Universitatea “Politehnica ” din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Descrierea conținutului vizual pe terminalul mobil

Lucrare de disertație

prezentată ca cerință parțială pentru obținerea titlului de
Master în domeniul Imagisticii Digitale
programul de studii de Tehnici Avansate in Imagistica Digitală

Conducători științifici
Prof. dr. Ing. Corneliu FLOREA
As. drd. Ing. Badea Mihai

Absolvent
George-Iulian NITROI

Anul 2020

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Programul de masterat **Tehnici avansate pentru imagistica digitală**

Anexa 2

TEMA LUCRĂRII DE DISERTAŢIE
a masterandului **NITROI E. George-Iulian , 421-TAID.**

1. Titlul temei: Descrierea conținutului vizual pe terminalul mobil

2. Descrierea contribuţiei originale a masterandului (în afara părţii de documentare) şi specificaţii de proiectare:
Tema își propune realizarea unei aplicații pentru terminalul mobil capabilă să descrie imaginile provenite de la acesta. Se propune realizarea unui prototip funcțional si contribuțiile în 2 categorii de analiza a datelor si respectiv de implementare. Pentru analiza datelor se urmareste: implementarea unei rețele convoluționale care identifică obiectele din imagine și a unei rețele neuronale recurente(decoder) care preia obiectele identificate de prima rețea și generează o descriere pentru poza respectivă .
Din punct de vedere aplicativ se urmareste crearea unei aplicații utilizând Android studio capabilă să facă o poză sau să ia o poză din galeria foto a terminalului mobil, crearea unui server utilizând Firebase care să asigure comunicația intre terminalul mobil și calculator, respectiv afișare rezultatului utilizatorului.

3. Resurse folosite la dezvoltarea proiectului:
Python, Tensorflow, Android Studio, Java, XML, Javascript, Firebase

4. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline:
MLAV,PAIC,ASTM

5. Proprietatea intelectuală asupra proiectului aparţine: U.P.B.

6. Data înregistrării temei: 2019-11-13 10:51:54

Conducător(i) lucrare,
Prof. dr. ing. Corneliu FLOREA

semnătura: .....

As. drd. ing Badea Mihai

semnătura: .....

Student,

semnătura:.....

Responsabil program master,
Prof. dr. ing. Constantin VERTAN

semnătura:.....

Decan,
Prof. dr. ing. Cristian NEGRESCU

semnătura:.....

Cod Validare:

DECLARAȚIE DE ORIGINALITATE A PROIECTULUI DE DIPLOMĂ/LUCRĂRII DE DISERTAȚIE[1)]

Subsemnatul NITROI E. GEORGE-IULIAN [2)], posesor al B.I./C.I./pașaport seria RR, nr. 427859, având CNP 1950718017009 2.

am întocmit proiectul de ~~absolvire/diplomă~~/lucrarea de disertație cu tema:[3)] DESCRIEREA CONȚINUTULUI VIZUAL PE TERMINALUL MOBIL

în vederea susținerii examenului de finalizare a studiilor universitare de licență/masterat, organizat de către Facultatea de ELECTRONICĂ TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI Departamentul ELECTRONICĂ APLICATĂ ȘI INGINERIA INFORMAȚIEI din cadrul Universității POLITEHNICA din București, sesiunea IUNIE, anul universitar 2020.

Luând în considerare conținutul art. 143 din Legea Educației Naționale nr. 1/2011, al. (4) *„Îndrumătorii proiectelor de diplomă/lucrărilor de disertație răspund în solidar cu autorii acestora de asigurarea originalității conținutului acestora"* și al. (5) *„Este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unui/unei proiect de diplomă/lucrări de disertație"*, declar pe propria răspundere, că acest/această proiect/lucrare este original(ă), fiind rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației în vigoare.

Cunosc faptul că plagiatul sau prezentarea unui/unei proiect/lucrări, elaborat(ă) de alt absolvent sau preluată de pe internet, din manuale și cărți, fără precizarea sursei constituie infracțiune (furt intelectual și nerespectarea dreptului de autor și a proprietății intelectuale) și atrage după sine anularea examenului de diplomă/disertație, precum și răspunderea penală.

Data: 07.06.2020 Semnătura: _____
(în original)

[1)] Declarația se completează *„de mână"* și reprezintă un document care se depune la înscrierea pentru susținerea examenului de finalizare a studiilor. Ea nu se include în proiect – vezi Anexa 5 în acest scop.
[2)] Numele, inițiala prenumelui tatălui și prenumele, cu majuscule.
[3)] Denumirea proiectului de diplomă/lucrării de disertație, cu majuscule.

Cuprins

Listă de figuri

Listă de Tabele

Listă de acronime

AI: Artificial Intelligence(Inteligența artificială)

ML: Machine Learning

UI: User interface(Interfața grafică)

Cuda: Compute Unified Device Achitecture

BaaS: Backend-as-a-Service

API: Application Programming Interface (colecția de clase/funcții/interfețe cu ajutorul cărora exploatăm funcționalitatea unui framework sau a unei biblioteci)

RNN: Recurent Neural Network(Rețea neuronală recurentă)

GRU: Gated Recurrent Units

LSTM: Long-short term memory

REST: Representational state transfer

ReLU: Rectified linear unit

IDE: Integrated development environment

AVD: Android Virtual Device

HTTP: Hypertext Transfer Protocol

Introducere

În zilele noastre dezvoltarea tehnologiei este mai rapidă iar acest lucru se poate observa mai ales în domeniul științei calculatoarelor. Inteligența artificială(AI) este la momentul de fața unul din vârfurile de lance în acest domeniu oferind o perspectivă diferită asupra rezolvării unor probleme care în programarea clasică s-au dovedit de-a lungul timpului dificil de abordat. Deși nu este un domeniu nou, el având bazele la începutul anilor 50 când metodele statistice care stau la baza algoritmilor de azi sunt găsite și perfecționate, implementarea lor și utilizarea acestor algoritmi in mediile de producție s-a dovedit dificilă până la începutul anilor 2000.

Apariția internetului și accesul la acesta pentru publicul larg la sfârșitul secolului 20 a fost unul din principalii factori ce au dus la reducerea în prim plan al inteligenței artificiale. Internetul rezolvă problema insuficienței datelor cu care cercetătorii în domeniu s-au lovit până în acel punct, facilitând crearea unor baze de date care ulterior să poată servi antrenării diferitelor modele matematice disponibile în acel moment.

Un alt lucru ce a împiedicat multă vreme ca acești algoritmi sa fie fezabili a fost lipsa puterii de procesare. A fost nevoie de zeci de ani pentru ca industria semiconductoarelor să ajungă la nivelul în care aceste modele să poate fi utilizate pe probleme complexe, cu un număr considerabil de clase, în scenarii utile aplicațiilor noastre de zi cu zi. În istoria recentă, Nvidia face parte din ceea ce se poate numi “Big bang-ul” rețelelor neuronale adânci, când în 2009 un astfel de model este antrenat pe o placă video sporind viteza de procesare a acestora de aproximativ 100 de ori.

În 2020 putem găsi componente AI în majoritatea programelor/aplicațiilor pe care noi le folosim de zi cu zi. Proiectul curent își propune realizarea un aplicații pe terminalul mobil care să folosească toate avansurile tehnologice menționate mai sus. Folosind un terminal mobil cu Android aplicația va fi capabilă sa facă poze sau să folosească poze deja existente în galeria telefonului cărora ulterior sa le adauge o descriere generată de un model de ML(machine learning). Aplicația va fi una te tip server-client unde, terminalul mobil(smatphone-ul) se va comporta ca un client pentru un server ce așteaptă să primească o poză de la telefon și ii va răspunde cu ieșirea modelului, o propoziție ce descrie acea poză.

Obiectivele pe care și le propune acest proiect sunt următoarele:

• Dezvoltarea unei aplicații Android

• Implementarea topologiei server client intre terminalele mobile și un server cu o putere de calcul îndeajuns de mare încât să ruleze modelul.

• Antrenarea modelului de rețea neuronală cară sa facă descrierea imaginilor

• Testarea aplicației

Capitol 1 Noțiuni teoretice necesare dezvoltării practice

1.1 Topologia aplicației

Aplicația noastră are 3 componente software:

- O aplicație dezvoltată în Android Studio ce va servi ca front-end pentru aplicația noastră și va facilita utilizatorului un meniu intuitiv și ușor de folosit pentru a profita de facilitățile oferite.
- O componenta Firebase ce va servi ca back-end și pe care noi ne vom stoca modelul antrenat pentru a genera descrierea imaginilor.
- Un model de rețea neuronală adâncă dezvoltat in Python cu ajutorul librăriei Tensorflow.

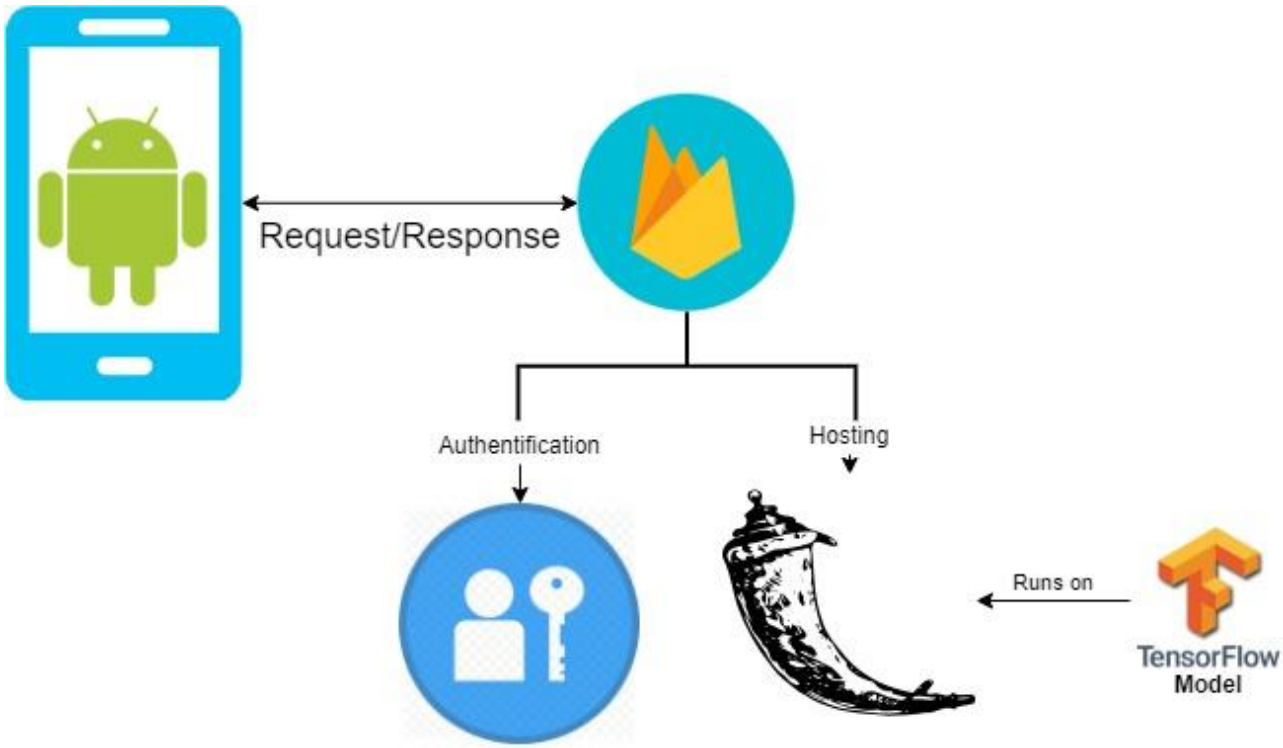


Figura 1-1 Topologie aplicație

Modelul obținut de noi în urma antrenării va fi prea mare pentru a putea fi stocat pe terminalul mobil. De asemenea nu orice terminal mobil are posibilitatea de a rula modele complexe pe arhitectura lor hardware așa că pentru a acomoda aplicația pe cat mai multe terminale mobile s-a luat decizia asupra acestei arhitecturi.

1.2 Arhitectură server-client

Modelul client-server este o structură de aplicație care distribuie sarcini (sau sarcini de lucru) între furnizorii unei resurse sau servicii, numiți servere și solicitanți de servicii, numiți clienți. Adesea, clienții și serverele comunică printr-o rețea de calculatoare pe hardware separat, dar atât clientul cât și serverul pot sta în același sistem.

Un server execută unul sau mai multe programe server, care își împărtășesc resursele cu clienții. Un client nu împărtășește nici una dintre resursele sale, dar solicită conținut sau serviciu de la un server. Prin urmare, clienții inițiază sesiuni de comunicare cu servere, care așteaptă solicitările.

Exemple de aplicații care utilizează modelul client-server sunt e-mail-ul, imprimarea în rețea (folosirea imprimantei prin internet) și WWW.

1.3 Terminalul mobil

Aplicația poate fi instalată și va rula pe orice telefon mobil, tabletă sau dispoziiv ce rulează o distribuție de android mai noua decăt Nougat 7.0 și care dispune de o camera foto și de conexiune la internet. Cerințele acestea vor fi minimale întrucât marea parte a puterii de procesare va fi oferită de server. Pentru a testa aplicația noiastra am utilizate 2 terminale mobile: un Samsung Galaxy S9 și un Google Pixel A3(folosit într-un emulator).

1.3.1 Platforma Android

Android este o platformă software și un sistem de operare pentru dispozitive și telefoane mobile bazată pe nucleul Linux, dezvoltată inițial de compania Google, iar mai târziu de consorțiul comercial Open Handset Alliance. Android permite dezvoltatorilor să scrie cod gestionat în limbajul Java și începând cu anul 2017 Kotlin controlând dispozitivul prin intermediul bibliotecilor dezvoltate de Google. Aplicațiile scrise în C și în alte limbaje pot fi compilate în cod mașină ARM și executate, dar acest model de dezvoltare nu este sprijinit oficial de către Google. [1]

Deși Java a fost limbajul de programare care a dominat o buna perioadă de timp dezvoltarea aplicațiilor pe platforma Android, dezvoltatorii au posibilitatea să folosească Kotlin care în mai puțin de 3 ani de la integrarea în platforma Android a ajuns sa fie limbajul oficial preferat de Google pentru Android. Kotlin face ca dezvoltarea aplicațiilor sa fie mai concisă (mai mult cu mai puține linii de cod) și elimină necesitatea dezvoltatorilor să lege fiecare element din UI(user interface) cu codul din spate. Fiind un limbaj nou, el oferă un nivel de abstractizare similar cu cel al JavaScript-ului sau Python-ului. Cu toate acestea el se poate integra ușor cu Java, dezvoltatorul având posibilitatea să folosească in Kotlin instanțe ale claselor scrise în Java și viceversa. [2]

1.3.2 Hardware

În momentul de față pe majoritatea terminalelor mobile pe lângă să asigure o comunicația audio fără fir sunt capabile să facă o multitudine de alte sarcini prin intermediul componentelor adiționale pe care le regăsim în formatul lor compact. Display-urle bazate pe tehnologii LED sunt capabile să redea imagini de calitate la rezoluții înalte și cu o gama largă de culori, procesoarele, memoria și stocarea sunt capabile să susțină sisteme de operare comparabile ca performanțe cu cele regăsite pe calculatoare, modem-urile integrate asigură comunicații de viteză și viteze înalte de transfer pe internet. Camerele foto integrate dispun de caracteristici precum: autofocus, senzor de

lumină, flash, stabilizare de imagine digitală sau mecanică. De asemenea putem regăsi senzori adiționali precum: accelerometru, giroscop, busolă digitală, senzori de proximitate, magnetometru etc.

Terminalele mobile de ultimă generație dispun de acceleratoare grafice cu arhitecturi CUDA, arhitectura utilizată și pe plăcile video regăsite pe calculatoare. Aceste acceleratoare permite rularea unor redarea unor jocuri/aplicații cu înaltă calitate grafică și pun la dispoziție o suită de biblioteci de software de ML accelerată complet pe GPU.

1.4 Firebase

Firebase este un backend ca un serviciu (BaaS). Firebase elimină necesitatea dezvoltatorului de a se concentra pe dezvoltarea și managementul unui server, și pune la dispoziție dezvoltatorului un API generic pentru a ii ușura munca. Printre facilitățile oferite de Firebase noi vom utiliza următoarele:

- Firebase Analytics pentru a obține statistici despre folosirea aplicației
- Firebase Auth pentru a permite utilizatorilor sa se autentifice pe un acont propriu și pentru a le proteja pozele trimise către server
- Firebase Hosting pentru a ține scripturile în Python
- ML Kit pentru a integra modelul antrenat

In momentul de față, Firebase este o soluție populara atât pentru aplicațiile Android si IOS cât și pentru

aplicațiile web oferind atât o interfață stabilă cu backend-ul aplicațiilor cat și metrici relevante legate de utilizarea aplicatiei.

1.5 Python

Python este un limbaj de programare interpretat, la nivel înalt, cu scop general. Creată de Guido van Rossum și lansată pentru prima dată în 1991, filozofia de proiectare a lui Python subliniază lizibilitatea codurilor prin utilizarea spațiilor pentru a delimita diferite structuri. Abordarea orientată spre obiect și multitudinea de funcții si librarii cu care el vine deja instalat face ca Python să fie in ziua de azi unul din limbajele de programare preferat pentru scriptare, proiecte de dimensiuni mici sau back-end în aplicații. Suportă mai multe paradigme de programare, incluzând programarea procedurală, orientată pe obiecte și funcționale

Pe lângă librăriile cu care python vine deja instalat, în cadrul proiectului nostru vom mai utiliza, pe lângă Tensorflow, și alte librarii/pachete precum:

- NumPy: o librărie care oferă suport pentru operațiile cu tensori, și un set de funcții pe care le putem folosi pe aceștia
- Matplotlib: librărie ce permite plotarea datelor in grafice
- OpenCV: o bibliotecă de funcții informatice specializată pe vedere pe care o vom folosi în prelucrarea pozelor
- Pandas: o librărie folosită pentru manipularea datelor

- Flask: un micro-framework pentru dezvoltare WEB

1.5.1 Medii virtuale în Python(Virtual Environments)

Aplicațiile Python vor folosi adesea pachete și module care nu fac parte din biblioteca standard. Aplicațiile vor avea uneori nevoie de o versiune specifică a unei biblioteci, deoarece aplicația poate necesita o rezolvare a unui anumit bug sau aplicația poate fi scrisă folosind o versiune învechită a interfeței bibliotecii.

Aceasta înseamnă că este posibil să nu fie posibilă o instalare Python să îndeplinească cerințele fiecărei aplicații. Dacă aplicația A are nevoie de versiunea 1.0 a unui anumit modul, dar aplicația B are nevoie de versiunea 2.0, atunci cerințele sunt în conflict și instalarea fiecărei versiuni 1.0 sau 2.0 va lăsa o aplicație să nu poată fi executată.

Soluția pentru această problemă este crearea unui mediu virtual, un arbore de directoare cu conținut independent care conține o instalare Python pentru o anumită versiune a Python, plus un număr de pachete suplimentare.

Aplicații diferite pot utiliza apoi diferite medii virtuale. Pentru a rezolva exemplul anterior de cerințe contradictorii, aplicația A poate avea propriul său mediu virtual cu versiunea 1.0 instalată în timp ce aplicația B are un alt mediu virtual cu versiunea 2.0. Dacă aplicația B necesită o bibliotecă modernizată la versiunea 3.0, aceasta nu va afecta mediul aplicației A.

1.5.2 Anaconda

Anaconda este o distribuție gratuită și open-source a limbajelor de programare Python și R pentru calcul științific (manipularea datelor, aplicații de învățare automată(ML), prelucrare a datelor la scară largă, analize predictive etc.), care urmărește simplificarea gestionării pachetelor și implementare. Distribuția include pachete științifice de date adecvate pentru Windows, Linux și macOS. Este dezvoltat și întreținut de Anaconda, Inc., care a fost fondată de Peter Wang și Travis Oliphant în 2012. Ca produs Anaconda, Inc., este cunoscut și sub denumirea de Anaconda Distribution sau Anaconda Individual Edition, în timp ce alte produse ale companiei sunt Anaconda Team Edition și Anaconda Enterprise Edition, care nu sunt gratuite.

Versiunile de pachete din Anaconda sunt gestionate de sistemul de gestionare a pachetelor conda. Acest manager de pachete a fost distribuit ca un pachet separat open-source, deoarece a sfârșit să fie util pe cont propriu și pentru alte lucruri decât Python. Există, de asemenea, o versiune mică, de bootstrap a Anaconda numită Miniconda, care include doar conda, Python, pachetele de care depind și un număr mic de alte pachete.

1.6 Flask

Flask a fost creat de Armin Ronacher de la Pocoo, un grup internațional de pasionați Python format în 2004. Deși inițial proiectul a fost dezvoltat ca o glumă de 1 Aprilie, acesta avea sa devina o alternativa solida pentru Django in dezvoltarea aplicațiilor web. Simplitatea și numărul de linii de cod redus au dus la popularitatea framework-ului de azi în aplicații bazate pe microservicii. Titulatura

de microframework se datorează faptului ca nu necesita alte librării sau software-uri pentru a funcționa.

Caracteristicile framework-ului:

- Permite crearea de servere de debug cât și de producție
- Permite testarea modulelor separate(Unit Testing)
- Oferă suport pentru API-uri de REST
- Asistență pentru cookie-uri sigure, pentru sesiunile din partea clientului
- Extensii disponibile pentru a îmbunătăți caracteristicile dorite, cum ar fi suport pentru manipularea bazelor de date [3]

1.7 Tensorflow

TensorFlow este o bibliotecă software gratuită și open-source pentru flux mare de date și programare tensorială pentru o gamă largă de sarcini. Este o bibliotecă simbolică de matematică și este, de asemenea, utilizată pentru aplicații de învățare automată, cum ar fi rețelele neuronale. Este utilizat atât pentru cercetare, cât și pentru producție la Google. Librăria este disponibilă pentru mai multe tehnologii de programare cum ar fi C, Java, Python și Javascript însă este mai stabilă pe ultimele 2. In cadrul proiectului nostru vom utiliza varianta librăriei din Python alături Keras.

Partea funcțională a TF este implementată în C++ și CUDA, în timp ce API-ul cel mai frecvent utilizat este creat în limbajul Python. Așadar, în TF se programează folosind Python, dar procesarea efectivă se realizează de către un engine dezvoltat in C++ / CUDA. [4]

1.7.1 Modul de funcționare

Un program TF are două componente esențiale:

- un model, ce conține totalitatea operațiilor ce se doresc a se efectua, precum și a datelor (tensorilor) ce se doresc a fi determinate. Modelul se implementează folosind o structură de tip graf, ce conține succesiunea de execuție a operațiilor și de evaluare a tensorilor (i.e. de determinare a valorilor variabilelor modelului).Acest graf este realizat conform principiului de programare data flow , de unde și denumirea de TensorFlow – o bibliotecă ce prelucrează tensori folosind operații structurate pe principiul data flow (pentru simplificare, prin tensor înțelegem un array multidimensional – scalar, vector, matrice 2D, 3D etc.) [4]
- sesiune – obiect ce permite execuția parțială sau completă a modelului menționat anterior. În cadrul unei sesiuni se rezolvă problema dorită cu ajutorul modelului definit în prealabil. [4]

1.7.2 Keras

Este API-ul la nivel înalt al TensorFlow-ului pentru construirea și antrenarea modelelor de ML. Este folosit pentru prototipare rapidă, cercetare de ultimă generație și producție, cu trei avantaje cheie:

- Ușor de utilizat: Keras are o interfață simplă, consistentă, optimizată pentru cazuri de utilizare obișnuită. Oferă feedback clar și acționabil pentru erorile utilizatorilor.
- Modulară și compozibilă: Modelele Keras sunt realizate prin conectarea blocurilor de construcții configurabile, cu câteva restricții.
- Ușor de extins: Scrierea blocurilor de construcții personalizate pentru a exprima noi idei de cercetare. Creați noi straturi, valori, funcții de pierdere și dezvoltați modele de ultimă generație.

1.8 Docker

Docker este un set de platforme ca servicii care utilizează virtualizare la nivel de sistem de operare pentru a livra software în pachete numite containere. Containerele sunt izolate unul de celălalt și împachetează propriul software, biblioteci și fișiere de configurare; ce pot comunica între ei prin canale bine definite. Toate containerele sunt administrate de un singur nucleu al sistemului de operare și, prin urmare, folosesc mai puține resurse decât mașinile virtuale.

Serviciul are atât niveluri gratuite, cât și premium. Programul care găzduiește containerele se numește Docker Engine. A fost început pentru prima dată în 2013 și este dezvoltat de Docker, Inc. [5]

Docker poate împacheta o aplicație și dependențele acesteia într-un container virtual care poate rula pe orice server Linux. Acest lucru ajută la asigurarea flexibilității și a portabilității, permițând rularea aplicației în diferite locații, fie pe loc, într-un cloud public sau într-un cloud privat. Docker folosește caracteristicile de izolare a resurselor a kernel-ului Linux (cum ar fi cgroups-uri și spații de nume kernel) și un sistem de fișiere capabil de unire (cum ar fi OverlayFS) pentru a permite rularea containerelor într-o singură instanță Linux, evitând generația de pornire și menținerea mașinilor virtuale. Deoarece containerele Docker sunt ușoare, un singur server sau o mașină virtuală poate rula mai multe containere simultan. O analiză din 2018 a constatat că un caz tipic de utilizare Docker implică rularea a opt containere pe gazdă, dar că un sfert din organizațiile analizate administrează 18 sau mai multe pe gazdă. [6]

Suportul kernel-ului Linux pentru spațiile de nume izolează în cea mai mare parte viziunea unei aplicații asupra mediului de operare, incluzând arbori de proces, rețea, ID-uri de utilizator și sisteme de fișiere montate, în timp ce cgrupurile kernel-ului asigură limitarea resurselor pentru memorie și CPU. Începând cu versiunea 0.9, Docker include propria componentă (numită "libcontainer") pentru a utiliza direct facilitățile de virtualizare furnizate de kernel-ul Linux, pe lângă utilizarea interfețelor de virtualizare abstractizate prin libvirt, LXC și systemd-nspawn.

Docker implementează un API de nivel înalt pentru a oferi containere ușoare care rulează procese izolate.

Software-ul Docker ca ofertă de servicii este format din trei componente:

- Software: demonul Docker, numit dockerd, este un proces persistent care gestionează containerele Docker și gestionează obiectele containerului. Daemon ascultă cererile trimise

prin API-ul Docker Engine. Programul client Docker, numit docker, oferă o interfață de linie de comandă care permite utilizatorilor să interacționeze cu daemoni Docker.

- Obiecte: Obiectele Docker sunt diferite entități utilizate pentru asamblarea unei aplicații în Docker. Principalele clase de obiecte Docker sunt imagini, containere și servicii.
  - Un container Docker este un mediu standard încapsulat care rulează aplicații. Un container este gestionat folosind API-ul Docker sau CLI.
  - imaginea Docker este un șablon care poate fi doar citit utilizat pentru a construi containere. Imaginile sunt utilizate pentru stocarea și expedierea aplicațiilor.
  - Un serviciu Docker permite scalarea containerelor pe mai multe demoni Docker. Rezultatul este cunoscut sub numele de "swarm", un set de demoni cooperanți care comunică prin API-ul Docker.
- Registre: Un registru Docker este un depozit pentru imagini Docker. Clienții Docker se conectează la registre pentru a descărca („pull") imagini pentru utilizare sau încărcare („push") a imaginilor pe care le-au construit. Registrele pot fi publice sau private. Două registre publice principale sunt Docker Hub și Docker Cloud. Docker Hub este registrul implicit în care Docker caută imagini. Registrele Docker permit, de asemenea, crearea notificărilor bazate pe evenimente. [6]

Unelte

Docker Compose este un instrument pentru definirea și rularea aplicațiilor Docker cu mai multe containere. Utilizează fișierele YAML pentru a configura serviciile aplicației și efectuează procesul de creare și pornire a tuturor containerelor cu o singură comandă. Utilitarul CLI pentru docker-compose permite utilizatorilor să execute comenzi pe mai multe containere simultan, de exemplu, construirea de imagini, containere de scalare, rularea containerelor care au fost oprite și multe altele. Comenzile legate de manipularea imaginii sau de opțiunile interactive ale utilizatorului nu sunt relevante în Docker Compose deoarece se adresează unui container. Fișierul docker-compose.yml este utilizat pentru a defini serviciile unei aplicații și include diferite opțiuni de configurare. De exemplu, opțiunea de construire definește opțiunile de configurare, cum ar fi calea Dockerfile, opțiunea de comandă permite una să înlocuiască comenzile Docker implicite și multe altele. Prima versiune beta publică a Docker Compose (versiunea 0.0.1) a fost lansată pe 21 decembrie 2013. Prima versiune gata de producție (1.0) a fost disponibilă la 16 octombrie 2014.

Docker Swarm oferă funcționalități native de clustering pentru containerele Docker, care transformă un grup de motoare Docker într-un singur motor Docker virtual. În Docker 1.12 și versiuni superioare, modul Swarm este integrat cu Docker Engine. Utilitarul CLI de *swarm* pentru docker permite utilizatorilor să ruleze containere Swarm, să creeze jetoane(token-uri) de descoperire, să listeze noduri în cluster și multe altele.Utilitarul CLI pentru nodul docker permite utilizatorilor să ruleze diverse comenzi pentru a gestiona nodurile într-un roi, de exemplu, listarea nodurilor într-un roi, actualizarea nodurilor și eliminarea nodurilor din roi. Docker gestionează *swarm-ul* folosind *Raft Consensus Algorithm*. Potrivit Raft, pentru a fi efectuată o actualizare, majoritatea nodurilor Swarm trebuie să fie de acord asupra actualizării. [5]

1.9 Git

Git este un sistem revision control care rulează pe majoritatea platformelor, inclusiv Linux, POSIX, Windows și OS X. Ca și Mercurial, Git este un sistem distribuit și nu întreține o bază de date comună. Este folosit în echipe de dezvoltare mari, în care membrii echipei acționează oarecum independent și sunt răspândiți pe o arie geografică mare. Git este dezvoltat și întreținut de Junio Hamano, fiind publicat sub licență GPL și este considerat software liber. [7]

1.9.1 Repository

Un repository de software, sau „repo” pe scurt, este o locație de stocare pentru pachete software. Adesea este stocat un conținut, precum și metadate. Aceste depozite grupează datele. Uneori gruparea este pentru un limbaj de programare, cum ar fi CPAN pentru limbajul de programare Perl, alteori pentru un întreg sistem de operare, alteori licența conținutului este criteriul. La partea clientului, un manager de pachete ajută la instalarea și actualizarea depozitelor.

La partea serverului, un depozit de software este gestionat în mod obișnuit de către controlul surselor sau administratorii de depozite. Unii dintre administratorii depozitelor permit agregarea altei locații a unui depozit într-o singură adresă URL și furnizează un proxy de memorie în cache. La realizarea continuă, multe artefacte sunt produse și deseori stocate central, astfel încât ștergerea automată a celor care nu sunt eliberate este importantă. [7]

Capitol 2 Modelul Rețelei Neuronale

Descrierea conținutului vizual (Image captioning) își propune descrierea conținutului vizual generând un text. La baza acestei idei a stat ideea de traducere bazată pe ML. Acolo întâlneam o arhitectura de tip autoecoder, unde prima jumătate a rețelei (encoder-ul) coda textul prezent la intrare într-un format cunoscut ca și vector-gând iar cea de-a doua parte a rețelei decoda semantica acestui vector într-o limbă noua. Acest vector-gând reprezintă semantica propoziției și permite ca aceasta sa fie decodat în altă limbă într-o propoziție a cărei lungime nu trebuie sa fie egala cu cea de la intrare, dar care păstrează informația primită. Atât encoder-ul cât si decoder-ul erau rețele neuronale recurente.

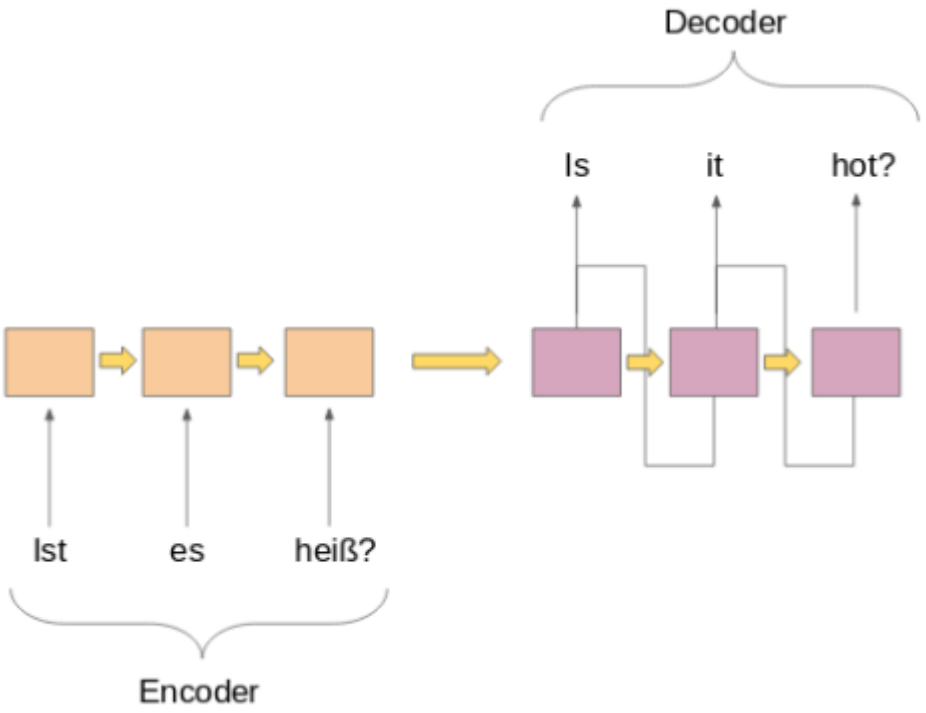


Figura 2-1 Machine Translation

Plecând de la aceeași idee vom înlocui encoder-ul cu un model de rețea neuronala convoluțională capabilă sa recunoască obiectele din imagine și care la ieșire va scoate un vector-gând similar cu cel prezent în arhitectura de traducere. Pentru a obține acest vector vom elimina din rețea ultimul strat de clasificare și vom redirecționa ieșirea penultimului strat către intrarea rețelei neuronale recurente. Cu toate acestea, dimensiunea internă a RNN-ului este de numai 512, de aceea avem nevoie de un strat intermediar complet (conectat) dens pentru a face maparea vectorului cu 4096 de elemente la un vector cu doar 512 elemente.

Decodorul folosește apoi acest vector împreună cu un marker de pornire „ssss” pentru a începe să producă cuvinte de ieșire. În prima iterație, este posibil să dea naștere cuvântului „big”. Apoi introducem acest cuvânt în decoder și sperăm că vom scoate cuvântul „brown” și așa mai departe. În cele din urmă, am generat textul „big brown bear sitting eeee" unde „eeee” marchează sfârșitul textului.

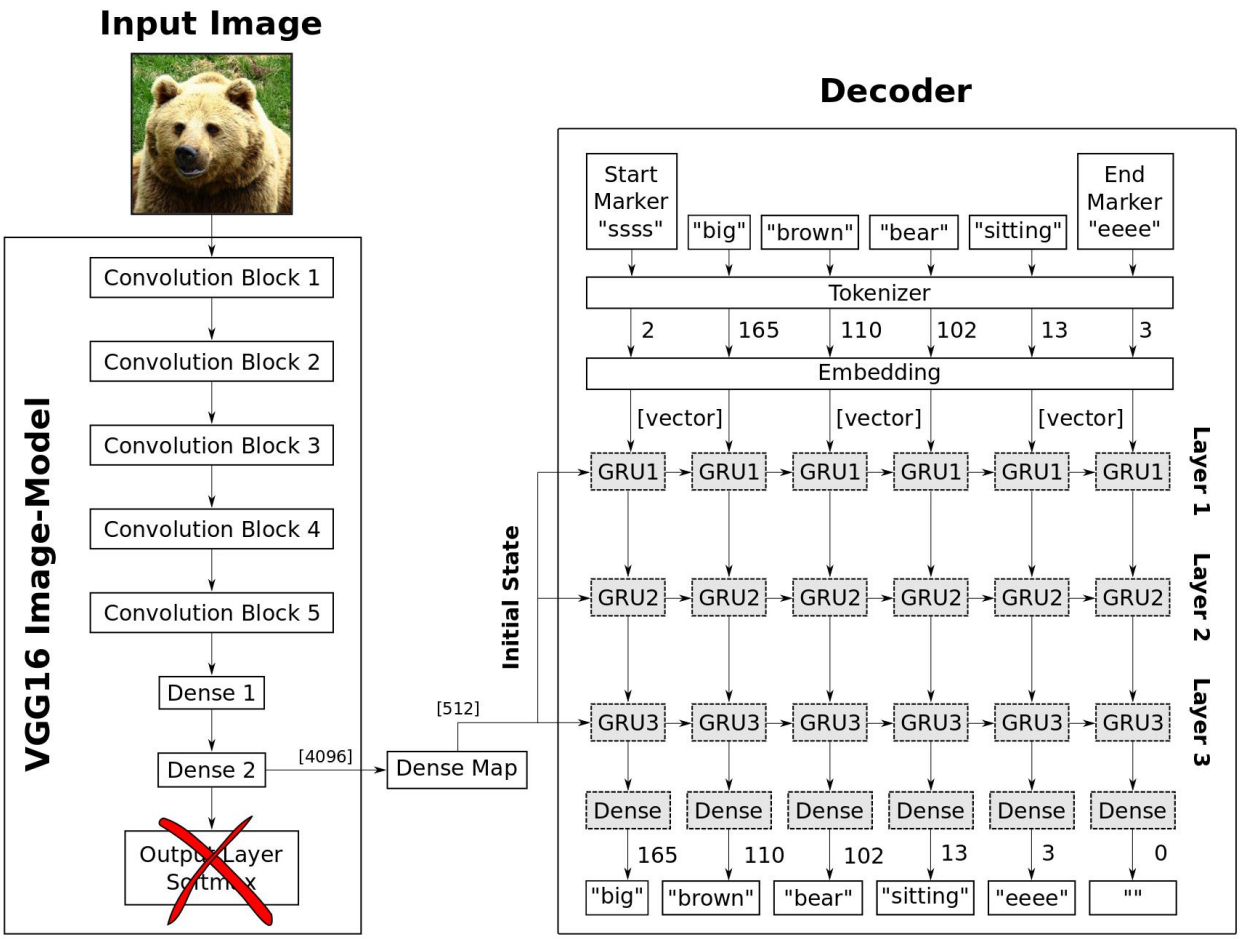


Figura 2-2 Topologie model Image Captioning [8]

Întrucât cele 2 arhitecturi diferă, VGG16 fiind unidirecțională(feed-forward), iar decoderul fiind o arhitectură recurentă, le vom antrena separat. Procedeul de antrenare este unul supervizat, în cazul rețelei convoluționale folosind "COCO data-set'' ce conține multe poze cu descrierile lor aferente, iar output-ul primei rețele va fi codat(tokenized). În cazul rețelei recurente alcătuită din 3 straturi GRU(Gated Recurrent Units), vom aplica același procedeu folosind descrierea pozelor codată in același format în care el o va primi in folosință de la rețeaua convoluțională.

2.1 Rețelele convoluționale

Este o arhitectură de rețea neuronală a capabilă sa extragă trăsături din imaginile propagate de-a lungul ei. Primele straturi convoluționale vor extrage trăsături de finețe, urmând ca pe măsură ce rețeaua se adâncește la nivelul unui strat sa putem observa trăsături ce descriu forme geometrice sau chiar obiecte de mici dimensiune. Acest tip de rețele au 3 straturi distincte:

Stratul convoluțional: este stratul de baza al rețelelor neuronale, fiind constituit dintr-o serie de filtre ce acoperă o mică porțiune din imagine. Operația matematica specifică este suma produsului element cu element intre fereastra acoperita de filtru și filtrul însăși. Filtrul cu care este aplicată convoluția pe o matrice este deplasat pe suprafața imaginii din colțul din stânga sus la dreapta și mai apoi pe liniile următoare până când ajunge in colțul din dreapta jos, obținându-se o noua matrice de valori.

Stratul convoluțional are proprietatea de a găsi caracteristici în imagine indiferent de poziția lor în imagine, datorită conectivității locale și a ponderilor comune pe fiecare porțiune a imaginii.

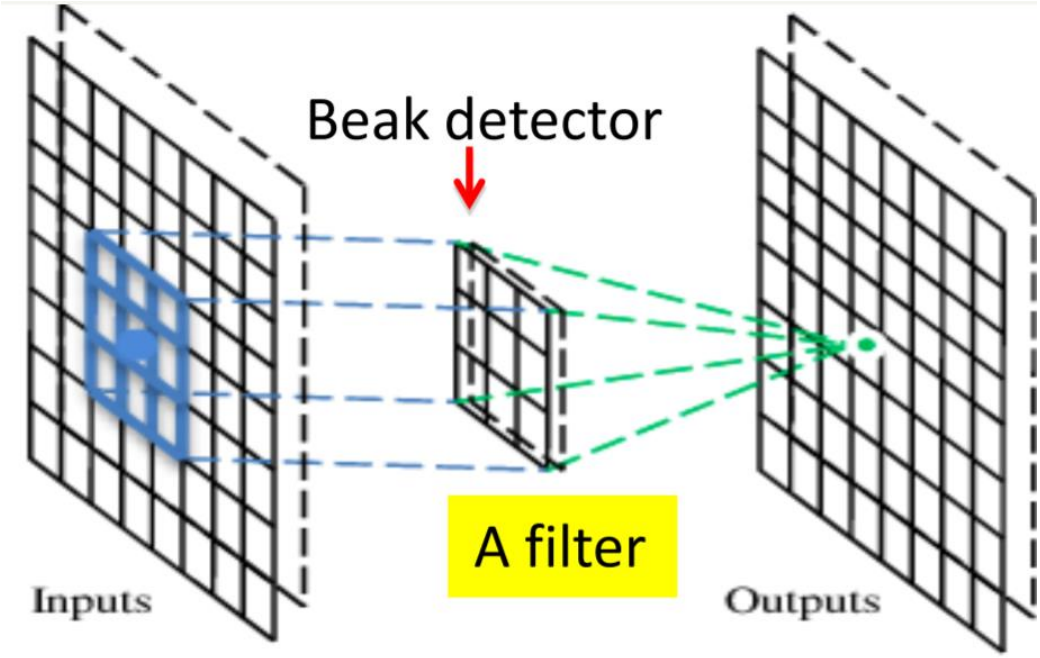


Figura 2-3 Convoluție [9]

O rețea convoluțională reprezintă o succesiune de straturi convoluționale intercalate cu funcțiile de activare. In mod uzual este utilizată funcția de activare ReLU(rectified linear unit). Comportamentul acestei funcții este descris astfel:

$$f(x) = x^{+} = \max(0, x) \ .$$

Funcția are un comportament similar cu modul de activare al neuronilor noștri și s-a dovedit a avea rezultate mai bune in faza de antrenare a modelelor de rețele neuronale datorită propagării mai eficiente a gradientului. Un alt avantaj ar fi simplitatea computațională a acestei funcții comparată cu alte funcții precum *tanh* sau *sigmoida.*

Dezavantajele utilizării sale sunt faptul ca nu este o funcție centrată in 0, că deși este o funcție invariantă cu un coeficient, în procesul de antrenare(backpropagation) funcția nu este derivabila pe intervalul $(-\infty, 0)$ și nu in ultimul rând, neuronii pot fi împinși într-o stare inactiva in procesul de antrenare. Alte variații: Noisy ReLU, Leaky ReLU, Parametric ReLU, ELU, încearcă să rezolve câteva din problemele menționate anterior. [10]

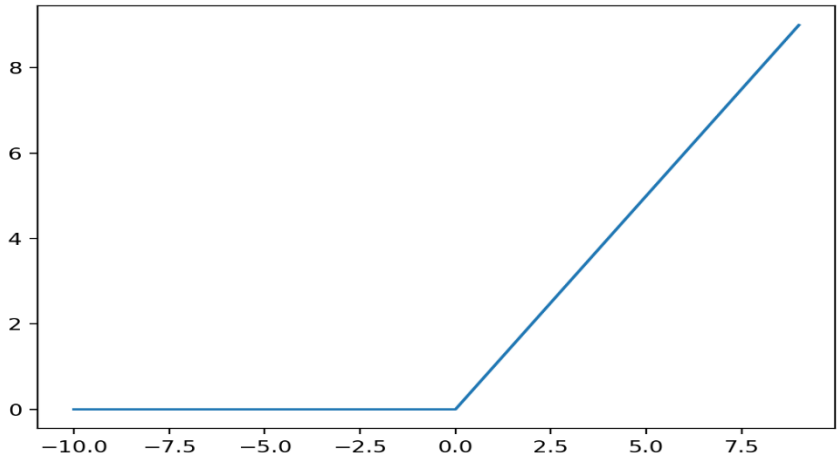


Figura 2-4 ReLU

Într-o înșiruire de straturi convoluționale, primele straturi vor fi capabile sa determine trăsături fine, contururi mici și muchii, iar pe măsură ce modelul devine din ce în ce mai adânc, putem observa trăsături de nivel înalt precum și chiar obiecte de dimensiune mai mică.

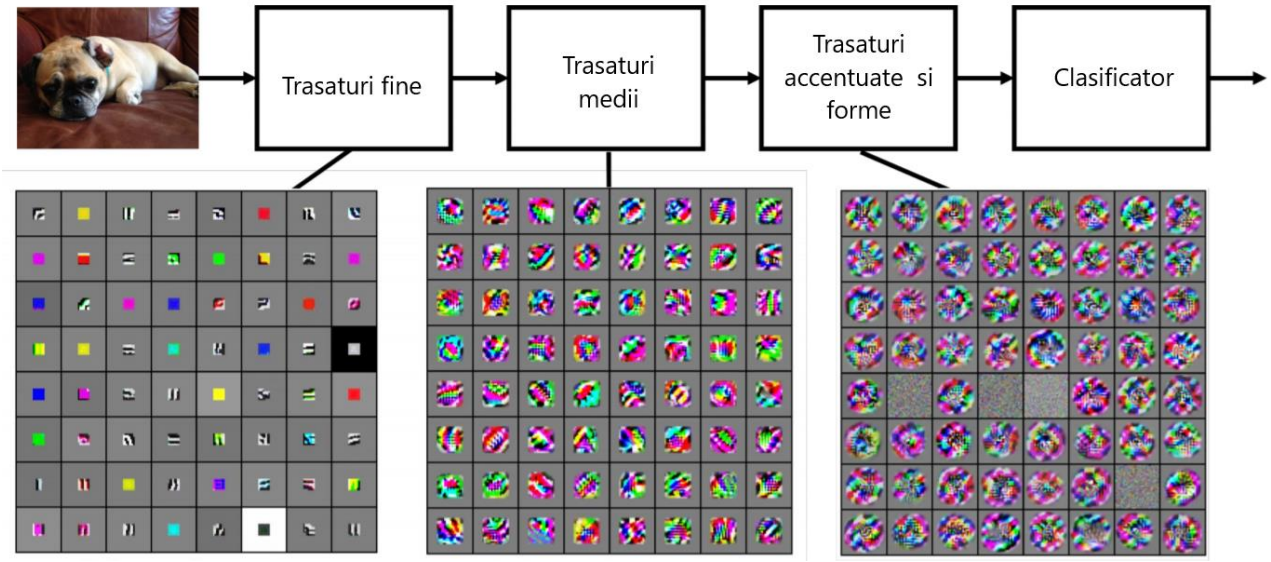


Figura 2-5 Imaginea prin straturile convoluționale [11]

Straturi de pooling al căror scop este să subeșantioneze informația in diferite stagii ale rețelei. Această subeșantionare nu va distorsiona informația din imagine, dar va reduce numărul de pixeli/calitatea imaginii pentru a putea fi mai ușor de lucrat cu aceasta.

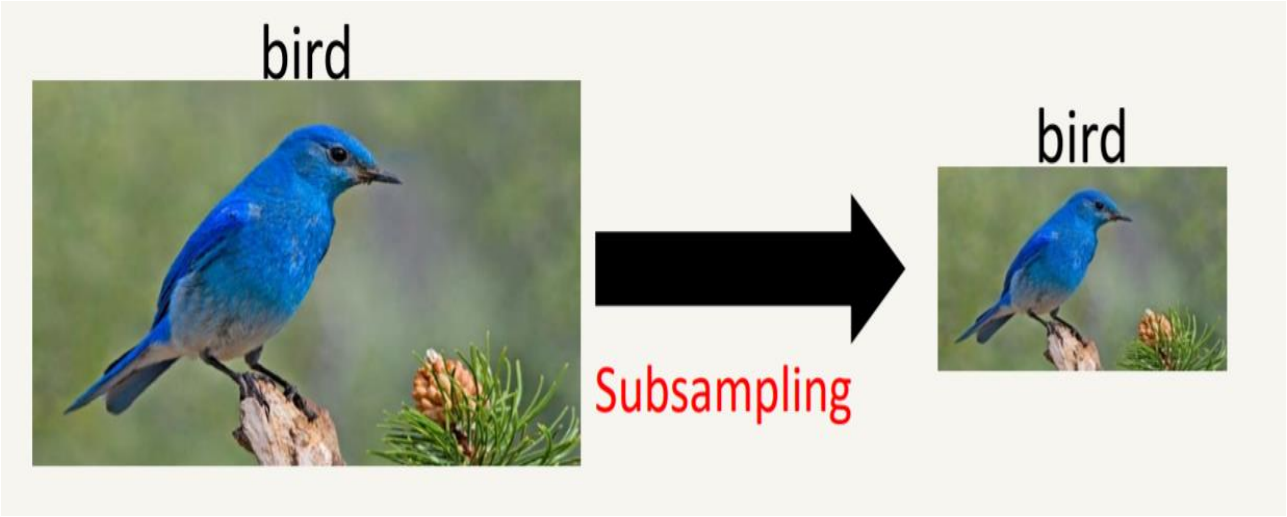


Figura 2-6 Subeșantionare [9]

De regulă aceste straturi sunt prezente într-o rețea la fiecare 1-2 straturi convoluționale și reduc la jumătate informația aflată în imagine la acel moment fie prin:

- Max-pooling-maximul dintr-o fereastră(2x2 de regulă)
- Avg-pooling- media dintr-o fereastră

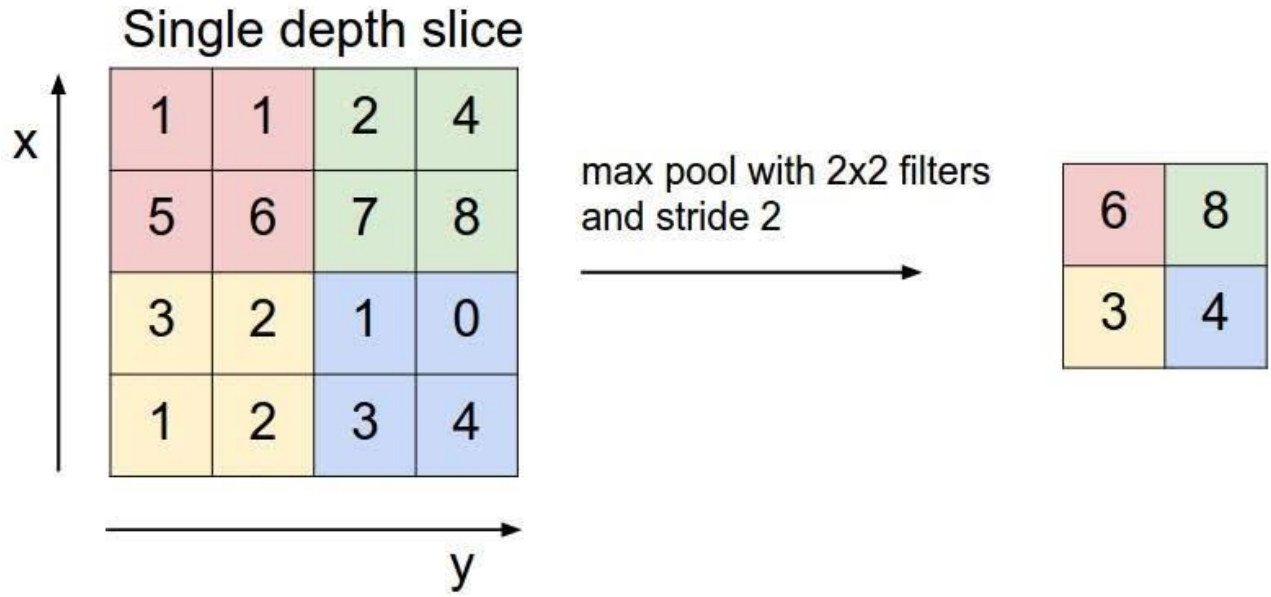


Figura 2-7 Max Pool [12]

Straturi Dense/Fully connected pe ultimele straturi dintr-o rețea convoluțională vom găsi straturi dense (fully connected) care preiau trăsăturile, determinate de straturile convoluționale și clasifică imaginile. Din punct de vedere computațional straturile dense trebuie să rețină un volum de ponderi semnificativ mai mare decât un strat convoluțional unde ponderile sunt refolosite pe fiecare fereastră. Intre straturile convoluționale și acestea mai exista un strat de liniarizare a informație care liniarizează imaginile venite din straturile convoluționale pentru a putea fi procesate de straturile fully connected. Ultimul va un număr de neuroni egal cu numărul de clase după care se dorește clasificarea.

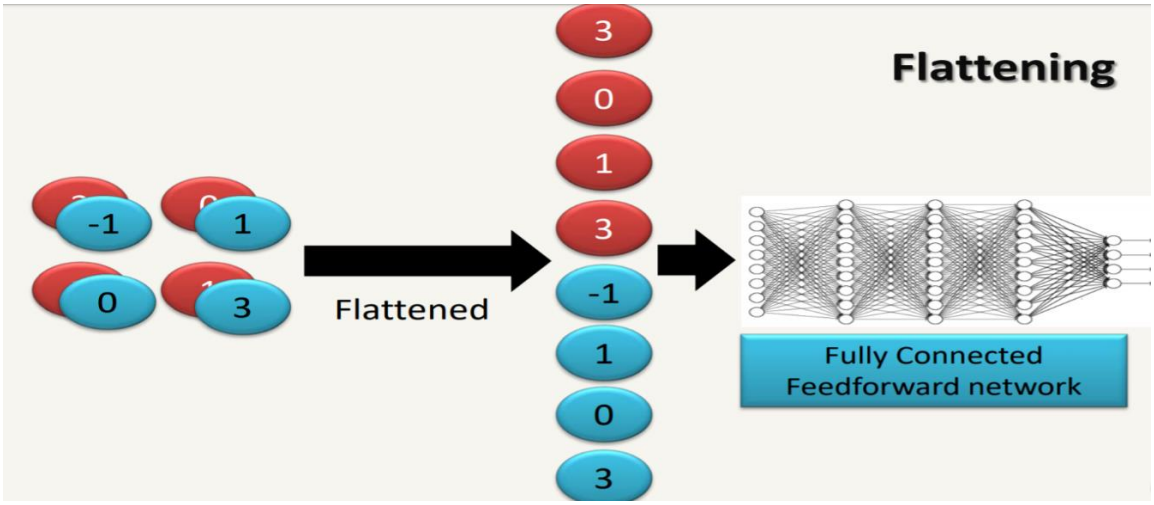


Figura 2-8 Liniarizarea straturilor convoluționale [9]

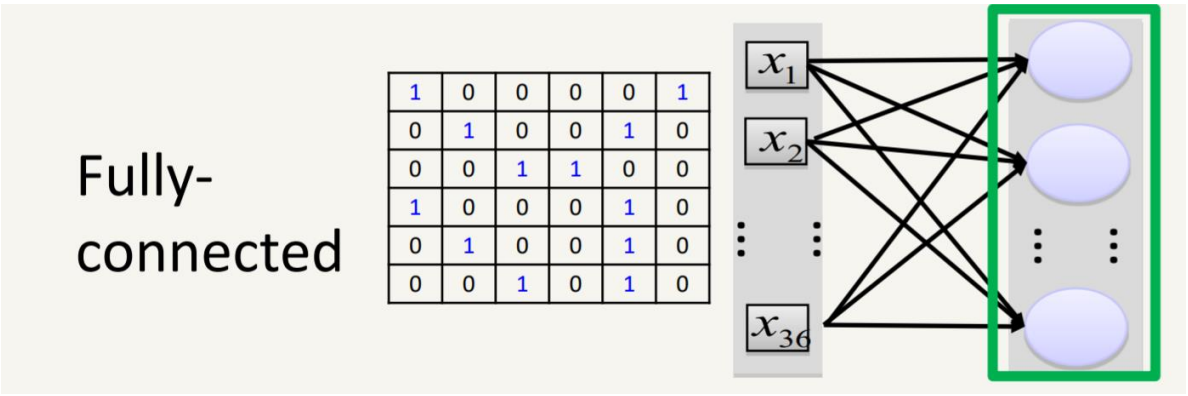


Figura 2-9 Fully connected [9]

Cele 3 tipuri de straturi menționate anterior se combină in ordinea specificată pentru a forma diferite arhitecturi de rețele neuronale convoluționale după cum se poate observa și in imaginea de mai jos.

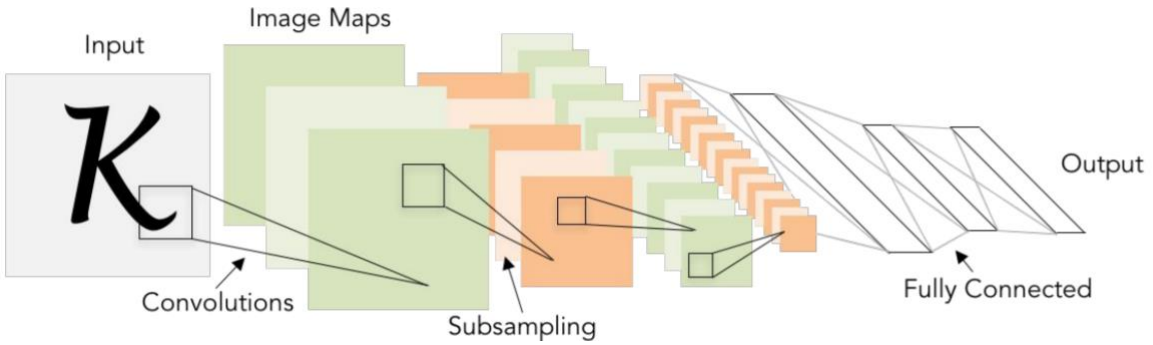


Figura 2-10 Rețea convoluțională [11]

2.2 Rețele recurente

Dacă o rețea feed forward, dens conectată primea la intrare ei toată informația, o rețea recurentă primește bucăți din informație la momente diferite de timp. Rețeaua e capabilă sa proceseze informația(cuvintele in cazul nostru) pe rând iar la momentul următor de timp, când aceasta va primi următorul cuvânt, să il proceseze, folosind aceleași ponderi și ținând cont de cuvintele pe care le-a primit anterior pentru a produce o ieșire care să clasifice întreaga secvență primită. Privită pe toată durata s-a de timp, adică pe lungimea vectorului de intrare(număr de cuvinte) ea poate fi convertită ca o rețea neuronală densă care după fiecare strat primește încă o bucată din informația de intrare, lucru ce se poate vedea și in Figura 2-12 Rețea recurentă desfășurată in timp.

Asemenea arhitecturi s-au dovedit utile în lucrul cu secvențele audio sau video dar și in modele care au ca scop traducerea dintr-o limba în alta. Sunt capabile să recunoască tipare în datele cu care lucrează, fapt ce s-a dovedit util atunci când încercăm să traducem dintr-o limbă în alta, aceeași idee fiind exprimată cu un număr diferit de cuvinte de la limbă la limbă. Similar proceselor de traducere și recunoaștere vocală, încercăm sa obținem o 'traducere' a pozelor de la intrarea modelului nostru interpretându-le inițial cu un model de rețea convoluțională ce va obține o secvența similara cu a celora obținute de encoder-ul arhitecturilor de recunoaștere vocală sau traducere dintr-o limbă in alta.

Principala problemă a acestor rețele este dată de lungimea secvenței. O dată cu trecerea timpului primele cuvinte care au intrat în rețea vor avea o pondere mai mică asupra rezultatului final fața de ultimele cuvinte. Această problemă se va reflecta și in procedeul de antrenare unde gradienții vor fi din ce in ce mai mici pe măsură ce ne întoarcem în timp pentru a modifica ponderile neuronilor.

Soluții precum LSTM(long-short term memory) și GRU au fost găsite pentru a păstra relevantă informația din pașii anteriori. Aceste unități sunt capabile să rețină informații de la aproximativ 1000 de cuvinte.

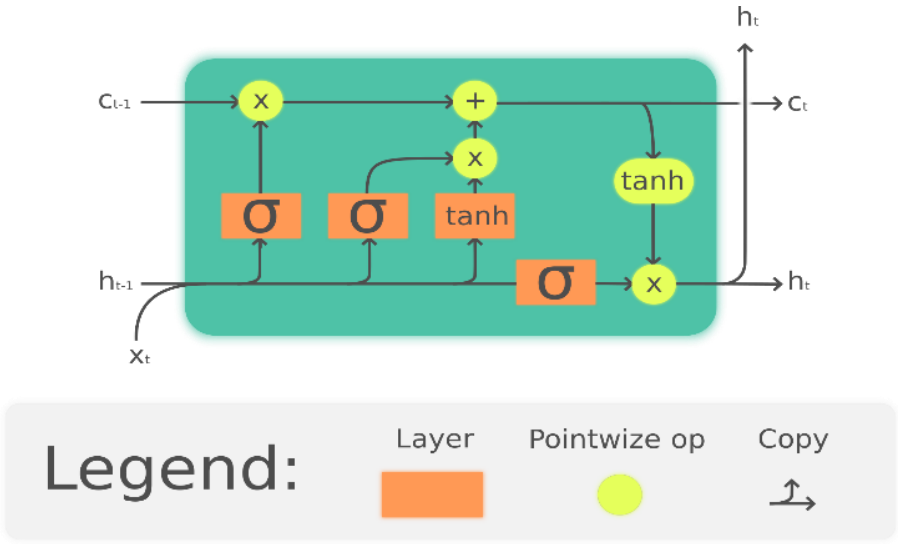


Figura 2-11 LSTM

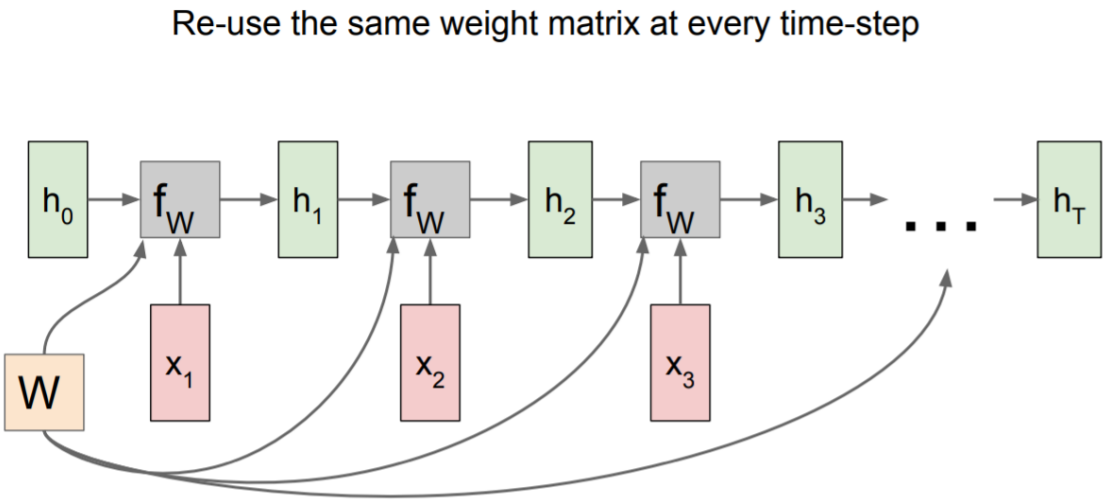


Figura 2-12 Rețea recurentă desfășurată in timp

În funcție de topologia rețelelor acestea pot avea diverse aplicații după cum se poate observa și in figura de mai jos.

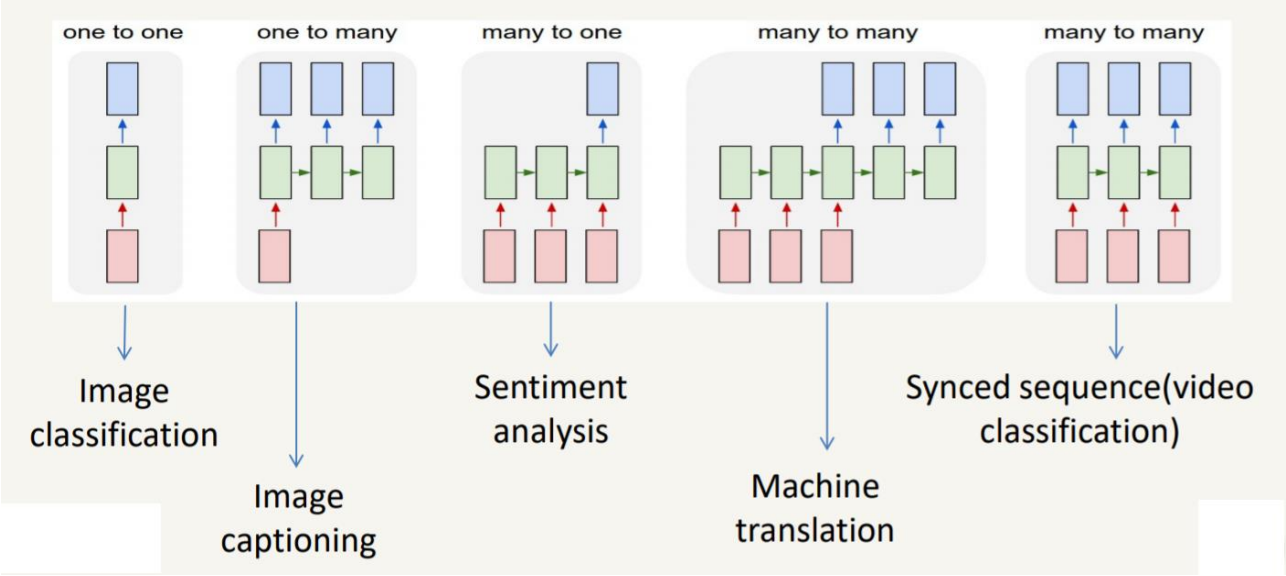


Figura 2-13Topologii de rețele recurente [11]

2.2.1 Gated recurrent unit

Gated recurrent units (GRUs) este mecanismul de neuron recurent pe care noi îl vom utiliza in descrierea conținutului vizual, în stagiul de decodare. Descoperiți in 2014 de Kyunghyun Cho [13] structura lor este similară cu cea a LSTM-urilor, având o poartă de uitare, pentru a putea reține doar informațiile relevante, însă are mai puțini parametrii ca acesta și nu are o poartă de ieșire. Deși nu la fel de puternic în aplicațiile de traducere precum LSTM, GRU au obținut performanțe similare cu acestea în recunoaștere audio și de discurs.

Modelul matematic

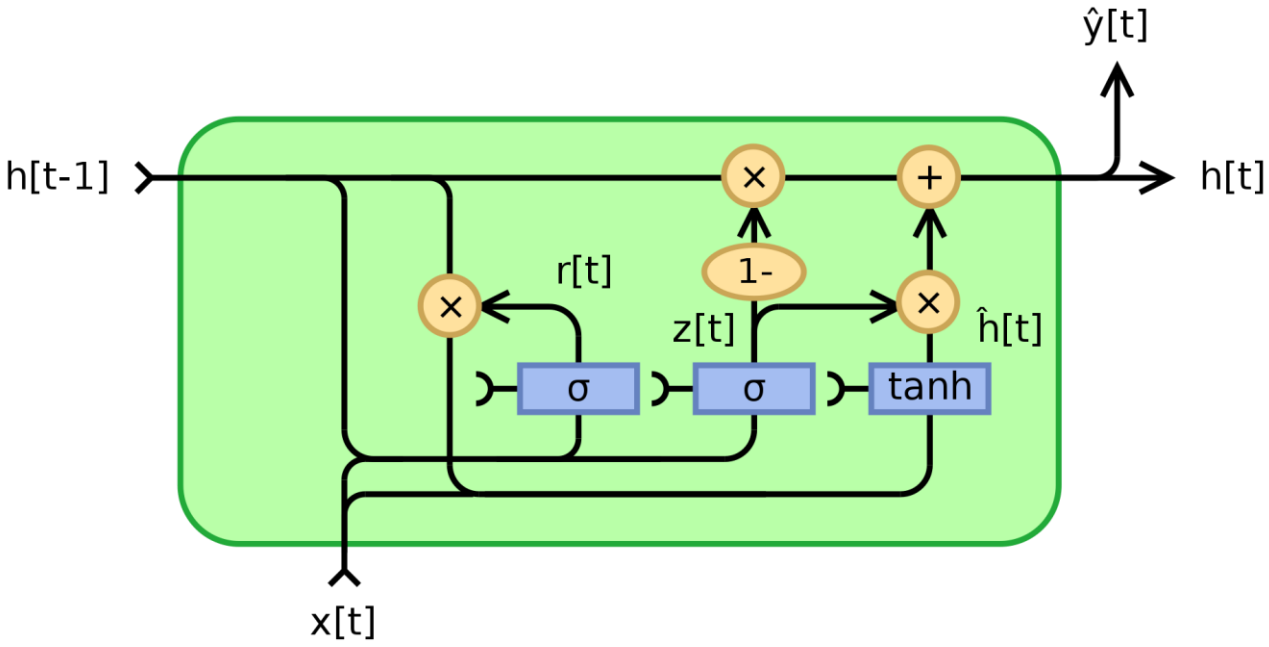


Figura 2-14 Gated Recurrent Unit [14]

La t=0, ieșirea $h_0 = 0$

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$

Unde,

- x_t -vectorul de intrare
- h_t -vectorul de ieșire
- z_t -vectorul actualizat
- r_t - vectorul de resetare al informației
- W, U și b parametrii, matricea și vectorul

Funcțiile de activare

- σ_g - Funcția de activare sigmoida
- ϕ_h - Funcția de activare tangentă hiperbolică

2.2.2 Straturile de embedding

Straturile de embedding(embedding layers) oferă o reprezentare densă a cuvintelor sub forma unui vector de valori întregi. Acest tip de strat a apărut ca o îmbunătățire a reprezentării baga de cuvinte(bag of words) datorită simplității sale. Valorile vectoriale atribuite fiecărui cuvânt poate fi învățate din texte și refolosite de la un proiect la altul. Pot fi și învățate în cadrul procesului de antrenare.

Spre deosebire de “bag of words”, straturile de embedding învață cuvintele în raport cu vecinii lor uzuali, iar, pe un vocabular de aceeași lungime, vectorii de reprezentare pentru un cuvănt au dimensiuni mai mici decât vectorii folosiți în algoritmul „bag of words”.

Este un strat flexibil care poate fi utilizat în mai multe moduri, cum ar fi:

- Poate fi folosit singur pentru a învăța o încorporare de cuvinte care poate fi salvată și folosită într-un alt model ulterior.
- Poate fi folosit ca parte a unui model de învățare adâncă, în care încorporarea(embedding-ul) este învățată împreună cu modelul însuși.
- Poate fi folosit pentru a încărca un model pre-instruit de încorporare a cuvintelor, un tip de învățare prin transfer.

În KERAS, când lucram cu un astfel de strat trebuie sa specificăm 3 parametri:

- input_dim: Aceasta este dimensiunea vocabularului de cuvinte. De exemplu, dacă datele sunt întregi codificate la valori cuprinse între 0-10, atunci dimensiunea vocabularului ar fi de 11 cuvinte.
- output_dim: Aceasta este dimensiunea spațiului vectorial în care vor fi încorporate cuvintele. Definim dimensiunea vectorilor de ieșire din acest strat pentru fiecare cuvânt.
- input_length: Aceasta este lungimea secvențelor de intrare. De exemplu, dacă toate documentele noastre ar fi alcătuite din 1000 de cuvinte, acesta ar fi 1000.

2.3 Transfer learning

Transfer learning (Învățarea prin transfer) se referă la folosirea unor cunoștințe deja acumulate prin experiențe anterioare sau probleme deja rezolvate pentru a găsi soluția unor probleme noi. În contextul ML ea face referirea la utilizarea unor modele deja antrenate pentru a rezolva probleme pe seturi de date noi. [15]

Oamenii pot adesea sa își folosească cunoștințele deja existente despre un domeniu pentru a rezolva probleme dintr-un domeniu adiacent cu acesta:

- Șah -> Dame
- Matematică -> Informatică
- Fotbal -> Rugby
- Patinaj -> Hochei

Cazuri în care am dori sa plecăm de la un model de rețea deja antrenată pentru a rezolva o problemă:

- Datele din setul nostru de antrenare nu sunt etichetate, așa ca folosim un model deja antrenat pentru a le grupa in clasa corespondentă.
- Detecția de obiecte. Poate ca modelul de la care am plecat nu era antrenat să detecteze clasele de care noi avem nevoie, însă putem reantrena ultimele straturi pentru a determina clasele de interes (Exemplu: Image Matting unde avem 2 clase fundalul și persoana). Formele pe care rețeaua noastră știe deja sa le detecteze, aflate in primele straturi ale rețelei noastre, pot fi reutilizate pentru a câștiga timp și a suplimenta pentru un volum de date de antrenare mai mic.
- Transferul de trăsături folosit în rezolvarea unei probleme.

2.4 Baza de date

Pentru a antrena modelul nostru am utilizat baza de date COCO(Microsoft COCO: Common Objects in Context). COCO este un set de date de detecție, segmentare și descriere a obiectelor la scară largă. COCO are mai multe caracteristici: obiectel segmentate, segmentarea materialelor superpixel, 330 de mi de imagini din care mai mult de 200 de mii etichetate, 91 de categorii de obiecte și peste 1,5 imagini per tip de obiect, cel puțin 5 descrieri pe o imagine. Scopul acestei baze de date a fost sa ajute la rezolvarea unei probleme actuale in domeniul ML, și anume înțelegerea scenelor.

Pentru construirea acestei baze de date, echipa formată din:Tsung-Yi, Lin Michael, Maire Serge, Belongie Lubomir, Bourdev Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollar au utilizat un instrument numit Amazon Mechanical Turk. Instrumentul le-a permis acestora adnotarea unui volum mare de date prin angajarea mai multor persoane.

Pentru a culege volumul mare de poze aceștia au efectuat căutări de baze de date interogând după tipuri de scene. În 328,000 de poze s-au etichetat aproximativ 2,5 milioane de obiecte. Comparativ cu Imagenet, COCO are mai puține clase însă are un număr mai mare de poze per clasă lucru ce poate fi util in problemele în care localizarea obiectelor în imagine este obiectivul principal. Numărul mare de imagini per clasă cat și numărul mare de descrieri textuale per imagine a reprezentat criteriul principal de selecție pentru sarcina de descriere conținutului vizual.

2.5 Implementare și Antrenare

Am încercat 2 medii pentru a antrena modelul de descriere a imaginii. Primul este laptopul personal ce dispune de o memorie de 16 GB, un procesor i7-8750H și o placa video Nvidia Geforce GTX 1060 de 6GB. Cel de al doilea este o instanță de GPU in CodeColab. Problemele întâmpinate cu cea din urma opțiune au fost datorate dimensiunii mari a bazei de date (40 GB arhiva+ date) și a duratei reduse de viață a instanței. Pentru a putea încărca baza de date de fiecare dată aș fi avut nevoie de un cont premium de google drive pentru a nu descărca și prelucra mereu setul de date. Altă problema a fost că durata unei epoci, în funcție de parametrii de antrenare era cuprinsă între 2 și 6 ore. Am preferat să utilizez resursele laptopului meu pentru a face antrenarea modelului.

Primul pas în antrenarea rețelei noastre a fost reprezentat de descărcarea setului de date. Pentru antrenare am avut 118287 de imagini. Pixelii imaginilor au fost scalați între 0 și 1. Pentru partea de encodare a rețelei am utilizat un model de VGG16 antrenat pe ImageNet.

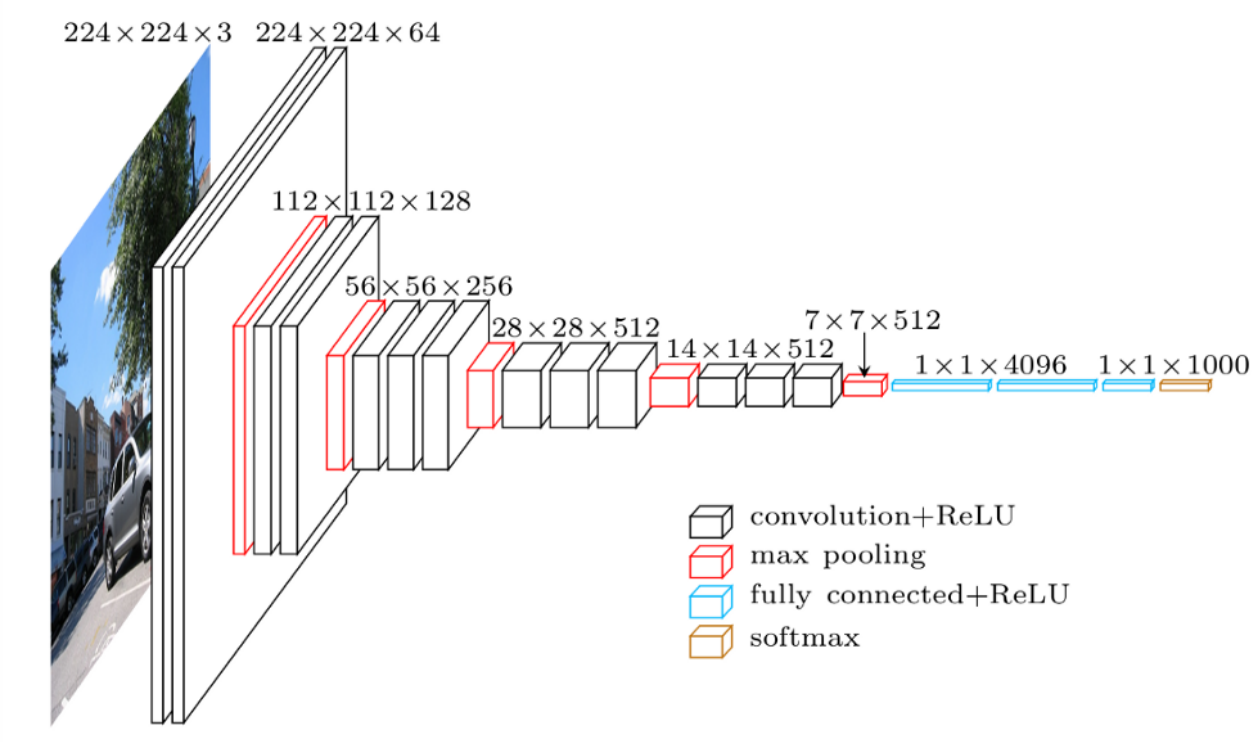


Figura 2-15 VGG16 [16]

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

Tabela 2-1 Parametrii VGG16

Modelul e capabil sa clasifice imaginile în 1000 de clase după cum putem observa din ulti mul strat. Pe noi ne interesează stratul **fc2** ce produce 4096 de valori pentru imaginea de la intrare. P entru a îmbunătății timpul de antrenare am spart modelul in 2, modelul convoluțional (encoderul) și modelul recurent(decoderul). Vom trece pozele din seturile din antrenare și validare prin primul mo del, VGG16, și vom salva cele 4096 de valori corespondente fiecare poze într-un fișier .pkl, pentru a

le putea ulterior încărca când vom antrena rețeaua recurentă. Cu toate acestea numărul de stări intern e ale rețelei convoluționale este de 512, așa ca vom avea nevoie de un strat dens care sa reducă dimensiunea de la 4092 la 512 intrări.

Pasul următor este să convertim cuvintele folosite pentru a descrie conținutul imaginilor din setul de antrenare și validare în numere întregi. Acest proces este cunoscut ca “tokenizare”, este utilizat in algoritmii de ML care lucrează pe text, deoarece rețelele neuronale nu pot opera direct pe cuvinte. Valorile întregi sunt convertite în vectori de lungimi egale ale căror elemente au valori raționale(floating). Acest va reprezenta stratul de embedding. Începutul și finalul propoziției/descrierii vor fi marcate prin 2 puțin probabile sa fac sens in contextul descrierii conținutului din imagine: 'ssss ' pentru start și ‘eeee’ pentru finalul ei.

Astfel, descrierile unei imagini înainte de tokenizare vor arata astfel:

```
['ssss Closeup of bins of food that include broccoli and bread. eeee',
 'ssss A meal is presented in brightly colored plastic trays. eeee',
 'ssss there are containers filled with different kinds of foods eeee',
 'ssss Colorful dishes holding meat, vegetables, fruit, and bread. eeee',
 'ssss A bunch of trays that have different food. eeee']
```

și post tokenizare:

```
[[2, 841, 5, 2864, 5, 61, 26, 1984, 238, 9, 433, 3],
 [2, 1, 429, 10, 3310, 7, 1025, 390, 501, 1110, 3],
 [2, 63, 19, 993, 143, 8, 190, 958, 5, 743, 3],
 [2, 299, 725, 25, 343, 208, 264, 9, 433, 3],
 [2, 1, 170, 5, 1110, 26, 446, 190, 61, 3]]
```

unde 2 și 3 reprezintă indexul pentru marker-ul de început, respectiv sfârșit.

Fiecare imagine are cel puțin 5 descrieri. Rețeaua recurentă va primi la intrare datele salvate anterior în fișierul .pkl, ce reprezintă ieșirea penultimului strat din VGG16. Numim epoca, trecerea unui număr de poze egal cu numărul de poze din setul de antrenare prin modelul nostru si ajustarea ponderilor pentru fiecare transă din acest set. Pentru a face antrenarea cât mai obiectiva pentru construirea unei tranșe de antrenare se va lua o poza la întâmplare din setul de antrenare alături de o descriere la întâmplare ce îi revine. Detrimentul acestei tehnici este ca o poza poate trece prin model de mai multe ori prin model in procesul de antrenare, în timp ce altele pot să nu se regăsească deloc în setul de antrenare

Am importat din tensorflow.keras.layers o implementare a GRU-ului și am construit modelul în felul următor.

- Din cele 4098 de valori obținute din rețeaua convoluțională cream o secvența de 128 de vectori prin embedding

  ```
  decoder_embedding=Embedding(input_dim=num_words,output_dim=embedding_size,
  name='decoder_embedding')
  ```

- Primind datele de la stratul de embedding, acest strat aduce datele de intrare la în formatul unui vector de 512 elemente

state_size=512
decoder_transfer_map=Dense(state_size,activation='tanh',name='decoder_transfer_map')

- Am aplicat o funcție de activare *tanh* pentru a obține valori cuprinse între -1 și 1

decoder_transfer_map=Dense(state_size,activation='tanh',name='decoder_transfer_map')

- Cream 3 starturi recurente.

decoder_gru1 = GRU(state_size, name='decoder_gru1', return_sequences=True)
decoder_gru2 = GRU(state_size, name='decoder_gru2', return_sequences=True)
decoder_gru3 = GRU(state_size, name='decoder_gru3', return_sequences=True)
net = decoder_gru1(net, initial_state=initial_state)
net = decoder_gru2(net, initial_state=initial_state)
net = decoder_gru3(net, initial_state=initial_state)

- Ultimul strat va fi reprezentat de un strat dens care va face o codare de tipul 'one-hot' pentru ieșirea rețelei recurente

decoder_dense = Dense(num_words,activation='softmax',name='decoder_output')
decoder_output = decoder_dense(net)

Modelul arată astfel:

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param
=================================================================
decoder_input (InputLayer)      [(None, None)]       0
_________________________________________________________________
transfer_values_input (InputLay [(None, 4096)]       0
_________________________________________________________________
decoder_embedding (Embedding)   (None, None, 128)    1280000
_________________________________________________________________
decoder_transfer_map (Dense)    (None, 512)          2097664
_________________________________________________________________
decoder_gru1 (GRU)              (None, None, 512)    986112

_________________________________________________________________
decoder_gru2 (GRU)              (None, None, 512)    1575936

_________________________________________________________________
decoder_gru3 (GRU)              (None, None, 512)    1575936

_________________________________________________________________
decoder_output (Dense)          (None, None, 10000)  5130000
=================================================================
Total params: 12,645,648
Trainable params: 12,645,648
```

Tabela 2-2 Model rețea recurentă

2.5.1 Antrenarea

În încercarea obținerii unor rezultate cât mai bune am încercat să variem diferiți "hyperparametri" pentru a îmbunătății acuratețea modelului nostru. Am aplicat mai multe variații asupra procesului de învățare asupra, ratei de învățare, optimizatorului, dimensiunii lotului și nu in ultimul rând, modelului recurent. Mai jos sunt enumerate câteva din încercările de îmbunătățire cât și soluția cu care s-a decis să se meargă mai departe.

1. În prima încercare am utilizat ADAM ca funcție de cost cu o rată fixă de învățare 10e-3, timp de 20 e epoci. Am observat că în primele epoci funcția de cost a scăzut constant pana la 1. Din epoca a 10-a valoarea funcției de cost a început sa fluctueze fără a mai converge către un minim
2. În încercarea îmbunătățirii am schimbat funcția de cost în RMSprop și am observat că deși mai încet decât cu configurația de hiper parametri anterior menționați, modelul a continuat sa conveargă către un minim local
3. Pentru a micșora funcția am încercat să modific modelul crescând numărul de neuroni din stratul de transfer de la 512 la 1024. Acest lucru a îngreunat antrenarea, timpul per epoca crescând de 4 ori. Nici rezultatele nu au fost vizibil mai bune
4. Combinația de parametrii care a dat cele mai bune rezultate a avut un optimizator RMSprop cu o rată de învățare variabilă, timp de 20 de epoci, cu un o dimensiune a lotului de învățare de 3000 de imagini. O epoca a durat în acest caz aproximativ 5 ore. Mai jos se poate observa evoluția funcției de cost.

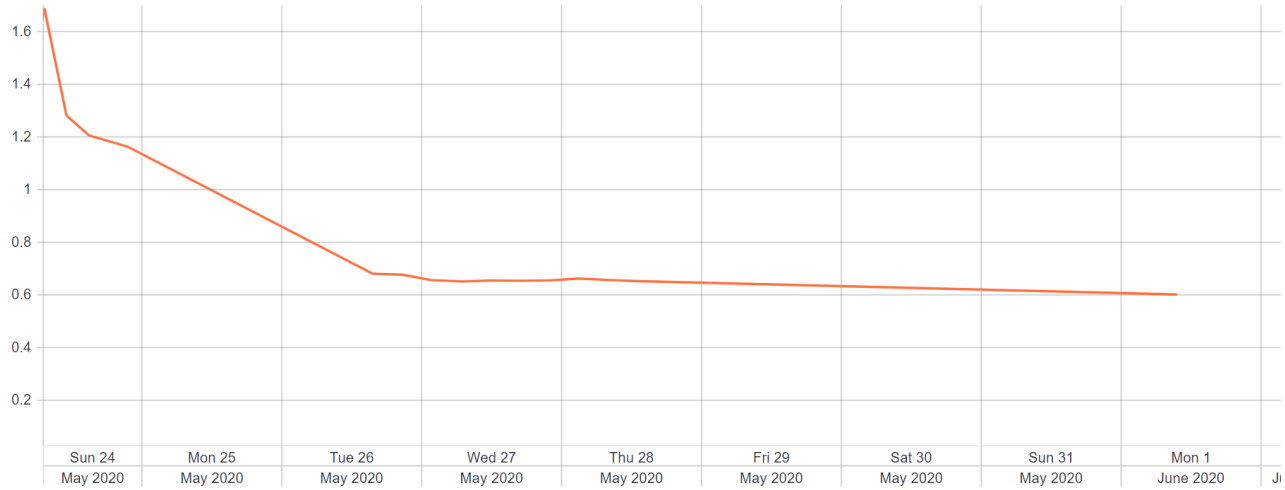


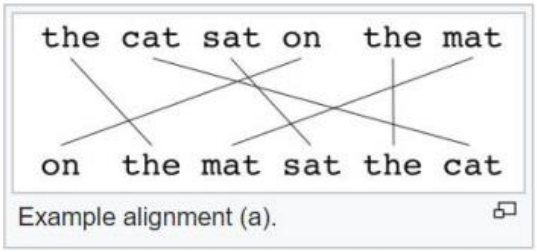
Figura 2-16 Evoluția funcției de cost

În final modelul cu performanța cea mai bună a fost salvat sub format protobuf.

2.5.2 Metrici de acuratețe

Meteor(2005)

Se bazează pe o potrivire explicită cuvânt cu cuvânt între ieșirea MT evaluată și una sau mai multe traduceri de referință. Se pot potrivi de asemenea sinonime. Calculează între descrierea candidată și legendă de referință.



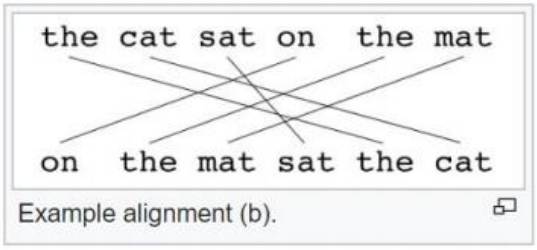


Figura 2-17 Mapare Meteor [17]

Acuratețea se va calcula după formulă

$$F_{mean} = \frac{10PR}{R + 9P}$$

BLEU(2002)

Candidate	the	the	the	the	the	the	the
Reference 1	the	cat	is	on	the	mat	
Reference 2	there	is	a	cat	on	the	mat

Figura 2-18 BLEU implementare [17]

Rezultatul este media geometrică a scorurilor de pe fiecare n descriere din setul de antrenare cu o penalitate de scurtare pentru a descuraja descrierile mai scurte.

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n\text{-}gram \in C} Count_{clip}(n\text{-}gram)}{\sum_{C' \in \{Candidates\}} \sum_{n\text{-}gram' \in C'} Count(n\text{-}gram')}$$

După cum se poate observa și în anexa 1, scorul pentru cele 2 metrici a fost pe setul de validare *0.3408*

2.6 Rezultate

Pe ultima soluție menționată anterior vom face o analiza a rezultatelor obținute de modelul nostru asupra unor imagini noi, cât și asupra unor imagini din setul de antrenare.

Pentru date noi:

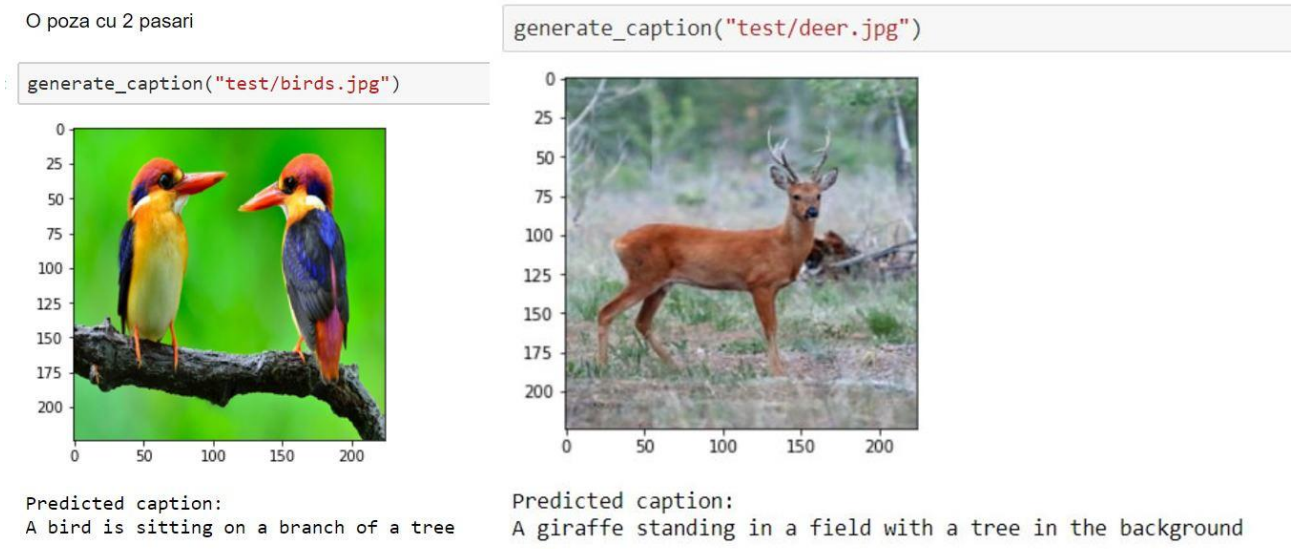
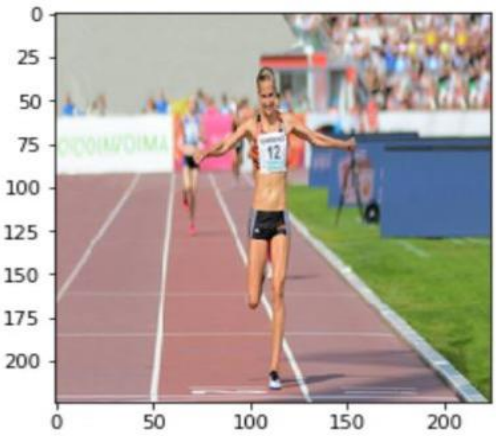


Figura 2-19 Rezultate: Păsări și capra sălbatică

Se poate observa că modelul poate recunoaște că în imagine se află păsări și ca stau pe ramura unui copac însă nu și faptul că sunt 2 păsări. Datorită culorilor similare, modelul confunda capra sălbatică cu o girafă.

O poza cu un atlet

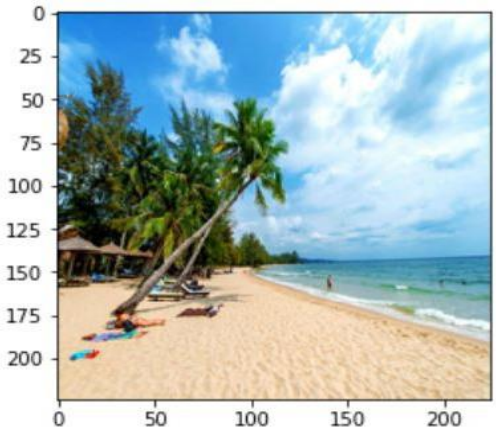
```
generate_caption("test/athlete.jpg")
```



```
Predicted caption:
A baseball player is up to bat at a game
```

O poza cu o plaja

```
generate_caption("test/beach.jpg")
```



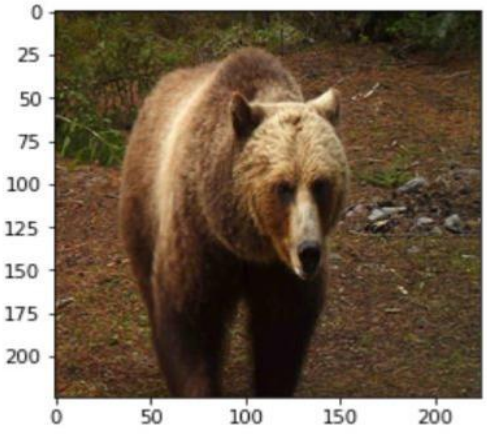
```
Predicted caption:
A person on a beach flying a kite
```

Figura 2-20 Rezultate: Atlet și plajă

Observăm din această poză, că modelul e concentrat mai mult pe ceea ce se întâmpla în fundalul imaginii și nu în prim-planul ei. În a doua poză identifică cu succes mediul însă datorită amplasării destul de răsfirate a persoanelor și a dimensiunii lor reduse trage o concluzie incorectă asupra acțiunii de pe plajă.

O poza cu un urs

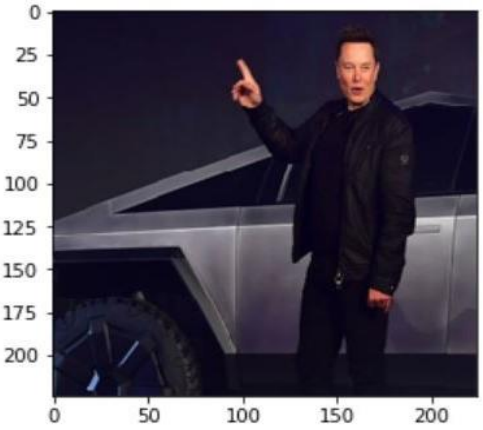
```
generate_caption("test/bear.jpg")
```



```
Predicted caption:
A bear is standing on a rocky surface
```

O poza cu un om

```
generate_caption("test/musk.jpeg")
```



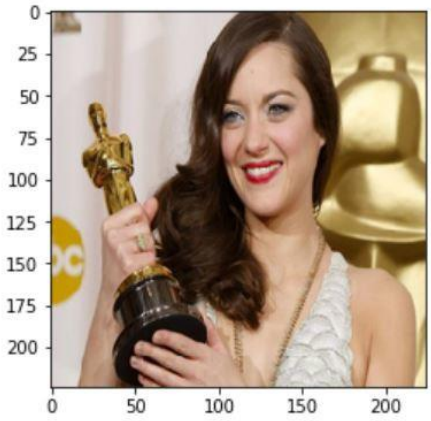
```
Predicted caption:
A man is standing in front of a mirror
```

Figura 2-21 Rezultate: Urs și Bărbat

Modelul este în a doua imagine aici păcălit de culoarea argintie a mașinii, și identifică bărbatul ca fiind în fața unei oglinzi.

O poza cu o femeie si un obiect

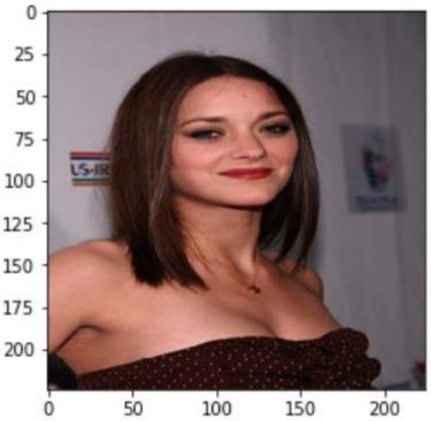
```
generate_caption("test/marion.jpg")
```



```
Predicted caption:
A young child is brushing his teeth with a toothbrush
```

O poza doar cu o femeie

```
generate_caption("test/marion1.jpg")
```

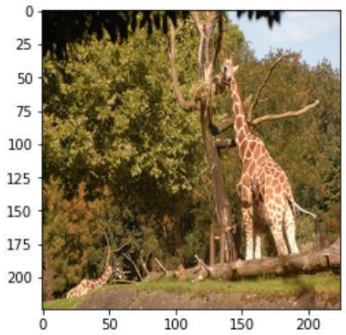


```
Predicted caption:
A woman with a cell phone in her hand
```

Figura 2-22 Rezultate: Femeie în diferite contexte

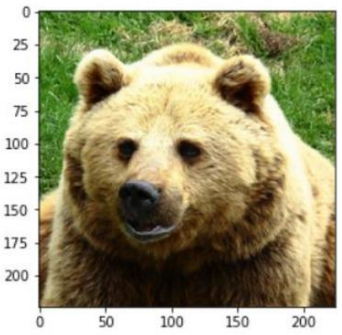
Modelul pare să aibă o problemă în a identifica femei atunci când poza nu este una de portret și chiar și atunci acesta v-a returna cu preponderența ultima descriere.

```
generate_caption_coco(idx=1, train=True)
```



```
Predicted caption:
A giraffe standing in a field of grass

True captions:
A giraffe eating food from the top of the tree.
A giraffe standing up nearby a tree
A giraffe mother with its baby in the forest.
Two giraffes standing in a tree filled area.
A giraffe standing next to a forest filled with trees.
```

```
generate_caption_coco(idx=1, train=False)
```



```
Predicted caption:
A dog is laying on the ground with a frisbee in its mouth

True captions:
A big burly grizzly bear is show with grass in the background.
The large brown bear has a black nose.
Closeup of a brown bear sitting in a grassy area.
A large bear that is sitting on grass.
A close up picture of a brown bear's face.
```

Figura 2-23 Rezultate: Date din setul de antrenare

Pentru scena în care este prezentată o girafă modelul se descurcă și este capabil să proceseze și informația de fundal similar cu descrierile din setul de antrenare. În cea de a doua poză unde informația este mai puțina, poza fiind in majoritate cu fața ursului, modelul identifică greșit animalul și adaugă informații eronate despre acțiunea din imagine.

Capitol 3 Aplicația Android

3.1 Introducere

Pentru a demonstra utilitatea unei astfel de algoritm, precum descrierea conținutului vizual, am decis să creez o aplicație care să semene cu un produs pe care l-am putea găsi pe piață. Mediul cel mai la îndemână s-a dovedit a fi prin intermediul unei aplicații pentru terminalul mobil. În momentul procesarea și analiza imaginilor reprezintă o categorie de interes pentru dezvoltatorii de Android/IOS.

Android este un proiect open-source, și nu necesită nici un tip de hardware proprietar dezvoltatorilor pentru a scrie aplicații, așa cum este în cazul IOS. Răspândirea largă a aplicațiilor android pe telefoane, tablete, ceasuri, televizoare, cât și comunitatea activă de dezvoltatori au influențat decizia de a dezvolta un soft care să se folosească de modelul prezentat anterior pentru a genera descrieri pentru imaginilor utilizatorilor.

3.2 Tehnologii folosite

Pentru a dezvolta aplicația am utilizat Android Studio pentru dezvoltarea aplicației, utilizând Kotlin ca principalul limbaj de dezvoltare. Spre deosebire de Java, codul este mult mai compact iar tratarea excepțiilor se face într-o manieră mai modernă. Principalul dezavantaj pe care Kotlin îl are este datorat vârstei sale, majoritatea aplicațiilor fiind scrie încă in Java.

Pentru definirea interfeței vizuale Android Studio folosește cod XML, într-o manieră similara cu html-ul. Mediul de dezvoltare îi permite utilizatorului să dezvolte atât utilizând limbajul scris, cât și cu ajutorul unui meniu drag&drop. Adesea lucrând la interfața vizuala am apelat la ambele modalități pentru a obține o experiența vizuală cât mai plăcută pentru utilizator. Butoanele vizibile în aplicație au fost create cu ajutorul unei aplicații web numite “Call-to-Action Button Optimizer”. Aplicația permite utilizatorului crearea unor butoane care să respect o paletă de culori similară. Logo Maker este o altă aplicație web care mi-a permis să creez un logo pentru softul nostru. Site-ul permite generarea de logo-uri sub forma de imagini vectorizate pentru a putea fi folosite atât în logoul aplicației cât și în layoutul ei.

3.2.1 Android Studio

Android Studio este un mediu de dezvoltare (software development environment, sau integrated development environment - "mediu integrat de dezvoltare) pentru colaborarea cu platforma Android, anunțată pe data de 16 mai 2013 în cadrul conferinței I / O Google.

IDE-ul este disponibil gratuit începând cu versiunea 0.1, publicată în mai 2013, apoi a trecut la testarea beta, începând cu versiunea 0.8, care a fost lansată în iunie 2014. Prima versiune stabilă 1.0 a fost lansată în decembrie 2014, apoi suportul pentru pluginul Android Development Tools (ADT) pentru Eclipse a încetat.

Android Studio este bazat pe software-ul IntelliJ IDEA de la JetBrains, este instrumentul oficial de dezvoltare a aplicațiilor Android. Acest mediu de dezvoltare este disponibil pentru

Windows, OS X și Linux. Pe 17 mai 2017, la conferința anuală Google I / O, Google a anunțat asistență pentru limbajul Kotlin utilizat de Android Studio ca limbaj de programare oficial pentru platforma Android, pe lângă Java și C ++. [18]

Android Studio permite utilizatorilor să creeze dispozitive virtuale android(AVD). Un dispozitiv virtual Android (AVD) este o configurație care definește caracteristicile unui telefon Android, tabletă, Android TV pe care dorim să le simulați în emulatorul Android. AVD Manager este o interfață în Android Studio care ne ajută să cream și să gestionăm AVD-uri.

3.2.2 Dependinte(Gradle File)

În scopul ușurării dezvoltării am utilizat o serie de pachete/librarii externe pentru a exploata resursele hardware ale terminalului mobil. Pentru a adăuga module într-un proiect tot ce trebuie să facem este să adăugam un link către repository-ul acestuia in gradle. Prin repository înțelegem un folder stocat pe un server ce oferă servicii de sub versionare a codului.

Gradle-ul este un sistem de compilare și construire (build) a codului care permite importarea altor librarii compatibile cu configurația/versiunea de Java/SDK-ul proiectului nostru. Un proiect are mai multe fișiere de configurare a build-ului pentru ca acesta sa poată fi tradus folosit de sistemul de operare android. Este bazat pe Java Virtual Machine iar rezultatul său este fișierul .apk. Se pot automatiza lucruri înainte și in timpul procesului de build, cum ar fi copierea de fișiere, teste de sistem și integrare cu diferite medii de dezvoltare. [19]

Dexter este una din aceste librării Android care simplifică procesul de solicitare a permisiunilor în timpul rulării. O dată cu introducere versiunii de Android 6.0 Marshmallow, aplicațiile nu mai aveau voie să ceară toate drepturile înainte de instalare din motive de securitate. Au existat multe cazuri în care aplicații care reprezentau jocuri aveau nevoie de acces la camera foto sau galeria de poze, lucru ce nu putea fi justificat și reprezenta o breșa de securitate. Această soluție cu care a venit Google a impus ca o aplicație să ceară utilizatorului un singur drept la un moment dat atunci când va avea nevoie de ea. Această abordare oferă utilizatorului mai mult control asupra aplicațiilor, dar necesită dezvoltatorilor să adauge o mulțime de coduri care să o susțină. Dexter eliberează codul de permisiune din activitățile și permite scrierea logicii oriunde dorim. Este o librărie open source și cu o comunitate activă care încă adaugă funcționalități la aceasta. [20]

OkHttp este modulul care ne-a ajutat să comunicăm cu serverul aplicației noastre. HTTP este principala tehnologie folosită în aplicațiile web. Este modul în care schimbăm date și media. Folosind HTTP face ca lucrurile noastre să se încarce mai rapid și economisește lățimea de bandă.

OkHttp este un client HTTP eficient în mod implicit:

- Suportul HTTP / 2 permite tuturor solicitărilor către aceeași gazdă să partajeze un socket.
- Combinarea conexiunilor reduce latența solicitărilor (dacă HTTP / 2 nu este disponibil).
- GZIP transparent reduce dimensiunile de descărcare.
- Memorarea în cache a răspunsului evită complet rețeaua pentru solicitări repetate.

OkHttp persistă atunci când rețeaua este încărcată: se va recupera în mod silențios de la problemele comune de conectare. Dacă serviciul are mai multe adrese IP, OkHttp va încerca alte adrese dacă prima conexiune nu reușește. Acest lucru este necesar pentru IPv4 + IPv6 și serviciile

găzduite în centrele de date redundante. OkHttp acceptă caracteristici moderne TLS (TLS 1.3, ALPN, fixarea certificatelor). Poate fi configurat să cadă înapoi pentru conectivitate largă. Folosirea OkHttp este ușoară. API-ul de cerere / răspuns este proiectat cu constructori fluenți și imuabilitate. Acceptă atât apeluri de blocare sincronă, cât și apeluri async cu apeluri de apel. [21]

Retrofit alături de OkHttp a ajutat la construirea unui REST API pentru a putea face request-uri la serverul nostru. Acesta este o încapsulare a HTTP în Java. Permite efectuarea de cere sincrone și asincrone la un server web, și permite conversia obiectelor din Java in payload HTML. [22]

Android PdfViewer o librărie ce permite afișarea fișierelor PDF în interfața vizuală. Permite utilizatorului să navigheze, să mărească sau micșoreze documentul, suportă gesturi și double tab.

3.3 Structura

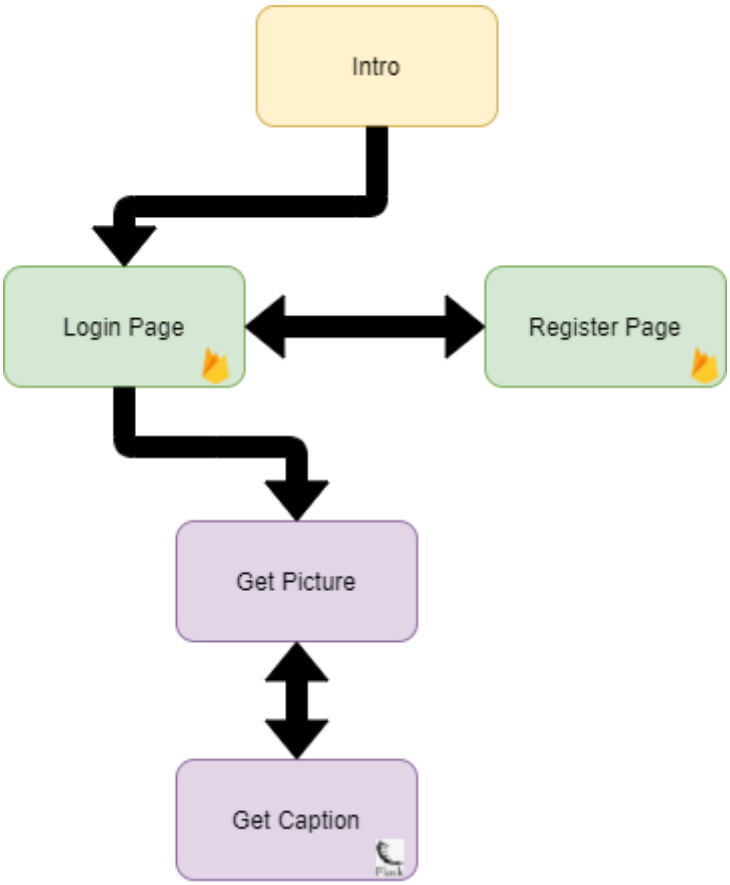


Figura 3-1 Diagrama Aplicației Android

O aplicație Android are la bază una sau mai multe activități. O activitate reprezintă o interfață cu elemente interactive în spatele căreia se află logica scrisă în Java sau Kotlin. Diagrama de mai sus descrie modul în care sunt dispuse activitățile utilizatorului. El este întâmpinat de o pagină de intrare în care este atașat un PDF-ul cu această lucrare și un buton care îi permite să treacă în pagina de LOGIN.

Ajuns în pagina de login, dacă utilizatorul are un cont, poate ajunge in pagina în care urmează să își selecteze poza dorită sau în caz contrar poate ajunge in pagina de înregistrare de unde, o dată creat un cont va fi redirecționat din nou in pagina de login. Cele 2 activități comunică cu baza de date de utilizatori din Firebase.

După ce utilizatorul a făcut o poză sau a încărcat o poză din galerie, el este redirecționat in activitatea ce comunică cu serverul de Flask și îi va returna o descriere.

3.3.1 Clase și resurse

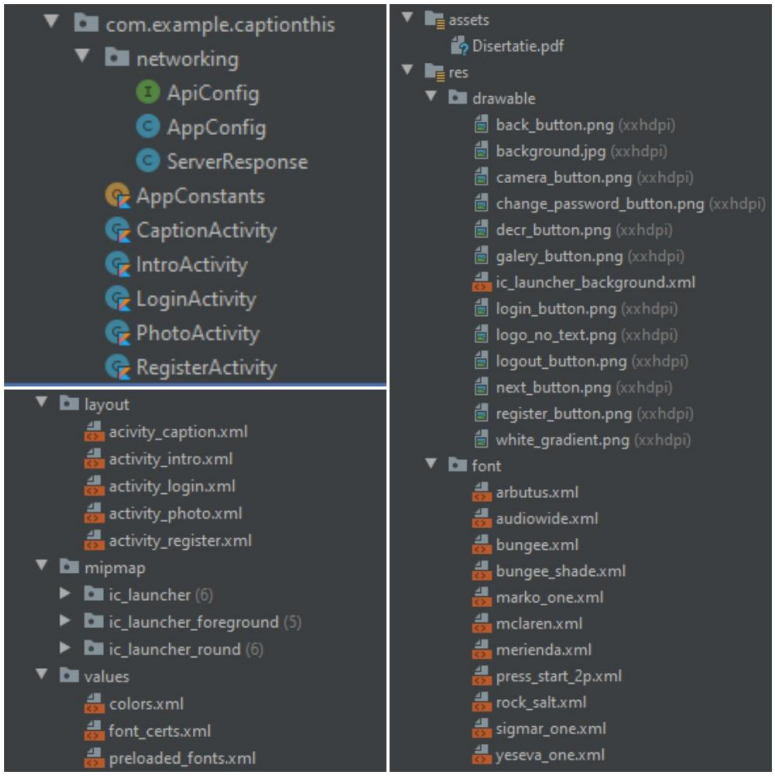


Figura 3-2 Fișierele proiectului

În figura clasele și resursele proiectului, așa cum sunt ele distribuite în Android Studio. CaptionActivity, IntroActivity, PhotoActivity, LoginActivity și RegisterActivity, sunt clasele corespondente activităților menționate mai sus și conțin logica interfeței dintre aplicație și utilizator.

Pe lângă cele 5 clase aferente activităților am mai definit încă 3 clase care să ne ajute sa ne folosim de librăriile **Retrofit** și **OKHTTP.** Folderul assets conține această lucrare în format PDF, iar in folderul res găsim resurse grafice, string-uri s-au fonturi pentru aplicația noastră. Clasele AppConfig și ApiConfig, conțin informații legate de server, de formatul request-ului și nu in ultimul rând de calea către care se va face reuquest-ul. Informația primită de la client/terminalul mobil este transformată într-un hashmap pentru a putea fi înțeleasă de server.

Imaginile sunt distribuite în sistemul de fișiere conform rezoluției lor. O dată lansată aplicația, sistemul de operare va ști ce rezoluție este potrivită pentru terminalul mobil pe care rulează. Valorile din proiect, precum codurile de culori, string-urile, și codurile culorilor sunt plasate în fișiere XML. Astfel, dacă ne dorim să modificăm o culoare undeva în interfața grafică a aplicației care s-ar putea regăsi în mai multe activități, intervenția noastră va fi minimă. String-urile stocate fac trecerea aplicației dintr-o limba in alta facilă, atât timp cat un alt String.xml este definit cu aceleași etichete dar valori diferite. Orice resursa pe ca o vedem aici poate fi apelată de oriunde din cod folosind sintaxa R.id.numele_resursei_dorite.

3.3.2 Activitatea introductiva

Figura 3-3 Activitatea introductivă

Prima interfață cu care va fi întâmpinat utilizatorul, conține o instanță de **Android PdfViewer** ce include această lucrare teoretică. Instanța va asigura utilizatorului o experiență similară cu orice alt cititor de fișiere pdf disponibil pentru android. Pe lângă acesta se mai poate observa logo-ul aplicației alături de numele ei și un buton care o dată acționat va instanța activitate de autentificare.

3.3.3 Activitățile de autentificare și înregistrare

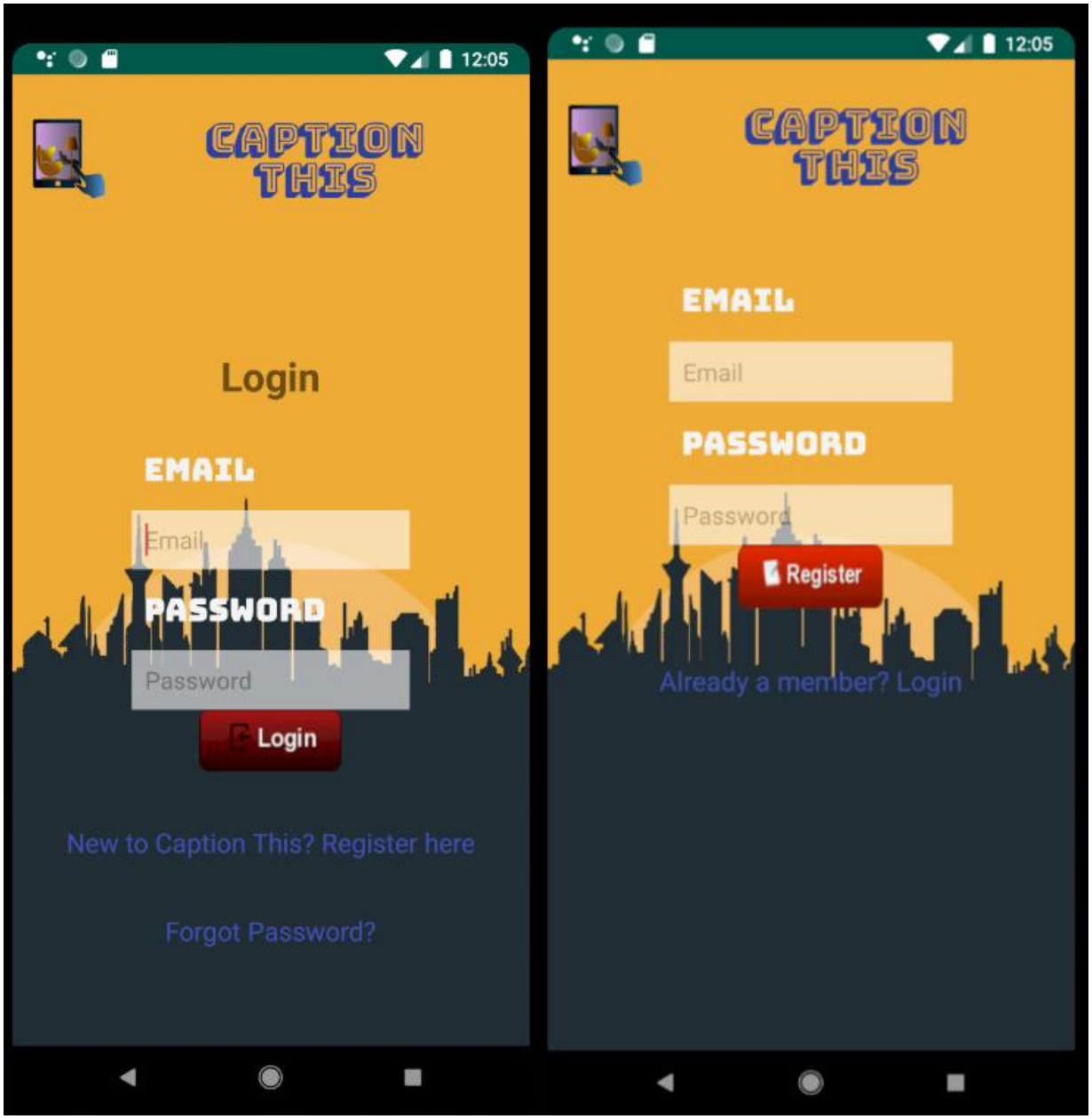


Figura 3-4 Activitățile de autentificare și înregistrare

O dată inițiată această activitate, softul verifică conexiunea la serviciul Firebase, și o dată inițiată conexiunea verifică daca un utilizator este deja logat pe terminalul mobil. În caz afirmativ această interfață nu mai este afișată utilizatorului și este inițiată direct activitatea de alegere a pozei.

Dacă nu s-a găsit nici un utilizator autentificat aplicația pune la dispoziție 2 câmpuri pentru cont și parolă, butonul de login, ce va trimite credențialele introduse de utilizator și verificate de soft pentru a fi valide. Serverul va căuta în baza s-a de date utilizatorul și dacă combinația de credențiale e validă va returna un răspuns în urma căruia va fi inițiată aplicația de alegere a pozei.

Dacă utilizatorul nu are cont, el va fi redirecționat către activitate de înregistrare. În cazul în care și-a uitat parola, un meniu îi va permite să își introducă parola, iar un link de resetare îi va fi trimis pe mail.

Similar cu activitatea de autentificare, cea de înregistrare va prelua datele de la utilizator, va valida formatul lor și le va trimite către server pentru a înregistra utilizatorul. În cazul în care adresa e email este validă, utilizatorul va primi un mail de confirmare, iar în caz contrar utilizatorul va fin anunțat ca adresa de mail introdusă nu este una validă. Utilizatorul poate naviga înapoi pe pagina de autentificare daca înregistrarea s-a făcut cu succes sau dacă apasă pe mesajul de sub butonul de înregistrare.

3.3.4 Activitatea de alegere a pozei

Figura 3-5 Activitatea de alegere a pozei

O dată autentificat utilizatorul are opțiunea de a alege o poză, fie din galerie fie să facă una în acel moment. Fiecare din cele 2 butoane vor iniția fie activitatea de camera foto, fie galeria, verificând mai întâi drepturile asupra acestor resurse. În cazul în care aplicația este utilizată pentru prima oară pe terminal, utilizatorul va trebui sa acorde aplicației dreptul la cameră și stocare. Rezultatul celor 2

activități, împreună cu imaginea este așteptat de această activitate. În cazul în care ele s-au terminat cu succes, activitate aferentă descrierii textuale este inițiată, iar poza este trimisă ca parametru.

3.3.5 Activitatea de obținere a descrieri textuale

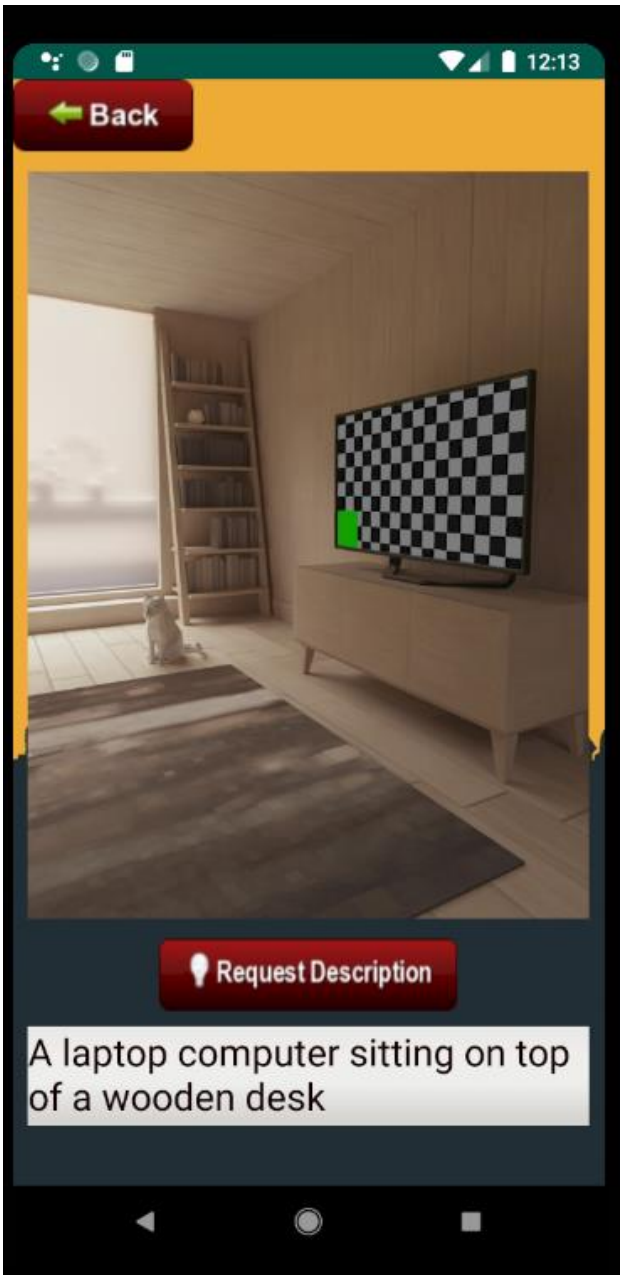


Figura 3-6 Activitatea de generare a descrierii

Poza preluată de la activitatea precedentă este preluată și afișată aici. Butonul **Request Description** va prelua imaginea din sistemul de fișiere al telefonului și o va împacheta în într-un HTTP request de tip POST, al cărui rezultat va fi descrierea. Descriere e preluată și afișată in textview-ul de sub buton. Utilizatorul are de asemenea posibilitatea să se întoarcă în meniul anterior și să aleagă altă poză.

Capitol 4 Backend

4.1 Motivatie

O aplicație care cere utilizatorului să încarce conținut vizual pentru a genera descrieri poate ridica suspiciuni legat de folosirea acestor informații pe care utilizatorul le oferă dezvoltatorilor. O aplicație scalabilă și sigură are nevoie de un backend care este ușor de întreținut și de dimensionat pentru cererea utilizatorilor. Firebase este un serviciu care oferă dezvoltatorilor instrumentele necesare pentru siguranță și protecție a datelor cât și pentru scalabilitate.

Algoritmul de machine learning ce generează descrierile pentru imagini nu poate rula direct in mediul android, așa că avem nevoie de un server care să ruleze acest algoritm la cererea clientului, ce în cazul nostru este aplicația android. Am ales să cream serverul nostru cu Flask deoarece algoritmul folosește python, iar astfel va trebui sa menținem un singur set de dependințe.

Din suita de funcții pe care firebase le ofera utilizatorilor noi am folosit funcționalitațile de hosting și autentificare. În android studio, Firebase este integrat ca un instrument ce permite generare de cod o dată ce un proiect este creat și legat la aplicație.

4.2 Autentificare

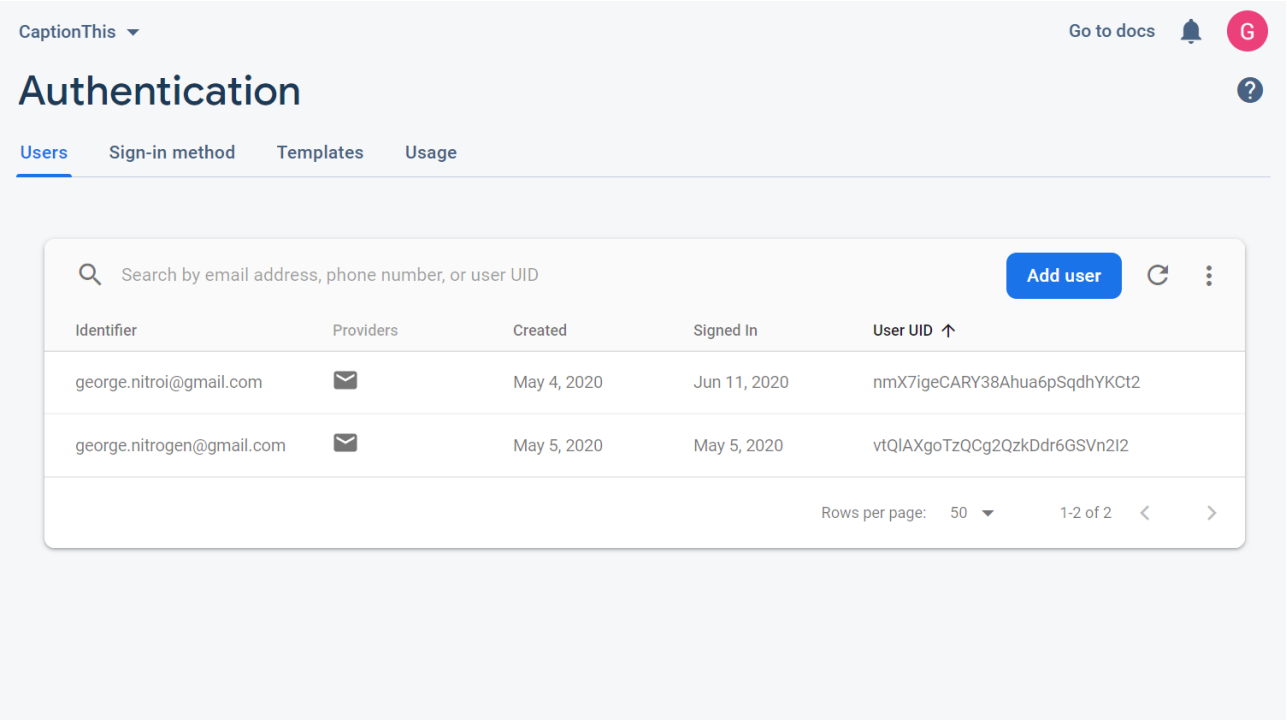


Figura 4-1 Autentificare Firebase

Majoritatea aplicațiilor trebuie să cunoască identitatea unui utilizator. Cunoașterea identității unui utilizator permite unei aplicații să salveze în siguranță datele utilizatorului în cloud și să ofere aceeași experiență personalizată pe toate dispozitivele utilizatorului.

Autentificarea Firebase oferă servicii de backend, SDK-uri ușor de utilizat și biblioteci UI gata făcute pentru a autentifica utilizatorii în aplicație. Acceptă autentificarea folosind parole, numere de telefon, furnizori populari de identitate federată precum Google, Facebook și Twitter și multe altele.

Firebase Authentication se integrează strâns cu alte servicii Firebase și se bazează pe standardele industriei precum OAuth 2.0 și OpenID Connect, astfel încât poate fi integrat ușor cu backend-ul personalizat.Pentru scopul aplicației noastre am optat pentru o autentificare bazată pe adresa de email a uitilzatorului.

Serviciul pune la dispoziție dezvoltatorului o perspectivă asupra utilizatorilor înregistrați, date despre aceștia și permite generarea unor modele pentru mail-urile pe care utilizatorul le va primi atunci când acesta își creează un cont, își uită parola sau dorește sa își șteargă contul sau dorește să își schimbe adresa de email pe care este creat acontul. Funcționalitatea asigură un domeniu de pe care aceste email-uri sunt trimise sau permite adăugarea domeniului dorit. Am configurat baza de date ca să poată exista un singur utilizator per adresa de mail. O altă măsură de siguranță pentru a proteja aplicația de abuzuri a fost de a limitat numărul de utilizatori autentificați la 100 per adresa de IP publică.

4.3 Server web

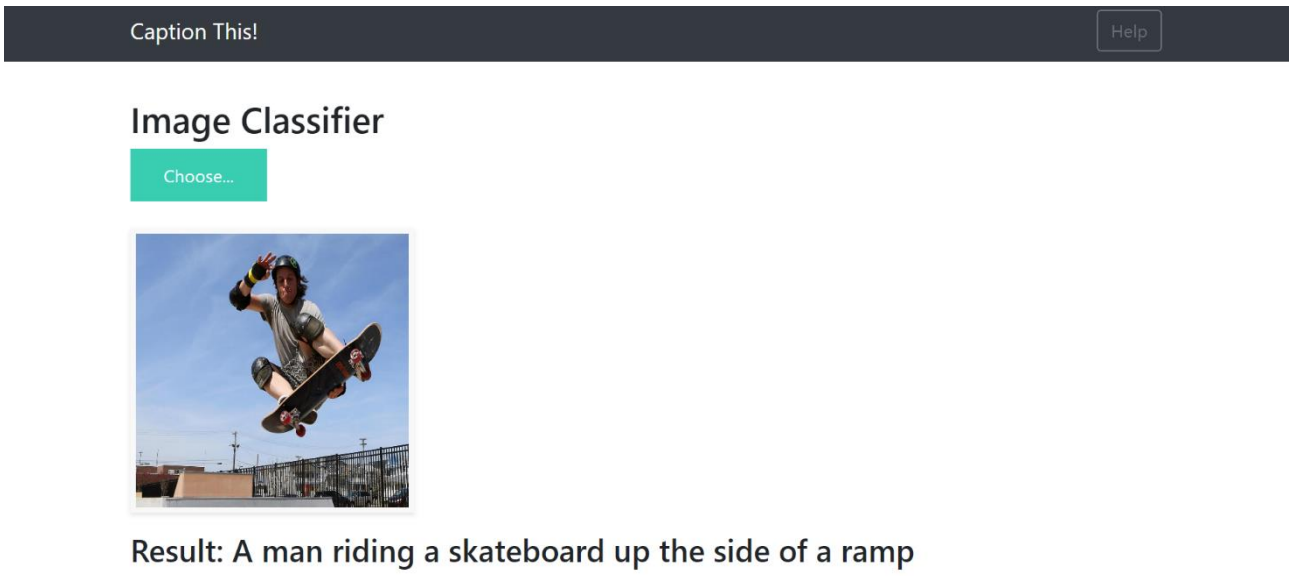


Figura 4-2 Aplicația web cu modelul de descriere

Când testam funcționalitatea server-ului nostru am decis ca cea mai bună metodă de a vedea rezultatele date de serverul nostru ar fi să cream o aplicație web care să ruleze modelul nostru în spate. Practic serverul nostru servește atât ca o aplicație web cât și ca o un endpoint pentru cererile utilizatorilor aplicației.

La pornire serverul încarcă modelul de rețea convoluțională și modelul de rețea recurentă pentru a fi folosite în continuare în cadrul unei funcții care încarcă o imagine de pe disc venită de la utilizator pentru a genera o descriere.

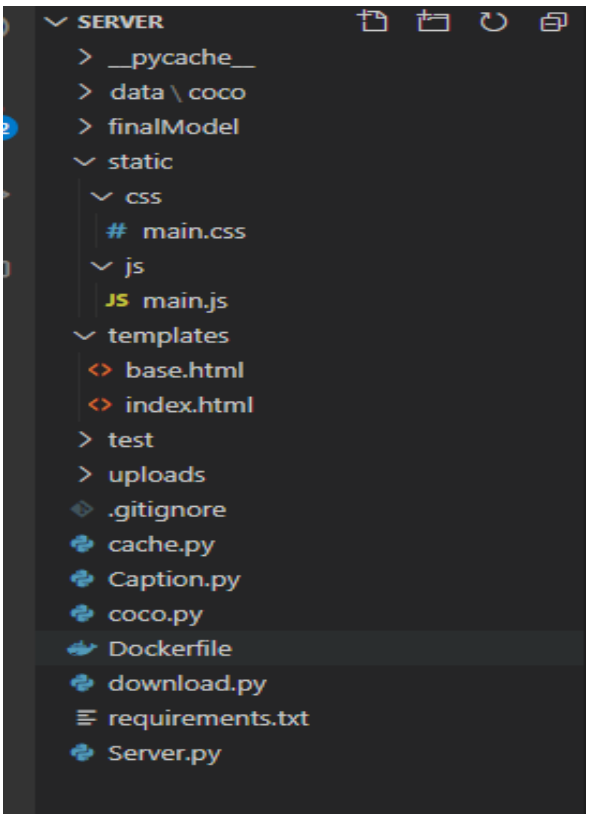


Figura 4-3 Structura serverului

Imaginea e salvată în folderul uploads, de unde e preluată de algoritm și după ce descrierea este generată ea este ștearsă pentru a preveni serverul să se umple. Pe lângă elementele de interfață grafică ale aplicației web, regăsim scripturile responsabile pentru menținerea serverului și gestionarea request-urilor venite de la utilizator. Caption.py este clasa ce gestionează încărcarea modelelor, legarea ieșirii primului model la cel de al doilea, încărcarea și prelucrarea imaginilor venite de la utilizatori și algoritmul de tokenizare. Pentru a se respecta indexarea corect cuvânt- numar întreg avem nevoie și de fișierul pkl generat în timpul antrenării. Modelul în format protobuf ce conține graful și ponderile modelului antrenat de noi.

Serverul web funcționează pe bază de cereri(request-uri HTTP). Pentru aplicația noastră, cele mai importante 2 cai de request sunt:

- POST http://127.0.0.1:5000/predict

Cale apelată atunci când utilizam aplicația din interfața web.

- POST http://127.0.0.1:5000/api/predict

Cale apelată atunci când utilizam aplicația prin API de pe terminalele mobile. Alături de acest request e necesar să adăugăm un fișier de tip .jpg sau .png, altfel serverul ne va răspunde cu “No picture in the request”. Răspunsul returnat de server este în format JSON, fiind ușor de convertit în hashmap de către terminalul mobil și afișat utilizatorului

```
{
  "message": "A man riding a skateboard up the side of a ramp "
}
```

4.4 Hosting

Pentru a obține un IP public la aplicația prezentată mai sus am folosit serviciul de hosting al firebase-ului. Pentru a înțelege însă cum funcționează acesta avem nevoie de câteva noțiuni de bază despre microservicii.

4.4.1 Docker

Proiectul de mai sus este inclus într-o imagine de docker pentru a putea fi folosit de serviciul de Firebase Hosting. Serviciul Folosește Google Cloud și necesită o subscripție a cărui preț variază cu uzul de resurse. Imaginea de docker este încărcată pe Google Cloud iar serviciul atribuie un IP public acestei aplicații. Dintr-o imagine se pot crea un număr nedefinit de instanțe care asigură acelașii serviciu.

Fișierul de docker va conține următoarele:

```
# 1 Versiunea de python
FROM python:3.6

# 2 Instalarea dependințelor
RUN pip install Flask gunicorn
RUN pip install -r requirements.txt

# 3 Pregatirea spațiului de lucru
COPY src/ /app
WORKDIR /app

# 4 Setarea portului pe care acest serviciu va raspunde
ENV PORT 8080

# 5 Pornirea serviciului web
CMD exec gunicorn --bind :$PORT --workers 1 --threads 8 app:app
```

4.4.2 Auto-scaling

Înțelegem prin auto-scaling, capacitatea unui serviciu, server, intanță de docker de a se replica de atâtea ori de câte ori e nevoie pentru a face față cererilor utilizatorilor. Asemenea tehnologii sunt folosite de multe servicii web precum: Netflix, Youbute, Amazon. Principiul din spate poartă numele de reverse-proxy. Clientul știe că trebuie sa își trimită traficul la un end-point cu un IP public. Acesta de regulă este un Load-Balancer. Load-Balancer-ul generează instanțele de servicii, creează sau

șterge din ele, și face rutarea cererilor clienților la una din instanțele disponibile ca mai apoi sa returneze rezultatul cererii.

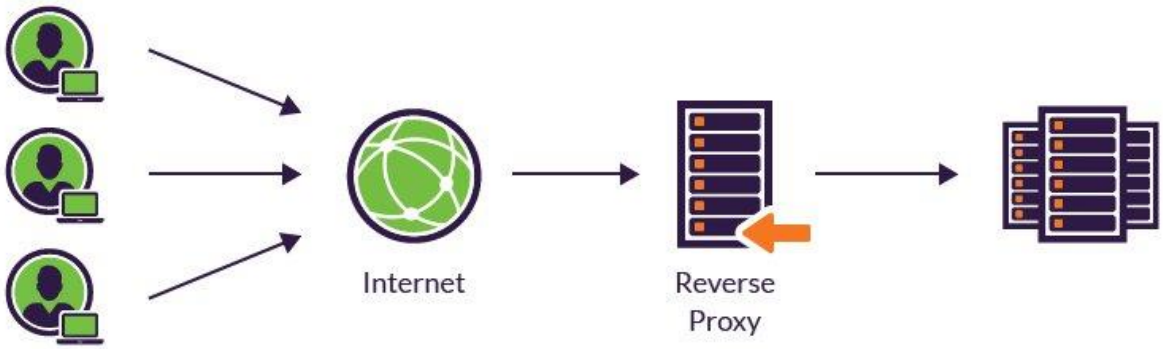


Figura 4-4 Reverse Proxy [23]

Capitol 5 Testare

5.1 Descrierea Utilizarii

Meniul aplicației este unul destul de intuitiv iar pașii pe care un utilizator trebuie să ii facă pentru a obține o descriere a fotografiilor sale sunt într-o manieră liniară de liniari. Utilizatorul primește instrucțiuni la fiecare pas, în fiecare activitate cu pașii pe care acesta trebuie să ii urmeze și este notificat in cazul în care ceva a mers prost, fie pe partea de aplicație, fie pe partea de server și îndemnat să încerce din nou.

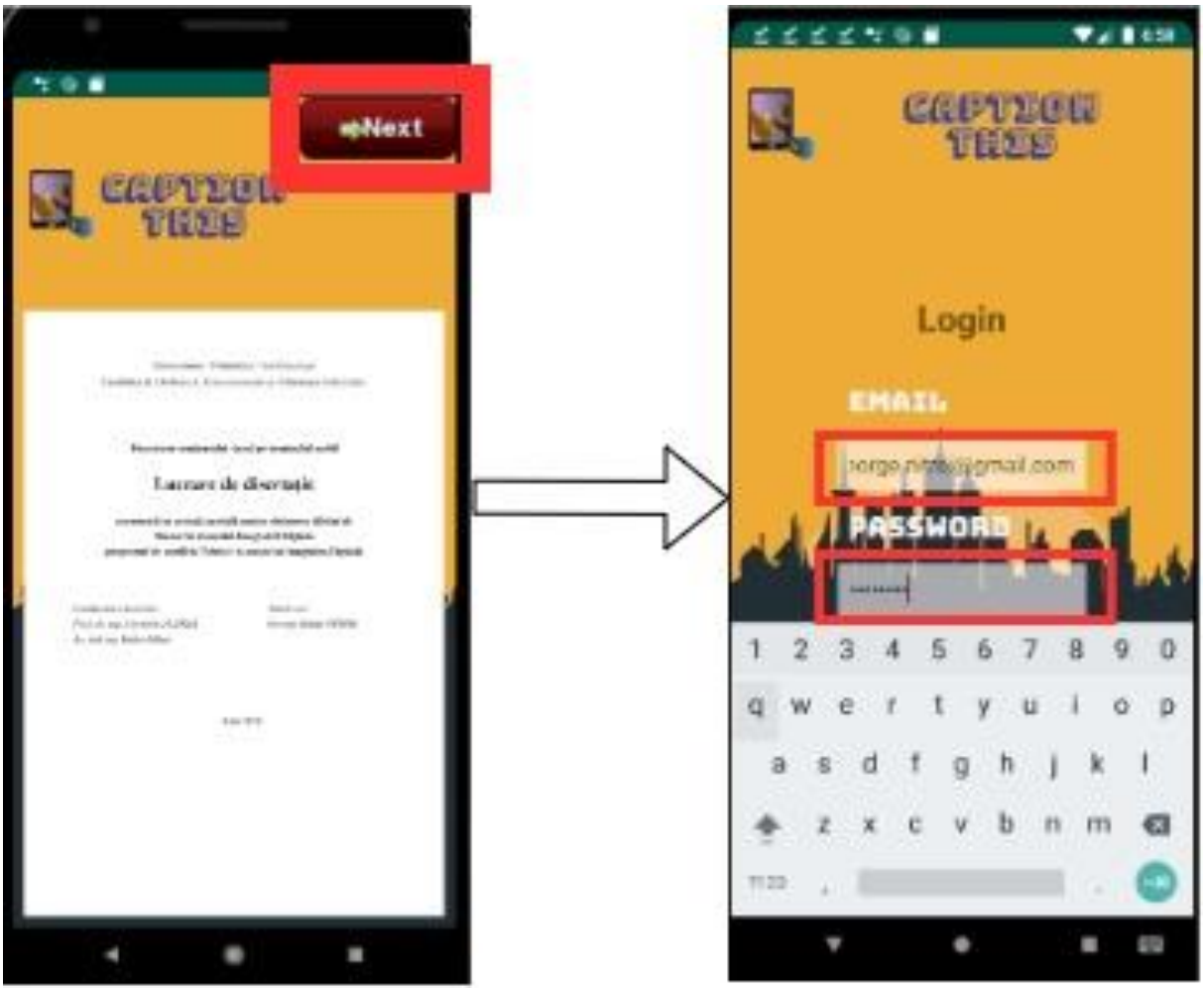


Figura 5-1 Cum ne autentificăm

Din primul ecran pentru a ajunge în pagina de autentificare tot ce trebuie să facem este să apăsam pe butonul Next. În pagina de autentificare vom introduce emailul și parola noastră pentru a merge mai departe. Dacă nu avem cont, vom apăsa pe textul care ne întreabă daca suntem noi pentru a fi redirecționați către pagina de înregistrare.

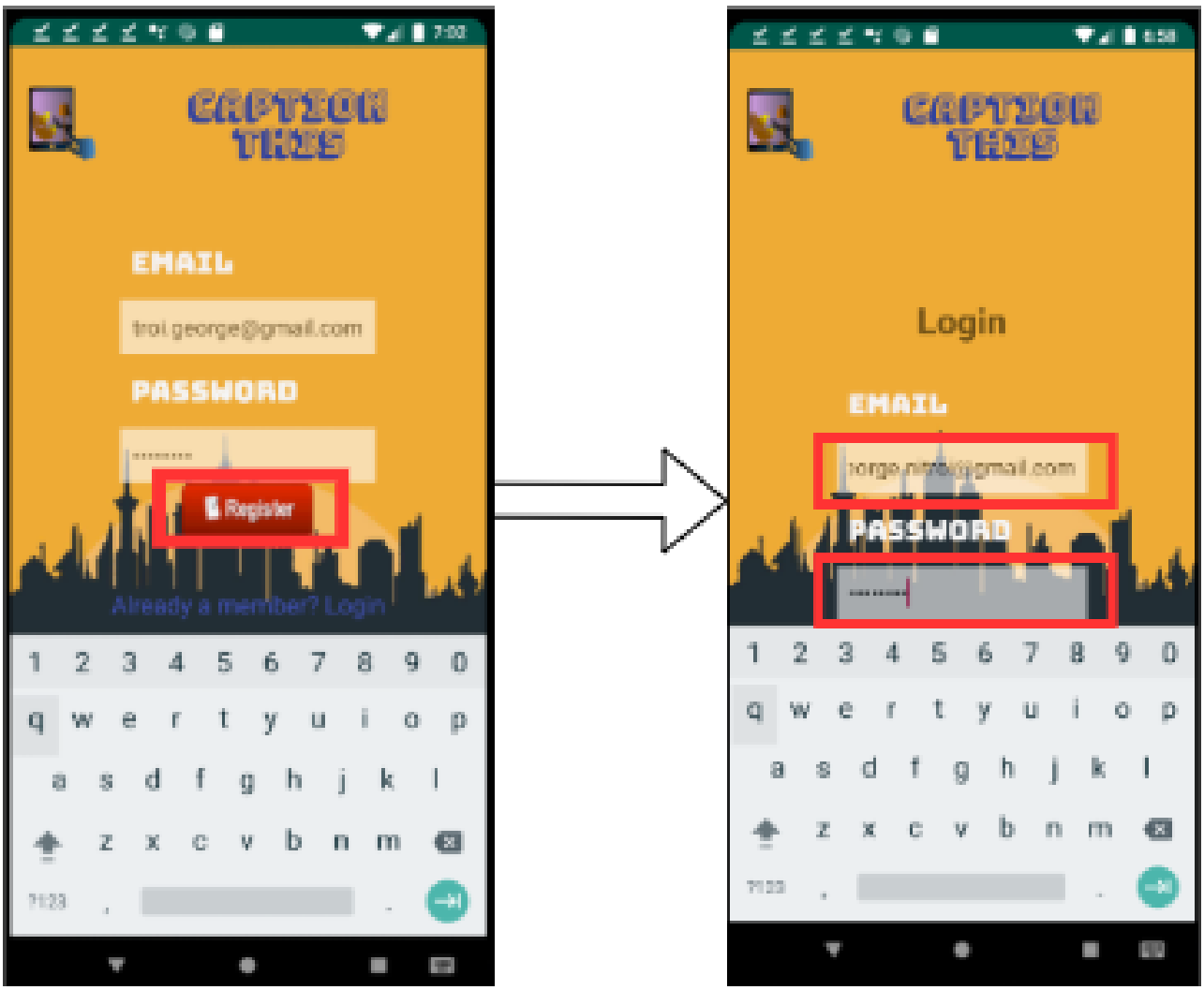


Figura 5-2 Cum ne înregistrăm

Adăugăm un email valid și o parolă și apăsam pe butonul Register după care ne întoarcem în pagina de login, confirmam contul de pe adresa noastră de mail și ne autentificăm pe aplicație.

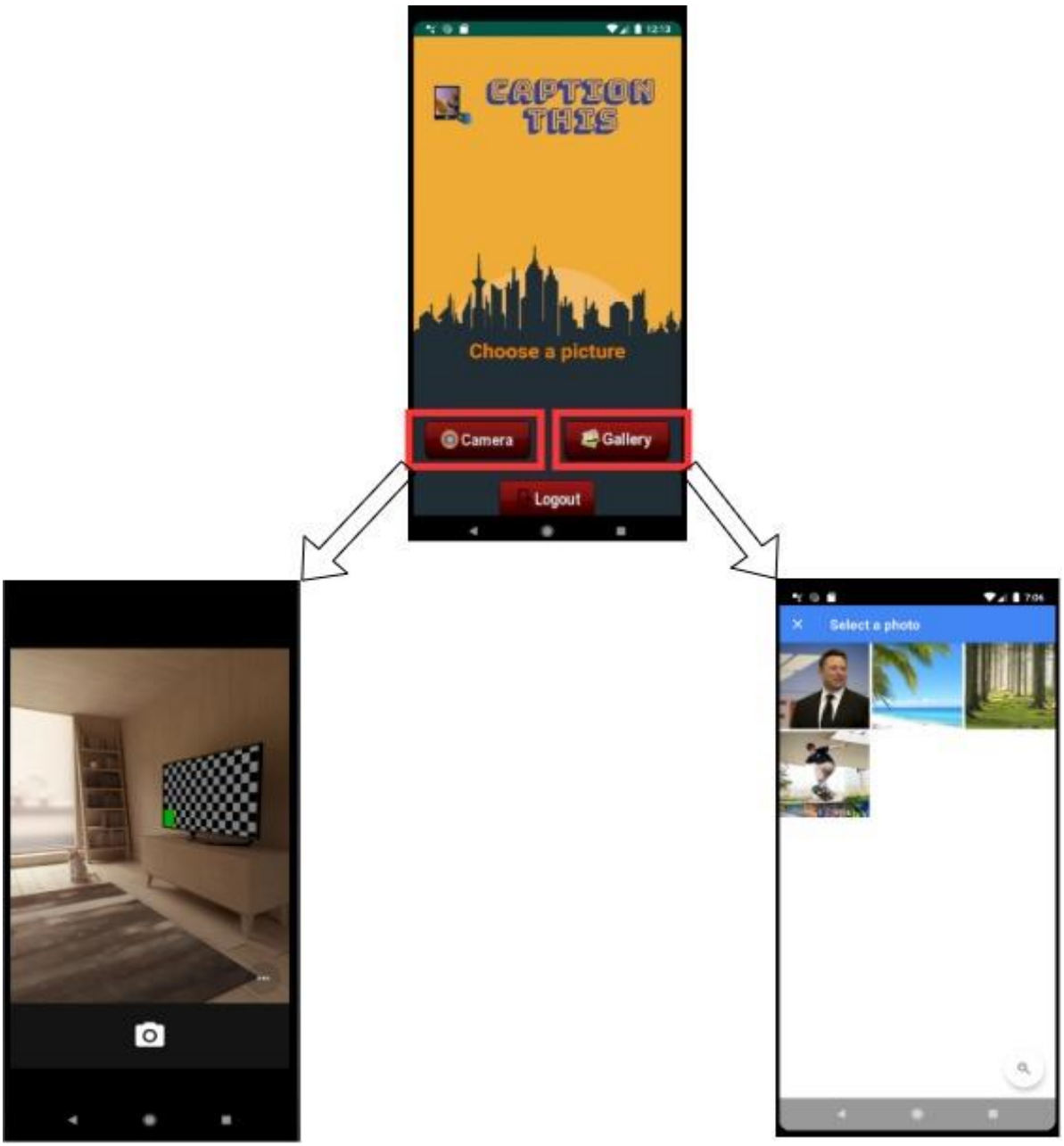


Figura 5-3 Cum alegem poza

Avem 2 opțiuni de a ne alegem poza, fie din camera, fie din galerie, fie să facem poza pe moment. Fiecare din cele două butoane va porni o activitate după cum se poate observa și în imagine. De asemenea mai există și butonul de logout care ne va permite să autentifică cu un alt cont. O dată ce imaginea a fost aleasă vom fi redirecționați către activitatea responsabilă de generare descrierii.

Figura 5-4 Cum obținem o decriere

Ajunși în această aplicație, vom putea vedea imaginea aleasă și apăsând pe butonul Request Description vom obține o descriere a pozei în aproximativ 0.5-1 secundă. Pentru a încărca o nouă poza tot ce trebuie să facem este să apăsam pe butonul back pentru a ne întoarce la activitatea anterioară.

5.2 Scalare

Dacă serverul este ținut local, pe calculatorul meu, el va putea rezolva cererile pe rând, deoarece nu a fost implementată nici o logică de multi-threading. Timpul în care scriptul returnează o descriere o dată ce rețelele sunt încărcate variază intre 0.5 și o secundă. Dacă adăugăm latența unui server local acest timp nu se modifică semnificativ (cca. 0.2 secunde in plus). Daca serverul este ținut in Google Cloud asta v-a mai adăuga încă 0.5 secunde, datorită resurselor mai reduse ale mașinii și folosirii unui CPU în loc de GPU.

În ceea ce privește scalarea aplicației, la un număr cat mai mare te utilizatori, acest lucru poate fi asigurat cu ușurință de sisteme precum Google Cloud, Amazon Web Services sau Microsoft Azure. În măsura în care sunt folosite mașini cu putere de calcul destul de mare încât sa returneze utilizatorului un rezultat în timp rezonabil, astfel de tehnologii sunt capabile să gestioneze orice nivel de cerere.

5.3 Perspective de dezvoltare

O pagină web care sa ofere aceleași facilitați de autentificare și generare a descrierii textuale ca aplicația Android. Astfel aplicația ar putea fi disponibilă pe mai multe platforme fiind ușor de folosit indiferent de terminalul folosit.

Un stocare temporară a pozele care să se comporte ca o memorie cache pentru pozele care deja au fost descrise. Acest lucru ar face generarea descrierii imaginilor ce deja au fost încărcate mult mai rapidă.

O interfață grafică mai complexă, ce să conțină animații și un sistem de notificări care să îl înștiințeze pe utilizator de eventuale modificări sau acțiuni pe care aplicația le face în fundal.

Mașini mai puternice în back-end capabile să ruleze algoritmul de machine learning mai repede pentru a spori viteza cu care descrierile sunt generate. Acest lucru e necesar dacă aplicația are un număr mare de utilizatori pentru a evita colapsul sistemului de back-end.

Îmbunătățirea modelului. După cum se poate vedea și din rezultatele prezentate în capitolul 2, modelul are dificultăți în generare unor descrieri pe anumite scene. Numărul de cuvinte din vocabular poate fi și el mărit pentru a evita generare unor descrieri generice.

Generare de descrieri pe categorii(artistice, concise, abstracte). Să avem mai multe modele recurente, fiecare antrenat pe descrieri tematice. În momentul în care utilizatorul încarcă imaginea să aibă posibilitatea de a alege o tematică a descrierii. Această tematică va decide ce model va rula pentru a genera descrierea.

Capitol 6 Concluzie

Această lucrare prezintă dezvoltarea unui aplicații pentru terminalele mobile ce îi permite utilizatorului să genereze descrieri textuale pentru pozele sale. Au fost accentuate diversele probleme tehnice apărute pe tot parcursul implementării și prezentate soluțiile de rezolvare găsite. Codul complet pentru dezvoltarea practică se poate găsi în anexe.

Testarea aplicației și a modelului de machine learning a fost realizată urmărind mai multe direcții de dezvoltare și utilizare. Rezultatele testelor realizate sunt satisfăcătoare și în concordanță cu obiectivele propuse la începutul proiectului. Conform acestora, aplicația poate fi utilizată de un număr mare de utilizatori și să genereze un set de rezultate satisfăcătoare

Proiectarea și implementarea aplicației s-a axat pe obținerea unor rezultate cat mai precise cu putinţă utilizând componente open-source și menținând un cost minim de realizare. S-au testat pe rând mai multe soluții atât hardware cât și software și s-au ales din acestea, cele care per total au avut cele mai bune rezultate în aplicația noastră. Rezultatele obținute in urma testării insuflă încrederea și convingerea utilizării unui astfel de aplicații pe terminale mobile.

Astfel de aplicații pot fi utilizate în viitor pentru a interpreta conținutul vizual din poze și secvențe video-uri. Cu un model ce oferă nivel de încredere destul de mare asemenea aplicații pot fi folosite cu scop didactic, pentru persoanele cu dizabilități, în roboți ce pot analiza diferite sporturi și oferi comentarii live, în sisteme de alarme, pentru a notifica proprietarul de cele observate.

Contribuțiile personale aduse acestei aplicații sunt:

- Proiectarea arhitecturii server-client
- Implementarea modelului de descriere a imaginilor
- Definirea modelului recurent de generare a descrierilor
- Antrenarea modelului recurent în mod repetat în încercarea de a găsi un set de parametri optimi.
- Dezvoltarea unui server web și incorporarea modelului menționat anterior ca microserviciu
- Dezvoltarea unei site web pentru serverul nostru
- Scalarea back-end ului prin tehnologii precum Docker și Google Cloud
- Dezvoltarea unei aplicații android
- Implementarea unui sistem de autentificare
- Testarea aplicației în diferite scenarii

În concluzie putem spune ca proiectul reuşeşte să obţină un set de rezultate convingătoare asupra problemei descrierii conținutului vizual pe terminalul mobil, demonstrând potenţialul unei astfel de aplicaţii şi aplicabilitatea ei în diferite domenii.

Bibliografie

[1] Google, „Androi Development,” Google, [Interactiv]. Available: https://developer.android.com/. [Accesat 18 1 2020].

[2] Android Authority, „Kotlin vs Java,” 18.10.2019. [Interactiv]. Available: https://www.androidauthority.com/kotlin-vs-java-783187/. [Accesat 18 1 2020].

[3] „Flask official page,” [Interactiv]. Available: https://flask.palletsprojects.com/en/1.1.x/.

[4] „Rețele neuronale în TensorFlow,” [Interactiv]. Available: http://myac.xhost.ro/inva/Lab9/InvA_Lab9.pdf. [Accesat 18 1 2020].

[5] „Docker,” [Interactiv]. Available: https://www.docker.com/.

[6] „What is docker?,” 2018. [Interactiv]. Available: https://opensource.com/resources/what-docker.

[7] „Git Ofiicial page,” [Interactiv]. Available: https://git-scm.com/.

[8] HVASS-Labs, „Image Captioning,” [Interactiv]. Available: https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/22_Image_Captioning.ipynb.

[9] C. Florea, „Curs MLAV,” [Interactiv]. Available: http://www.master-taid.ro/Cursuri/MLAV_files/07_08_MLAV_ConvNets_CF.pdf.

[10] M. A. C., Introduction to Machine Learning with Python: A Guide for Data Scientists, Paperback, 2012.

[11] F.-F. L. &. J. J. &. S. Yeung, „C231,” 2017. [Interactiv]. Available: http://cs231n.stanford.edu/slides/2017/. [Accesat 2020].

[12] S. Tejani. [Interactiv]. Available: https://shafeentejani.github.io/assets/images/pooling.gif.

[13] B. v. M. C. G. B. B. H. S. B. Kyunghyun Cho, „Learning Phrase Representations using RNN Encoder–Decoder,” [Interactiv]. Available: https://arxiv.org/pdf/1406.1078.pdf.

[14] „Gated recurrent unit,” GDCoder, [Interactiv]. Available: https://gdcoder.com/what-is-a-recurrent-neural-networks-rnns-and-gated-recurrent-unit-grus/.

[15] C. Florea, „Curs MLAV,” [Interactiv]. Available: http://www.master-taid.ro/Cursuri/MLAV_files/11_12_MLAV_En_Transfer_2018.pdf.

[16] „Extract Features, Visualize Filters and Feature Maps in VGG16 and VGG19 CNN Models,” 2020. [Interactiv]. Available: https://mc.ai/extract-features-visualize-filters-and-feature-maps-in-vgg16-and-vgg19-cnn-models/.

[17] Q. N. Vikash Sehwag, „Towards Image Captioning,” Princeton Univeristy, [Interactiv]. Available: https://www.cs.princeton.edu/courses/archive/spring18/cos598B/public/outline/Towards%20image%20captioning.pdf.

[18] „Android Studio release notes," [Interactiv]. Available: https://developer.android.com/studio/releases/index.html.

[19] „Google I/O 2013 - The New Android SDK Build System," 2013. [Interactiv]. Available: https://www.youtube.com/watch?v=LCJAgPkpmR0.

[20] „Karumi," [Interactiv]. Available: https://github.com/Karumi/Dexter.

[21] „OKHTTP," [Interactiv]. Available: https://square.github.io/okhttp/.

[22] „Retrofit," [Interactiv]. Available: https://square.github.io/retrofit/.

[23] „Imperva Official WebSite," [Interactiv]. Available: https://www.imperva.com/learn/performance/reverse-proxy/.

Anexa 1 Codul modelului de descriere

Image Caption Model¶

Algoritmul de descriere a imaginilor poate fi vazut ca o rețea de tipul encoder-decoder. Reteaua convolutionala codeaza imaginea intr-un format pe care reteaua recurenta il va putea folosi pentru a genera o descriere textuala.

Encoder¶

Vom folosi un model de VGG16 preantrenat pentru clasificarea imaginilor. Vom elimina unltimul layer de clasificare pentru a redirectiona iesirea stratului anterior. Stratul anterior are 4096 de neuroni, ale caror iesiri vor reprezenta un vector de 4096 de elemente ce descriu continutul imaginii. Acest vector va servi ca intrarea initiala a retelei neuronale recurente (Gated Recurrent Units sau GRU). Starea interna a retelei recurente are 512 elemente asa ca vectorul nostru va trebui redus la aceasta dimensiune cu ajutorul unui strat de 512 neuroni conectat complet la stratul anterior.

Decoder¶

Decodorul foloseste starea initiala impreauna cu un marker de start "ssss" pentru a produce cuvintele ce descriu imaginea. In prima iteratie speram ca va da la iesire cuvantul "big". Imaginea este apoi servita din nou la intrarea retelei de data aceasta cu cuvintele obtinute in iteratiile anterioare pentru a putea obtine o propozitie. În final vom obține "big brown bear sitting eeee", unde "eeee" marchează finalul propozitiei.

O schemă a algoritmului arată in felul următor:

Flowchart

Importuri¶

In [2]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import sys
import os
from PIL import Image
from cache import cache
from tensorflow.keras import backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, GRU, Embedding
from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import RMSprop,Adam

from tensorflow.keras.callbacks import ModelCheckpoint, TensorBoard
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

Am folosit Python 3.6 (Anaconda) and si urmatoarele versiuni de tensorflow si keras:

In [3]:

```
tf.__version__
```

Out[3]:

```
'2.2.0'
```

In [4]:

```
tf.keras.__version__
```

Out[4]:

```
'2.3.0-tf'
```

Importul bazei de date¶

In [5]:

```
import coco
```

Descarcarea bazei de date¶

In [6]:

```
coco.maybe_download_and_extract()
```

```
Downloading http://images.cocodataset.org/zips/train2017.zip
Data has apparently already been downloaded and unpacked.
Downloading http://images.cocodataset.org/zips/val2017.zip
Data has apparently already been downloaded and unpacked.
Downloading http://images.cocodataset.org/annotations/annotations_trainval2017.zip
Data has apparently already been downloaded and unpacked.
```

Obtinerea numelor imaginilor si a descrierilor din setul de antrenare

In [7]:

```
_, filenames_train, captions_train = coco.load_records(train=True)
```

```
- Data loaded from cache-file: data/coco/records_train.pkl
```

Numarul de imagini din setul de antrenare.

In [8]:

```
num_images_train = len(filenames_train)
num_images_train
```

Out[8]:

```
118287
```

Obtinerea numelor fisierelor si a descrierilor pentru imaginile din setul de validare

In [9]:

```
_, filenames_val, captions_val = coco.load_records(train=False)
```

```
- Data loaded from cache-file: data/coco/records_val.pkl
```

Functii ajutatoare¶

Incarcarea si redimensionarea imaginilor

In [10]:

```
def load_image(path, size=None):
    """
    Load the image from the given file-path and resize it
    to the given size if not None.
    """

    # Load the image using PIL.
    img = Image.open(path)

    # Resize image if desired.
    if not size is None:
        img = img.resize(size=size, resample=Image.LANCZOS)

    # Convert image to numpy array.
    img = np.array(img)

    # Scale image-pixels so they fall between 0.0 and 1.0
    img = img / 255.0

    # Convert 2-dim gray-scale array to 3-dim RGB array.
    if (len(img.shape) == 2):
        img = np.repeat(img[:, :, np.newaxis], 3, axis=2)

    return img
```

Afisarea imaginii si descrierilor aferente

In [11]:

```
def show_image(idx, train):
    """
    Load and plot an image from the training- or validation-set
    with the given index.
    """

    if train:
        # Use an image from the training-set.
        dir = coco.train_dir
        filename = filenames_train[idx]
        captions = captions_train[idx]
    else:
        # Use an image from the validation-set.
        dir = coco.val_dir
        filename = filenames_val[idx]
        captions = captions_val[idx]

    # Path for the image-file.
    path = os.path.join(dir, filename)

    # Print the captions for this image.
    for caption in captions:
        print(caption)

    # Load the image and plot it.
    img = load_image(path)
```

```
    plt.imshow(img)
    plt.show()
```

Example¶

In [12]:

```
show_image(idx=1, train=True)
show_image(idx=11, train=True)
show_image(idx=111, train=True)
```

```
A giraffe eating food from the top of the tree.
A giraffe standing up nearby a tree
A giraffe mother with its baby in the forest.
Two giraffes standing in a tree filled area.
A giraffe standing next to a forest filled with trees.
```

Pre-Trained Image Model (VGG16)¶

Vom crea o instanta de VGG16 folosind API-ul de Keras. Modelul a fost deja preantrenat pe setul de date ImageNet. VGG16 contine un segment de straturi convolutionale urmata de staturi fully-connected, folosite pentru clasificarea imaginilor.

include_top=True va descarca intreaga retea ce are 528 de MB. Avem nevoie de intreaga retea deoarce vom folosi si o parte din straturile interconectate pentru descrierea imaginilor.

In [13]:

```
image_model = VGG16(include_top=True, weights='imagenet')
```

Print a list of all the layers in the VGG16 model.

In [14]:

```
image_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
```

```
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

Folosim penultimul layer de clasificare, numit *fc2*. Acesta e un strat dens/complet conecatat.

In [15]:

```
transfer_layer = image_model.get_layer('fc2')
```

Il numim "transfer-layer" deoarece va transfera informatiile venite de la reteaua convolutionala.

Cream un nou model ce are aceeasi intrare ca reteaua de mai sus insă iesirea este penultimul strat

In [16]:

```
image_model_transfer = Model(inputs=image_model.input,
                             outputs=transfer_layer.output)
```

Modelul se asteapta ca imaginile sa aiba urmatoarea dimensiune

In [17]:

```
img_size = K.int_shape(image_model.input)[1:3]
img_size
```

Out[17]:

```
(224, 224)
```

Pentru fiecare imagine modelul va obtine un vector de 4096 valori

In [18]:

```
transfer_values_size = K.int_shape(transfer_layer.output)[1]
transfer_values_size
```

Out[18]:

```
4096
```

Procesarea tuturor imaginilor¶

Vom crea functii ce au rol sa preproceseze imaginile din seturile de date folosind modelul preantrenat si salvand valorile intr-un fisier pentru a putea fi reincarcate rapid.

Obtinem astfel un nou set de date de intrare cu valorile preprocesate. Nu vom schimba toți parametrii modelului VGG16. Avem nevoie de valorile de transfer pentru a instrui modelul de subtitrare a imaginii pentru mai multe epoci, astfel încât economisim mult timp calculând valorile de transfer o dată și salvându-le într-un fișier *.pkl in prealabil.

Aceasta este o funcție de ajutor pentru afisarea progresului.

In [19]:

```
def print_progress(count, max_count):
    # Percentage completion.
    pct_complete = count / max_count

    # Status-message. Note the \r which means the line should
    # overwrite itself.
    msg = "\r- Progress: {0:.1%}".format(pct_complete)

    # Print it.
    sys.stdout.write(msg)
    sys.stdout.flush()
```

Aceasta este funcția pentru procesarea fișierelor date folosind modelul VGG16 și returnarea valorilor de transfer ale acestora.

In [20]:

```
def process_images(data_dir, filenames, batch_size=32):
    """
    Process all the given files in the given data_dir using the
    pre-trained image-model and return their transfer-values.

    Note that we process the images in batches to save
    memory and improve efficiency on the GPU.
    """

    # Number of images to process.
    num_images = len(filenames)

    # Pre-allocate input-batch-array for images.
    shape = (batch_size,) + img_size + (3,)
    image_batch = np.zeros(shape=shape, dtype=np.float16)

    # Pre-allocate output-array for transfer-values.
    # Note that we use 16-bit floating-points to save memory.
    shape = (num_images, transfer_values_size)
    transfer_values = np.zeros(shape=shape, dtype=np.float16)

    # Initialize index into the filenames.
    start_index = 0
```

```
    # Process batches of image-files.
    while start_index < num_images:
        # Print the percentage-progress.
        print_progress(count=start_index, max_count=num_images)

        # End-index for this batch.
        end_index = start_index + batch_size

        # Ensure end-index is within bounds.
        if end_index > num_images:
            end_index = num_images

        # The last batch may have a different batch-size.
        current_batch_size = end_index - start_index

        # Load all the images in the batch.
        for i, filename in enumerate(filenames[start_index:end_index]):
            # Path for the image-file.
            path = os.path.join(data_dir, filename)

            # Load and resize the image.
            # This returns the image as a numpy-array.
            img = load_image(path, size=img_size)

            # Save the image for later use.
            image_batch[i] = img

        # Use the pre-trained image-model to process the image.
        # Note that the last batch may have a different size,
        # so we only use the relevant images.
        transfer_values_batch = \
            image_model_transfer.predict(image_batch[0:current_batch_size])

        # Save the transfer-values in the pre-allocated array.
        transfer_values[start_index:end_index] = \
            transfer_values_batch[0:current_batch_size]

        # Increase the index for the next loop-iteration.
        start_index = end_index

    # Print newline.
    print()

    return transfer_values
```

Procesarea imagnilor de antrenare

In [21]:

```
def process_images_train():
    print("Processing {0} images in training-set ...".format(len(filenames_train)))

    # Path for the cache-file.
    cache_path = os.path.join(coco.data_dir,
                              "transfer_values_train.pkl")

    # If the cache-file already exists then reload it,
    # otherwise process all images and save their transfer-values
    # to the cache-file so it can be reloaded quickly.
    transfer_values = cache(cache_path=cache_path,
```

```
                            fn=process_images,
                            data_dir=coco.train_dir,
                            filenames=filenames_train)

    return transfer_values
```

Procesarea imagnilor de validare

In [22]:

```
def process_images_val():
    print("Processing {0} images in validation-set ...".format(len(filenames_val)))

    # Path for the cache-file.
    cache_path = os.path.join(coco.data_dir, "transfer_values_val.pkl")

    # If the cache-file already exists then reload it,
    # otherwise process all images and save their transfer-values
    # to the cache-file so it can be reloaded quickly.
    transfer_values = cache(cache_path=cache_path,
                            fn=process_images,
                            data_dir=coco.val_dir,
                            filenames=filenames_val)

    return transfer_values
```

Procesam toate imaginile din setul de antrenament și le salvam valorile de transfer într-un fișier cache. Acest proces a durat aproximativ 90 de minute pentru o GPU 1060 GTX.

In [23]:

```
%%time
transfer_values_train = process_images_train()
print("dtype:", transfer_values_train.dtype)
print("shape:", transfer_values_train.shape)
```

```
Processing 118287 images in training-set ...
- Data loaded from cache-file: data/coco/transfer_values_train.pkl
dtype: float16
shape: (118287, 4096)
Wall time: 1.88 s
```

Procesam toate imaginile din setul de validare și le salvam valorile de transfer într-un fișier cache. Acest proces a durat aproximativ 10 minute pentru un GPU 1060 GTX.

In [24]:

```
%%time
transfer_values_val = process_images_val()
print("dtype:", transfer_values_val.dtype)
print("shape:", transfer_values_val.shape)
```

```
Processing 5000 images in validation-set ...
- Data loaded from cache-file: data/coco/transfer_values_val.pkl
dtype: float16
shape: (5000, 4096)
Wall time: 148 ms
```

Tokenizer¶

Retelele neuronale nu pot lucra direct pe cuvinte. Pentru a putea obtine o descriere textuala vom folosi un procedeu denumit tokenizare. Vom desemna un vocabular de 10000 de cuvinte si fiecarui cuvant ii vom atribui un numar intreg. Inceputul si finalul propozitiei vor fi formate din 2 cuvinte care sunt putin probabile sa faca parte din setul de antrenare.

In [25]:

```
mark_start = 'ssss '
mark_end = ' eeee'
```

Functia va adauga markerul de inceput si final capturilor din setul de antrenare si verificare

In [26]:

```
def mark_captions(captions_listlist):
    captions_marked = [[mark_start + caption + mark_end
                        for caption in captions_list]
                        for captions_list in captions_listlist]

    return captions_marked
```

In [27]:

```
captions_train_marked = mark_captions(captions_train)
captions_train_marked[0]
```

Out[27]:

```
['ssss Closeup of bins of food that include broccoli and bread. eeee',
 'ssss A meal is presented in brightly colored plastic trays. eeee',
 'ssss there are containers filled with different kinds of foods eeee',
 'ssss Colorful dishes holding meat, vegetables, fruit, and bread. eeee',
 'ssss A bunch of trays that have different food. eeee']
```

In [28]:

```
def flatten(captions_listlist):
    captions_list = [caption
                     for captions_list in captions_listlist
                     for caption in captions_list]

    return captions_list
```

Transformam setul de antrenare intr-un vector de numere intregi

In [29]:

```
captions_train_flat = flatten(captions_train_marked)
```

Setam numarul maxim de cuvinte distincte cu care vom antrena.

In [30]:

```
num_words = 10000
```

Un wrapper asupra functiei de tokenizare deja existenta in Keras

In [31]:

```
class TokenizerWrap(Tokenizer):
    """Wrap the Tokenizer-class from Keras with more functionality."""
```

```
    def __init__(self, texts, num_words=None):
        """
        :param texts: List of strings with the data-set.
        :param num_words: Max number of words to use.
        """

        Tokenizer.__init__(self, num_words=num_words)

        # Create the vocabulary from the texts.
        self.fit_on_texts(texts)

        # Create inverse lookup from integer-tokens to words.
        self.index_to_word = dict(zip(self.word_index.values(),
                                      self.word_index.keys()))

    def token_to_word(self, token):
        """Lookup a single word from an integer-token."""

        word = " " if token == 0 else self.index_to_word[token]
        return word

    def tokens_to_string(self, tokens):
        """Convert a list of integer-tokens to a string."""

        # Create a list of the individual words.
        words = [self.index_to_word[token]
                 for token in tokens
                 if token != 0]

        # Concatenate the words to a single string
        # with space between all the words.
        text = " ".join(words)

        return text

    def captions_to_tokens(self, captions_listlist):
        """
        Convert a list-of-list with text-captions to
        a list-of-list of integer-tokens.
        """

        # Note that text_to_sequences() takes a list of texts.
        tokens = [self.texts_to_sequences(captions_list)
                  for captions_list in captions_listlist]

        return tokens
```

Cream o instanta a clasei

In [32]:

```
%%time
tokenizer = TokenizerWrap(texts=captions_train_flat,
                          num_words=num_words)
```

```
Wall time: 8.83 s
```

Obtinem indexul markerului de start('ssss').

In [33]:

```
token_start = tokenizer.word_index[mark_start.strip()]
token_start
```

Out[33]:

```
2
```

Obtinem indexul markerului de final('eeee').

In [34]:

```
token_end = tokenizer.word_index[mark_end.strip()]
token_end
```

Out[34]:

```
3
```

In [35]:

```
%%time
tokens_train = tokenizer.captions_to_tokens(captions_train_marked)
```

```
Wall time: 8.22 s
```

Cuvintele convertite in numere intregi

In [36]:

```
tokens_train[0]
```

Out[36]:

```
[[2, 841, 5, 2864, 5, 61, 26, 1984, 238, 9, 433, 3],
 [2, 1, 429, 10, 3310, 7, 1025, 390, 501, 1110, 3],
 [2, 63, 19, 993, 143, 8, 190, 958, 5, 743, 3],
 [2, 299, 725, 25, 343, 208, 264, 9, 433, 3],
 [2, 1, 170, 5, 1110, 26, 446, 190, 61, 3]]
```

Propozitiile corespondente

In [37]:

```
captions_train_marked[0]
```

Out[37]:

```
['ssss Closeup of bins of food that include broccoli and bread. eeee',
 'ssss A meal is presented in brightly colored plastic trays. eeee',
 'ssss there are containers filled with different kinds of foods eeee',
 'ssss Colorful dishes holding meat, vegetables, fruit, and bread. eeee',
 'ssss A bunch of trays that have different food. eeee']
```

Generatorul de date¶

Fiecare imagine are cel putin 5 descrieri ale continutului. Reteaua neuronala va fi antrenata cu batch-uri ale valorilor intermediare venite de la vgg16 si o secvente de numere intregi reprezentand descrierea lor. Dacă ar trebui să avem matricile de numpy potrivite pentru setul de antrenament (adica ce obtinem la intrarea retelei recurente cu cele cel putin 5 descrieri valide), fie

ar trebui să folosim o singură legendă pentru fiecare imagine și să ignorăm restul acestor date valoroase, fie ar trebui să repetăm valorile de transfer de imagine pentru fiecare dintre legendele, care ar irosi multă memorie.

O soluție mai bună este crearea unui generator de date personalizat pentru Keras care va crea un lot de date cu valori de transfer selectate aleatoriu și secvențe de simboluri.

Această funcție returnează o listă de secvențe ale tokenizer-ului aleatoriu pentru imagini cu indicii date în setul de antrenament.

In [38]:

```
def get_random_caption_tokens(idx):
    """
    Given a list of indices for images in the training-set,
    select a token-sequence for a random caption,
    and return a list of all these token-sequences.
    """

    # Initialize an empty list for the results.
    result = []

    # For each of the indices.
    for i in idx:
        # The index i points to an image in the training-set.
        # Each image in the training-set has at least 5 captions
        # which have been converted to tokens in tokens_train.
        # We want to select one of these token-sequences at random.

        # Get a random index for a token-sequence.
        j = np.random.choice(len(tokens_train[i]))

        # Get the j'th token-sequence for image i.
        tokens = tokens_train[i][j]

        # Add this token-sequence to the list of results.
        result.append(tokens)

    return result
```

This generator function creates random batches of training-data for use in training the neural network.

In [39]:

```
def batch_generator(batch_size):
    """
    Generator function for creating random batches of training-data.

    Note that it selects the data completely randomly for each
    batch, corresponding to sampling of the training-set with
    replacement. This means it is possible to sample the same
    data multiple times within a single epoch - and it is also
    possible that some data is not sampled at all within an epoch.
    However, all the data should be unique within a single batch.
    """

    # Infinite loop.
    while True:
```

```
        # Get a list of random indices for images in the training-set.
        idx = np.random.randint(num_images_train,
                                size=batch_size)

        # Get the pre-computed transfer-values for those images.
        # These are the outputs of the pre-trained image-model.
        transfer_values = transfer_values_train[idx]

        # For each of the randomly chosen images there are
        # at least 5 captions describing the contents of the image.
        # Select one of those captions at random and get the
        # associated sequence of integer-tokens.
        tokens = get_random_caption_tokens(idx)

        # Count the number of tokens in all these token-sequences.
        num_tokens = [len(t) for t in tokens]

        # Max number of tokens.
        max_tokens = np.max(num_tokens)

        # Pad all the other token-sequences with zeros
        # so they all have the same length and can be
        # input to the neural network as a numpy array.
        tokens_padded = pad_sequences(tokens,
                                      maxlen=max_tokens,
                                      padding='post',
                                      truncating='post')

        # Further prepare the token-sequences.
        # The decoder-part of the neural network
        # will try to map the token-sequences to
        # themselves shifted one time-step.
        decoder_input_data = tokens_padded[:, 0:-1]
        decoder_output_data = tokens_padded[:, 1:]

        # Dict for the input-data. Because we have
        # several inputs, we use a named dict to
        # ensure that the data is assigned correctly.
        x_data = \
        {
            'decoder_input': decoder_input_data,
            'transfer_values_input': transfer_values
        }

        # Dict for the output-data.
        y_data = \
        {
            'decoder_output': decoder_output_data
        }

        yield (x_data, y_data)
```

Pasi per epoca¶

Dat fiind modul de generare al datelor putem intalni cazuri in care in aceeasi epoca se repeta o imagine de mai multe ori cu descrieri diferite, sau o imagine nu este utilizata deloc. Vom folosi totusi termenul de epoca pentru a masura timpul de antrenament al modelului

In [40]:

```
num_captions_train = [len(captions) for captions in captions_train]
```

Numarul total de descrieri din setul de antrenare

In [41]:

```
total_num_captions_train = np.sum(num_captions_train)
```

Crearea Retelei Recurente¶

Vom crea o retea recurenta ce va primi la intrare valorile de transfer de la reteaua convolutionala ce are ca scop sa determine obiectele din imagine, cat si propozitia construita deja pentru a genera o descriere textuala a imaginii de la intrare. Aceasta retea este decoder-ul arhitecturii noastre.

Acest model va fi insa detasat de primul model. El va fi compilat si antrenat separat. Acesta este alcatuit din 3 straturi de

Gated Recurent Units a caror stare interna este:

In [42]:

```
state_size = 512
```

Stratul de embedding converteste valorile intregi de la intrare in vectori de lungimea:

In [43]:

```
embedding_size = 128
```

Valorile de intrare pentru decoder

In [44]:

```
transfer_values_input = Input(shape=(transfer_values_size,),
                              name='transfer_values_input')
```

La iesirea penultimului strat din VGG16 avem 4096 valori dar GRU-ul nostru are doar 512 stari interne. Vom reduce numarul de valori cu ajutorul unui strat dens intermediar de 512 neuroni ce va preceda arhitectura recurenta.

Folosim functia de activare *tanh* pentru a avea valori cuprinse intre -1 si 1.

In [45]:

```
decoder_transfer_map = Dense(state_size,
                             activation='tanh',
                             name='decoder_transfer_map')
```

Intrarea in acest decoder poate avea lungimi arbitrare

In [46]:

```
decoder_input = Input(shape=(None, ), name='decoder_input')
```

In [47]:

```
decoder_embedding = Embedding(input_dim=num_words,
                              output_dim=embedding_size,
                              name='decoder_embedding')
```

Cream secventa de straturi recurente.

In [48]:

```
decoder_gru1 = GRU(state_size, name='decoder_gru1',
                   return_sequences=True)
decoder_gru2 = GRU(state_size, name='decoder_gru2',
                   return_sequences=True)
decoder_gru3 = GRU(state_size, name='decoder_gru3',
                   return_sequences=True)
```

La iesire vom avea un vector de forma *[batch_size, sequence_length, state_size]*. Fiecare cuvant ce va alcatui descrierea va veni sub forma unui vector(format din activarile lui GRU3). Aceste secvente trebuie convertite in numere intregi pentru a putea obtine din maparea numar-cuvant anterior definita, o propozitie.

Aceasta mapare se va face cu ajutorul unui strat de iesire ce va avea numarul de neuroni egal cu numarul vocabularului nostru

In [49]:

```
decoder_dense = Dense(num_words,
                      activation='softmax',
                      name='decoder_output')
```

Conectarea intrarii si crearea modelului de antrenare¶

Conectarea retelei recurente la encoder ul nostru(VGG16)

In [50]:

```
def connect_decoder(transfer_values):
    # Map the transfer-values so the dimensionality matches
    # the internal state of the GRU layers. This means
    # we can use the mapped transfer-values as the initial state
    # of the GRU layers.
    initial_state = decoder_transfer_map(transfer_values)

    # Start the decoder-network with its input-layer.
    net = decoder_input

    # Connect the embedding-layer.
    net = decoder_embedding(net)

    # Connect all the GRU layers.
    net = decoder_gru1(net, initial_state=initial_state)
    net = decoder_gru2(net, initial_state=initial_state)
    net = decoder_gru3(net, initial_state=initial_state)

    # Connect the final dense layer that converts to
    # one-hot encoded arrays.
    decoder_output = decoder_dense(net)

    return decoder_output
```

In [51]:

```
decoder_output = connect_decoder(transfer_values=transfer_values_input)
```

```
decoder_model = Model(inputs=[transfer_values_input, decoder_input],
                      outputs=[decoder_output])
```

Compilarea modelului¶

Stratul de iesire va fi reprezentat de un set de neuroni peste care se va aplica o decodare de tipul 'one-hot'. Cu toate aceastea validarea contine un set de valori tokenizate. Ca functie loss, vom utiliza sparse cross-entropy ce face o conversie interna intre o valoare intreaga si un vector cu codare one hot pentru fiecare cuvant.

Am testat aici atat cu Adam optimizer cat si cu RMSprop si am ajuns la concluzia ca cel din urma are perfomantele cele mai mari. Adam functioneaza bine in primele epoci insa esueaza in a face o convergenta dupa ceva vreme.

In [52]:

```
decoder_model.compile(optimizer=tf.keras.optimizers.RMSprop(
    learning_rate=0.001, rho=0.9, momentum=0.0, epsilon=1e-07, centered=False),
                      loss='sparse_categorical_crossentropy')
```

Callback Functions¶

In [53]:

```
path_checkpoint = 'captionThisModel.h5'
callback_checkpoint = ModelCheckpoint(filepath=path_checkpoint,
                                      verbose=1,
                                      save_weights_only=True)
```

Log-uri de TensorBoard(Inca nu m-am uitat peste ele). Pot fi utile in lucrarea teoretica

In [54]:

```
callback_tensorboard = TensorBoard(log_dir='./captionThis_logs/',
                                   histogram_freq=0,
                                   write_graph=False)
```

In [55]:

```
callbacks = [callback_checkpoint, callback_tensorboard]
```

Load Checkpoint¶

Incarcarea ultimului model.

In [56]:

```
try:
    decoder_model.load_weights(path_checkpoint)
except Exception as error:
    print("Error trying to load checkpoint.")
    print(error)
```

Train the Model¶

Vom antrena modelul decoder-ului.

O epoca a durat 4-5 ore pe un GTX 1060 GPU, 6GB cu un batch size de 3000.

In [57]:

```
batch_size = 3000
generator = batch_generator(batch_size=batch_size)
steps_per_epoch = int(total_num_captions_train / batch_size)
steps_per_epoch
```

Out[57]:

```
197
```

In [141]:

```
%%time
decoder_model.fit(x=generator,
                  steps_per_epoch=steps_per_epoch,
                  epochs=10,
                  callbacks=callbacks)
```

```
Epoch 1/10
295/295 [==============================] - ETA: 0s - loss: 0.6806
Epoch 00001: saving model to captionThisModel.h5
295/295 [==============================] - 17519s 59s/step - loss: 0.6806
Epoch 2/10
295/295 [==============================] - ETA: 0s - loss: 0.6770
Epoch 00002: saving model to captionThisModel.h5
295/295 [==============================] - 18075s 61s/step - loss: 0.6770
Epoch 3/10
295/295 [==============================] - ETA: 0s - loss: 0.6564
Epoch 00003: saving model to captionThisModel.h5
295/295 [==============================] - 17881s 61s/step - loss: 0.6564
Epoch 4/10
295/295 [==============================] - ETA: 0s - loss: 0.6512
Epoch 00004: saving model to captionThisModel.h5
295/295 [==============================] - 19080s 65s/step - loss: 0.6512
Epoch 5/10
295/295 [==============================] - ETA: 0s - loss: 0.6546
Epoch 00005: saving model to captionThisModel.h5
295/295 [==============================] - 18077s 61s/step - loss: 0.6546
Epoch 6/10
295/295 [==============================] - ETA: 0s - loss: 0.6537
Epoch 00006: saving model to captionThisModel.h5
295/295 [==============================] - 17695s 60s/step - loss: 0.6537
Epoch 7/10
295/295 [==============================] - ETA: 0s - loss: 0.6550
Epoch 00007: saving model to captionThisModel.h5
295/295 [==============================] - 18391s 62s/step - loss: 0.6550
Epoch 8/10
295/295 [==============================] - ETA: 0s - loss: 0.6620
Epoch 00008: saving model to captionThisModel.h5
295/295 [==============================] - 17478s 59s/step - loss: 0.6620
Epoch 9/10
295/295 [==============================] - ETA: 0s - loss: 0.6572
Epoch 00009: saving model to captionThisModel.h5
295/295 [==============================] - 17055s 58s/step - loss: 0.6572
Epoch 10/10
295/295 [==============================] - ETA: 0s - loss: 0.6526
Epoch 00010: saving model to captionThisModel.h5
295/295 [==============================] - 17615s 60s/step - loss: 0.6526
Wall time: 2d 1h 50min 16s
```

Generarea descrierilor¶

Functia incarca o imagine si genereaza o descriere

In [58]:

```
def generate_caption(image_path, max_tokens=30):
    """
    Generate a caption for the image in the given path.
    The caption is limited to the given number of tokens (words).
    """

    # Load and resize the image.
    image = load_image(image_path, size=img_size)

    # Expand the 3-dim numpy array to 4-dim
    # because the image-model expects a whole batch as input

    # so we give it a batch with just one image.
    image_batch = np.expand_dims(image, axis=0)

    # Process the image with the pre-trained image-model
    # to get the transfer-values.
    transfer_values = image_model_transfer.predict(image_batch)

    # Pre-allocate the 2-dim array used as input to the decoder.
    # This holds just a single sequence of integer-tokens,
    # but the decoder-model expects a batch of sequences.
    shape = (1, max_tokens)
    decoder_input_data = np.zeros(shape=shape, dtype=np.int)

    # The first input-token is the special start-token for 'ssss '.
    token_int = token_start

    # Initialize an empty output-text.
    output_text = ''

    # Initialize the number of tokens we have processed.
    count_tokens = 0

    # While we haven't sampled the special end-token for ' eeee'
    # and we haven't processed the max number of tokens.
    while token_int != token_end and count_tokens < max_tokens:
        # Update the input-sequence to the decoder
        # with the last token that was sampled.
        # In the first iteration this will set the
        # first element to the start-token.
        decoder_input_data[0, count_tokens] = token_int

        # Wrap the input-data in a dict for clarity and safety,
        # so we are sure we input the data in the right order.
        x_data = {
            'transfer_values_input': transfer_values,
            'decoder_input': decoder_input_data
        }

        # Note that we input the entire sequence of tokens
        # to the decoder. This wastes a lot of computation
        # because we are only interested in the last input
        # and output. We could modify the code to return
        # the GRU-states when calling predict() and then
        # feeding these GRU-states as well the next time
```

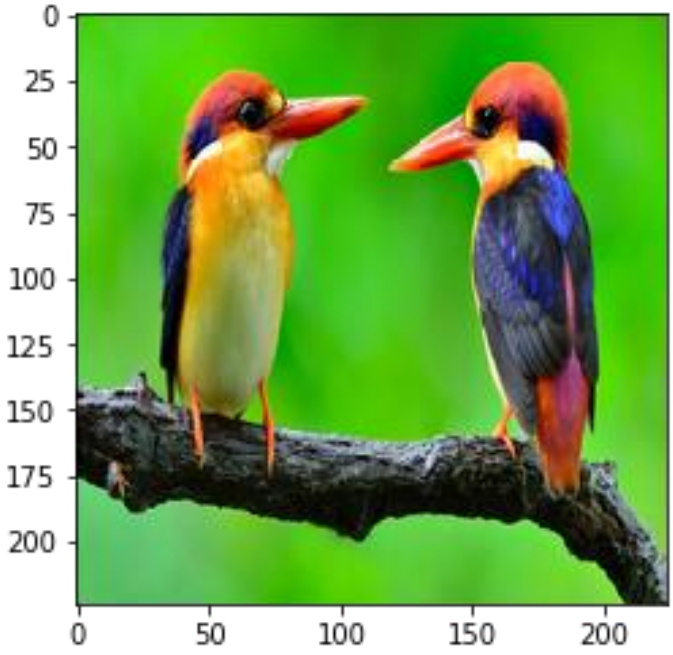
```
        # we call predict(), but it would make the code
        # much more complicated.

        # Input this data to the decoder and get the predicted output.
        decoder_output = decoder_model.predict(x_data)

        # Get the last predicted token as a one-hot encoded array.
        # Note that this is not limited by softmax, but we just
        # need the index of the largest element so it doesn't matter.
        token_onehot = decoder_output[0, count_tokens, :]

        # Convert to an integer-token.
        token_int = np.argmax(token_onehot)

        # Lookup the word corresponding to this integer-token.
        sampled_word = tokenizer.token_to_word(token_int)

        # Append the word to the output-text.
        output_text += " " + sampled_word

        # Increment the token-counter.
        count_tokens += 1

    # This is the sequence of tokens output by the decoder.
    output_tokens = decoder_input_data[0]

    # Plot the image.
    plt.imshow(image)
    plt.show()
    output_text=output_text[1].upper()+output_text[2:-4]

    # Print the predicted caption.
    print("Predicted caption:")
    print(output_text)
    print()
```

Example¶

O poza cu 2 pasari

In [59]:

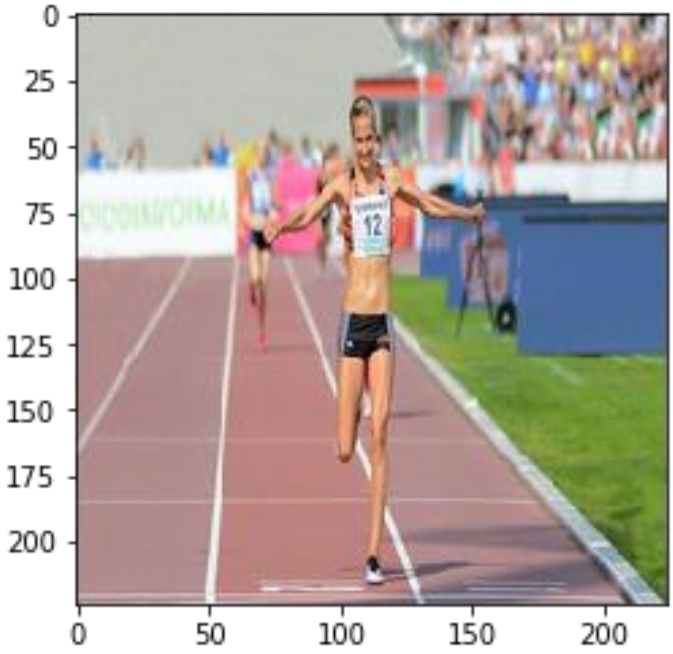
```
generate_caption("test/birds.jpg")
```



O poza cu un atlet

In [60]:

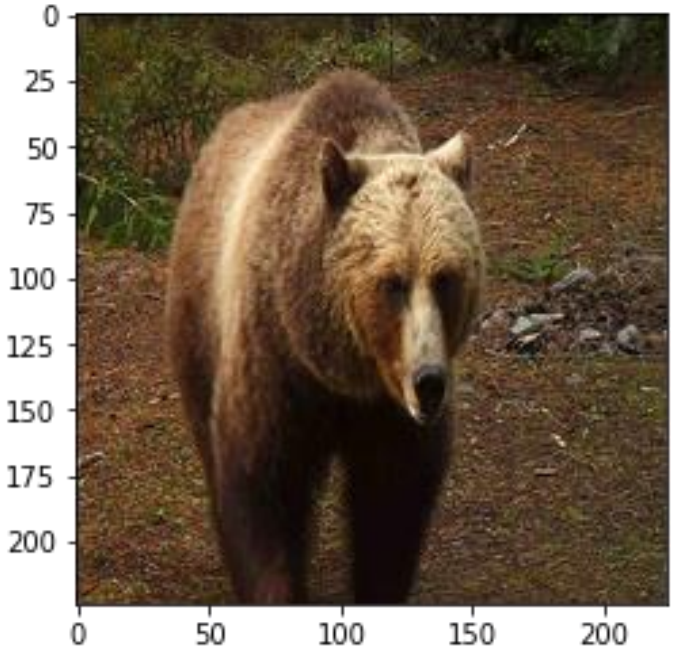
```
generate_caption("test/athlete.jpg")
```



O poza cu un urs

In [61]:

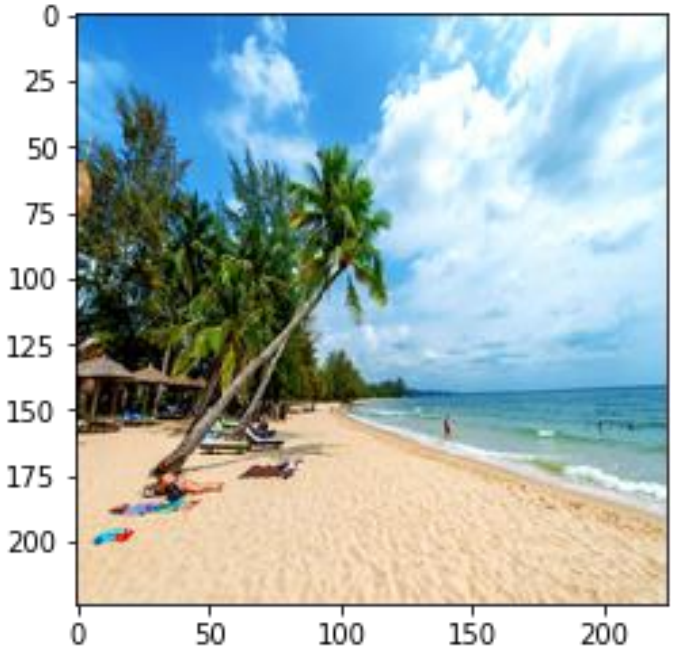
```
generate_caption("test/bear.jpg")
```



O poza cu o plaja

In [62]:

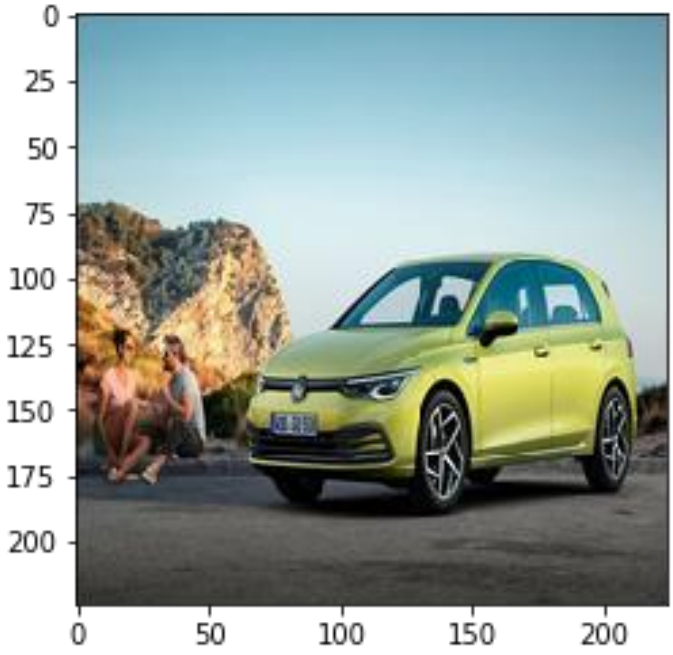
```
generate_caption("test/beach.jpg")
```



O poza cu o masina

In [63]:

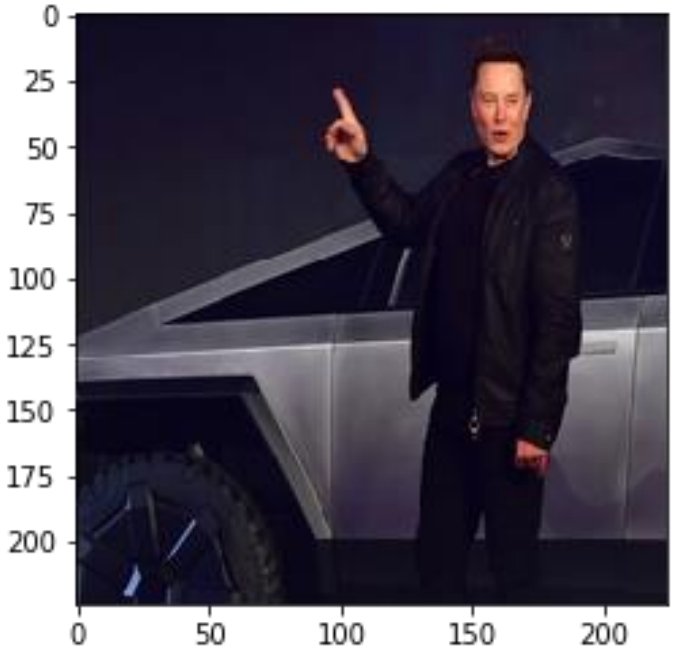
```
generate_caption("test/golf.jpg")
```



O poza cu un om

In [64]:

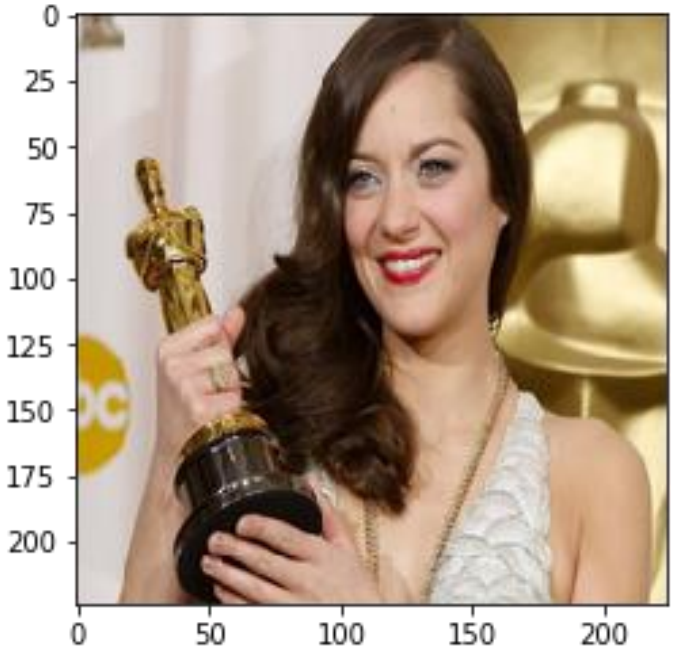
```
generate_caption("test/musk.jpeg")
```



O poza cu o femeie si un obiect

In [66]:

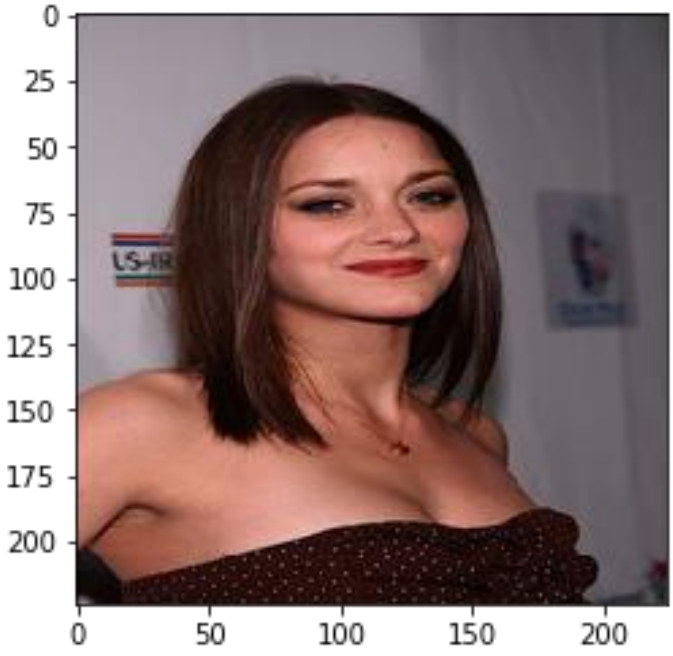
```
generate_caption("test/marion.jpg")
```



O poza doar cu o femeie

In [67]:

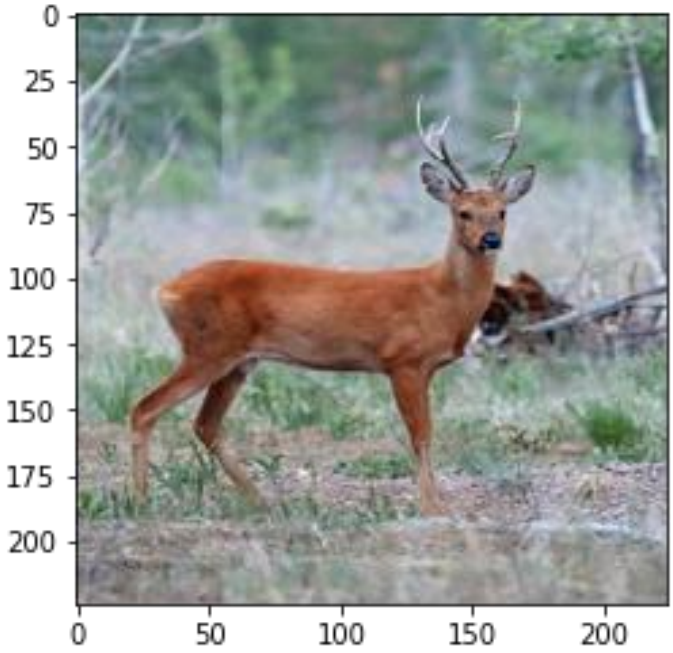
```
generate_caption("test/marion1.jpg")
```



Sa incercam cu poza unei capre

In [68]:

```
generate_caption("test/deer.jpg")
```



Urmeaza sa definim o functie care sa ne afiseze o imagine din setul de antrenare/validare si rezultatul pe care il produce modelul nostru.

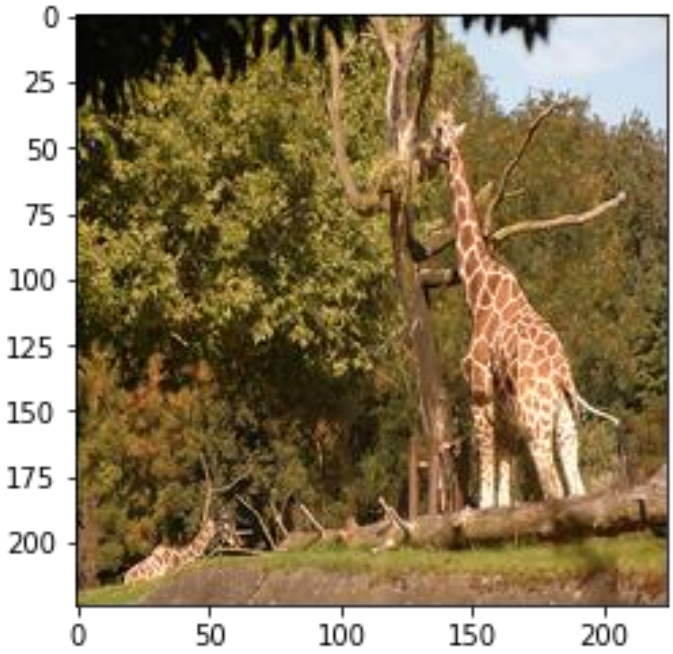
In [73]:

```
def generate_caption_coco(idx, train=False):
    """
    Generate a caption for an image in the COCO data-set.
    Use the image with the given index in either the
    training-set (train=True) or validation-set (train=False).
    """

    if train:
        # Use image and captions from the training-set.
        data_dir = coco.train_dir
        filename = filenames_train[idx]
        captions = captions_train[idx]
    else:
        # Use image and captions from the validation-set.
        data_dir = coco.val_dir
        filename = filenames_val[idx]
        captions = captions_val[idx]

    # Path for the image-file.
    path = os.path.join(data_dir, filename)

    # Use the model to generate a caption of the image.
    generate_caption(image_path=path)

    # Print the true captions from the data-set.
    print("True captions:")
    for caption in captions:
        print(caption)
```

Alta imagine din setul de antrenare

In [74]:

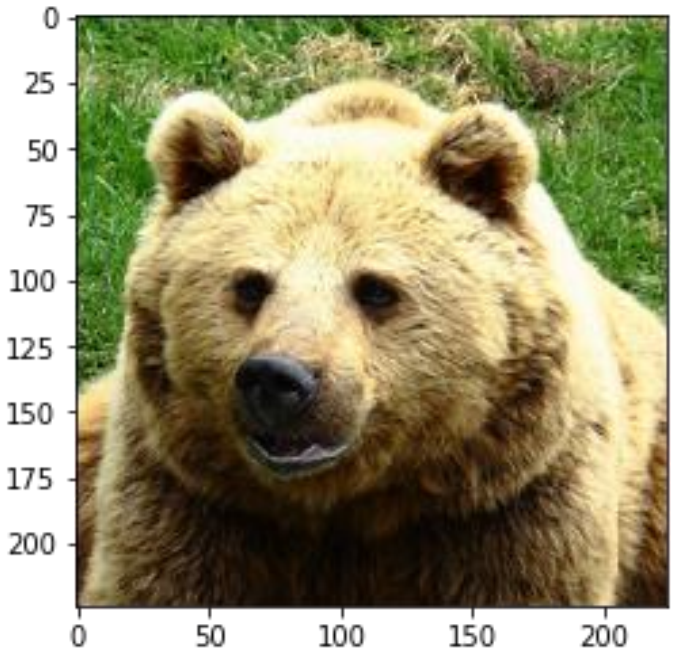
```
generate_caption_coco(idx=1, train=True)
```



Alta imagine din setul de validare

In [75]:

```
generate_caption_coco(idx=1, train=False)
```



Acuratețea

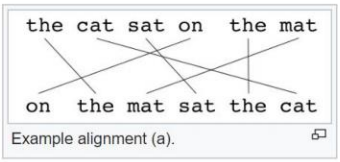
METEOR (2005)

It is based on an explicit word-to-word matching between the MT output being evaluated and one or more reference translations. It can also match synonyms.

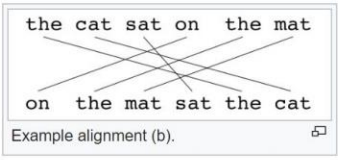
Calculate mapping between the candidate and reference caption. In conflict, mapping between least crosses is selected.

$$F_{mean} = \frac{10PR}{R + 9P}$$

Extend it to longer n-grams with a pernaly for matching.



Example alignment (a).



Example alignment (b).

In [204]:

```
import nltk
nltk.data.path=[r'C:\Users\George\Desktop\Dizertatie\Python']
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

In [220]:

```
def MeteorScore():
    data_dir = coco.val_dir
    acc=0
    for idx in range(num_images_val):
        filename = filenames_val[idx]
        caption_val_set = captions_val[idx]
        path=os.path.join(data_dir, filename)
        caption=generate_caption(image_path=path,plot=False,printc=False)
        acc+=round(nltk.meteor(caption_val_set, caption),4)
    return acc/num_images_val
```

In [212]:

```
print('METEOR score is {}'.format(round(MeteorScore(),4)))
```

```
METEOR score is 0.3408
```

BLEU (2002)

Precision = 7/7

Modified precision= 2/7

Max . score is limited by freq. in reference.

Candidate	the	the	the	the	the	the	the
Reference 1	the	cat	is	on	the	mat	
Reference 2	there	is	a	cat	on	the	mat

Output is the geometric mean of n-gram score with a brevity penalty to discourage shorter translation.

Definition:

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n\text{-}gram \in C} Count_{clip}(n\text{-}gram)}{\sum_{C' \in \{Candidates\}} \sum_{n\text{-}gram' \in C'} Count(n\text{-}gram')}$$

Ref: Papineni, Kishore, et al. "BLEU: a method for automatic evaluation of machine translation." *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2002.

Image credits: https://en.wikipedia.org/wiki/BLEU

In [ ]:

```
import nltk.translate.bleu_score as bleu
```

In [213]:

```
def BleuScore():
    data_dir = coco.val_dir
    acc=0
    for idx in range(num_images_val):
        filename = filenames_val[idx]
        caption_val_set = captions_val[idx]
        path=os.path.join(data_dir, filename)
        caption=generate_caption(image_path=path,plot=False,printc=False)
        acc+=round(bleu.sentence_bleu(caption_val_set, caption),4)
    return acc/num_images_val
```

In [218]:

```
print('BLEU score is {}'.format(round(BleuScore(),4)))
```

```
BLEU score is 0.3408
```

Salvam Modelul

In [76]:

```
decoder_model.summary()
```

```
Model: "model_1"
_________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=========================================================================================
==========
decoder_input (InputLayer)      [(None, None)]       0
_________________________________________________________________________________________
transfer_values_input (InputLay [(None, 4096)]       0
```

```
__________________________________________________________________________________________
___________
decoder_embedding (Embedding)   (None, None, 128)    1280000     decoder_input[0][0]
__________________________________________________________________________________________
___________
decoder_transfer_map (Dense)    (None, 512)          2097664     transfer_values_input[
0][0]
__________________________________________________________________________________________
___________
decoder_gru1 (GRU)              (None, None, 512)    986112      decoder_embedding[0][0
]
                                                                 decoder_transfer_map[0
][0]
__________________________________________________________________________________________
___________
decoder_gru2 (GRU)              (None, None, 512)    1575936     decoder_gru1[0][0]
                                                                 decoder_transfer_map[0
][0]
__________________________________________________________________________________________
___________
decoder_gru3 (GRU)              (None, None, 512)    1575936     decoder_gru2[0][0]
                                                                 decoder_transfer_map[0
][0]
__________________________________________________________________________________________
___________
decoder_output (Dense)          (None, None, 10000)  5130000     decoder_gru3[0][0]
==========================================================================================
===========
Total params: 12,645,648
Trainable params: 12,645,648
Non-trainable params: 0
__________________________________________________________________________________________
___________
```

In [77]:

```
decoder_model.save(r'C:\Users\George\Desktop\Dizertatie\Python\Caption Model\finalModel
')
```

```
WARNING:tensorflow:From C:\Users\George\Anaconda3\envs\tf2\lib\site-packages\tensorflow
\python\ops\resource_variable_ops.py:1817: calling BaseResourceVariable.__init__ (from
tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be
removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: C:\Users\George\Desktop\Dizertatie\Python\Caption Mo
del\finalModel\assets
```

Verificam ca modelul se incă

In [78]:

```
newModel=tf.keras.models.load_model('finalModel')
newModel.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________
___________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
===========
decoder_input (InputLayer)      [(None, None)]       0
```

```
__________________________________________________________________________________________
___________
transfer_values_input (InputLay [(None, 4096)]       0
__________________________________________________________________________________________
___________
decoder_embedding (Embedding)   (None, None, 128)    1280000     decoder_input[0][0]
__________________________________________________________________________________________
___________
decoder_transfer_map (Dense)    (None, 512)          2097664     transfer_values_input[
0][0]
__________________________________________________________________________________________
___________
decoder_gru1 (GRU)              (None, None, 512)    986112      decoder_embedding[0][0
]
                                                                 decoder_transfer_map[0
][0]
__________________________________________________________________________________________
___________
decoder_gru2 (GRU)              (None, None, 512)    1575936     decoder_gru1[0][0]
                                                                 decoder_transfer_map[0
][0]
__________________________________________________________________________________________
___________
decoder_gru3 (GRU)              (None, None, 512)    1575936     decoder_gru2[0][0]
                                                                 decoder_transfer_map[0
][0]
__________________________________________________________________________________________
___________
decoder_output (Dense)          (None, None, 10000)  5130000     decoder_gru3[0][0]
==========================================================================================
===========
Total params: 12,645,648
Trainable params: 12,645,648
Non-trainable params: 0
__________________________________________________________________________________________
___________
```

Anexa 2 Server.py

```
from __future__ import division, print_function
# coding=utf-8
import sys
import os
import glob
import re
import numpy as np

import shutil

import Caption as caption

# Flask utils
from flask import Flask, redirect, url_for, request, render_template, jsonify
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer

UPLOAD_FOLDER = 'uploads/'

# Define a flask app
app = Flask(__name__)

app.secret_key = "secret key"

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
# Model saved with Keras model.save()
MODEL_PATH = 'finalModel'

# Load your trained model
model=caption.Caption(MODEL_PATH)

ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg'])

print('Model loaded. Check http://127.0.0.1:5000/')
```

```
def model_predict(img_path, model):

    preds = model.generate_caption(img_path)
    return preds


@app.route('/', methods=['GET'])
def index():
    # Main page
    return render_template('index.html')



def allowed_file(filename):
        return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


@app.route('/predict', methods=['GET', 'POST'])
def upload():
    basepath = os.path.dirname(__file__)
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['file']

        # Save the file to ./uploads

        file_path = os.path.join(
            basepath, app.config['UPLOAD_FOLDER'], secure_filename(f.filename))
        f.save(file_path)

        # Make prediction
        preds = model_predict(file_path, model)
        resp = jsonify({'message' : preds})
        resp.status_code = 400
        return preds
```

```
    return None


@app.route('/api/predict', methods=['POST'])
def upload_file():
    basepath = os.path.dirname(__file__)
    # check if the post request has the file part
    if 'file' not in request.files:
        print([x for x in request.files.items()])
        resp = jsonify({'message' : 'No picture in the request'})
        resp.status_code = 400
        return resp
    file = request.files['file']
    if file!=None and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file_path = os.path.join(basepath, app.config['UPLOAD_FOLDER'],
secure_filename(filename))
        file.save(file_path)
        # Make prediction
        preds = model_predict(file_path, model)
        resp = jsonify({'message' : preds})
        resp.status_code = 201
        return resp
    else:
        resp = jsonify({'message' : 'Failed'})
        resp.status_code = 400
        return resp



if __name__ == '__main__':
    app.run()
```

Anexa 3 Caption.py

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
```

```
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Dense, GRU, Embedding
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.callbacks import ModelCheckpoint
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
import coco

import time

class TokenizerWrap(Tokenizer):
    """Wrap the Tokenizer-class from Keras with more functionality."""

    def __init__(self, texts, num_words=None):
        """
        :param texts: List of strings with the data-set.
        :param num_words: Max number of words to use.
        """

        Tokenizer.__init__(self, num_words=num_words)

        # Create the vocabulary from the texts.
        self.fit_on_texts(texts)

        # Create inverse lookup from integer-tokens to words.
        self.index_to_word = dict(zip(self.word_index.values(),
                                      self.word_index.keys()))

    def token_to_word(self, token):
        """Lookup a single word from an integer-token."""

        word = " " if token == 0 else self.index_to_word[token]
        return word

    def tokens_to_string(self, tokens):
        """Convert a list of integer-tokens to a string."""
```

```
        # Create a list of the individual words.
        words = [self.index_to_word[token]
                 for token in tokens
                 if token != 0]

        # Concatenate the words to a single string
        # with space between all the words.
        text = " ".join(words)

        return text

    def captions_to_tokens(self, captions_listlist):
        """
        Convert a list-of-list with text-captions to
        a list-of-list of integer-tokens.
        """



        tokens = [self.texts_to_sequences(captions_list)
                  for captions_list in captions_listlist]

        return tokens

class Caption():

    def __init__(self,model_path):
        print('='*10+' Initiating the caption model '+10*'=')
        self.img_size=(224, 224)
        num_words = 10000
        image_model = VGG16(include_top=True, weights='imagenet')
        transfer_layer = image_model.get_layer('fc2')
        self.image_model_transfer = Model(inputs=image_model.input, outputs=transfer_layer.output)
        self.decoder_model= tf.keras.models.load_model(model_path)
        self.mark_start = 'ssss '
        self.mark_end = ' eeee'
```

```
        self.token_start = 2
        self.token_end = 3
        _, filenames_train, captions_train = coco.load_records(train=True)
        captions_train_marked = self.mark_captions(captions_train)
        captions_train_flat = self.flatten(captions_train_marked)
        self.tokenizer = TokenizerWrap(texts=captions_train_flat, num_words=num_words)
        print(self.decoder_model.summary())

    def mark_captions(self,captions_listlist):
        captions_marked = [[self.mark_start + caption + self.mark_end   for caption in captions_list] for captions_list in captions_listlist]
        return captions_marked



    def connect_decoder(self,transfer_values):
        """
        Map the transfer-values so the dimensionality matches
        the internal state of the GRU layers. This means
        we can use the mapped transfer-values as the initial state
        of the GRU layers.
        """
        initial_state = self.decoder_transfer_map(transfer_values)



        net = self.decoder_input

        net = self.decoder_embedding(net)

        net = self.decoder_gru1(net, initial_state=initial_state)
        net = self.decoder_gru2(net, initial_state=initial_state)
        net = self.decoder_gru3(net, initial_state=initial_state)

        decoder_output = self.decoder_dense(net)

        return decoder_output
```

```
def flatten(self,captions_listlist):
    captions_list = [caption
                     for captions_list in captions_listlist
                     for caption in captions_list]

    return captions_list

def load_image(self,path, size=None):
    """
    Load the image from the given file-path and resize it
    to the given size if not None.
    """

    # Load the image using PIL.
    img = Image.open(path)

    # Resize image if desired.
    if not size is None:
        img = img.resize(size=size, resample=Image.LANCZOS)

    # Convert image to numpy array.
    img = np.array(img)

    # Scale image-pixels so they fall between 0.0 and 1.0
    img = img / 255.0

    # Convert 2-dim gray-scale array to 3-dim RGB array.
    if (len(img.shape) == 2):
        img = np.repeat(img[:, :, np.newaxis], 3, axis=2)

    return img

def generate_caption(self,image_path, max_tokens=30):
    """
    Generate a caption for the image in the given path.
    The caption is limited to the given number of tokens (words).
```

```
"""

image = self.load_image(image_path, size=self.img_size)

image_batch = np.expand_dims(image, axis=0)

transfer_values = self.image_model_transfer.predict(image_batch)

shape = (1, max_tokens)
decoder_input_data = np.zeros(shape=shape, dtype=np.int)

token_int = self.token_start

output_text = ''

count_tokens = 0

while token_int != self.token_end and count_tokens < max_tokens:

    decoder_input_data[0, count_tokens] = token_int

    x_data = {
        'transfer_values_input': transfer_values,
        'decoder_input': decoder_input_data
    }

    decoder_output = self.decoder_model.predict(x_data)
```

```
            token_onehot = decoder_output[0, count_tokens, :]

            token_int = np.argmax(token_onehot)

            sampled_word = self.tokenizer.token_to_word(token_int)

            output_text += " " + sampled_word

            count_tokens += 1

        output_tokens = decoder_input_data[0]

        print("Predicted caption:")
        output_text=output_text[1].upper()+output_text[2:-4]
        print(output_text)
        print()
        return(output_text)

#Test
# caption=Caption(model_path='finalModel')
# now=time.time()
# caption.generate_caption(image_path='test\\beach.jpg')
# then=time.time()
# print('Time lapsed: {}'.format(then-now))
# now=time.time()
# caption.generate_caption(image_path='test\\bear.jpg')
# then=time.time()
# print('Time lapsed: {}'.format(then-now))
```

Anexa 4 Clase Kotlin

```kotlin
package com.example.captionthis

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import kotlinx.android.synthetic.main.activity_intro.*

class IntroActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_intro)
        pdfView.fromAsset("Disertatie.pdf").load()
        next_button.setOnClickListener {
            val intent= Intent(this,LoginActivity::class.java)
            startActivity(intent)
        }
    }
}
```

```kotlin
package com.example.captionthis

import android.content.DialogInterface
import android.content.Intent
import android.os.Bundle
import android.util.Patterns
import android.widget.EditText
import android.app.AlertDialog
import android.widget.Toast
import android.util.Log
import androidx.appcompat.app.AppCompatActivity
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.FirebaseUser
import kotlinx.android.synthetic.main.activity_login.*


class LoginActivity : AppCompatActivity() {
    private lateinit var auth: FirebaseAuth
    companion object {
        val TAG = "Login Activity"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)
        auth = FirebaseAuth.getInstance()
        login_button.setOnClickListener {
            doLogin()
        }

        register_link.setOnClickListener {
```

```
            val intent= Intent(this,RegisterActivity::class.java)
            startActivity(intent)
        }




    }

    private fun forgotPassword(username : EditText){
        if (username.text.toString().isEmpty()) {
            return
        }

        if
(!Patterns.EMAIL_ADDRESS.matcher(username.text.toString()).matches()) {
            return
        }

        auth.sendPasswordResetEmail(username.text.toString())
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    Toast.makeText(this,"Email sent.",
Toast.LENGTH_SHORT).show()
                }
            }

    }

    private fun doLogin() {
        if (email_input.text.toString().isEmpty()) {
            email_input.error = "Please enter email"
            email_input.requestFocus()
            return
        }

        if
(!Patterns.EMAIL_ADDRESS.matcher(email_input.text.toString()).matches()) {
            email_input.error = "Please enter valid email"
            email_input.requestFocus()
            return
        }

        if (txt_pwd.text.toString().isEmpty()) {
            txt_pwd.error = "Please enter password"
            txt_pwd.requestFocus()
            return
        }

        auth.signInWithEmailAndPassword(email_input.text.toString(),
txt_pwd.text.toString())
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
```

```
                    val user = auth.currentUser
                    updateUI(user,false)
                } else {

                    updateUI(null,false)
                }
            }
    }

    public override fun onStart() {
        super.onStart()
        val currentUser = auth.currentUser
        if(currentUser!=null) {
            currentUser.reload()
        }
        updateUI(currentUser,true)
    }

    private fun updateUI(currentUser: FirebaseUser?,  onStart: Boolean ) {

        if (currentUser != null) {
            if(currentUser.isEmailVerified) {
                startActivity(Intent(this, PhotoActivity::class.java))
                finish()
            }else{
                Toast.makeText(
                    baseContext, "Please verify your email address.",
                    Toast.LENGTH_SHORT
                ).show()
                Log.d(TAG,"Gets to verify email")
                currentUser.sendEmailVerification()
            }
        } else {
            if (onStart == true){
                Toast.makeText(
                    baseContext, "No user logged in",
                    Toast.LENGTH_SHORT
                ).show()
                Log.d(TAG,"Fails Login")
            }
            else{
                Toast.makeText(
                    baseContext, "Login failed.",
                    Toast.LENGTH_SHORT
                ).show()
                Log.d(TAG,"Fails Login")
            }

        }
    }

}
```

```
package com.example.captionthis

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Patterns
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.FirebaseUser
import kotlinx.android.synthetic.main.activity_register.*
import android.widget.Toast
import android.util.Log
class RegisterActivity : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth

    companion object {
        val TAG = "Login Activity"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_register)
        auth = FirebaseAuth.getInstance()

        register_button.setOnClickListener {
            val intent= Intent(this,LoginActivity::class.java)
            registerUser()
            startActivity(intent)
        }
        login_link.setOnClickListener {
            val intent= Intent(this,LoginActivity::class.java)
            startActivity(intent)
        }


    }



    private fun registerUser() {


        if (email_input.text.toString().isEmpty()) {
            email_input.error = "Please enter email"
            email_input.requestFocus()
            return
        }

        if
(!Patterns.EMAIL_ADDRESS.matcher(email_input.text.toString()).matches()) {
            email_input.error = "Please enter valid email"
            email_input.requestFocus()
            return
```

```
        }

        if (pwd_input.text.length < 6){
            Toast.makeText(applicationContext,"Password too short, enter
mimimum 6 charcters" , Toast.LENGTH_LONG).show()
            return
        }

        if (pwd_input.text.toString().isEmpty()) {
            pwd_input.error = "Please enter password"
            pwd_input.requestFocus()
            return
        }
        val email=email_input.text.toString()
        val password= pwd_input.text.toString()
        auth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                    // Sign in success, update UI with the signed-in user's
information
                    Log.d(TAG, "createUserWithEmail:success")
                    sendEmailVerification()
                    startActivity(Intent(this, LoginActivity::class.java))
                    finish()
                } else {
                    // If sign in fails, display a message to the user.
                    Log.w(TAG, "createUserWithEmail:failure",
task.exception)
                    Toast.makeText(baseContext, "Authentication failed.",
                        Toast.LENGTH_SHORT).show()

                }


            }
    }

    private fun sendEmailVerification() {
        // Disable button

        // Send verification email
        // [START send_email_verification]
        val user = auth.currentUser
        if (user !=null) {
            user.sendEmailVerification()
                .addOnCompleteListener(this) { task ->
                    // [START_EXCLUDE]
                    // Re-enable button

                    if (task.isSuccessful) {
                        Toast.makeText(baseContext,
                            "Verification email sent to ${user.email} ",
                            Toast.LENGTH_SHORT).show()
```

```
                    } else {
                        Log.e(TAG, "sendEmailVerification", task.exception)
                        Toast.makeText(baseContext,
                            "Failed to send verification email.",
                            Toast.LENGTH_SHORT).show()
                    }
                    // [END_EXCLUDE]
                }
        }

        // [END send_email_verification]
    }
}
```

```
package com.example.captionthis

import android.Manifest
import android.app.Activity
import android.app.AlertDialog
import android.content.ContentValues
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.provider.MediaStore
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.karumi.dexter.Dexter
import com.karumi.dexter.MultiplePermissionsReport
import com.karumi.dexter.PermissionToken
import com.karumi.dexter.listener.PermissionRequest
import com.karumi.dexter.listener.multi.MultiplePermissionsListener
import kotlinx.android.synthetic.main.activity_photo.*
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.FirebaseUser
import java.lang.Exception

class PhotoActivity : AppCompatActivity() {

    var fileUri: Uri? = null
    private lateinit var auth: FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_photo)
        auth=FirebaseAuth.getInstance()
        gallery_button.setOnClickListener {
            pickPhotoFromGallery()
        }
        camera_button.setOnClickListener{
            askCameraPermission()
        }
        logout_button.setOnClickListener {
            val user =auth.currentUser
```

```
            if (user!=null){
                try {
                    auth.signOut()
                    Toast.makeText(
                        baseContext, "Logout successful",
                        Toast.LENGTH_LONG
                    ).show()
                    startActivity(Intent(this, LoginActivity::class.java))

                }catch (e:Exception){
                    // an error
                    Toast.makeText(
                        baseContext, "Logout failed due to: "+e.toString(),
                        Toast.LENGTH_LONG
                    ).show()
                }

            }
        }

    }
    //pick a photo from gallery
    private fun pickPhotoFromGallery() {
        val pickImageIntent = Intent(Intent.ACTION_PICK,
            MediaStore.Images.Media.EXTERNAL_CONTENT_URI)

        startActivityForResult(pickImageIntent,
AppConstants.PICK_PHOTO_REQUEST)
    }

    //launch the camera to take photo via intent
    private fun launchCamera() {
        val values = ContentValues(1)
        values.put(MediaStore.Images.Media.MIME_TYPE, "image/jpg")
        fileUri = contentResolver
            .insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
                values)
        val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
        if(intent.resolveActivity(packageManager) != null) {
            intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri)
            intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION
                    or Intent.FLAG_GRANT_WRITE_URI_PERMISSION)
            startActivityForResult(intent, AppConstants.TAKE_PHOTO_REQUEST)
        }
    }

    //ask for permission to take photo
    fun askCameraPermission(){
        Dexter.withActivity(this)
            .withPermissions(
            Manifest.permission.CAMERA,
            Manifest.permission.WRITE_EXTERNAL_STORAGE)
            .withListener(object : MultiplePermissionsListener
```

```
            {

                override fun onPermissionsChecked(report:
MultiplePermissionsReport) {/* ... */
                    if(report.areAllPermissionsGranted()){
                        //once permissions are granted, launch the camera
                        launchCamera()
                    }else{
                        Toast.makeText(this@PhotoActivity, "All permissions
need to be granted to take photo", Toast.LENGTH_LONG).show()
                    }
                }

                override fun onPermissionRationaleShouldBeShown(permissions:
List<PermissionRequest>, token: PermissionToken) {/* ... */
                    //show alert dialog with permission options
                    AlertDialog.Builder(this@PhotoActivity)
                        .setTitle(
                            "Permissions Error!")
                        .setMessage(
                            "Please allow permissions to take photo with
camera")
                        .setNegativeButton(
                            android.R.string.cancel,
                            { dialog, _ ->
                                dialog.dismiss()
                                token?.cancelPermissionRequest()
                            })
                        .setPositiveButton(android.R.string.ok,
                            { dialog, _ ->
                                dialog.dismiss()
                                token?.continuePermissionRequest()
                            })
                        .setOnDismissListener({
                            token?.cancelPermissionRequest() })
                        .show()
                }

            }).check()

    }

    //override function that is called once the photo has been taken
    override fun onActivityResult(requestCode: Int, resultCode: Int,
                                  data: Intent?) {
        intent=Intent(this,CaptionActivity::class.java)
        if (resultCode == Activity.RESULT_OK
            && requestCode == AppConstants.TAKE_PHOTO_REQUEST) {
            //photo from camera
            intent.putExtra("imagePath",fileUri.toString())
            startActivity(intent)
        }else if(resultCode == Activity.RESULT_OK
            && requestCode == AppConstants.PICK_PHOTO_REQUEST){
```

```
            //photo from gallery
            fileUri = data?.data
            intent.putExtra("imagePath",fileUri.toString())
            startActivity(intent)
        } else {
            super.onActivityResult(requestCode, resultCode, data)
        }
    }
}
```

```
@file:Suppress("DEPRECATION")

package com.example.captionthis

import android.app.ProgressDialog
import android.content.Intent
import android.os.Bundle
import android.provider.MediaStore
import android.widget.Toast

import java.io.File

import androidx.appcompat.app.AppCompatActivity
import android.net.Uri
import kotlinx.android.synthetic.main.acivity_caption.*

import com.example.captionthis.networking.ApiConfig
import com.example.captionthis.networking.AppConfig
import com.example.captionthis.networking.ServerResponse
import okhttp3.MediaType
import okhttp3.RequestBody
import android.util.Log
import retrofit2.Call
import retrofit2.Callback
import retrofit2.Response

class CaptionActivity : AppCompatActivity() {

    private var postPath: String? = null



    private var contentDescription: String?= null


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.acivity_caption)

        var image_path : String = intent.getStringExtra("imagePath");
        var fileUri = Uri.parse(image_path)
        var photo=
MediaStore.Images.Media.getBitmap(this.getContentResolver(), fileUri);
```

```
        picture_image_view.setImageBitmap(photo)
        back_button.setOnClickListener {
            return_to_photo()
        }
        describe_button.setOnClickListener{
            get_description()
        }
        val selectedImage = fileUri
        val filePathColumn = arrayOf(MediaStore.Images.Media.DATA)
        val cursor = contentResolver.query(selectedImage!!, filePathColumn,
null, null, null)
        assert(cursor != null)
        cursor!!.moveToFirst()
        val columnIndex = cursor.getColumnIndex(filePathColumn[0])
        postPath =cursor.getString(columnIndex)

    }
    fun return_to_photo(){
        val intent= Intent(this,PhotoActivity::class.java)
        startActivity(intent)
    }

    fun get_description(){
        uploadFile()
    }

    private fun uploadFile() {
        if (postPath == null || postPath == "") {
            Toast.makeText(this, "please select an image ",
Toast.LENGTH_LONG).show()
            return
        } else {

            // Map is used to multipart the file using okhttp3.RequestBody
            val map = HashMap<String, RequestBody>()
            val file = File(postPath!!)

            // Parsing any Media type file
            val requestBody = RequestBody.create(MediaType.parse("*/*"),
file)
            map.put("file\"; filename=\"" + file.name + "\"", requestBody)
            val getResponse =
AppConfig.getRetrofit().create(ApiConfig::class.java)
            val call = getResponse.upload("token", map)
            call.enqueue(object : Callback<ServerResponse> {
                override fun onResponse(call: Call<ServerResponse>,
response: Response<ServerResponse>) {
                    if (response.isSuccessful) {
                        if (response.body() != null) {
                            val serverResponse = response.body()
                            //Toast.makeText(applicationContext,
serverResponse?.message, Toast.LENGTH_SHORT).show()
                            contentDescription=serverResponse?.message
```

```
                            description_text.text=contentDescription
                        }
                    } else {
                        Toast.makeText(applicationContext, "problem
uploading image", Toast.LENGTH_SHORT).show()
                        description_text.text="problem uploading image"
                    }
                }

                override fun onFailure(call: Call<ServerResponse>, t:
Throwable) {
                    Log.v("Response gotten is", t.message)
                }
            })
        }
    }



}
```

Anexa 5 Cod Interfață grafică

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background"
    tools:context=".IntroActivity">

    <LinearLayout
        android:id="@+id/horizontal_layout"
        android:layout_width="match_parent"
        android:layout_height="179dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/title_text_view"
            android:layout_width="0dp"
            android:layout_height="189dp"
            android:layout_weight="1"
            android:drawableStart="@drawable/logo_no_text"
            android:fontFamily="@font/bungee_shade"
            android:text="@string/title_text_view"
            android:textColor="#303F9F"
            android:textSize="30sp"
```

```
            android:gravity="center"
            android:padding="10sp"
            android:textStyle="bold"
            tools:text="@string/title" />

        <ImageButton
            android:id="@+id/next_button"
            android:layout_width="138dp"
            android:layout_height="55dp"
            android:layout_gravity="end"
            android:background="@drawable/next_button"
            android:contentDescription="@string/general" />
    </LinearLayout>

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="10sp">

        <com.github.barteksc.pdfviewer.PDFView
            android:id="@+id/pdfView"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </FrameLayout>

</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background"
    android:orientation="vertical"
    tools:context=".IntroActivity">

    <LinearLayout
        android:id="@+id/title_linear_layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:id="@+id/title_text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:drawableStart="@drawable/logo_no_text"
            android:fontFamily="@font/bungee_shade"
            android:gravity="center"
            android:padding="10sp"
            android:text="@string/title_text_view"
            android:textColor="#303F9F"
```

```
            android:textSize="30sp"
            android:textStyle="bold"
            tools:text="@string/title" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:orientation="vertical">

        <TextView
            android:id="@+id/registerTitle"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginTop="80dp"
            android:text="@string/login"
            android:textSize="30sp"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/fstTxt"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:fontFamily="@font/bungee"
            android:paddingStart="100sp"
            android:paddingEnd="100sp"
            android:text="@string/email"
            android:textColor="#F5F2F2"
            android:textSize="24sp" />

        <EditText
            android:id="@+id/email_input"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:ems="10"
            android:hint="@string/email"

android:background="@color/common_google_signin_btn_text_dark_default"
            android:alpha="0.6"
            android:importantForAutofill="no"
            android:inputType="textEmailAddress"
            android:padding="10sp" />

        <TextView
            android:id="@+id/secTxt"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:fontFamily="@font/bungee"
            android:paddingStart="100sp"
```

```
            android:paddingEnd="100sp"
            android:text="@string/password"
            android:textColor="#F5F5F5"
            android:textSize="24sp" />

        <EditText
            android:id="@+id/txt_pwd"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:padding="10sp"
            android:hint="@string/password"

android:background="@color/common_google_signin_btn_text_dark_default"
            android:alpha="0.6"
            android:inputType="textPassword"
            android:ems="10"
            android:importantForAutofill="no" />

        <ImageButton
            android:id="@+id/login_button"
            android:layout_width="108dp"
            android:layout_height="47dp"
            android:layout_gravity="center"
            android:background="@drawable/login_button"
            android:contentDescription="@string/general"
            android:padding="30sp" />

        <TextView
            android:id="@+id/register_link"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="40dp"
            android:gravity="center"
            android:text="@string/new_to"
            android:textColor="#3F51B5"
            android:textSize="20sp" />

        <TextView
            android:id="@+id/forgot_link"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="40dp"
            android:gravity="center"
            android:text="@string/forgot"
            android:textColor="#3F51B5"
            android:textSize="20sp" />

    </LinearLayout>

</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background"
    android:orientation="vertical"
    tools:context=".IntroActivity">

    <TextView
        android:id="@+id/title_text_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:drawableStart="@drawable/logo_no_text"
        android:fontFamily="@font/bungee_shade"
        android:gravity="center"
        android:padding="10sp"
        android:text="@string/title_text_view"
        android:textColor="#303F9F"
        android:textSize="30sp"
        android:textStyle="bold"
        tools:text="@string/title" />

    <TextView
        android:id="@+id/email_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:fontFamily="@font/bungee"
        android:paddingStart="100sp"
        android:paddingEnd="100sp"
        android:text="@string/email"
        android:textColor="#F5F2F2"
        android:textSize="24sp" />

    <EditText
        android:id="@+id/email_input"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:ems="10"
        android:hint="@string/email"

android:background="@color/common_google_signin_btn_text_dark_default"
        android:alpha="0.6"
        android:importantForAutofill="no"
        android:inputType="textEmailAddress"
        android:padding="10sp" />

    <TextView
        android:id="@+id/password_text_view"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:fontFamily="@font/bungee"
        android:paddingStart="100sp"
        android:paddingEnd="100sp"
        android:text="@string/password"
        android:textColor="#F5F5F5"
        android:textSize="24sp" />

    <EditText
        android:id="@+id/pwd_input"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:ems="10"

android:background="@color/common_google_signin_btn_text_dark_default"
        android:alpha="0.6"
        android:hint="@string/password"
        android:importantForAutofill="no"
        android:inputType="textPassword"
        android:padding="10sp" />

    <!--<TextView
        android:id="@+id/re_password_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="@font/bungee"
        android:paddingStart="100sp"
        android:paddingEnd="100sp"
        android:text="@string/re_pwd"
        android:textColor="#F5F5F5"
        android:textSize="24sp" />

    <EditText
        android:id="@+id/re_pwd_input"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:ems="10"
        android:hint="@string/re_pwd"
        android:importantForAutofill="no"
        android:inputType="textPassword"
        android:padding="10sp" />-->

    <ImageButton
        android:id="@+id/register_button"
        android:layout_width="108dp"
        android:layout_height="47dp"
        android:layout_gravity="center"
        android:background="@drawable/register_button"
        android:contentDescription="@string/general"
        android:padding="30sp" />
```

```
    <TextView
        android:id="@+id/login_link"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="40dp"
        android:gravity="center"
        android:text="@string/already"
        android:textColor="#3F51B5"
        android:textSize="20sp" />

    <!--<TextView
        android:id="@+id/registerTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="80dp"
        android:text="@string/register"
        android:textSize="30sp"
        android:textStyle="bold" />-->

</LinearLayout>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="@drawable/background"
    tools:context=".PhotoActivity"
    >


    <LinearLayout
        android:id="@+id/main_linear"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@drawable/background"
        android:orientation="vertical">

        <LinearLayout
            android:id="@+id/title_layout"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:orientation="horizontal"
            tools:visibility="visible">

            <TextView
                android:id="@+id/title_text_view"
                android:layout_width="0dp"
```

```
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:padding="25sp"
            android:drawableStart="@drawable/logo_no_text"
            android:fontFamily="@font/bungee_shade"
            android:gravity="center"
            android:text="@string/title_text_view"
            android:textColor="#303F9F"
            android:textSize="45sp"
            android:textStyle="bold"
            tools:text="@string/title" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <Space
            android:layout_width="match_parent"
            android:layout_height="251dp" />

        <TextView
            android:id="@+id/main_instruction"
            android:layout_width="match_parent"
            android:layout_height="128dp"
            android:lineSpacingExtra="18sp"
            android:lines="4"
            android:text="@string/main_text_instruction"
            android:textAlignment="center"
            android:textAllCaps="false"
            android:textColor="#F57C00"
            android:textSize="30sp"
            android:textStyle="bold"
            android:typeface="normal"
            tools:visibility="visible" />

    </LinearLayout>

    <LinearLayout
        android:id="@+id/buttons_layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center|top"
        android:orientation="horizontal">

        <Space
            android:layout_width="25dp"
            android:layout_height="42dp" />

        <ImageButton
            android:id="@+id/camera_button"
            android:layout_width="154dp"
```

```
                android:layout_height="60dp"
                android:background="@drawable/camera_button"

android:contentDescription="@string/camera_button_description"
                android:scaleType="center" />

            <Space
                android:layout_width="50dp"
                android:layout_height="42dp" />

            <ImageButton
                android:id="@+id/gallery_button"
                android:layout_width="154dp"
                android:layout_height="62dp"
                android:background="@drawable/galery_button"

android:contentDescription="@string/document_button_descriptio"
                android:scaleType="center" />

            <Space
                android:layout_width="25dp"
                android:layout_height="42dp" />

        </LinearLayout>

        <Space
            android:layout_width="match_parent"
            android:layout_height="29dp" />

        <ImageButton
            android:id="@+id/logout_button"
            android:contentDescription="@string/general"
            android:layout_width="140dp"
            android:layout_height="64dp"
            android:layout_gravity="center"
            android:background="@drawable/logout_button" />
    </LinearLayout>

</LinearLayout>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="@drawable/background"
    tools:context=".IntroActivity"
    >
```

```
<LinearLayout
    android:id="@+id/main_linear"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background"
    android:orientation="vertical">

    <ImageButton
        android:id="@+id/back_button"
        android:layout_width="120dp"
        android:layout_height="48dp"
        android:background="@drawable/back_button"
        android:contentDescription="@string/general" />

    <LinearLayout
        android:id="@+id/top_layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:padding="10sp">

            <ImageView
                android:id="@+id/picture_image_view"
                android:layout_width="match_parent"
                android:layout_height="503dp"
                android:contentDescription="@string/general"
                tools:srcCompat="@tools:sample/avatars" />
        </FrameLayout>

        <ImageButton
            android:id="@+id/describe_button"
            android:layout_width="198dp"
            android:layout_height="48dp"
            android:layout_gravity="center"
            android:background="@drawable/decr_button"
            android:contentDescription="@string/general"
            android:scaleType="fitCenter" />

        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:padding="10sp">

            <TextView
                android:id="@+id/description_text"
                android:layout_width="match_parent"
                android:layout_height="66dp"
```

```
                    android:background="@drawable/white_gradient"
                    android:ems="10"
                    android:hint="@string/image_description_dummy"
                    android:importantForAutofill="no"
                    android:text="@string/picture_description"
                    android:textColor="#110C0C"
                    android:textSize="25dp"/>
            </FrameLayout>

        </LinearLayout>

    </LinearLayout>

</LinearLayout>
```