbii române. Pentru Word2Vec avem la dispoziție un set de embeddings antrenate de către Nordic Language Processing Laboratory [20], antrenate folosind pagini de Wikipedia.

Un alt algoritm este FastText, propus de Bojanowski et al. [5], care împarte cuvintele în n-grame de litere. Ulterior, modelul învață astfel părți comune ale cuvintelor, cum ar fi prefixe și sufixe, fiind informație transferabilă și pentru cuvintele mai rare din corpus. Vocabularul este compus tot din cuvinte întregi.

Ca și opțiuni disponibile, există embeddings generate de către Grave et al. [13] pe un corpus format din Wikipedia și Common Crawl, o companie specializată în "copierea" internetului în scopuri de cercetare.

Un alt proiect folosind acest algoritm este și CoRoLa [45, 8, 32], care folosește un corpus al limbii române alcătuit din publicații jurnalistice, site-uri, literatură, documente, artă; adunate din mai multe colțuri ale țării, inclusiv din Republica Moldova, pentru a include cât mai multe dialecte și informații.

O altă metodă de obținere a embeddings este prin utilizarea unui transformer, de exemplu BERT [10]. Acest transformer este unul bidirecțional și de dimensiune mare (adânc), antrenat să învețe reprezentarea cuvintelor, din ambele sensuri ale propoziției, observând în deplin contextul acestora. Un model BERT adaptat pe limba română este cel antrenat de Masala et al. [27], RoBERT, cu ajutorul datelor de pe Wikipedia, cărți, articole, corpusul Oscar și diferite bucăți din RoTEX.

4 EXPERIMENTELE PROPUSE

4.1 Setul de date Flickr30k

Flickr30k[54] este un set de date compus din imagini și descrieri, foarte utilizat pentru antrenare modelelor tip "Image Captioning". Acesta conține 31,783 de imagini cu câte 5 descrieri fiecare (în total 158.915). Imaginile sunt preluate de pe platforma "Flickr", și constau în mare parte în oameni care fac diverse activități sau participă la diverse evenimente. Descrierile originale au fost adăugate de adnotatori, manual. Avantajele acestui set de date sunt dimensiunea medie, (alte alternative populare sunt Flickr8k sau MSCOCO care inițial avea 164K imagini, adăugându-se ulterior mai multe imagini pentru testare) și numărul mare de descrieri. Astfel pot fi prototipate și antrenate mai multe modele cu resurse restrânse.

4.1.1 Traducere și corectare

Setul de date Flickr30k a fost tradus automat folosind modelul NLLB[44] dezvoltat de Facebook, varianta cu 3.3 miliarde de parametrii. NLLB este un model mulți lingvistic, de traducere automată, având suport pentru 200 de limbi, incluzând limba Română. De asemenea, acesta este un model open source.

Traducerea nu este perfectă, fiind observabile și greșeli gramaticale sau posibile halucinații (secvențe care se repetă). Astfel, necesită corectare manuală. Această corectare a fost realizată de către 3 studenți, incluzându-mă și pe mine. În total cele 158.915 de secvențe de text au fost împărțite între 3 bucăți egale de aproximativ 53k.

Datorită unui interval scurt de timp (aproximativ 3 luni), și unui volum mare de secvențe, apare în mod normal și eroarea umană. Unele cuvinte nu au fost corectate, pot avea în continuare greșeli gramaticale, pot fi conflicte privind corectarea unor termeni în limba română (de exemplu termenul din engleză "skateboarder" nu are o traducere exactă așa că se va regăsi în traducere atât "skateboarder" cat și variațiuni: "skateboardist", "skateboard-er", "skateboard-ist").

4.1.2 Vocabular și variante

În primul rând, textul este curățat de semnele de punctuație. Acestea îngreunează tokenizarea. Dacă folosim un tokenizator clasic, care separă cuvintele prin spațiul dintre ele, o să obținem tokeni diferiți pentru fiecare cuvânt urmat de un semn de punctuație.

Tabela 2: Cele mai comune cuvinte din setul de date în funcție de numărul lor de apariții

Cuvânt	Nr. apariții	Cuvânt	# apariții	Cuvânt	# apariții	Cuvânt	# apariții	Cuvânt	# aparitii
un	127381	cu	57788	bărbat	25028	stă	14353	într-un	11794
o	101294	și	45530	femeie	20062	timp	14346	lângă	10829
în	92560	se	32707	într-o	15864	cămașă	13030	unui	9553
de	79244	care	31801	ce	15078	oameni	12770	fața	9302
pe	57976	la	26411	doi	14862	om	12337	unei	8913

În cazul antrenării unui strat de Embeddings de la zero, putem să ne construim propriul vocabular. Pentru a evita greșelile gramaticale, putem să alegem un prag de frecvență al cuvintelor: de exemplu, includem doar cuvintele care se repetă de cel puțin două ori. Acest lucru ne ajută să eliminăm din greșelile gramaticale și termeni ce nu vor fi învățați foarte mult. Restul termenilor pot fi înlocuiți un token special, $<UNK>$ (din engleză "unknown", necunoscut).

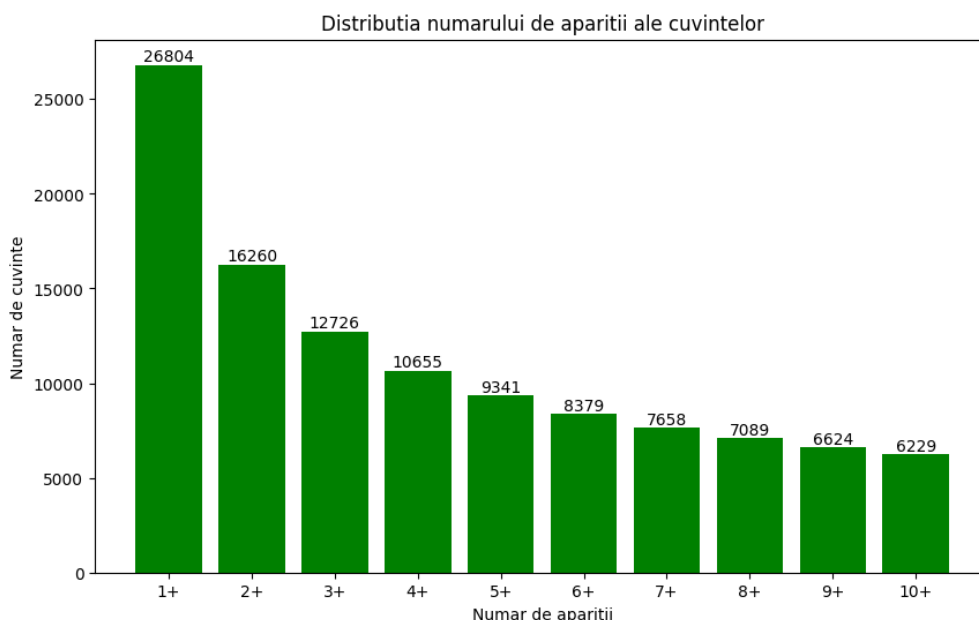


Figura 14: Grafic reprezentând distribuția numărului de cuvinte în funcție de pragul de apariție

În cazul utilizării de Embeddings preantrenate, putem să reutilizăm vocabularul acestora. Treceam prin fiecare token din propozițiile noastre, vedem dacă acesta se regăsește în vocabularul propus și îl adăugăm în noul vocabular. Astfel, eliminăm și greșelile gramaticale și cuvinte care nu există și putem să reducem și vocabularul propus astfel încât să utilizăm doar cuvintele noastre.

De asemenea, lungimea frazelor variază foarte mult (de la 2 la 115 că număr de cuvinte). Deseori informația din propoziții este mai mică de atât, așa că am preferat reducerea lungimii tuturor propozițiilor la 25 de cuvinte.

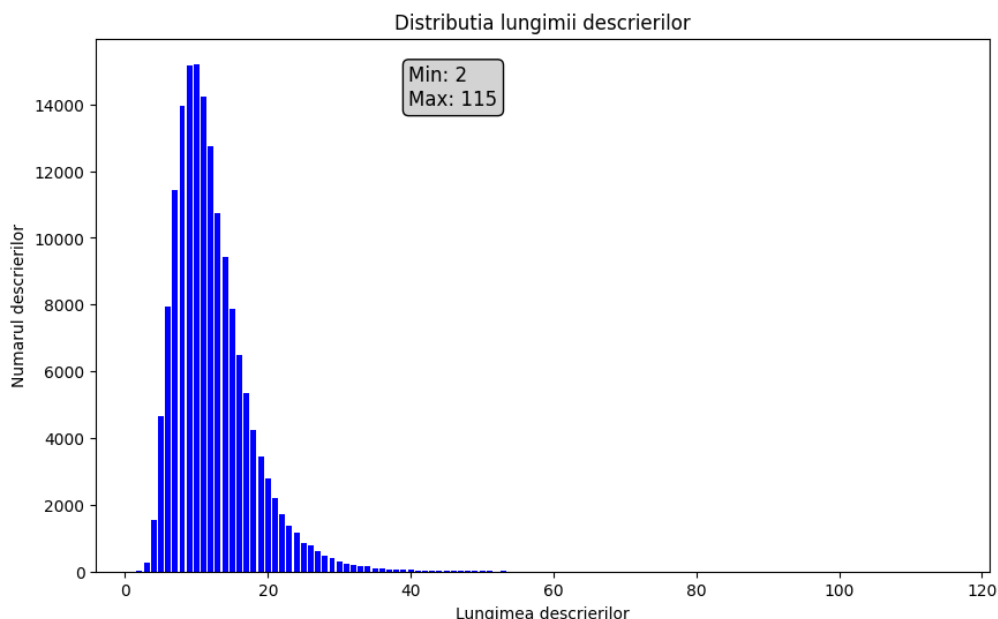


Figura 15: Grafic reprezentând distribuția lungimilor propozițiilor

4.2 Arhitectura modelelor propuse

Modelele folosite pentru experimente sunt alcătuite din două componente: Encoder și Decoder. Folosind conceptele menționate mai sus putem să construim modele bazate pe arhitecturi CNN-RNN sau arhitecturi pe bază de transformeri sau hibride.

4.2.1 CNN-Encoder RNN-Decoder

Modulul Encoder preia o imagine și frazele asociate, imaginea este trecută printr-o serie de transformări, proces numit și augmentare, iar apoi cu ajutorul unei Rețele Convoluționale, din care eliminăm straturile FCN, extragem un vector de caracteristici unidimensional (figura 16).

Fiecare propoziție este tokenizată, iar apoi pentru învățare secvențială aceasta este trunchiată iterativ, pornind de la primul cuvânt, și ajungând la secvențe din ce în ce mai lungi. Aceste secvențe sunt umplute ("padding") cu 0 la început pentru a păstra lungimea inițială.

Ulterior, următorul token din fiecare secvență va deveni predicția perechii dintre secvența respectivă și vectorul imaginii.

Pentru arhitectura clasică Encoder-Decoder (figura 17) sunt folosite Rețele Recurente, respectiv LSTM și GRU, iar pentru a putea implementa mecanismele de atenție Bahdanau și Luong vom folosi rețele Recurente Bidirecționale (permit parcurgerea în ambele sensuri ale secvenței), respectiv BiLSTM și BiGRU. Aceste rețele Recurente sunt responsabile doar de învățarea secvențelor text.

Înainte de a intra în RNN, secvența tokenizată trebuie să treacă prin procesul de "Embedding",

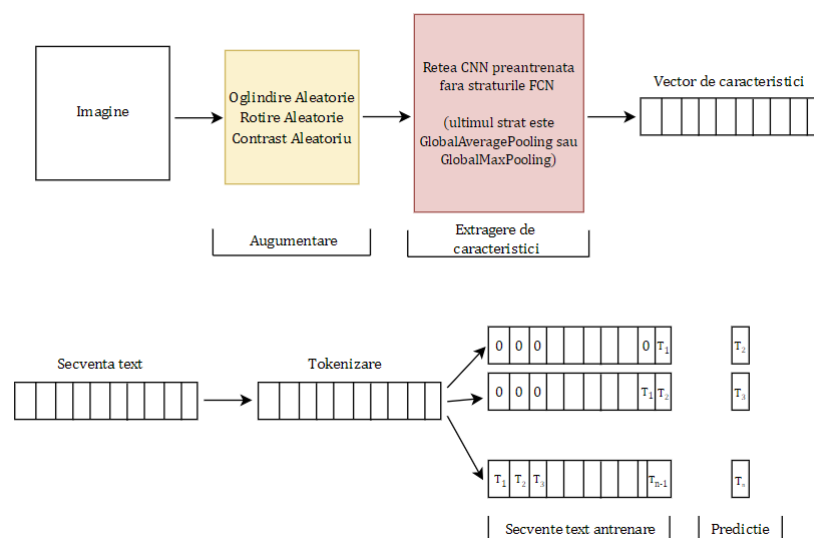


Figura 16: Ilustrare grafică a encodării imaginilor și secvențelor de text

adică asignăm un vector fiecărui token din secvență și creștem dimensiunea spațială, pentru a putea reprezenta și învăța informația mai bine. Acesta este ultimul pas al Encoderului. Ieșirea RNN, a decoderului, este îmbinată cu vectorul imaginii prin adunare sau concatenare. La final, avem o serie de câteva straturi FCN pentru a putea prezice următorul token din secvență. Pentru a preveni overfitting-ul, putem adăuga un mecanism de Dropout, pentru a seta un procent fix din caracteristicile imaginilor și din Embedding-ul textului cu 0. Astfel, încercăm să nu lăsăm modelul să caute "pattern-uri" nedorite, vrem să generalizeze informația învățată.

Un alt tip de arhitectură (figura 18) folosit se folosește de mecanisme de atenție (Bahdanau și Luong). Pentru acestea avem nevoie în schimb de un RNN bidirecțional. Acesta crește dimensiunile datelor, așa că vom aplatiza ("flatten") rezultatul stratului bidirecțional.

Cum fiecare din aceste arhitecturi poate fi folosită atât cu LSTM cât și GRU, și mecanisme de îmbinare prin adunare sau concatenare, obținem câte 4 variante de modele pentru fiecare, în total 8:

1. Decoder LSTM și Adunare
2. Decoder LSTM și Concatenare
3. Decoder GRU și Adunare
4. Decoder GRU și Concatenare
5. Decoder BiLSTM cu mecanism de atenție Luong și Concatenare
6. Decoder BiLSTM cu mecanism de atenție Bahdanau și Concatenare
7. Decoder BiGRU cu mecanism de atenție Luong și Concatenare
8. Decoder BiGRU cu mecanism de atenție Bahdanau și Concatenare

Cu fiecare din acești Decodoare putem folosi orice CNN-Encoder propus: ResNet50, ResNet50V2, InceptionV3, InceptionResNetV2, EfficientNet B4; rezultând în 40 de modele.

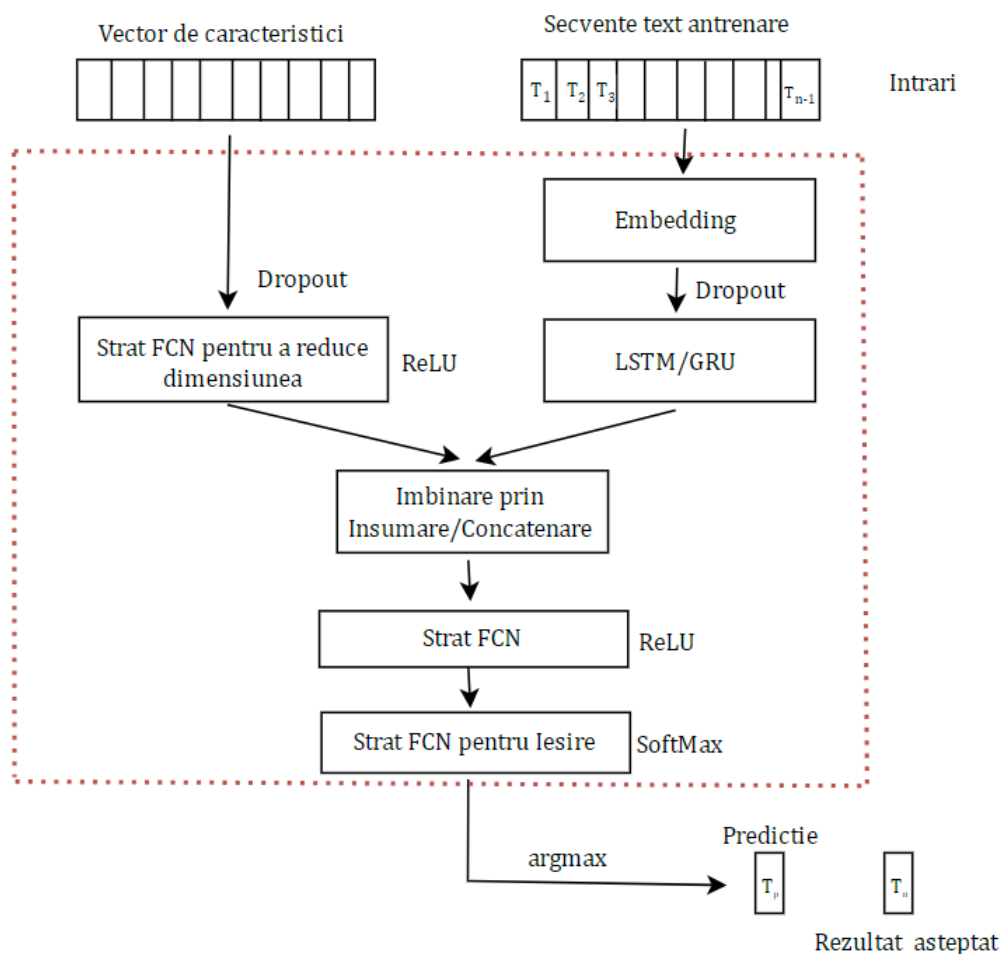


Figura 17: Arhitectura CNN-Encoder RNN-Decoder

4.2.2 Vision Transformer-Encoder RNN-Decoder

Printr-un procedeu similar putem folosi și transformerul vizual ViT-b16, eliminând capătul MLP, pentru a codifica imaginea într-un vector unidimensional. Astfel, obținem încă 8 modele.

4.2.3 CNN-Transformer-Encoder Transformer-Decoder

Un al treilea tip de arhitectură (figura 19) este compusă dintr-un Transformer Encoder Vizual, adaptat cerinței noastre. Spre deosebire de ViT, în loc să spargem imaginea în petice și să le antrenăm, aceasta este trecută printr-o rețea convoluțională preantrenată. Pentru a nu crește complexitatea rețelei, trecem aceste embeddings printr-un strat FFN și reducem dimensiunea acestora, și prin mecanismul Multi-Head Attention cu conexiune reziduală. Decoderul funcționează pe același principiu, însă va primi ca intrare secvențele de text. Similar, putem folosi și ViT în loc de CNN-uri. Encoderul are rolul de a învăța să reprezinte într-un alt mod informația din imagini, adaptându-se la task-ul curent. Această arhitectură este inspirată și

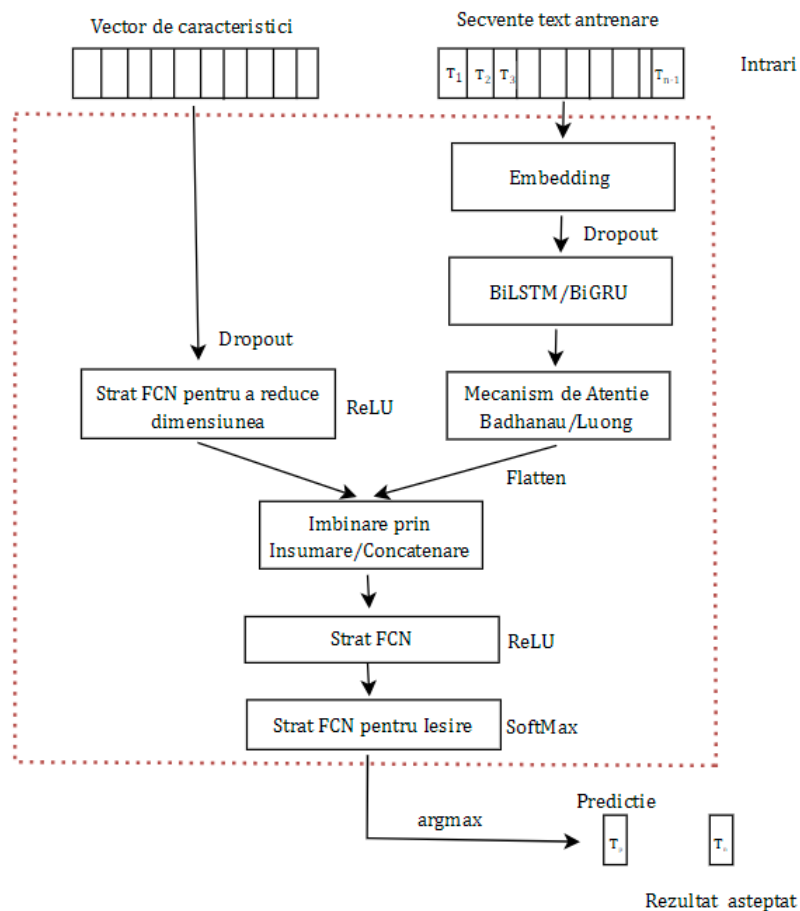


Figura 18: Arhitectura CNN-Encoder RNN-Decoder cu mecanism de atenție

implementată cu ajutorul unui tutorial pus la dispoziție de librăria "Keras"¹.

4.3 Tehnici pentru generarea descrierilor

Modelele noastre pornesc de la o secvență (de exemplu: "<START>"), și împreună cu imaginea transformată în caracteristici, prezic următorul token cel mai probabil. Procesul se repetă până întâlnim limita de lungime sau token-ul "<END>". Astfel, pentru a genera descrieri, avem două posibilități, Greedy Search și Beam Search.

4.3.1 Greedy Search

Greedy Search alege mereu token-ul cel mai probabil (cu valoarea cea mai mare din predicție), îl concatenează la secvență și repetă procesul până când una dintre cele două condiții menționate sunt îndeplinite.

¹https://keras.io/examples/vision/image_captioning/

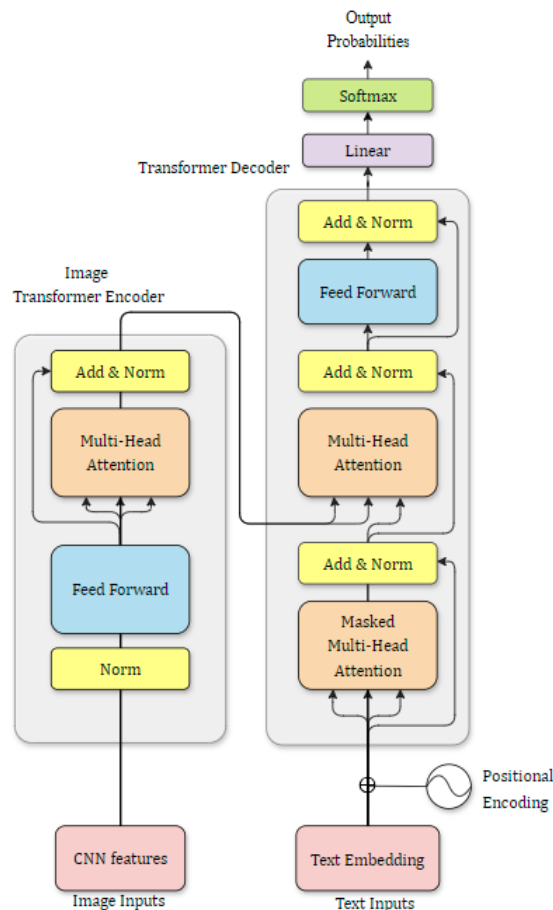


Figura 19: Arhitectura Transformer-Encoder Transformer-Decoder folosind CNN/ViT pentru extragerea de informații din imagine ca și intrare a encoderului

4.3.2 Beam Search

Beam Search este un tip de căutare ramificată (de unde provine și termenul Beam). Pentru aceasta este necesară stabilirea numărului de ramificații de la fiecare secvență. Ca exemplu, o căutare cu 3 beam-uri ar începe cu aceeași secvență "**<START>**", iar peste ea sunt appenduite pe rând top 3 tokene prezise, reținând și probabilitățile acestor tokeni. Pentru următoarea iterație, pornim cu secvențele "**<START><TOK1>**", "**<START><TOK2>**", "**<START><TOK3>**". De la fiecare predicție, concatenăm cei mai probabili 3 tokeni rezultând în potențial 9 candidați. Adunăm în același timp iarăși și probabilitatea secvenței. Următoarea căutare va începe cu top 3 secvențe în funcție de scorul de până acum. Procesul se repetă până dăm de un token "**<END>**" sau atingem limita. Beam Search este o alternativă mai bună decât Greedy pentru că ne permite o explorare mai bună a spațiului răspunsurilor.

4.4 Evaluarea modelelor

Pentru a evalua descrierea generată de modele, modul cel mai bun de lucru ar fi evaluarea umană. Această verificare ar sesiza diferențele dintre textul generat și cel de referință, ar sesiza contextul și sensul propoziției și cum diferă acestea. Însă, există câteva dificultăți, prima fiind cuantificarea greșelilor, modul în care acestea sunt contorizate, dacă diferă doar cuvinte din propoziția de referință, dacă are complet alt sens, dacă diferă cuvinte și are același sens, dacă sunt în mare parte aceleași cuvinte dar cu alt sens. La fel de complexă este și atribuirea unui scor pe baza acestor factori. A doua problemă în acest sens este și costul unei astfel de verificări manuale, atât monetar cât și ca timp. Din aceste motive, au fost implementate diferite metrice și diferiți algoritmi pentru o verificare automată, fără a depinde de subiectivitate, și având un set clar de reguli pentru jurizare. Pentru antrenament dispunem atât de descrieri cât și de imagini. Astfel, putem considera doar descrierile pentru jurizarea modelului, ținând cont că acestea sunt corecte (iar dacă nu ar fi, tot pe baza acestora învață modelul).

4.4.1 Bleu

Scorurile BLEU au fost propuse de Papineni et. al [36] în 2002, cu scopul accelerării cercetării și dezvoltării algoritmilor pentru traduceri automate. Acestea sunt considerate foarte corelate cu jurizarea umană și au fost folosite inițial pentru a jura traducerile automate și rezumatele textului ("Machine Translation" și "Text Summarization" în engleză). Aceste două domenii pot fi încadrate în generarea de text în limbaj natural, la fel cum poate fi încadrată și generarea de descrieri, fiind o metrică relevantă pentru experimentele propuse.

Pentru a defini scorul BLEU, trebuie să înțelegem mai întâi câteva concepte: N-grame, precizia și precizia trunchiată ("clipped precision" din engleză). Să spunem că avem următoarea propoziție: "Doi prieteni se bucură de timpul petrecut împreună". O N-gramă este o secvență de n cuvinte alăturate, respectiv pentru scorul BLEU, de până la 4 cuvinte alăturate. 1-gramele acestei propoziții sunt cuvintele în sine: "Doi", "prieteni", etc. 2-gramele sunt secvențele: "Doi prieteni", "prieteni se", etc. 3-gramele sunt: "Doi prieteni se", "prieteni se bucură", etc. 4-gramele sunt: "Doi prieteni se bucură", "prieteni se bucură de", etc. Mai departe, o să ne referim la propozițiile generate automat ca fiind "candidați" iar cele din setul de date ca fiind "referințe".

Precizia se referă la numărul de n-grame din candidat care se află în referință.

$$Precizie_n = \frac{\text{Număr de n-grame din candidat care sunt și în referință}}{\text{Număr de n-grame din candidat}} \quad (49)$$

Avem următorul exemplu:

Referință : Doi prieteni se bucură de timpul petrecut împreună (50)

Candidat : Doi amici se bucură de timpul petrecut împreună (51)

Pentru calcularea preciziei unei 1-grame, toate cuvintele din candidat, în afară de "amici", se repetă în referință, precizia fiind 7/8. Problema cu această precizie ar fi în cazul unui cuvânt care se repetă, de exemplu să avem drept candidat o secvență: "Doi doi doi". Precizia acesteia ar fi 3/3, adică 100%, ceea ce nu este corect.

Precizia trunchiată are rolul de a remedia această problemă, impunând un prag maximal pentru care un cuvânt poate fi contorizat, la numărul de apariții al acestuia în referință. Pentru exemplul cu "Doi doi doi", cuvântul "doi" apare o singură dată în referință, deci ar fi numărat o singură dată. Scorul actualizat este de 1/3.

$$PT_n = \text{Precizie trunchiată}_n = \frac{\text{Număr trunchiat de n-grame din candidat care sunt și în referință}}{\text{Număr de n-grame din candidat}} \quad (52)$$

Scorul BLEU se calculează astfel:

$$BLEU_n = Penalizare * \exp\left(\sum_{n=1}^N w_n * \log Precizie_n\right) \quad (53)$$

Propozițiile candidat prea lungi sunt deja penalizate prin precizia n-gramelor. O altă problemă este atunci când propozițiile candidat sunt prea scurte, au șansa să aibă un scor mult mai mare. Astfel, dorim să penalizăm cu 1.0 (adică deloc) atunci când propoziția este la fel de lungă sau mai mare; iar când propoziția este prea scurtă vrem să micșorăm scorul. Formula propusă este următoarea:

$$Penalizare = \begin{cases} 1 & \text{dacă } c > r \\ e^{1-r/c} & \text{dacă } c \leq r \end{cases} \quad (54)$$

unde r și c reprezintă lungimea propozițiilor referință și candidat. Termenul w_n din formula BLEU-n, reprezintă o greutate uniformă aplicată preciziei, respectiv:

$$w_n = \frac{1}{n} \quad (55)$$

Pentru mai multe fraze de referință, se alege scorul BLEU maxim în raport cu fiecare. Scorul BLEU este pe scară [0-1], respectiv poate fi reprezentat în procente [36].

4.4.2 Rouge

Metrica Rouge a fost dezvoltată de către Chin-Yew Lin [24], dorind să completeze lipsurile metricii BLEU, cu accent pe recall. Metrica BLEU, bazată pe precizie, calculează scorul

împărțind numărul de n-grame identice la numărul total de n-grame din candidat. În schimb, metrica Rouge împarte la numărul de n-grame din referință. De asemenea, în loc de a considera doar scorul maxim al unei singure referințe, se adună numărul de bi-grame, oferind mai multă flexibilitate în variabilitatea propoziției candidat.

Spre deosebire de BLEU, scorurile ROUGE bazate pe n-grame sunt calculate doar pentru unigramă și bigramă, utilizând formula:

$$ROUGE_n = \frac{\sum_{\text{ref}} \sum_{n\text{-gramă}} \text{număr n-grame potrivite cu ref}}{\sum_{\text{ref}} \sum_{n\text{-gramă}} \text{număr n-grame în ref}} \quad (56)$$

Colecția de metrici ROUGE include și ROUGE-L, o metrică care calculează cea mai lungă secvență comună ("Longest Common Subsequence" sau LCS). Deși a fost propusă inițial pentru a evalua calitatea rezumării textului ("Text Summary"), poate fi folosită pentru a evalua și coerența frazei generate în raport cu textul de referință. Definim conceptul de LCS astfel: avem două secvențe $[x_1, x_2, \dots, x_n]$ și $[y_1, y_2, \dots, y_m]$. Dacă există o secvență comună, cea mai lungă, sub formă de indici consecutivi $[i_1, i_2, \dots, i_k]$, astfel încât $x_{i_1} = y_{i_1}, x_{i_2} = y_{i_2}, \dots, x_{i_k} = y_{i_k}$, atunci aceasta se va numi cea mai lungă secvență comună. Putem calcula recall-ul și precizia acestora folosind formulele următoare, unde m reprezintă lungimea frazei de referință R și n lungimea frazei candidat C :

$$R_{lcs} = \frac{LCS(R, C)}{m} \quad (57)$$

$$P_{lcs} = \frac{LCS(R, C)}{n} \quad (58)$$

Putem calcula apoi media armonică ponderată, care este și formula pentru ROUGE-L:

$$ROUGE_L = \frac{(1 + \beta^2) \cdot R_{lcs} \cdot P_{lcs}}{R_{lcs} + \beta^2 \cdot P_{lcs}} \quad (59)$$

unde β este un parametru dinamic. Formula oferă următoarele proprietăți: $ROUGE_L = 1$ când $R = C$ (frazele sunt identice) și $ROUGE_L = 0$ când R și C nu au nimic în comun [24].

4.4.3 Meteor

Metrica METEOR a fost dezvoltată de către Banerjee et. al[4], încercând să completeze de asemenea lipsurile metricii BLEU. Acesta a identificat câteva probleme principale privind scorurile BLEU: lipsa recall-ului, N-grame de ordin prea mare (probleme pe care le rezolvă și implementarea ROUGE), lipsa unei potriviri explicite între N-grame ceea ce duce la numărări incorecte și folosirea mediei geometrice în calcul (dacă scorul unei n-grame este 0 atunci și media este 0).

Pentru calculul metricii METEOR, se realizează mapări între unigramele din propoziția candidat și referință, prin utilizarea a mai multor tehnici: mapare directă ("băiatul" cu "băiatul"), prin trunchiere la forma din dicționar ("băiatul" cu "băiat") sau prin sinonime. Dintre acestea se alege setul care conține cele mai puține încrucișări între mapări (dacă suprapunem cele două fraze, referință și candidat, și trasăm linii între unigramele mapate, fiecare intersecție de linii este numărată). Această idee poate fi transpusă astfel: avem două mapări (c_i, r_j) și (c_k, r_l) . O intersecție a acestora ar însemna că următoarea formulă să fie evaluată la un număr negativ.

$$(pos(c_i) - pos(c_k)) * (pos(r_j) - pos(r_l)) \quad (60)$$

Următorul pas, este calculul preciziei P și recall-ului R:

$$P = \frac{\text{număr de unigrame mapate}}{\text{număr unigrame totale în candidat}} \quad (61)$$

$$R = \frac{\text{număr de unigrame mapate}}{\text{număr unigrame totale în referință}} \quad (62)$$

Calculăm media armonică cu P și R:

$$Medie = \frac{10 * P * R}{R + 9 * P} \quad (63)$$

De asemenea, se calculează și o penalizare, dată de raportul dintre numărul de secvențe continue ce se regăsesc în referință și candidat și numărul de unigrame potrivite. Astfel, cu cât secvențele sunt mai lungi, penalizarea este mai mică.

$$Penalizare = 0.5 * \frac{\text{număr secvențe}}{\text{număr unigrame}} \quad (64)$$

Scorul METEOR este dat de următoarea formulă:

$$METEOR = Medie * (1 - Penalizare) \quad (65)$$

Prin această implementare, a fost observată o corelare mai bună între scorul acordat de juriul uman și scorul calculat automat, fiind o îmbunătățire față de metrica BLEU[4].

4.4.4 Ter

O altă metrică este metrica TER ("Translation Error Rate") propusă de Snover et. al[37] după apariția metricii METEOR. Această metrică abordează problema diferit, în loc să compare n-grame sau secvențe, măsurăm calitatea textului generat în numărul minim de pași și acțiuni

pe care trebuie să le facem pentru a ajunge la referință. Un astfel de pas poate însemna inserție, ștergere, deplasare de cuvinte sau secvențe de cuvinte. Fiecare operație, de orice tip, are același cost. Pentru a putea deplasa textul, toate cuvintele trebuie să fie identice, inclusiv majusculile. Formula pe scurt pentru calculul metricii este următoarea[37]:

$$TER = \frac{\text{număr de editări necesare}}{\text{media numărului de cuvinte din referință}} \quad (66)$$

4.4.5 Cider

Metrică CIDER este propusă mai recent, cele menționate anterior datând din cel mult 2005. Aceasta a fost propusă în 2015 de Vedantam et al.[47], și este gândită pentru evaluarea descrierilor generate pe bază de imagini, față de cele anterioare al căror scop inițial a fost evaluarea textelor traduse. Fată de metricile anterioare, scopul nu mai este potrivirea exactă a unigramelor sau n-gramelor, ci evaluarea cantității de informații regăsite în candidat.

În primul rând, cuvintele, unigramele, sunt reduse la forma lor din dicționar (rădăcină). Următorul pas este să evaluăm cele mai comune n-gramme regăsite în frazele de referință. Astfel, vrem să atribuim ponderi mai mici cuvintelor comune, care nu oferă foarte multă informație: de exemplu prepoziții, pronume, iar în cazul setului de date propus, cuvintele și secvențele cele mai comune sunt articole vestimentare însoțite de atribute (culori, aspect). Calculăm frecvența inversă termenilor ("TF-IDF" sau "term frequency-inverse document frequency" din engleză) pentru fiecare n-grama astfel:

$$g_k(s_{ij}) = \frac{h_k(s_{ij})}{\sum_{\omega_l \in \Omega} h_l(s_{ij})} \log \frac{|I|}{\sum_{I_p \in I} \min(1, \sum_q h_k(s_{pq}))} \quad (67)$$

Unde $g_k(s_{ij})$ reprezintă numărul de apariții ale n-grammei w_k în fraza s_{ij} , Ω reprezintă vocabularul tuturor n-grammelor și I setul întreg de imagini. Primul termen al ecuației măsoară frecvența n-grammei w_k , iar al doilea termen măsoară raritatea acesteia. Primul termen plasează o pondere mai mare n-grammelor frecvente din propozițiile imaginii, iar al doilea o pondere mai mică pentru termenii regăsiți în toate imaginile.

Scorul CIDER pentru n-gramme este calculată prin similaritatea cosinusului:

$$CIDER_n(c_i, S_i) = \frac{1}{m} * \sum_j \frac{g^n(c_i) * g^n(s_{ij})}{\|g^n(c_i)\| * \|g^n(s_{ij})\|} \quad (68)$$

iar scorul final este dat de formula:

$$CIDER(c_i, S_i) = \sum_{n=1}^N w_n * CIDER_n(c_i, S_i) \quad (69)$$

cu $w_n = 1/n$. Această metrică s-a dovedit a fi foarte precisă pentru acest subdomeniu, fiind folosită ca și benchmark de majoritatea modelelor state-of-the-art.[47]

5 REZULTATE

Modelele au fost implementate cu ajutorul bibliotecilor "Tensorflow" [1] și "Keras" [7]. Transformerul ViT este folosit cu implementarea din biblioteca "vit-keras". Pentru transformer-ul RoBERT am folosit biblioteca "HuggingFace" [52]. Mecanismele de atenție sunt implementate cu ajutorul bibliotecii "keras-self-attention". Scorurile sunt calculate cu ajutorul librăriilor "Evaluate" [48] și "Cidereval".

Pentru antrenare am folosit platforma Google Collab, care pune la dispoziție diverse plăci grafice performante. În cazul de față, am folosit o placă video Nvidia T4 cu aproximativ 16GB VRAM și un mediu de dezvoltare cu 51GB RAM.

Setul de date îl împărțim în două bucăți, cel folosit pentru antrenare și constă în 26,783 de imagini a câte 5 descrieri fiecare și cel folosit pentru validare 5,000 de imagini și 25k descrieri.

Pentru antrenarea modelelor vom folosi optimizatorul Adam și funcția de loss "Categorical Cross Entropy". De asemenea, pentru a spori acuratețea modelului, vom augmenta imaginile înainte de a fi trecute prin CNN sau ViT. Acest proces presupune aplicarea unor transformări aleatorii, mai exact oglindire pe orizontală, o rotație cu factor de 0.2 și schimbarea contrastului cu un factor de 0.3. Astfel, pentru 10 epoci, modelele vor avea la dispoziție 267,830 imagini diferite. Deși acestea sunt practic repetitive, ajută modelul să învețe mai multă informație și să prevină overfit-ul.

5.1 Rezultate cantitative

5.1.1 Rezultate obținute utilizând arhitectura CNN-Encoder RNN-Decoder

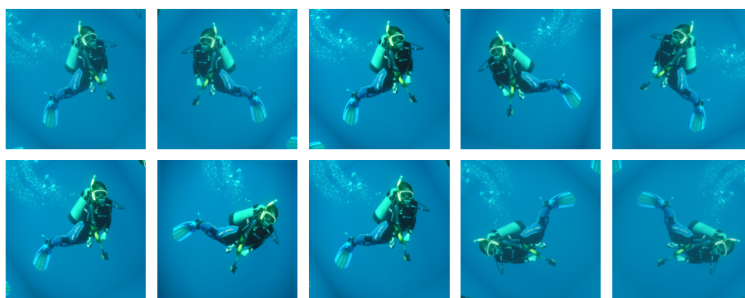


Figura 20: O singură imagine este transformată în 10 moduri diferite prin augmentare

Pentru RNN-Decoder vom stabili următoarele abrevieri:

- C1 - îmbinare prin Adunare
- C2 - îmbinare prin Concatenare
- C3 - Decodor cu mecanism de atenție Luong și Concatenare drept metodă de îmbinare
- C4 - Decodor cu mecanism de atenție Bahdanau și Concatenare drept metodă de îmbinare

Pentru a putea face o comparație corectă stabilim că vom antrena toate modelele timp de 10 epoci.

Pentru acest prim experiment vom folosi un strat de Embeddings neinitializat, pe care îl vom antrena noi. Includem în vocabular doar cuvintele care se repetă de cel puțin două ori. Hiperparametrii acestui experiment sunt:

- Număr de epoci - 10
- Rata de învățare - 0.0003 ($3 * 10^{-4}$)
- Dimensiune batch - 32
- Rata dropout - 0.3 (30%)
- Dimensiune Embeddings - 128
- Dimensiune LSTM/GRU - 256
- Dimensiune FFN - 256

Tabela 3: Rezultatele experimentelor CNN-Encoder RNN-Decoder cu Embeddings antrenate de la zero

Model			BLEU				CIDEr	ROUGE			METEOR	TER
			B-1	B-2	B-3	B-4		R-1	R-2	R-L		
RN50	LSTM	C1	46.06%	15.98%	6.39%	2.36%	27.87%	35.62%	15.98%	31.53%	28.38%	91.64
		C2	44.83%	15.55%	6.20%	2.24%	28.02%	35.44%	16.01%	31.51%	28.22%	94.77
	BiLSTM	C3	46.10%	15.89%	6.18%	2.23%	26.55%	35.58%	16.04%	31.67%	28.39%	90.77
		C4	46.54%	15.62%	6.29%	2.32%	24.91%	35.31%	15.47%	31.21%	27.92%	88.96
	GRU	C1	44.42%	15.37%	6.15%	2.25%	27.67%	35.56%	16.20%	31.55%	28.82%	97.31
		C2	45.29%	15.71%	6.28%	2.29%	28.23%	36.00%	16.56%	31.89%	29.01%	94.92
	BiGRU	C3	45.18%	14.99%	5.93%	2.18%	25.32%	35.41%	15.72%	31.28%	28.29%	95.25
		C4	43.91%	14.43%	5.71%	2.11%	25.56%	34.60%	15.05%	30.59%	27.78%	97.27
RN50V2	LSTM	C1	28.23%	6.47%	1.91%	0.54%	8.29%	26.03%	8.23%	22.03%	20.47%	146.50
		C2	17.30%	3.85%	1.23%	0.31%	5.24%	19.10%	5.86%	16.71%	15.78%	190.67
	BiLSTM	C3	32.73%	6.52%	1.83%	0.48%	8.04%	28.03%	7.65%	22.86%	21.82%	136.81
		C4	19.11%	3.71%	0.99%	0.28%	5.35%	19.29%	5.12%	16.96%	16.48%	192.19
	GRU	C1	18.32%	3.79%	1.00%	0.27%	5.30%	18.54%	5.36%	16.43%	16.06%	195.91
		C2	22.03%	4.29%	1.20%	0.35%	7.12%	21.98%	6.12%	19.17%	17.77%	185.32
	BiGRU	C3	19.21%	4.22%	1.20%	0.34%	4.84%	19.74%	5.97%	17.49%	16.40%	187.87
		C4	33.53%	6.29%	1.74%	0.51%	7.63%	27.95%	7.13%	22.56%	21.77%	132.53
IV3	LSTM	C1	33.47%	7.69%	2.38%	0.62%	8.40%	28.11%	9.17%	24.61%	20.81%	120.05
		C2	34.03%	8.80%	2.90%	0.85%	9.75%	28.76%	9.84%	24.72%	23.00%	122.00
	BiLSTM	C3	38.30%	9.60%	3.32%	0.95%	9.76%	30.35%	10.79%	26.17%	22.62%	101.79
		C4	36.04%	9.37%	2.88%	0.80%	8.64%	30.49%	10.68%	25.38%	24.00%	123.23
	GRU	C1	37.05%	8.01%	2.40%	0.63%	8.46%	29.54%	9.39%	25.36%	21.11%	106.36
		C2	37.80%	9.16%	2.87%	0.80%	8.90%	30.55%	9.88%	25.43%	23.21%	114.50
	BiGRU	C3	38.93%	10.04%	3.19%	0.90%	9.27%	30.86%	10.74%	26.01%	23.11%	105.26
		C4	37.31%	8.56%	2.61%	0.70%	8.38%	29.77%	9.48%	25.36%	21.96%	106.75
IRNV2	LSTM	C1	32.06%	8.44%	2.53%	0.6%	9.31%	27.73%	9.51%	21.53%	24.15%	162.60
		C2	40.18%	7.40%	2.40%	0.62%	8.23%	31.04%	8.82%	25.49%	22.18%	103.62
	BiLSTM	C3	23.02%	4.73%	1.30%	0.34%	6.18%	21.22%	5.48%	18.35%	18.43%	182.72
		C4	37.52%	6.14%	1.99%	0.48%	6.62%	28.46%	6.48%	23.01%	21.45%	112.80
	GRU	C1	39.09%	7.41%	2.14%	0.60%	7.27%	29.43%	7.40%	24.35%	21.16%	99.50
		C2	13.68%	2.94%	0.86%	0.22%	3.08%	16.74%	5.41%	14.71%	13.39%	200.36
	BiGRU	C3	22.55%	6.04%	1.98%	0.49%	6.69%	23.36%	8.37%	20.92%	18.55%	179.96
		C4	36.36%	6.53%	1.67%	0.49%	6.19%	29.54%	8.52%	23.55%	18.51%	100.76
EN4	LSTM	C1	47.20%	18.97%	8.01%	3.07%	41.23%	37.75%	18.73%	34.06%	31.50%	92.91
		C2	47.32%	19.01%	8.06%	3.14%	42.39%	37.91%	18.92%	34.22%	31.70%	92.10
	BiLSTM	C3	49.87%	19.48%	8.32%	3.30%	39.38%	38.22%	18.79%	34.37%	31.37%	85.72
		C4	49.15%	19.04%	7.80%	2.95%	38.84%	38.05%	18.53%	34.02%	31.42%	88.15
	GRU	C1	48.76%	19.24%	8.03%	3.14%	40.18%	38.31%	18.92%	34.50%	31.24%	87.09
		C2	49.20%	19.79%	8.62%	3.51%	41.70%	38.54%	19.25%	34.66%	31.93%	88.39
	BiGRU	C3	49.17%	19.16%	8.06%	3.18%	39.07%	37.97%	18.73%	34.21%	31.40%	86.72
		C4	48.99%	19.15%	8.02%	3.15%	39.31%	38.25%	18.86%	34.27%	31.64%	88.90
EN7	LSTM	C1	44.00%	17.23%	7.23%	2.80%	42.01%	37.36%	18.36%	33.48%	31.85%	104.22

Pe baza acestor rezultate (tabela 3) putem observa câteva lucruri. Deși rețelele convoluționale ResNet50V2 și InceptionResNetV2 sunt mai performante pe task-ul de clasificare decât ResNet50 și respectiv InceptionV3, acestea au avut diferențe negative și rezultate instabile în procesul învățării prin transfer.

Rețeaua convoluțională cea mai performantă atât în task-ul de clasificare cât și în experimentele propuse este EfficientNet B4. Aceasta deține cele mai bune scoruri în cadrul tuturor metricilor.

Din punct de vedere al decodului RNN, în teorie LSTM este mai performant din punct de vedere al acuratetei iar GRU este mai eficient ca timp. Rezultatele însă nu ne ajută să tragem aceeași concluzie. Dacă vrem să comparăm rezultatele dintre același CNN și aceeași metodă de îmbinare, observăm că LSTM și GRU sunt la egalitate. Pe partea de antrenare, diferența de timp nu este semnificativă.

Ca și metodă de îmbinare, concatenarea pare să dea deseori rezultate mai bune. Există și cazuri în care rezultatele sunt complet opuse: IRNV2 cu GRU, RN50V2 cu LSTM.

Pentru mecanismul de atenție, iarăși avem un comportament similar atât pentru tipul Badhau cat și pentru tipul Luong.

Cele mai bune rezultate, având cel mai mare scor în 7 din 10 metrice este modelul cu encodori EfficientNetB4 și decodori GRU cu metoda de concatenare.

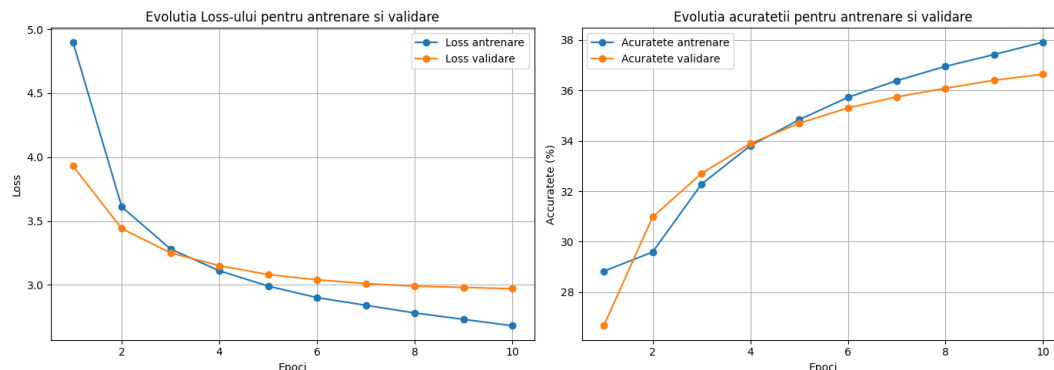


Figura 21: Metrice din timpul de antrenare și validare pentru arhitectura EfficientNetB4 + GRU + Concatenare

Metricele înregistrate, respectiv acuratetea și loss-ul pentru unul dintre modelele antrenate (figura 21) ne arată că acest model nu întâmpină probleme de overfit, însă începe să atingă un platou privind valorile de validare.

5.1.2 Rezultate obținute utilizând arhitectura ViT-Encoder RNN-Decoder

Prin aceeași metodologie și hiperparametrii vom testa și transformerul vizual ViT.

Tabela 4:
Rezultatele experimentului cu ViT-Encoder RNN-Decoder

Model			Bleu				Cider	Rouge			Meteor	Ter
			B-1	B-2	B-3	B-4		R-1	R-2	R-L		
ViT	LSTM	C1	37.63%	9.52%	3.28%	1.03%	11.69%	30.49%	10.34%	25.83%	23.53%	113.84
		C2	36.07%	9.11%	3.11%	0.92%	10.96%	29.84%	10.69%	26.12%	21.48%	104.48
	BiLSTM	C3	37.51%	9.08%	3.12%	0.97%	10.82%	30.02%	9.92%	27.75%	22.72%	105.38
		C4	37.43%	9.16%	2.99%	0.87%	10.04%	30.46%	10.35%	25.93%	22.62%	105.30
	GRU	C1	37.81%	9.31%	3.11%	0.93%	11.15%	30.63%	10.24%	25.97%	23.56%	112.49
		C2	37.72%	9.63%	3.36%	1.01%	11.36%	29.88%	10.17%	25.74%	22.39%	104.98
	BiGRU	C3	37.73%	9.22%	3.11%	0.94%	10.63%	29.97%	10.00%	25.77%	22.14%	100.40
		C4	37.31%	9.31%	3.28%	1.11%	11.81%	30.51%	10.36%	25.83%	23.52%	112.297

Deși transformer-ul ViTb16 este printre cele mai bune în task-urile de clasificare, inclusiv mai bun decât CNN-urile, nu a reușit să depășească scorurile anterioare (tabela 4). De asemenea, scorurile sale sunt împărțite, nu putem identifica o "rețetă" pentru cea mai bună combinație. Metricile din figura 22 ne arată că valoarea funcției de loss începe să atingă un prag de minim local.

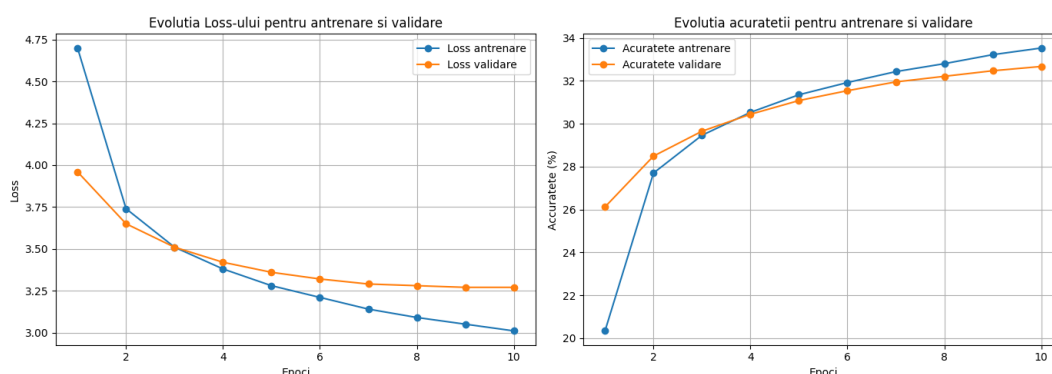


Figura 22: Metrici din timpul de antrenare și validare pentru arhitectura ViTb16 + GRU + Concatenare

5.1.3 Rezultate obținute utilizând Embeddings preantrenate

Mai departe vom încerca să aplicăm diferite embeddings pe una dintre rețelele construite cu ajutorul EfficientNetB4. Aceste embeddings nu mai sunt antrenate ulterior, iar avantajul lor ar fi că avem un plus de performanță din start și sunt approximate din punct de vedere statistic mult mai bine deoarece au fost antrenate pe seturi de date mai largi și pe perioadă îndelungată. Hiperparametrii sunt în continuare aceași că la primul experiment.

Am reantrenat modelul de bază cu EfficientNet. Pentru acesta am mărit pragul de frecvență al cuvintelor din vocabular, la un număr de minim 5 apariții. Contrar așteptărilor, cele mai bune rezultate sunt obținute tot cu modelul original (tabela 5). Un posibil motiv este faptul că putem antrena aceste embeddings mult mai bine pe task-ul de image captioning și setul de date propus, obținând rezultate mai bune decât cu cele preantrenate. Embeddings generate cu

Tabela 5:

Modelul format din rețeaua convolutionala EfficientNet4, LSTM și Adunare că metodă de îmbinare, utilizând Embeddings preantrenate prin diverși algoritmi și seturi de date.

Embeddings	Dimensiune	Bleu				Cider	Rouge			Meteor	Ter
		B-1	B-2	B-3	B-4		R-1	R-2	R-L		
Base	128	49.58%	20.02%	8.49%	3.30%	41.70%	38.69%	19.24%	35.01%	31.61%	85.53
NLPL - Word2Vec	100	33.08%	12.62%	5.01%	1.86%	34.51%	32.28%	16.05%	29.67%	26.70%	116.28
CC - FastText	300	45.31%	17.59%	7.01%	2.58%	40.63%	37.31%	18.22%	33.57%	31.04%	96.31
CoRoLa - FastText	300	45.69%	17.86%	7.35%	2.78%	41.48%	37.80%	18.78%	34.14%	31.37%	94.98
RoBERT-large	1024	26.64%	0.89%	0.01%	0.0%	8.09%	19.36%	2.71%	16.87%	8.28%	72.15

ajutorul transformerului RoBERT au avut un rezultat foarte slab. Acest lucru poate semnifica o nepotrivire a arhitecturii BERT în cadrul experimentului. Este posibil să avem nevoie de un transformer generativ care produce embeddings. De asemenea, nu putem exclude și o utilizare greșită a acestuia.

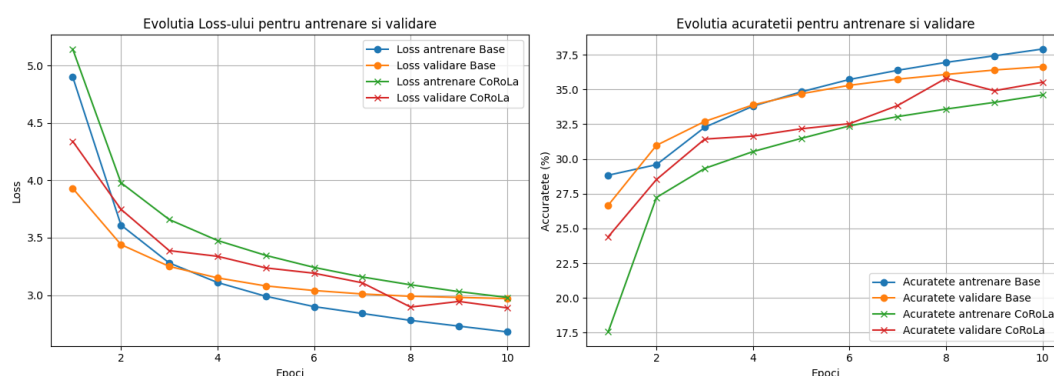


Figura 23: Metrici din timpul de antrenare și validare pentru embeddings Base și embeddings CoRoLa

5.1.4 Rezultate obținute utilizând arhitectura Transformer-Encoder Transformer-Decoder

Următoarele experimente sunt cele cu arhitectura transformer. Pentru aceste experimente am ales următorii hiperparametrii:

- Număr de epoci - 10
- Rata de învățare - 0.0001 ($1 * 10^{-4}$)
- Dimensiune batch - 64
- Dropout encodor - 0.0
- Dropout decodor - 0.1, 0.3, 0.5 (pentru stratul de mulți head attention, respectiv pentru straturile MLP)
- Dimensiune Embeddings - 512
- Dimensiune FFN - 512
- Capete de atenție - 6 (atât pentru encodor cât și pentru decodor)

Tabela 6:

Rezultatele experimentelor cu arhitectura CNN-Transformer-Encoder Transformer-Decoder

CNN	Bleu				Cider	Rouge			Meteor	Ter
	B-1	B-2	B-3	B-4		R-1	R-2	R-L		
ResNet50	33.18%	5.59%	1.39%	0.36%	5.37%	25.47%	5.43%	21.08%	18.24%	121.45
ResNet50V2	40.33%	4.45%	0.69%	0.14%	2.40%	27.64%	4.34%	23.70%	18.28%	91.88
InceptionV3	33.51%	3.91%	1.11%	0.24%	4.81%	24.40%	3.75%	22.01%	15.92%	97.72
InceptionResNetV2	36.93%	6.09%	1.67%	0.49%	6.70%	28.76%	6.65%	23.03%	21.62%	117.20
EfficientNetB4	38.78%	8.66%	2.30%	0.69%	7.22%	27.29%	7.61%	24.07%	19.18%	85.98

Tabela 7:

Rezultatele experimentelor cu modelul ViT-Transformer-Encoder Transformer-Decoder

CNN	Bleu				Cider	Rouge			Meteor	Ter
	B-1	B-2	B-3	B-4		R-1	R-2	R-L		
ViT	27.80%	3.68%	0.79%	0.12%	3.68%	22.65%	3.88%	19.65%	16.27%	133.07

Vocabularul este același ca în primul experiment.

În continuare putem observa că rețeaua EfficientNetB4 dă cele mai bune rezultate (tabela 6). Însă arhitectura transformer nu a depășit arhitectura CNN-RNN. Acest lucru poate însemna că nu a fost antrenat suficient, are o convergență mai lentă sau nu au fost aleși bine hiperparametrii. Încă o dată, valoarea de loss pentru validare (figura 24) atinge un prag ce nu poate

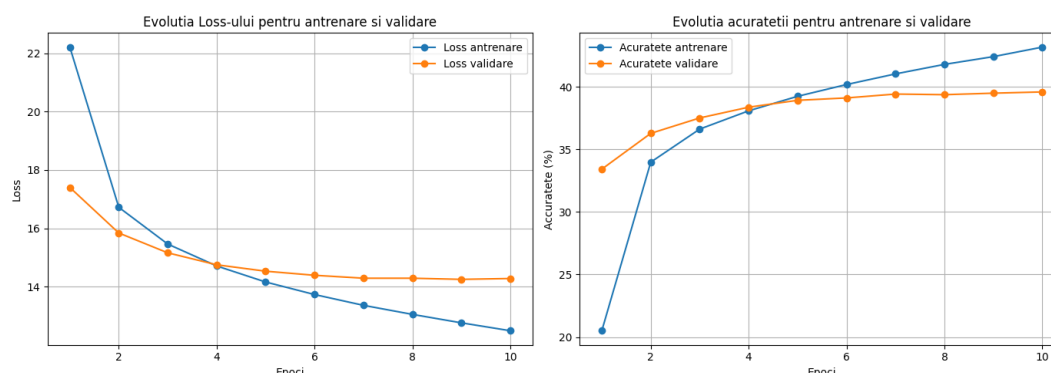


Figura 24: Metrice din timpul de antrenare și validare pentru arhitectura tip transformer cu EfficientNetB4

fi depășit.

Rezultate (tabela 7) slabe a avut și transformerul vizual ViT în raport cu arhitectura transformer.

5.1.5 Rezultate obținute utilizând tehnica Beam Search pentru generarea descrierilor

Ca și ultim experiment, vom compara metodele de generare ale descrierilor: Greedy și Beam Search. În esență, dacă aplicăm beam search cu un singur cap de căutare, obținem același

Tabela 8:

Modelul format din rețeaua convoluțională EfficientNet4, GRU și Concatenare folosind Beam Search pentru generarea descrierilor

Beam Search width	Bleu				Cider	Rouge			Meteor	Ter
	B-1	B-2	B-3	B-4		R-1	R-2	R-L		
k=1 (base)	49.58%	20.02%	8.49%	3.30%	41.70%	38.69%	19.24%	35.01%	31.61%	85.53
k=3	53.60%	23.07%	10.24%	4.51%	47.23%	40.56%	21.26%	37.04%	32.21%	74.73

algoritm ca la greedy. Putem observa (tabela 8) că prin metoda beam search cu 3 capete, descrierile generate au fost mai apropiată de realitate, obținând scoruri cele mai bune de până acum.

Cumulativ, antrenarea modelelor a durat aproximativ 150 de ore. În medie un singur model pentru antrenare și evaluare a durat aproximativ 2 ore, excepție făcând modelul cu transformer BERT care a durat aproximativ 30 de ore.

5.2 Rezultate calitative

Putem observa și o comparație a descrierilor generate de modele (tabela 9). Arhitectura tip CNN-RNN reușește să producă descrieri bune, coerente, care au legătură cu imaginile și propozițiile de referință. Pe de altă parte, VIT-RNN și Transformerii produc descrieri greșite în raport cu referințele.

O comparație a modalităților de construire ale descrierilor, respectiv Greedy și Beam Search (tabela 10), ne arată că putem obține descrieri și mai bune prin metoda Beam Search. O îmbunătățire clară au descrierile pozelor 3 și 4.

Tabela 9: Comparatie a descrierilor generate de modelele EfficientNetB4 + GRU + Concatenare (Model 1), ViT + GRU + Concatenare (Model 2) si EfficientNetB4 cuplat cu ahitectura Transformer (Model 3)


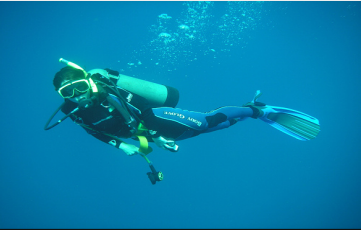




		
<p>Referinta: Un bărbat într-o cămașă galbenă cu o geantă înaltă gri stă lângă un alt bărbat într-o cămașă albă și pantaloni gri</p> <p>Model 1: O femeie cu o jachetă roz și o jachetă albastră stă pe o bancă de ciment</p> <p>Model 2: Un om care poartă o cămașă albastră și pantaloni scurți negri se plimbă pe o bicicletă</p> <p>Model 3: Un om în picioare în fața unui perete acoperit cu graffiti</p>	<p>Referinta: O persoană care poartă echipament de scuba înoată sub apă</p> <p>Model 1: Un scafandru înoată în apă</p> <p>Model 2: Un scafandru înoată în apă</p> <p>Model 3: Un om care poartă o pălărie de baseball se plimbă pe o pistă de pământ</p>	<p>Referinta: O tânără în costum de baie cu dungi sare în ocean</p> <p>Model 1: O fată tânără se plimbă pe plajă</p> <p>Model 2: Un om care poartă o cămașă albastră și pantaloni scurți negri se plimbă pe o plajă</p> <p>Model 3: Un om care se plimbă pe un teren de noapte</p>
		
<p>Referinta: Un câine maro care poartă o vestă roșie aleargă prin iarbă</p> <p>Model 1: Un câine maro și alb se plimbă pe iarbă</p> <p>Model 2: Un jucător de baseball în uniformă roșie și albă se pregătește să arunce mingea</p> <p>Model 3: Un om în apă</p>	<p>Referinta: Oamenii se distrează pe stradă și râd cu un ofițer de securitate</p> <p>Model 1: Un grup de oameni se plimbă pe stradă</p> <p>Model 2: Un bărbat în cămașă albastră și pantaloni scurți negri se plimbă pe o stradă aglomerată</p> <p>Model 3: Un om în apă</p>	<p>Referinta: Un biciclist de munte care poartă o cască colorată coboară cu bicicleta</p> <p>Model 1: Un motociclist de munte care merge cu bicicleta pe o potecă</p> <p>Model 2: Un bărbat în cămașă albastră și pantaloni scurți negri stă pe o bancă cu bicicleta în fața unei clădiri</p> <p>Model 3: Un om în cămașă neagră și pantaloni scurți albi se plimbă pe un trotuar</p>

Tabela 10: Comparație a descrierilor generate de modelul EfficientNetB4 + GRU + Concatenare, prin mijloace diferite de construire ale descrierilor: Greedy (Model 1) și Beam Search cu 3 capete (Model2)

		
<p>Model 1: O femeie cu o jachetă roz și o jachetă albastră stă pe o bancă de ciment</p> <p>Model 2: O femeie cu ochelari de soare și ochelari de soare stă în fața unei clădiri de cărămidă</p>	<p>Model 1: Un scafandru înoată în apă</p> <p>Model 2: Un scafandru înoată în apă</p>	<p>Model 1: O fată tânără se plimbă pe plajă</p> <p>Model 2: O fată în costum de baie sare în apă</p>
		
<p>Model 1: Un câine maro și alb se plimbă pe iarbă</p> <p>Model 2: Un câine maro și alb aleargă pe iarbă</p>	<p>Model 1: Un grup de oameni se plimbă pe stradă</p> <p>Model 2: Un grup de oameni se plimbă pe stradă</p>	<p>Model 1: Un motociclist de munte care merge cu bicicleta pe o potecă</p> <p>Model 2: Un om care merge cu bicicleta prin pădure</p>

6 CONCLUZII ȘI DIRECȚII VIITOARE

Rezultatele obținute sunt departe de cele state-of-the-art. Factori principali sunt arhitecturile mai slabe și cantitatea de informație restrânsă la care am avut acces, alături de timpii de antrenare scăzuți. De asemenea, fiind un set de date nou propus, trebuie explorat ce funcționează cel mai bine pe acesta. Una dintre problemele întâmpinate este și accentul pe anumite cuvinte care sunt foarte comune în setul de date însă nu reflectă neapărat multă informație: articole vestimentare și culori.

Pe viitor ar trebui găsit un mecanism care să încerce să ia mai puțin în considerare cuvintele cele mai comune și să învețe mai mult din cuvintele rare sau cu un grad de apariție normal. Trebuie realizate experimente și cu alte tehnologii și arhitecturi care au dovedit că sunt bune pentru acest task. În special, ar trebui folosiți și transformeri generativi sau LLM-uri sau transformeri încrucișați pentru a îmbunătății performanța. De asemenea, este necesară obținerea mai multor seturi de date, de dimensiuni și mai mari pentru a putea avea o șansă adevărată în obținerea unor rezultate bune. Pentru asta este posibil să fie necesar un tradeoff, respectiv lipsa corectării manuale, lucru care poate să introducă ușor greșeli gramaticale. Pentru problema respectivă, putem găsi alte soluții la momentul respectiv.

BIBLIOGRAFIE

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from [tensorflow.org](https://www.tensorflow.org).
- [2] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. In *International conference on machine learning*, pages 584–592. PMLR, 2014.
- [3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [4] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005.
- [5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.
- [6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [7] François Chollet et al. Keras. <https://keras.io>, 2015.
- [8] Dan Cristea, Nils Diewald, Gabriela Haja, Cătălina Măranduc, Verginica Barbu Mititelu, and Mihaela Onofrei. How to find a shining needle in the haystack. querying corola: solutions and perspectives. 2019.
- [9] Ringki Das and Thoudam Doren Singh. Assamese news image caption generation using attention mechanism. *Multimedia Tools and Applications*, 81(7):10051–10069, 2022.

- [10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [12] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [13] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. *arXiv preprint arXiv:1802.06893*, 2018.
- [14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016.
- [16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [17] Jia Cheng Hu, Roberto Cavicchioli, and Alessandro Capotondi. Exploiting multiple sequence lengths in fast end to end training for image captioning. In *2023 IEEE International Conference on Big Data (BigData)*, pages 2173–2182. IEEE, 2023.
- [18] Xiaowei Hu, Zhe Gan, Jianfeng Wang, Zhengyuan Yang, Zicheng Liu, Yumao Lu, and Lijuan Wang. Scaling up vision-language pre-training for image captioning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 17980–17989, 2022.
- [19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [20] Andrei Kutuzov, Murhaf Fares, Stephan Oepen, and Erik Velldal. Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *Proceedings of the 58th Conference on Simulation and Modelling*, pages 271–276. Linköping University Electronic Press, 2017.

- [21] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [22] Chenliang Li, Haiyang Xu, Junfeng Tian, Wei Wang, Ming Yan, Bin Bi, Jiabo Ye, Hehong Chen, Guohai Xu, Zheng Cao, et al. mplug: Effective and efficient vision-language learning by cross-modal skip-connections. *arXiv preprint arXiv:2205.12005*, 2022.
- [23] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023.
- [24] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.
- [25] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.
- [26] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.
- [27] Mihai Masala, Stefan Ruseti, and Mihai Dascalu. Robert—a romanian bert model. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6626–6637, 2020.
- [28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [29] Santosh Kumar Mishra, Mahesh Babu Peethala, Sriparna Saha, and Pushpak Bhattacharyya. An information multiplexed encoder-decoder network for image captioning in hindi. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3019–3024. IEEE, 2021.
- [30] Santosh Kumar Mishra, Gaurav Rai, Sriparna Saha, and Pushpak Bhattacharyya. Efficient channel attention based encoder–decoder approach for image captioning in hindi. *Transactions on Asian and Low-Resource Language Information Processing*, 21(3):1–17, 2021.
- [31] Santosh Kumar Mishra, Sriparna Saha, and Pushpak Bhattacharyya. A scaled encoder decoder network for image captioning in hindi. In *Proceedings of the 18th international conference on natural language processing (ICON)*, pages 251–260, 2021.
- [32] Verginica Barbu Mititelu, Dan Tufiş, and Elena Irimia. The reference corpus of the contemporary romanian language (corola). In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

- [33] Thanh Tin Nguyen, Long H Nguyen, Nhat Truong Pham, Liu Tai Nguyen, Van Huong Do, Hai Nguyen, and Ngoc Duy Nguyen. viecap4h-vlsp 2021: Vietnamese image captioning for healthcare domain using swin transformer and attention-based lstm. *arXiv preprint arXiv:2209.01304*, 2022.
- [34] Van-Quang Nguyen, Masanori Suganuma, and Takayuki Okatani. Grit: Faster and better image captioning transformer using dual visual features. In *European Conference on Computer Vision*, pages 167–184. Springer, 2022.
- [35] Emilio Soria Olivas, Jos David Mart Guerrero, Marcelino Martinez-Sober, Jose Rafael Magdalena-Benedito, L Serrano, et al. *Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques: Algorithms, methods, and techniques*. IGI global, 2009.
- [36] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.
- [37] Mathew Snover, Bonnie Dorr, Richard Schwartz, John Makhoul, Linnea Micciulla, and Ralph Weischedel. A study of translation error rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA 06)*, pages 223–231, 2005.
- [38] Bipesh Subedi and Bal Krishna Bal. Cnn-transformer based encoder-decoder model for nepali image captioning. In *Proceedings of the 19th International Conference on Natural Language Processing (ICON)*, pages 86–91, 2022.
- [39] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.
- [40] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [41] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.
- [42] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2820–2828, 2019.

- [43] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.
- [44] NLLB Team, Marta R Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, et al. No language left behind: Scaling human-centered machine translation (2022). URL <https://arxiv.org/abs/2207.04672>, 2022.
- [45] Dan Tufiş, Verginica Barbu Mititelu, Elena Irimia, Vasile Păiş, Radu Ion, Nils Diewald, Maria Mitrofan, and Mihaela Onofrei. Little strokes fell great oaks: creating corola, the reference corpus of contemporary romanian. 2019.
- [46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [47] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575, 2015.
- [48] Leandro Von Werra, Lewis Tunstall, Abhishek Thakur, Sasha Luccioni, Tristan Thrush, Aleksandra Piktus, Felix Marty, Nazneen Rajani, Victor Mustar, and Helen Ngo. Evaluate & evaluation on the hub: Better best practices for data and model measurements. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 128–136, 2022.
- [49] W3Techs.com. Historical quarterly trends in the usage statistics of content languages for websites, 2024. https://w3techs.com/technologies/history_overview/content_language/ms/y [Accesat: 27.05.2024].
- [50] Jianfeng Wang, Zhengyuan Yang, Xiaowei Hu, Linjie Li, Kevin Lin, Zhe Gan, Zicheng Liu, Ce Liu, and Lijuan Wang. Git: A generative image-to-text transformer for vision and language. *arXiv preprint arXiv:2205.14100*, 2022.
- [51] Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International Conference on Machine Learning*, pages 23318–23340. PMLR, 2022.
- [52] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface’s transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

- [53] Hao Yang, Junyang Lin, An Yang, Peng Wang, Chang Zhou, and Hongxia Yang. Prompt tuning for generative multimodal pretrained models. *arXiv preprint arXiv:2208.02532*, 2022.
- [54] Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2:67–78, 2014.