UNIVERSITATEA DIN BUCUREȘTI FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

LUCRARE DE LICENȚĂ

Coordonator:
Prof. Dr. Radu Ionescu

Absolvent:
Moldovan George-Alexandru

București
Iunie (fingers crossed), 2020

UNIVERSITATEA DIN BUCUREȘTI FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

Sistem pentru detectarea anomaliilor in video

Coordonator:
Prof. Dr. Radu Ionescu

Absolvent:
Moldovan George-Alexandru

București
Iunie (fingers crossed), 2020

Cuprins

Rezumat

Având în vedere contextul actual, detectarea anomaliilor în video este un subiect de interes în mai multe arii, în mod special în securitatea publică. Putem spune că această problemă este încă nerezolvată, deoarece sistemele actuale, deocamdată, nu depăşesc omul cand vine vorba de detectarea anomaliilor. De asemenea, o altă problemă a sistemelor de detectare a anomaliilor în video este nevoia acestora de resurse computaţionale mari în partea de inferență, făcând aproape imposibilă rularea acestora direct pe hardware-ul existent al sistemelor de supraveghere video actuale, acolo unde acestea prezintă un maxim interes. Astfel, putem spune că dezvoltarea unui sistem capabil să transforme sistemele de supraveghere actuale în sisteme ce pot recunoaşte evenimente anormale este un subiect ce poate revoluţiona domeniul supravegherii video. Această lucrare îşi propune o implementare al sistemului state-of-the-art la momentul redactării, aşa cum este prezentat de *Ionescu et al.* [10]. Obiectivul este obţinerea unei arhitecturi ce foloseşte o soluţie PaaS si expunerea etapei de inferenţă printr-un API astfel încât convertirea unui sistem de supraveghere clasic într-unul inteligent să devină doar o problemă de implementare, fără a fi nevoie de schimbarea hardware-ului. Utilizarea unei soluţii PaaS pentru etapa de inferenţă rezolvă problema executării cererilor fără complexitatea creeri şi întreţinerii unei infrastructuri de maşini virtuale sau fizice.

Capitolul 1

Introducere

1.1 Motivatie

Detectarea anomaliilor în video este în strânsă legătură cu sistemele de supraveghere inteligente,un domeniu care a fost şi este de interes pentru mine. La rândul lor, sistemele de supraveghere inteligente, au o mare importanţă în securitatea publică. Cu toţii ne dorim o lume în care apelurile de urgenţă în caz de incendiu se fac automat, alunecările de teren sunt descoperite înainte să fie prea târziu, iar oamenii rău intenţionaţi sunt opriţi înainte să se întâmple tragedii.

Astfel, arhitectura folosită se bazează pe detecţia caracteristicilor spaţio-temporale ale evenimentelor prezente în video, care mai apoi sunt împărţite în clase de normalitate. Aceste caracteristici sunt extrase trecând evenimentul printr-o serie de auto-encodere pentru a folosi mai apoi reprezentarea latentă în cadrul clasificării finale. În etapa de antrenare, se folosesc filmări ce prezintă comportamentul normal în scenariul analizat. Analizând aceste video-uri putem creea un model capabil să recunoască dacă un eveniment aparţine unei clase de normalitate analizate până acum, sau dacă este un caz anormal. Deoarece un eveniment poate fi ales în multe moduri, în cadrul acestui sistem un eveniment reprezintă orice obiect aflat în cadru. Analiza asupra fiecarui obiect conţine şi un cadru precedent dar si unul viitor, asemănător cu modalitatea folosită de *Ionescu et al.* [10]. O menţiune în acest sens ar fi că pentru fiecare obiect sunt analizate si cadrele de la poziţiile t-3 si t+3 respectiv la poziţia t a cadrului analizat. Din acest motiv, atunci când sistemul va analiza un video în sistem live-feed, analiza se va face cu un decalaj de 3 cadre. Având în vedere că pentru un video actual viteza de redare este de minim 15 cadre pe secundă, acest decalaj este neglijabil.

Pe lângă partea algoritmică a detectării anomaliilor, o altă arie de interes a acestei lucrări este cloud computing. Această parte analizează un nou mod de rulare, ce facilitează atât dezvoltarea cât şi execuţia ulterioară a unor sisteme complexe. Acest nou mod constă în folosirea unei arhitecturi plasată în cloud, ce oferă dezvoltatorului posibilitatea să creeze sisteme ce necesită multe resurse în timpul rulării, fără costurile asociate creeri şi menţinerii unei infrastructuri proprii. Pe de altă parte, având în

vedere că toate operaţiunile sunt executate in cloud, utilizatorii serviciului au nevoie doar de conexiune la internet şi cerinţe minime pentru sistemele proprii, fara a fi nevoiţi să achiziţioneze echipamente noi pentru a folosi sisteme de detecţie a anomaliilor.

1.2 Context

Detectarea anomaliilor în video poate fi văzută ca o problemă subiectivă, deoarece un eveniment este normal sau anormal doar dacă este luat în considerare şi contextul în care acesta apare. Un exemplu foarte bun este comparaţia între două persoane care se luptă şi o persoană care se plimbă. Care dintre aceste evenimente este anormal ? Desigur, depinde de context. Dacă sistemul supraveghează o arenă de lupte, atunci persoana care se plimbă în ring prezintă un comportament anormal, în timp ce luptătorii prezintă comportamentul aşteptat. Din acest motiv majoritatea lucrărilor din domeniu [5, 10, 13]... abordează un mod de lucru bazat pe antrenarea folosind video-uri ce provin din aceeasi locaţie cu cele de test. Tocmai din cauza dependenţei de context, detectarea anomaliilor nu este o problemă ce poate fi generalizată, astfel fiecare scenariu necesită o antrenare şi un model propriu.

Ca şi moduri de expunere a soluţiei software către utilizatori, aceasta se poate face în 2 moduri :

- Folosind servere proprii
- Folosind servicii cloud

Folosirea serverelor proprii presupune, pe lângă prezenţa fizică a serverelor şi cumpărarea tuturor serviciilor conexe, cum ar fi servere de baze de date, sisteme de distribuire a fişierelor, infrastructură de reţea, ş.a.m.d., este nevoie si de o echipă dedicată pentru întreţinerea infrastructurii şi repararea eventualelor probleme ce pot apărea. Aceste considerente, împreuna cu faptul că scalarea soluţiilor software este foarte anevoioasă atunci când este folosită infrastructura proprie, fac această soluţie să nu mai fie folosită în mod curent deoarece incetineşte dezvoltarea aplicaţiei, iar rezultatul final este şi el unul mai puţin calitativ decât ce se poate obţine folosind soluţii în cloud.

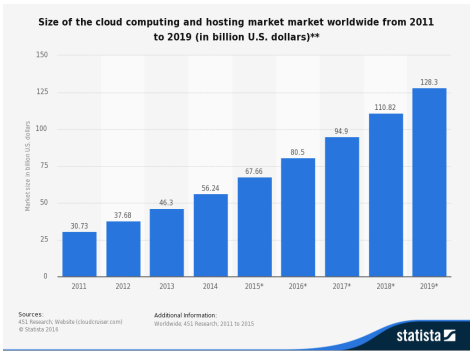

Figura 1.1: Statistică ce evidenţiază importanţa domeniului cloud computing în ultimii ani

Sursă: https://www.statista.com/

Folosirea serviciilor cloud oferă multiple posibilităţi de dezvoltare a aplicaţiilor, de la creeare unei infrastructuri complete care este administrată în totalitate de către dezvoltator, la medii de execuţie serverless care sunt complet administrate de către provider-ul de servicii cloud. Deşi creearea unei infrastructuri proprii încă este necesară acolo unde legislaţia nu permite ca datele sa fie stocate în cloud, în toate celelalte cazuri se face migrarea spre soluţii cloud, lucru vizibil şi în evoluţia cotei de piaţă a domeniului cloud-computing la nivel global, aşa cum se poate observa şi in figura **??**.

1.3 Conţinutul lucrării

Ca şi structură, lucrarea este împărţită în 2 părţi:

- Partea algoritmică a sistemului de detectare a anomaliilor în video
- Partea de deployment a sistemului

În prima parte sistemul va fi analizat si detaliat din punct de vedere algoritmic si teoretic studiind problema detecţiei propriu zisă. Ca şi tehnologii, in această parte am ales sa folosesc Python3 ca şi limbaj de programare pentru anvantajele pe care le are. Printre acestea se numără faptul ca este un limbaj orientat pe obiecte care pune la dispozitia dezvoltatorului numeroase librării specifice pentru AI/ML mutând astfel atenţia dinspre detalii de implementare spre detalii de arhitectură şi probleme mai abstracte ale programului care sunt cu adevărat importante pentru rezultatul final. Una dintre librăriile care s-a dovedit esenţială dezvoltarii este *Keras* [6], care oferă un API ce uşurează dezvoltarea unei reţele neuronale adânci/convoluţionale dar şi optimizează timpul de antrenare pentru modelele create.

În cea de a doua parte este analizat tipul de deployment al aplicaţiei. Aici vor fi prezentate analize detaliate ale avantajelor si dezavantajelor soluţiilor cloud, dar si motivele pentru care modul final de deployment a fost ales.
Soluţiile de cloud-computing analizate vor fi:

- Infrastructure as a Service (IaaS) - analiză pe Amazon EC2
- Platform as a Service (PaaS) - analiză pe Amazon Elastic Beanstalk
- Function as a Service (FaaS) - analiză pe Amazon Lambda

Capitolul 2

Analiza arhitecturii si a tehnologiilor folosite

Pentru o mai bună înţelegere a contextului în care tehnologiile prezentate au fost folosite, acestea vor fi prezentate mai jos in cadrul secţiunii corespunzătoare locului în care a fost folosită în cadrul proiectului. Pentru toate etapele sistemului, limbajul folosit este Python3, deoarece, împreună cu librariile si framework-urile existente pentru inteligenţă articială, reprezintă mediul ideal pentru dezvoltarea unei soluţii modulare, rapide si uşor de modificat.

2.1 Etapa de antrenare

Pentru etapa de antrenare, sistemul trece prin următoarea serie de etape:

- Detectează obiectele din setul de date
- Antrenează un autoencoder pe imaginile obiectelor si un alt autoencoder pe gradienţii obiectelor.
- Obţine reprezentarea latentă a fiecărui eveniment
- Stabileşte k clase de normalitate folosind reprezentările latente
- Antrenează k clasificatori de tipul one-versus-rest

Ca prim pas, pentru obţinerea obiectelor dintr-o imagine, atât în etapa de antrenare cât şi în cea de inferenţă este folosit un detector de obiecte pre-antrenat. Un detector de obiecte este un sistem ce primeşte ca input o imagine si după procesarea acesteia rezolvă problema detecţiei de obiecte şi întoarce poziţiile la care sunt plasate obiectele în imagine, şi etichetele asociate acestora. Deşi detectorul de obiecte face parte din arhitectură, deoarece detectarea de obiecte este un subiect în sine, implementarea unui astfel de algoritm nu face obiectul acestei lucrări. Datorită acestui lucru, este folosit un detector deja antrenat din biblioteca *gluoncv* si testat pe setul de date COCO.

După aceasta etapă sunt antrenate cele două autoencodere convoluţionale. Unul pentru imaginea propriu-zisă şi altul pentru gradient. Un autoencoder este un tip de reţea neuronală alcătuit din 2 parţi (encoder si decoder) care este folosit pentru a obţine reprezentarea latentă a unui obiect folosind un mod de învăţare nesupervizată. În timpul antrenării, autoencoder-ele au ca scop modificarea parametrilor interni pentru a obţine la ieşire, datele primite la intrare. Deşi ieşirea unui autoencoder nu prezintă interes, ceea ce este folositor este reprezentarea latentă (rezultatul encoder-ului) a datelor. Folosind această tehnică, se obţine o reducere a dimensionalităţii ce imbunătăţeşte semnificativ performanţele clasificatorilor ulteriori.

Obţinerea reprezentării latente a fiecărui obiect se realizează trecând imaginea, respectiv gradientul său prin encoderul autoencoderului corespunzător si păstrarea rezultatului.

Odată obţinut vectorul de caracteristici pentru toate obiectele din setul de date, urmează stabilirea claselor de normalitate. Acest lucru se realizează prin aplicarea algoritmului k-means de clustering. Pentru implementare a fost folosit algoritmul *LLoyd* implementat în biblioteca python *sklearn*. Aplicând acest algoritm peste vectorii de caracteristici obţinuţi, rezultă k categorii de normalitate cu vectorii de caracteristici aferenţi.

Folosind categoriile de normalitate obţinute la pasul anterior, putem spune că faţă de o anumită categorie *i*, celelalte k-1 categorii reprezintă categorii *artificial* anormale. Le numim *artifical* anormale deoarece în mod obiectiv ele sunt acţiuni normale pentru sistemul de dectare a anomaliilor, dar pentru antrenarea unor clasificatori conform schemei one-vs-rest acestea sunt tratate drept anormale. Astfel, putem antrena un clasificator binar g(i) în aşa fel încat să separăm elementele din categoria i de cele din categoriile {1,2...k}/i generând funcţia :

$$f_i(x) = \sum_{1}^{n} w_j * x_j + b$$

. Unde x $\in R^n$ reprezintă vectorul de caracteristici, w este vectorul de parametri a funcţiei, iar b reprezintă bias-ul funcţiei. [10].

Astfel, generăm k astfel de funcţii corespunzătoare celor k clasificatori ce vor fi folosiţi pentru a stabili daca un eveniment este anormal. Conform schemei one-vs-rest, un eveniment este anormal daca este clasificat drept anormal de către toţi cei k clasificatori.

2.2 Etapa de inferenţă

În cadrul etapei de inferenţă, sistemul foloseşte detectorul de obiecte, autoencoderele pre-antrenate şi cei k clasificatori binari pentru a stabili dacă un anumit eveniment este sau nu anormal. Astfel, parcursul sistemului este următorul :

- Extragerea cadrelor necesare din video
- Extragerea obiectelor din imagini
- Obţinerea reprezentării latente
- Clasificarea evenimentelor

Pentru analiza unui eveniment care apare la un indice dat t sunt necesare 3 cadre. Mai precis cadrele de la indicii *t-3*, *t+3* şi *t*. Din cadrul t se va extrage vectorul de caracteristici specific aparenţei vizuale, iar din celelalte 2 cadre se vor extrage vectorii de caracterisitici specifici mişcării obiectului, prin analiza gradienţilor. Prin concatenarea acestor 3 vectori, se obţine vectorul final de caracteristici ce va fi folosit drept input pentru clasificatorii finali.

Pentru extragerea obiectelor din cadrele analizate, se va folosi acelaşi detector de obiecte ca în etapa de antrenare. Acesta va fi rulat pe cadrul principal t, urmând apoi sa se folosească coordonatele obiectelor de la cadrul t, si pentru cadrele t+3 si t-3 deoarece din cauza diferenţei mici de indici, obiectele nu se pot mişca indeajuns incât sa fie necesară rularea pe toate cele 3 cadre.

Odată obţinute obiectele din cadrul analizat, dupa obţinerea gradienţilor din cadrele t-3 si t+3, se poate obţine reprezentarea latentă a acestor informaţii. Reprezentarea latentă a imaginii obiectului constă în rezultatul generat de encoderul autoencoderului pentru imagini iar reprezentarea latentă a gradienţilor constă în rezultatul generat de encoderul autoencoderului pentru gradienţi.

Odată obtinuţi vectorii de caracteristici pentru reprezentarea vizuală şi pentru reprezentarea mişcării obiectului, prin concatenarea lor se obţine vectorul final, ce poate fi folosit drept input pentru clasificarea finală. Conform schemei *one-vs-rest* acest vector este clasificat de toţi cei k clasificatori, iar rezultatul final este scorul maxim obţinut în urma clasificării.

2.3 Etapa de deployment

Pentru a face sistemul public, acesta este lansat drept un API ce rulează etapa de inferenţă pe un server web plasat in cloud. Pentru dezvoltarea serverului am ales să folosesc framework-ul *Flask*. Flask este un micro-framework de python folosit pentru dezvoltarea soluţiilor web. Motivele pentru care acest framework a fost ales sunt :

- Acesta adaugă un overhead foarte mic aplicaţiei, lucru esenţial atunci cand aplicaţie se doreşte a fi plasată in cloud, din cauza limitărilor de memorie.

- Oferă un suport foarte bun pentru planificarea rutelor de intrare în aplicaţie, lucru foarte important atunci când se doreşte dezvoltarea unui API.

- Este unul dintre framework-urile suportate de Amazon Elastic Beanstalk

Comparativ cu alte framework-uri, Flask este diferit deoarece nu impune linii clare dezvoltatorilor atunci când vine vorba de forma sau componentele aplicaţiei ce urmează a fi dezvoltată. Astfel, dezvoltatorul are control complet asupra aplicaţiei si îşi poate manifesta creativitatea sau ideile fara a fi restricţionat de framework. Flask a fost creat tocmai cu ideea de a fi construit peste el. Deşi poate nu oferă aceeaşi viteză de dezvoltare comparativ cu celelalte frameworkuri, acesta oferă libertatea de alegere la fiecare pas. Are suport pentru toate tipurile de baze de date, fie ele relaţionale sau nerelaţionale, nu are preferinţe când vine vorba de metode de autentificare sau de creare a rolurilor, totul este suportat şi totul este la latitudinea dezvoltatorului. [9]

Pentru a lansa serverul in cloud, am folosit serviciul Amazon Elastic Beanstalk. Acesta este un serviciu complex, ce însumează la rândul lui mai multe servicii cloud oferite de Amazon Web Services(AWS). Mai jos sunt prezentate câteva dintre avantajele si dezavantajele acestui serviciu, fiind analizat in detaliu in capitolele ce urmează. Elastic Beanstalk este un serviciu de tipul *PaaS* ce ofera servicii de deployment si administrate complete.Pentru o mai bună detaliere, mai jos sunt definite toate serviciile cloud incluse de Elastic Beanstalk:

- Amazon EC2 : este un serviciu web oferit de Amazon ce constă în oferirea unui mediu de execuţie sigur in cloud. Este echivalentul unei maşini fizice mutate in cloud. Acesta a fost creat pentru a uşura misiunea dezvoltatorilor de a migra serviciile proprii spre cloud computing. Aceste instanţe sunt extrem de configurabile, punând la dispoziţia dezvoltatorului mai mult de 50 de tipuri de instanţe, plus diferite opţiuni de optimizare a unor părţi specifice, cum ar fi memoria sau placa video. [2]

- Amazon S3 : este un serviciu ce oferă medii de stocare în cloud. Stocarea este de tipul cheie-obiect, unde cheia identifică unic la nivel global un fişier. Acesta oferă o securitate sporită a datelor, şi o disponibilitate de 99.999999% deoarece datele sunt distribuite în sisteme diferite şi în zone diferite. [4]

- Auto Scaling Group : acest serviciu oferă un mod automat de lansare a instanţelor EC2 astfel încât traficul să nu depăşească niciodata puterea de execuţie a unui aplicaţii. Scopul acestui serviciu este de a oferi capacitatea de a scala pentru a menţine o performanţă optimă, menţinând costul în tot acest timp la valoarea minimă. [1]

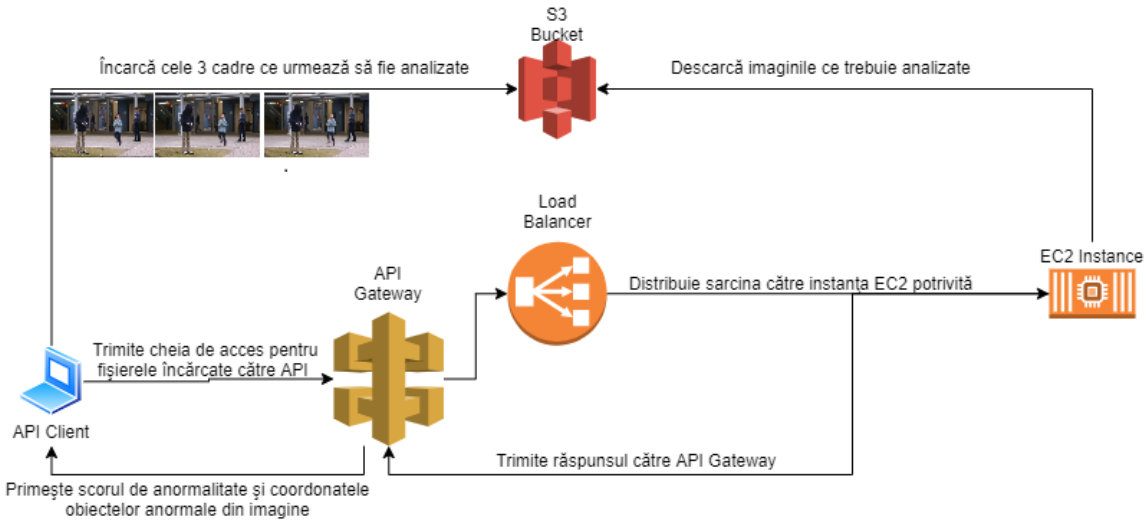

Figura 2.1: Arhitectura abstractizată a API-ului

- Elastic Load Balancing: este un serviciu care împarte traficul administrat de aplicaţie către instanţele lansate astfel încât acestea să fie utilizate intr-un. mod optim. Astfel, poate suporta încarcătura variabilă a aplicaţiei şi o poate distribui în aşa fel încat, în funcţie de modul ales(*accesibilitate crescută*, *(*scalare automată), *securitate ridicată*) aplicaţia să nu scadă sub performanţele dorite. [3]

Dezvoltatorul se ocupă doar de aplicaţia/serverul propriu zis, crează pachetul de deployment, iar Elastic Beanstalk crează instanţele EC2 necesare, administrează si rutează traficul către instanţe folosind un Load Balancer, iar atunci cand aplicaţia este suprasolicitată, lansează în mod automat noi servere folosindu-se de avantajele unui Auto Scaling Group.

Aşa cum se poate observa în figura **??**, API-ul este creat în aşa fel încât evaluarea se face pentru fiecare cadru în parte. Dezvoltatorul ce implementează clientul pentru API are doar responsabilitatea urcării cadrelor necesare într-un spaţiu S3 prestabilit. Preprocesarea, extragerea obiectelor din imagine, şi rularea detecţiei de anomalii sunt toate executate in cloud. Astfel, efortul computaţional asupra clientului este minim. La momentul accesării API-ului, serverul trebuie să primeasca ca parametru în apelul HTTP cheia de acces pentru cadrele urcate de catre client. Folosind această informaţie, pe server se descarcă aceste cadre şi sunt analizate pentru a detecta anomaliile din cadrul central. Ca şi rezultat, clientul primeşte scorul de anormalitate al cadrului împreună cu toate poziţiile obiectelor anormale din cadru, date ce pot fi folosite pentru notificări ulterioare sau diferite aplicaţii pe partea de client.

Capitolul 3

Sistemul de detecţie a anomaliilor

În acest capitol va fi analizat în detaliu sistemul de detecţie a anomaliilor în video, atât din punct de vedere al arhitecturii alese cât şi din punct de vedere al implementării. În acest sens, etapa de antrenare, şi cea de inferenţă a sistemului vor fi analizate în secţiuni separate.

În ceea ce priveşte detecţia de obiecte, ce este comună tuturor etapelor sistemului, este folosit un detector de obiecte pre-antrenat pe setul de date COCO ce foloseşte o arhitectură SSD-Mobilenet, din biblioteca *gluoncv model-zoo*. Acesta are ca prim avantaj viteza de analiză a unei imagini, deoarece aşa cum se poate observa şi în analiza timpilor de execuţie a sistemului, o mare parte din timp este petrecută detectând obiecte, şi doar o mică parte analizând daca evenimentul este unul normal sau anormal. Astfel, viteza a fost cel mai important aspect luat în considerare pentru alegerea detectorului de obiecte.

3.1 Analiza antrenării sistemului

Din cauza complexităţii problemei de a identifica comportamentul anormal al obiectelor prezente în video, arhitectura sistemului presupune multiple etape de prelucare a datelor până la momentul clasificării finale. Astfel, etapa de antrenare este împărţită la rândul ei în 2 etape:

- Etapa de reducere a dimensionalităţii (antrenarea autoencoderelor)
- Etapa de antrenare a clasificatorilor finali

În etapa de reducere a dimensionalităţii sunt definite şi antrenate autoencoderele pe baza imaginilor si gradienţilor extraşi din setul de date. Motivul pentru care obiectele sunt procesate de către autoencodere este că datorită antrenării doar pe obiectele din videourile de antrenare, acestea vor învăţa să reprezinte doar evenimentele normale. Astfel, atunci cand prin aceste autoencodere vor trece evenimente anormale, ce nu sunt asemanătoare cu cele de antrenament, autoencoderele vor genera o eroare de reconstrucţie ce va uşura sarcina clasificatorilor finali.

Arhitectura aleasă pentru autoencodere este cea descrisă de *Ionescu et al.* [10] şi constă intr-o arhitectură convoluţională rapidă, formată dintr-un encoder cu 3 blocuri convoluţional + max-pooling şi un decoder format din 3 blocuri upsampling + convoluţional. Fiecare autoencoder primeşte ca input date de dimensiune 64×64 si creează după encoder un vector de caracteristici de dimensiune $8 \times 8 \times 16$. Structura detaliată a autoencoderelor este :

- Stratul de intrare, de dimensiune $64 \times 64 \times 1$
- Bloc Convoluţional + MaxPooling format din : Un strat convoluţional bazat pe 32 de filtre de dimensiune 3×3 urmat de funcţia de activare *Relu* şi un strat max-pooling bazat pe filtre 2×2 cu *stride* 2.
- Bloc Convoluţional + MaxPooling format din : Un strat convoluţional bazat pe 32 de filtre de dimensiune 3×3 urmat de funcţia de activare *Relu* şi un strat max-pooling bazat pe filtre 2×2 cu *stride* 2.
- Bloc Convoluţional + MaxPooling format din : Un strat convoluţional bazat pe 16 de filtre de dimensiune 3×3 urmat de funcţia de activare *Relu* şi un strat max-pooling bazat pe filtre 2×2 cu *stride* 2.
- Bloc Convoluţional + UpSampling fprmat din : Un strat convoluţional bazat pe 16 de filtre de dimensiune 3×3 urmat de funcţia de activare *Relu* şi un strat UpSampling bazat pe filtre 2×2.
- Bloc Convoluţional + UpSampling fprmat din : Un strat convoluţional bazat pe 32 de filtre de dimensiune 3×3 urmat de funcţia de activare *Relu* şi un strat UpSampling bazat pe filtre 2×2.
- Bloc Convoluţional + UpSampling fprmat din : Un strat convoluţional bazat pe 32 de filtre de dimensiune 3×3 urmat de funcţia de activare *Relu* şi un strat UpSampling bazat pe filtre 2×2.
- Un ultim strat Convoluţional ce are ca rol reducerea dimensiunii de ieşire de la $64 \times 64 \times 32$ la $64 \times 64 \times 1$[10] . Aceste este bazat pe 1 filtru de dimensiune 3×3 urmat de funcţia de activare *Sigmoid*

Primele 4 straturi reprezintă encoder-ul, în timp ce ultimele 4 straturi reprezintă decoderul.

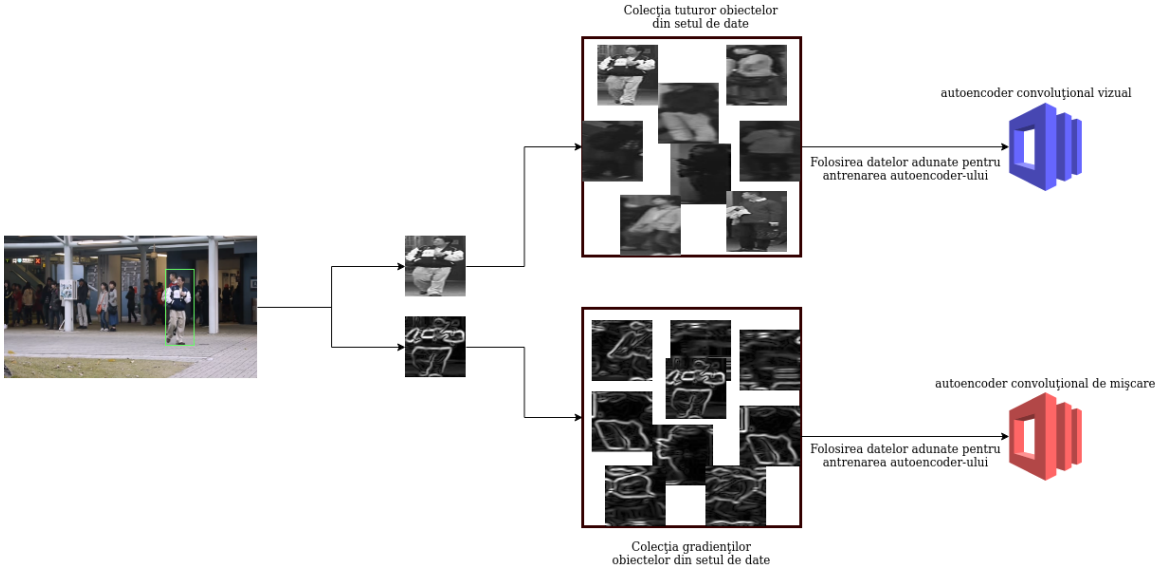

Figura 3.1: Arhitectura primei etape de antrenare

Implementarea arhitecturii prezentate în figura **??** constă în următorii paşi:

- Pentru toate videourile de antrenare se execută detecţia de obiecte pentru fiecare cadru, extrăgând astfel toate obiectele din video.Fiecare cadru este transformat în alb-negru pentru a fi prelucrat în etapele următoare. Pentru fiecare obiect extras, imaginea acestuia este redimensionată pentru a respecta dimensiunea de intrare a autoencodere-lor la 64×64 iar apoi este calculat gradientul, ce reflectă mişcarea obiectului. Gradientul este calculat dupa formula:

$$\sqrt{G_x^2 + G_y^2}$$

  Unde G_x, G_y sunt imagini ce în fiecare punct conţin derivata orizontală respectiv verticală a imaginii iniţiale, obţinută prin aplicarea unui kernel Sobel de 5×5.

- Imaginile si gradienţii astfel obţinuţi sunt adaugaţi intr-o colecţie generală pentru tot setul de date. Cele 2 colecţii astfel create vor servi drept input pentru antrenarea autoencoderelor.

- Folosind cele 2 colecţii (de imagini, respectiv de gradienţi) este antrenat câte un autoencoder pentru fiecare colecţie. Înainte de a fi folosite pentru antrenarea autoencoderelor, atât imaginile, cât şi gradienţii, sunt normalizate în intervalul [0,1] .Antrenarea se realizează folosind optimizatorul Adam [12] şi funcţia loss ce constă în diferenţa medie a pătratelor, dată de formula : $L(I, O) = \frac{1}{h*w} \sum_1^h \sum_1^w (I_{ij} * O_{ij})^2$ [10] unde I si O reprezintă imaginea de input respectiv de output si h,w sunt dimensiunile imaginilor. Aceasta este executată timp de 100 de epoci cu o rată de invaţare de 10^{-3}, dar având şi o regula de oprire rapidă, ce

inseamnă că dacă timp de 2 epoci funcţia loss nu se imbunătaţeşte, antrenarea este finalizată.

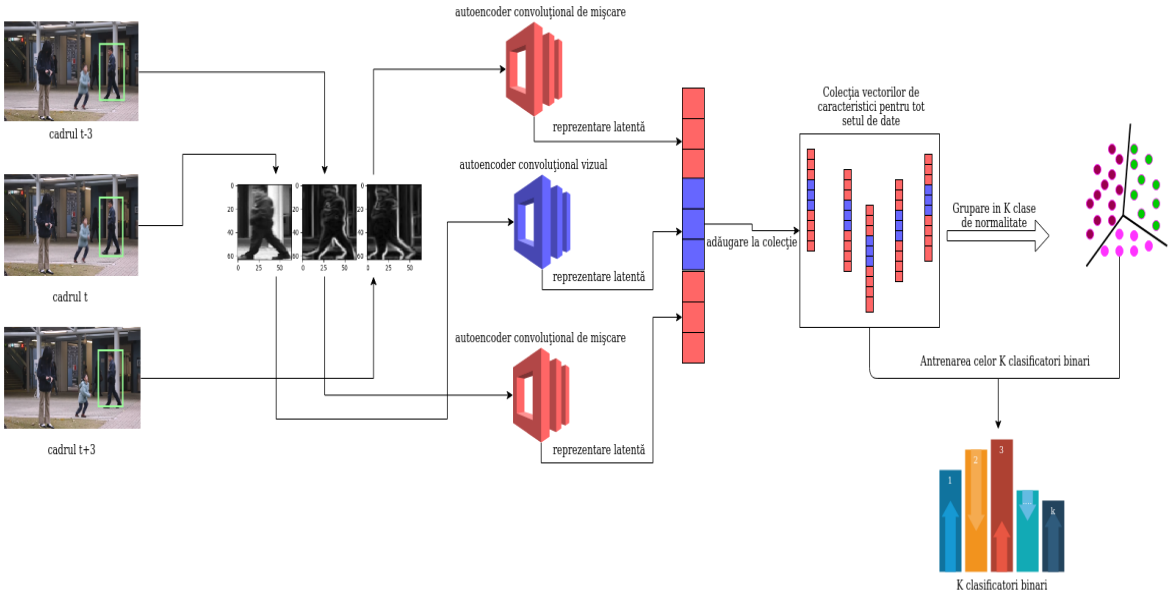

Figura 3.2: Arhitectura etapei finale de antrenare

În etapa de antrenare a clasificatorilor finali, se folosesc autoencoderele antrenate în etapa precedentă pentru a obţine vectorii de caracteristici specifici fiecărui eveniment prezent în setul de date. Ca şi prim pas, se detectează toate obiectele prezente în fiecare cadru din video-urile de antrenare, iar pentru fiecare obiect, se folosesc coordonatele acestuia pentru a decupa acelaşi obiect din cadrul *t-3* şi *t+3*, respectiv la indexul *t* curent. Se calculează gradienţii pentru imaginile decupate, astfel este reprezentată mişcarea obiectului faţă de cadrul curent, iar apoi se obţin vectorii caracteristici ai fiecarui gradient prin trecerea acestora prin autoencoderul corespunzător antrenat în etapa precendentă. Odată obţinuti cei 3 vectori caracteristici specifici evenimentului, aşa cum este ilustrat şi în figura **??**, concatenarea acestora este stocată intr-o colecţie globală ce reţine datele pentru tot setul de date.

Aplicând algoritmul de *k-means clustering* se obţin k categorii de normalitate. Astfel, pentru fiecare eveniment din colecţia globală este cunoscută categoria i din care face parte. Folosind aceste date, putem antrena cei k clasificatori binari. Pentru fiecare categorie, un clasificator binar este antrenat folosind evenimentele din categoria curent drept date de antrenare pozitive, iar celelalte k-1 categorii drept date de antrenare negative. În final, se obţin k clasificatori binari ce au rolul de a evalua daca un eveniment aparţine sau nu categoriei de normalitate i unde i este indicele clasificatorului g_i.

3.2 Analiza etapei de inferenţă

În cadrul etapei de inferenţă este analizat un eveniment cu scopul de a determina daca acesta este sau nu anormal. Analiza unui singur eveniment stă la baza tuturor modurilor de utilizare a sistemului, deoarece, avand aceste date, putem obţine scoruri de normalitate pentru fiecare cadru, sau scoruri de normalitate la nivel de pixel sau chiar şi clasificări generale de normalitate a unui video în totalitate.

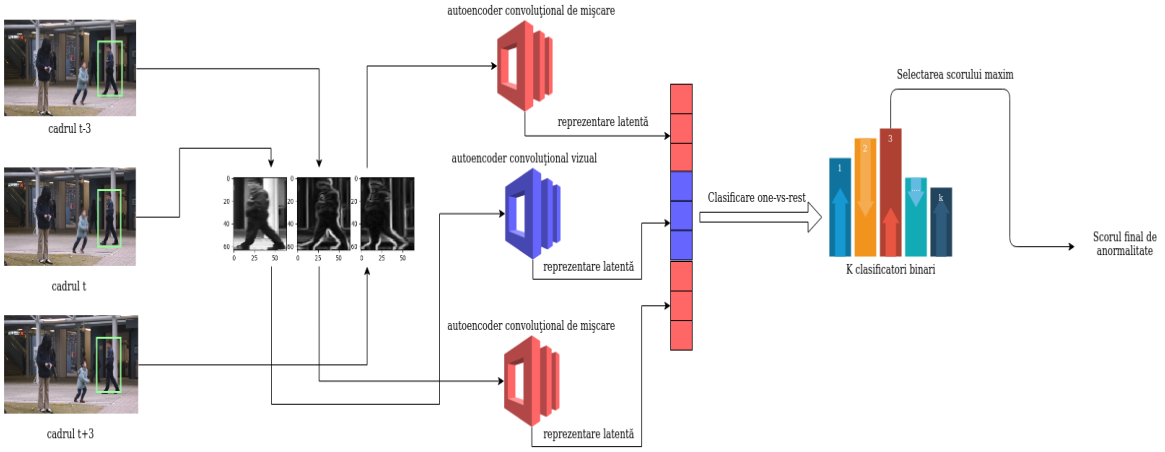

Figura 3.3: Arhitectura etapei de inferenţă

Pentru a obţine scorul unui eveniment, un pas foarte important este obţinerea vectorului de caracteristici corespunzător. Acesta se obţine folosind autoencoderele antrenate in etapa precedentă, după pre-procesarea cadrelor corespunzătoare evenimentului. Pre-procesarea este asemanătoare etapei de antrenare, şi anume, pentru fiecare obiect dintr-un cadru *t*, se selectează aceeaşi zonă din cadrele *t-3* si *t+3* cu scopul de a calcula gradienţii ce reflectă mişcarea obiectului relativ la cadrul t. Odată obţinute aceste 3 date (imaginea obiectului şi cei 2 gradienţi) aşa cum este descris şi in figura **??**, acestea sunt trecute prin autoencodere şi sunt obţinute reprezentările latente, ce sunt concatenate pentru a se obţine vectorul final de caracteristici. Vectorul de caracteristici este apoi clasificat de toţi cei k clasificatori binari antrenaţi anterior, iar scorul final de anormalitate este dat de formula :

$$score = max(g_i(v)), i \in [1..k]$$

unde v este vectorul de caracteristici, iar g_i este un clasificator binar.

Filmarea propriu zisă şi selecţia cadrelor este realizată pe partea de client, în timp ce pre-procesarea cadrelor, detecţia obiectelor si calcularea scorului pentru fiecare dintre aceste obiecte este executat pe server. Astfel, atât timp cât există o conexiune la reţea, clientul nu este limitat de puterea de calcul proprie, şi poate analiza mai multe videouri simultan. Totuşi clientul trebuie sa încarce cadrele ce vor fi analizate catre destinaţie S3 a serverului. Pentru a nu încetini semnificativ procesul de analiză,

trimitere cadrelor în reţea trebuie facută în paralel, folosind eventualele posibilităţi de multi-threading ale clientului.

În ceea ce priveşte serverul pe care rulează analiza propriu zisă, acesta este un HTTP API, ce expune pentru public capacitatea de a rula inferenţa. În stadiul curent, API-ul este configurat cu un singur endpoint, de tip POST : "upload/frame_key" , unde frame_key reprezintă cheia de acces cu care au fost urcate în cloud cadrele ce urmează sa fie analizate. Pentru ca o cheie de acces sa fie considerată validă de catre server, acesta trebuie sa găsească in destinaţia S3 specifică 3 fişiere ce au următoarele chei :

- frame_key
- frame_key_d3
- frame_key_p3

Frame_key este parametrul primit prin HTTP, iar frame_key_d3 şi frame_key_p3 sunt chei obţinute prin concatenarea sufixului **_d3** respectiv **_p3** la cheia primită ca parametru. Daca parametrul primit reprezintă o cheia valida, atunci cadrele sunt descărcate temporar pe server, unde sunt folosite pentru a rula intregul proces de inferenţă, iar in final, rezultatele sunt transmise clientului în format json. JSON-ul rezultat conţine 3 câmpuri :

- Codul de stare: ce reprezintă codul http rezultat în urma apelului
- Result: ce conţine scorul cadrului de anormalitate, ce reprezintă maximul dintre scorurile obiectelor din cadru.
- boxes: o listă ce conţine coordonatele obiectelor ce sunt considerate anormale din cadrul principal

Codul de stare respectă standardul HTTP1.1, mai precis *RFC7231*[8] şi respectă valorile prezentate în tabelul de mai jos:

Valoarea codului	Semnificaţie
1xx	Apelul a fost primit, dar procesarea încă este în desfaşurare
2xx	Apelul a fost primit şi procesat cu succes
3xx	Clientul trebuie să execute paşi suplimentari pentru ca apelul sa fie procesat
4xx	Apelul este greşit din punct de vedere sintactic sau nu poate fi procesat
5xx	Apelul este valid dar serverul a întâmpinat o eroare în timpul procesării

Tabela 3.1: Descrierea codurilor returnate de către server

Modul în care este structurat serverul face foarte uşoară crearea de noi API-uri, cu aceeaşi funcţionalitate, dar orientate spre tipuri de anormalitate diferite. Aşa cum a fost descris şi în introducere, detectarea anomaliilor este dependentă de setul de

date de referinţă. Astfel, fiecare model antrenat pe un set de date diferit, acoperă o arie diferită de anomalii. Altfel spus, în funcţie de setul de date folosit la antrenare, sistemul va detecta un alt set de evenimente drept anomalii. În fine, un nou API este necesar pentru fiecare astfel de model.

Deoarece arhitectura curentă a serverului încarcă modelele pre-antrenate dintr-o destinaţie S3 la momentul iniţializării, pentru a creea un nou API ce a fost antrenat pe un alt set de date, este necesară doar crearea unei noi destinaţii S3, încarcarea modelelor pre-antrenate la acea destinaţie, schimbarea în cod a denumirii vechii destinaţii în cea nouă şi apoi un nou deployment. Astfel, cu un număr minim de paşi, poate fi creat un nou API ce execută etapa de inferenţă pentru un sistem ce a fost antrenat pe un nou set de date.

Capitolul 4

Execuţia în cloud a sistemului

4.1 Cloud-computing în inteligenţă artificială

Execuţia în cloud este un domeniu cu o creştere substanţială în ultimii ani, înlocuind practic o bună parte din infrastructurile clasice existente. Acest lucru are implicaţii şi în dezvoltarea sistemelor de inteligenţă artificială, acesta fiind un domeniu ştiut drept unul cu necesar de putere de execuţie mare, dar si cu un potenţial de scalare extraordinar. Tocmai acest potenţial, face ca inteligenţa artificială sa fie candidatul perfect pentru execuţia în cloud. Faptul că efortul de dezvoltare pentru a creea un sistem în cloud pregătit să deservească milioane de utilizatori este egal cu dezvoltarea unui sistem pentru câteva mii de utilizatori în mod clasic, arată din nou de ce această soluţie a devenit alegerea perfectă pentru multe companii.

Deşi opţiunile pentru execuţia în cloud sunt vaste, de la sisteme complet adminsitrate de către utilizator, până la funcţii cloud complet administrate de distribuitor, tendinţa este ca acolo unde este posibil, clientul să se ocupe cât mai puţin de administrare, şi cât mai mult de dezvoltarea propriu zisă a produsului. Un alt subiect de interes pentru execuţia în cloud este comparaţia între arhitecturi *serverfull* si *serverless* dar alegerea între cele două diferă de la sistem la sistem deoarece pentru a alege o arhitectură serverless, sistemul trebuie construit pentru asta încă de la inceputul dezvoltării.

Sistemele de ML, sau IA, sunt deobicei folosite pentru a îmbunătaţii sisteme deja existente, sau pentru a uşura luarea deciziilor vitale. Din acest motiv, monitorizarea execuţiei, asigurarea disponibilităţii sistemului şi posibilitatea de a controla costurile sunt lucruri foarte importante pentru alegerea modului de execuţie a sistemului. Pe lângă asigurarea tuturor acestor facilităţi, soluţiile cloud oferă de multe ori si performanţe mai bune decât infrastructurile fizice echivalente, oferind astfel toate bazele necesare pentru găzduirea sistemelor IA/ML.

La ora actuală, primii 3 distribuitori de soluţii cloud (Amazon,Microsoft,Google) oferă şi soluţii speciale de implementare rapidă a inteligenţei artificiale în aplicaţii deja existente, folosind sisteme special gândite să suporte întreg procesul de dezvoltare si execuţie, totul înglobat într-un singur serviciu :

- Amazon SageMaker
- Google AI Platform
- Azure Machine Learning

4.2 Analiza opţiunilor

Opţiunile în ceea ce priveşte execuţia în cloud sunt diverse şi în continua dezvoltare. Deşi categoriile de soluţii sunt bine definite, serviciile propriu zise sunt modificate destul de des, avand din ce în ce mai multe facilităţi.
Categoriile de servicii cloud sunt :

- Infrastructure as a Service (IaaS)
- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Function as a Service (FaaS)

Soluţiile de tip *IaaS* presupun punerea la dispoziţia clientului doar a infrastructurii cerute, fara nici o administrare din partea distribuitorului. Din acest punct de vedere, soluţiile IaaS pot fi vazute de către client drept infrastructuri fizice, singura diferenţă fiind că locul în care este amplasată aparatura nu este deţinută de către client. În ceea ce priveşte securitatea, update-urile, softurile şi toate legăturile din cadrul infrastructurii, acestea sunt administrate strict de către client. Între a rula un sistem pe o infrastructură locală şi a rula un sistem pe o infrastructura aflată in cloud dar folosind o soluţie IaaS există doar diferenţe de natură economica.

SaaS reprezintă o metoda pentru distribuirea softurilor pe internet la cerere şi pe bază de subscripţie. Soluţiile SaaS sunt folosite pentru găzduirea şi adminstrarea softurilor pentru a facilita distribuţia de update-uri sau pathcuri de securitate.

Soluţiile *PaaS* reprezintă un mod de a crea medii de dezvoltare, testare sau producţie la cerere. Este construit pentru a asigura o modalitate rapidă de a crea aplicaţii software diverse, de la aplicaţii web, la cele mobile sau API-uri ce fac parte dintr-un alt sistem. Practic, este un mod de a creea si administra soluţii IaaS în mod automat, asigurând toate uneltele necesare pentru scalabilitate automată, distribuirea traficului între servere, mentenanţă, asigurarea securităţii şi lansarea de noi versiuni. Deşi este un serviciu serverfull, acesta ia de pe umerii clientului sarcina creeri şi

administrării infrastructurii, grăbind astfel procesul de dezvoltare şi lansare a unui nou sistem software.

In ceea ce priveşte FaaS, acesta este un domeniu nou, deoarece a apărut pentru prima data in 2010 fiind oferit de câteva start-upuri la acea vreme. Acest mod de dezvoltare orientat spre microservicii a devenit trendul in industrie în ultimii ani pentru sisteme cu potenţial de scalare mare, deoarece prezintă numeroase avantaje din punct de vedere al modului de dezvoltare şi de execuţie in industrie. În momentul de faţă, pentru servicii de tip FaaS sunt 3 mari jucători: Amazon cu AWS Lambda, Google cu Google Cloud Functions si Microsoft cu Azure Functions.[11]. Numeroase lucrări din domeniu [7, 14] arată ca rularea algoritmilor de machine learning folosind soluţii FaaS (Function as a service) precum AWS Lambda sau Google cloud functions, este în sine o problemă ce necesită soluţii de optimizare a codului pentru a indeplini restricţiile soluţiilor de rulare serverless, cum ar fi memoria limitată a mediului de execuţie.

O altă caracteristica a soluţiilor cloud este tipul de execuţie şi anume: *serverfull* sau *serverless*. Prin serverfull, se intelege o soluţie ce rulează pe servere ce pot fi indentificate în mod unic, care rulează în mod continuu, dar pe care se execută diferite operaţii în funcţie de nevoile sistemului. IaaS si PaaS sunt mereu soluţii serverfull, în timp ce un serviciu SaaS poate fi atat serverfull cât si serverless. Prin serverless se inţelege o soluţie ce lansează noi micro-instanţe pentru fiecare cerere, ce este mai apoi oprită dupa execuţia codului. Costul unei soluţii serverless este direct proporţional cu timpul de execuţie, în timp ce pentru soluţiile serverfull costul este constant.

Caracteristicile celor 2 tipuri sunt prezentate în tabelul de mai jos :

Caracteristici	Soluţii AWS serverfull	Soluţii AWS serverless
Când este activ serviciul	Când este declanşat de un eveniment	Continuu, până la oprire
Limbaj de programare	Python, Java, Go, C#, şi altele...	Orice limbaj
Max RAM	0.125 - 3 Gb	0.5-1952 Gb
Max Capacitate de stocare	0.5 Gb	0-3600 Gb
Max Timp de rulare	900 secunde	Nelimitat
Unitatea minimă taxată	0.1 secunde	60 de secunde
Preţ minim pe unitate	$0.0000002	$0.0000867
Sistem de operare	Ales de distribuitorul de soluţii cloud	Ales de către client

4.3 Comparaţie între soluţiile PaaS si FaaS

Atât soluţiile PaaS cât şi cele FaaS sunt pretabile sistemelor de inteligenţă artificială, deoarece oferă adminisitrare automată, putere mare de calcul şi avantajul că suportă optimizari prin rularea pe GPU.

Soluţiile PaaS, deşi oferă servicii de adminstrare automată, în spatele acestui mecanism se ascunde tot o infrastructură clasică, cu servere fizice ce rulează în mod continu, ce necesită echilibrarea traficului între acestea, lansarea de noi servere, lansarea de noi versiuni a aplicaţiei pe serverele deja pornite sau închiderea unor servere atunci cand nu mai sunt folosite. Toate acestea sunt realizate de către distribuitorul de soluţii cloud, punând la dispoziţia clientului un mediu gata să suporte orice tip de trafic către aplicaţie. Astfel dezvoltatorul se poate gândi doar la construirea unui server cât mai eficient, pentru a folosi resursele unei instanţe la maxim. Un aspect important pentru soluţiile PaaS este costul asociat instanţelor folosite. Deşi numarul lor este variabil, în orice moment măcar o instanţa trebuie sa fie activă. Asta inseamnă că dacă sistemul poate rula doar pe instanţe cu resurse suplimentare, ce au un cost orar mai mare, sistemul va avea un cost de operare mare chiar şi atunci cand este subutilizat. Un sistem ce foloseşte soluţii PaaS în mod ideal este de preferat sa ruleze pe mai multe instanţe cu capacitate de execuţie mică faţă de mai puţine instanţe cu capacitate mare, deoarece lansarea sau inchiderea de noi instanţe este gratuită, în timp ce rularea acestora atunci când sunt sub-utilizate nu este.

Soluţiile FaaS, pe de alta parte, abstractizează si mai mult infratructura existentă, luând orice responsabilitate de pe umerii clienţilor atunci când vine vorba de adminstrarea acesteia. FaaS presupune existenţa unei funcţii, ce simbolizează o aplicaţie, ce este executată de fiecare dată când apare un eveniment declanşator. Avantajul acestei abordări este costul asociat, şi anume faptul ca plata se realizează strict pentru timpul în care funcţia a fost executată. Timpul petrecut pentru a aştepta apariţia unui eveniment nu este taxat. Spre exemplu, soluţie FaaS a celor de la Amazon, AWS Lambda, poate fi declanşată în numeroase moduri, de la încarcarea unui fişier într-o destinaţie S3 prestabilită, la un apel HTTP către un endpoint ataşat funcţiei. Aceste soluţii suportă majoritatea limbajelor de programare, şi pot fi configurate în aşa fel încat sa acopere orice nevoie în ceea ce priveşte puterea de execuţie. Soluţiile FaaS prezintă şi o serie de dezavantaje printre care se numară:

- Există o limită asupra dimensiunii codului ce urmează sa fie executat de către funcţie.
- Timpul de execuţie a unei funcţii este deasemenea limitat.
- Starea programului nu este pastrată între execuţii facând necesară stocarea datelor ce trebuie sa persiste în locaţii dedicate (S3 sau o bază de date)
- Limitarea executării concurente. (Lambda nu permite mai mult de 1000 de funcţii sa fie executate concomitent)

Deşi din descrierea acestui tip de soluţii, pare alegerea ideală pentru execuţia etapei de inferenţă pentru un model de *machine learning*, situaţia devine complicată atunci când sistemul este unul complex, iar respectarea acestei limite de dimensiune devine

un blocaj. De multe ori, aplicaţiile de AI/ML depind de biblioteci si frameworkuri cunoscute în domeniu precum *tensorflow, keras, mxnet*, dar care au fost dezvoltate având în minte eficienţa execuţiei, şi nu minimizarea codului sursă. Având în vedere ca AWS Lambda limitează dimensiunea codului ataşat la 250MB, lansarea sistemelor ce se bazează pe reţele neuronale este o sarcină grea din punct de vedere al încadrării în aceste restricţii.

4.4 Descrierea soluţiei curente

Soluţia aleasă pentru execuţia etapei de inferenţă în cloud este *AWS Elastic Beanstalk*. Acest serviciu de tip PaaS oferă mediul perfect pentru lansarea aplicaţiei, atât din punct de vedere al configurabilităţii cât şi al securităţii oferite. Pentru a lansa o aplicaţie folosind elastic beanstalk, este necesar să creezi un pachet de lansare, ce conţine codul sursă si celelalte instrucţiuni pentru iniţializarea serviciului. Deşi acesta este limitat la 512MB, este mai mult decât indeajuns deoarece mediul de rulare si dependinţele programului nu sunt adaugate în acest pachet, ci sunt descarcate pe server la momentul lansării pe baza unui fişier de configurare adaugat în pachet numit *requirements.txt*.
În timpul lansării aplicaţiei, sunt create automat următoarele resurse :

- O flotă , sau o singură instanţă *EC2*, în funcţie de configuraţie
- Un *Elastic Load Balancer* ce distribuie traficul între instanţe
- Un *AutoScaling Group* ce defineşte regulile de scalare a numărului de instanţe
- O destinaţie *S3* în care este stocat codul sursă a aplicaţiei
- Şi un *API Gateway* ce joacă rolul de punct de intrare al traficului în aplicaţie

Deoarece la bază se află instanţe EC2, aceasta conferă un mare avantaj din punct de vedere al execuţiei concurente acestei soluţii faţă de AWS Lambda sau alte soluţii FaaS asemanătoare. Deşi mai multe instanţe Lambda pot fi executate concurent, în cadrul unei singure instanţe, apelurile concurente nu sunt permise. Luând în calcul ca în etapa de inferenţă sunt şi părţi de input/output în care resursele sunt nefolosite, folosirea instanţelor EC2 ce pot rula mai multe apeluri în mod concurent este opţiunea optimă din punct de vedere al folosirii resurselor.

Capacitatea de multi-threading din cadrul serverelor, la care se adaugă scalarea automată a numarului acestora în funcţie de traficul ce accesează API-ul, face ca resursele sa fie folosite la maxim, atât din punct de vedere al performanţei disponibile, cât şi din punct de vedere economic.

Un alt avantaj al acestei soluţii este usurinţa cu care se poate schimba mediul de execuţie. Spre exemplu, într-un anumit scenariu, în care se observă în panoul de

monitorizare ca instanţele EC2 curente sunt suprasolicitate, cu ajutorul interfeţei web, sau cea din linia de comandă, pot fi schimbate tipurile de instanţe EC2 folosite cu unele mai performante fără a avea serviciul indisponibil pe durata acestei modificări.

Listă de figuri

Listă de tabele

Bibliografie

[1] Amazon autoscaling description, 04 2020.

[2] Amazon ec2 description, 04 2020.

[3] Amazon elastic load balancing description, 04 2020.

[4] Amazon s3 description, 04 2020.

[5] K. Cheng, Y. Chen, and W. Fang. Video anomaly detection and localization using hierarchical feature representation and gaussian process regression. pages 2909–2917, 2015.

[6] F. Chollet et al. Keras description, 04 2020.

[7] A. Christidis, R. Davies, and S. Moschoyiannis. Serving machine learning workloads in resource constrained environments: a serverless deployment example. pages 55–63, 2019.

[8] R. Fielding and J. Reschke. Hypertext transfer protocol (http/1.1): Semantics and content. RFC 7231, RFC Editor, June 2014.

[9] M. Grinberg. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, Inc., 1st edition, 2014.

[10] R. T. Ionescu, F. S. Khan, M. Georgescu, and L. Shao. Object-centric auto-encoders and dummy anomalies for abnormal event detection in video. pages 7834–7843, 2019.

[11] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, et al. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019.

[12] Peng Qi, Wei Zhou, and Jizhong Han. *A method for stochastic L-BFGS optimization*. 2017.

[13] W. Sultani, C. Chen, and M. Shah. Real-world anomaly detection in surveillance videos. pages 6479–6488, 2018.

[14] H. Wang, D. Niu, and B. Li. Distributed machine learning with a serverless architecture. pages 1288–1296, 2019.