UNIVERSITATEA “ALEXANDRU IOAN CUZA“ IAŞI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENŢĂ

Învățare nesupervizata din video-uri

propusă de

Coțofană Victor

Sesiunea: iulie, 2019

Coordonator științific

Lect.dr. Ignat Anca

UNIVERSITATEA “ALEXANDRU IOAN CUZA“ IAŞI

FACULTATEA DE INFORMATICĂ

Învățare nesupervizata din video-uri

Coțofană Victor

Sesiunea: iulie, 2019

Coordonator științific

Lect.dr. Ignat Anca

Avizat,

Îndrumător Lucrare de Licenţă

Titlul, Numele şi prenumele _____

Data _____ Semnătura _____

DECLARAŢIE privind originalitatea conţinutului lucrării de licenţă

Subsemntatul(a) ……

domiciliul în ……..

născut(ă) la data de ……..…., identificat prin CNP ……..……., absolvent(a) al(a) Universităţii „Alexandru Ioan Cuza" din Iaşi, Facultatea de …….. specializarea ……., promoţia ……., declar pe propria răspundere, cunoscând consecinţele falsului în declaraţii în sensul art. 326 din Noul Cod Penal şi dispoziţiile Legii Educaţiei Naţionale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licenţă cu titlul: _____

_____

_____elaborată sub îndrumarea dl. / d-na _____, pe care urmează să o susţină în faţa comisiei este originală, îmi aparţine şi îmi asum conţinutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă să fie verificată prin orice modalitate legală pentru confirmarea originalităţii, consimţind inclusiv la introducerea conţinutului său într-o bază de date în acest scop.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice in vederea facilitării fasificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diploma sau de disertaţie şi în acest sens, declar pe proprie răspundere că lucrarea de faţă nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, …… Semnătură student ……

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Învățare nesupervizata din video-uri*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.
De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași,

Absolvent *Coțofană Victor*

_____

(semnătura în original)

Cuprins

Introducere

In ultimii ani domeniul inteligentei artificiale si in special invatarea automata au avut o crestere semnificativa pe plan mondial. O mare parte a atentiei industriei este axata pe partea de Deep Learning a domeniului, aceasta aducand rezultate exceptionale. Cu ajutorul acesteia, s-au creat modele care “invata” mult mai rapid si mai bine in comparatie cu un creier uman, oferind rezultate mai bune mai bune chiar si fata de specialistii dintr-un anumit domeniu.

Fiind intr-o era a informatiei, in care fluxul de intrare si creare a continutului depaseste capacitatea de procesarea a acesteia, apare dorinta si necesitatea de a gasi solutii mult mai eficiente decat cele actuale. Un exemplu de o astfel de problema este fluxul de incarcare de video-uri pe platforma YouTube. Conform statisticilor in fiecare minuta pe platfomra sunt incarcate in jur de 500 de ore de continut video nou[1]. Nu e vorba de problema de stocare, dar mai degraba e o problema la un nivel mai inalt, cum determini ce continut apare in aceste video-uri in cel mai scurt timp posibil? Problema nu se pune doar in scopul crearii categoriilor pentru utilizatori, dar si pentru filtrarea video-urilor interzise pe platforma. Avand problema de fata si avand toate progresele in domeniul inteligentei artificiale is a Deep Learning poate fi propusa o poisibila abordare catre solutie.

Lucrarea de fata prezinta implementarea unui model de retea neuronala care este antrenata sa clusterizeze video-uri nefolosind etichetele (eng. label) acestora de antrenare, fiind astfel invatare complet nesuprevizata. Modelul implementat si pe care il prezint in prezenta lucrare este bazat pe aritcolul original “*Unsupervised learning from videos using temporal coherency deep networks*” [2].

In **Capitolul 1** este descrisa problemei invatarii nesupervizate din video-uri si de ce difera fata de retelele neuronale traditionale si invatarea supervizata.

In **Capitolul 2** este prezentata in detaliu arhitectura modelului si implementarea acesteia. De asemenea sunt introduse conceptele teoretice folosite in arhitectura.

In **Capitolul 3** este descrisa multimea de date folosita pentru antrenarea modelului si algoritmul necesar de preprocesare a datelor pentru model.

In **Capitolul 4** sunt prezentate API-urile terte folosite atat si procesul de setare a masinii virtuale folosite.

In **Capitolul 5** sunt prezentate rezultatele, argumentarea si vizualizarea acestora si comparatia cu rezultatul din artioclul original,.

[1] https://expandedramblings.com/index.php/youtube-statistics/
[2] https://www.sciencedirect.com/science/article/pii/S1077314218301772

Contributii

Pentru realizarea lucrarii aveam nevoie de un anumit fundal de cunostinte pentru a intelege articolul mai bine. Fiind un domeniu nou pentru mine si avand doar cunostintele de la materia Retele Neuronale din anul 3 sem 1, am avut nevoie de mai multa informatie si resurse. Pentru acestea, am urmarit 2 saptamani ale cursului lui Andrew Ng a tutorialului online *Convolutional Neural Networks*[3] de pe platforma online de invatare Coursera. Avand deja o baza relativ buna, am putut intelege conceptele ofeirte de acest curs.

Avand cunostintele teoretice necesare, urmatorul pas a fost sa acumulez cunostinte de implementare a acestora. Pentur aceasta am folosit Keras[4], o librarie open-source pentru retele neuronale scrisa in Python. O introducere buna in aceasta librarie sunt tutorialele de pe site-ul oficial Tensorflow[5], unul din posibilele backend-uri folosite pentru Keras. Keras-ul ofera solutii relative rapide pentru cazuri generale si des folosite in domeniu, insa daca implementarea necesita abordari mai avansate este necesar ceva timp pentru a gasi solutia.

Avand o retea neuronala de antrenat am optat pentru o masina virtuala cu placa video. Aceasta a sporit considerabil viteza cu care modelul invata. Pentru a seta masina virtuala sa fie capabila sa ruleze framework-ul pe placa video a fost necesara instalarea anumitor drivere specifice programarii pe placa video.

Pentru determinarea hyperparametrilor, a fost necesara consultarea codului sursa[6] a autorilor articolului orignal. Pentru a monotoriza procesul invatarii am afisat la fieare iteratie intr-un spreadsheet online detalii necesare. Neavand control asupra masinii virtuale, si avand in vedere ca invatarea ia mult timp am optat pentru salvarea modelului la anumite perioade.

In final dupa invatarea modelului si avand rezultatele am creat o reprezentare a datelor care usureaza evaluarea modelului.

[3] https://www.coursera.org/learn/convolutional-neural-networks/home/welcome
[4] https://keras.io/
[5] https://www.tensorflow.org/guide/keras
[6] https://github.com/gramuah/unsupervised

Capitolul 1. Invatare nesupervizata din video-uri

Majoritatea modelelor de retele neuronale au un proces de invatare supervizat. Multimea de date de antrenare contine si etichete (eng. label) care reprezinta ce contine acea unitate de data, elementul esential pe baza cauruia modelul invata. In asta si consta invatarea supervizata, cand datele de intrare sunt cunoscute si ceea ce reprezinta ele (eng. ground truth) este cunoscut si ajuta la invatare.

Invatarea nesupervizata, pe de alta parte, nu foloseste etichete pentru invatare, si foloseste o alta arhitectura si o alta abordare. Astfel un model poate invata o anumita multime de date fara sa stie efectiv ce reprezinta acea multime de date.

Pentru a crea o retea neuronala care invata din video-uri nesupervizat, am unit doua arhitecturi de retele in una singura. Si anume, am creat o retea siameza ce contine doua retele neuronale convolutive (eng. Convolutional Neural Network). Retelele convolutive contin o arhitectura ce faciliteaza invatarea supervizata a imaginilor, iar retelele siameze au ca idee de baza invatarea nesupervizata pe baza a doua arhitecturi de retele neuronale.

Pentru a le combina am eliminat ultimul nivel din arhitectura retelei convolutive, care deseori e un SoftMax, si am oferit catre functia de invatare din reteaua siameza care e contrastive loss function rezultatul penultimului nivel, care reprezinta lista cu trasaturi a datelor intrate. Avand doua retele convoluitve, avem 2 reprezentari ale datelor care sunt introduse in functia contrastive loss care invata cat de asemanatoarea sau nu sunt acestea, astfel invatand nesuprevizat multimea de date de intrare.

Multimea de date de intrare este constituita din video-uri. Invatarea in sine constituie din antrenarea modelului pe baza unor video-uri in urmatorul mod. Din fiecare video sunt extrase cadrele necesare procesarii, si apoi sunt introduse in model fie ca perechi pozitive, cadre din acelasi video, fie ca perechi negative, cadre din video-uri diferite. Scopul e sa antrenam modelul sa plaseze cadrele din acelasi video-uri aproape in spatiul de reprezentare a trasaturilor si sa plaseze cadrele din video-uri diferite la o distanta de ele.

Capitolul 2. Arhitectura si implementare

2.1 Arhitectura detaliata a modelului

Modelul costa din doua retele neuronale convolutive, cu arhitectura inspirata din AlexNet, unite intr-o retea siameza. Parametrii celor doua retele sunt aceleasi. Inputul modelului este format dintr-o pereche de cadre, si un label care reprezinta relatia dintre aceste doua cadre-uri, fie pozitiva fie negativa. Pozitiva inseamna ca cadrele sunt din acelasi video si sunt succesive, adica una dupa cealalta, iar negative inseamna ca fac parte din video-uri complet diferite.

Arhitectura retelei neuronale convolutive folosita este bazata pe arhitectura AlexNet, avand aceasi dimensiune pentru intrare si nivele conectate la sfarsit, dar nivelele intermediare sunt diferite. Arhitectura este inspirata din codul sursa al articolului original. Cel mai probabil autorii au decis sa foloseasca o astfel de arhitectura pentru a mari capacitatea retelei de a extragere trasaturile cat mai diferite.

Retelele convolutive se unesc la ultimul nivel. Neavand nevoia de a clasifica datele de intrare, nu se ia nivelul de SoftMax. La unire se face un layer de dimensiunea 8193 care consta din doua reprezentari a trasaturilor cadrelor procesate, care sunt de dimensiunea 4096 fiecare si un label care indica relatiile dinre cadre. Cu ajutorul acestor date, reteaua invata. Fiind o retea siameza, esenta invatarii se face in functia de contrastive loss. Aceasta determina cat de asemanatoare sau nu sunt cadrele incarcate in model.

2.2 Implementarea in Keras

Implementarea AlexNet-ului poate fi usor gasita online. Adunand mai multe surse si consultand tutorialul din Coursera, si lucrare originala, am creat propria mea implementare a AlexNet-ului in Keras. Ceea ce m-a ajutat ulterior sa ajung la arhitectura folosita in articol.

Implementarea AlexNet-ului nu a fost grea, pentru ca e relativ straightforward si nu necesita aspecte avansate. Dificultatile au aparut cand am dorit sa adaptez ce aveam deja existent in ce doream. De exemplu, unirea a doua nivele (eng. layers) in Keras. Cel mai greu a fost construirea functiei constrastive loss. Precum functia necesita o conditie, natural m-am inclinat sa implementez solutia printr-o conditie `if`. Cum am spus, ultimul nivel al modelului are dimensiunea de 8193, naiv am presupus ca aceasta dimensiune reprezinta dimensiunea unui `numpy.array`, insa dupa mai multe ore am aflat ca nu aveam dreptate. Folosind Tensorflow ca backend pentru Keras, la initalizare se creaza in spate un graf pe care Tensorflow-ul il compileaza si apoi il ruleaza. Ce am eu ca intrare in

functia custom contrastive loss, nu era un `numpy.array` dar era un obiect `Tensor` din libraria Tensorflow. Acest obiect desi detine si el o dimensiune ca un `numpy.array`, se comporta foarte diferit. E mai mult o functie ce are un input si un output, care reprezinta un nod intr-un graf decat un simplu vector. Si deci nu puteam face conditionarea direct pe obiect. Astfel am incadrat conditia de evaluare in functia in sine. Am extras valorile trasaturilor celor doua cadre si label-ul folosit in trei obiecte de tip `Tensor`.

Keras ofera posibilitatea de a salva modelul sau greutatile (eng. weights) acestuia in orice moment al antrenarii. Acest lucru este foarte folositor luand in considerare ca antrenarea necesita mult timp de procesare, si in caz ca apare vreo eroare in codul sursa sau masina in care este rulat modelul intampina o eroare, sa nu fie pierdute datele antrenate deja. Pentru a salva tot modelul creat care face parte atat din nivelele create cat si greutatile (eng. weights) invatarii, este folosita functia `save` a obiectului `Model` din Keras. Rezultatul acestei salvari este un fisier de tip `HDF5`. Iar pentru incarcarea modelului salvat, se poate folosi functia `get_saved_model` din acelasi obiect `Model`, dand-ui ca parametru calea catre fisierul salvat. Avand o funcite custom, trebuie oferita explicit functiei de incarcare a modelului, altfel arunca o eroare.

Capitolul 3. Preprocesarea video-urilor

3.1 Multimea de date folosita

Pentru invatarea unui model de retele neuronale, fie supervizate sau nu, este necesara o anumita multime de date pe baza careia modelul invata. Datele pot fi de orice tip, fie text, imagine, video etc. Aceasta multime de date poate fi selectata manual pentru a satisface cerinta unei anumite probleme sau poate fi folosita o multime de date deja creata in urma unor anumite studii care se axeaza pe un anumit domeniu. Multe multimi de date pot fi gasite online si descarcate gratuit, astfel incat majoritatea cercetatorilor le fac publice dupa cercetare. Exemple de multimi de date publice: MNIST, ImageNet, CIFAR-10 etc.

Astfel incat acumularea unei multimi de date specifice unei probleme ar lua mult prea mult timp am decis sa folosesc o multime de date deja existenta si care contine datele de care am nevoie. Multimia de date folosita pentru antrenarea modelului este UCF101[7]. UCF101 este o multime de date pentru recunoastere de actiuni ce contine 13,320 video-uri colectate din YouTube si clasificate in 101 categorii. Aceste categorii pot fi impartite in 5 tipuri: 1) Interactiunea om-obiect 2) Miscarea corpului 3) Interactiunea om-om 4) Actiunea de a canta la un instrument muzical 5) Actiuni sportive

Fiecare categorie contine 25 de grupe. O grupa de video-uri este in esenta un video lung dedicat unei actiuni ce a fost impratit in 4-7 video-uri mai scurte, in lungimi ce variaza intre 5 si 10 secunde. Video-urile din aceeasi grupa pot sa impartaseasca aceleasi trasaturi ca: fundal asemanator, acelasi punct de vedere etc. Aceasta abordarea este binevenita astfel incat usureaza procesul de preprocesare.

Pe langa arhiva ce contine multimea de date, pe site este publicata o alta arhiva ce contine 3 impartiri diferite a multimii pentru antrenare si testare. Fiecare impartire contine in jur de 9500 de video-uri de antrenare si aproximativ 3500 de vido-uri de testare. Aceste impartiri sunt exprimate ca si liste. Atat listele pentru antrenare si pentru testare folosesc caile relative catre fiecare video ce trebuie procesat. La listele de antrenarea este prezent si eticheta (*eng. label*) acestui video ce reprezinta categoria din care face parte. Eticheta aceasta este reprezentata ca un numar, iar relatia dintre aceste numere si categoriile din multimea de date este gasita intr-un alt fisier, de asemenea prezent in arhiva.

La inceput am dorit sa reproduc cat de mult posibil ce au facut autorii articolului original, astfel am decis sa iau acleasi pachet de antrenare/testare de video-uri folosita de ei. Acesta constitue

[7] https://www.crcv.ucf.edu/data/UCF101.php

din 9537 de video-uri pentru invatare si 3783 pentru testare. Insa, precum este detaliat in **Capitolul 4. Antrenarea modelului is masina virtuala**, am fost nevoit sa reduc dimenisunea multimii de date, din cauza lipsei de putere de procesare. Astfel din 101 categorii am scazut la doar 10, din 9537 de video-uri pentru invatare am scazut la 1029, iar din 3783 de video-uri de testare am scazut la 409.

Avand libertatea de a alege tipul de categorie, am selectat *Actiuni sportive*. Categoriile selectate sunt:

1. Tragerea cu arcul (eng. Archery)
2. Biliard (eng. Billiards)
3. Bowling
4. Lovitura de golf (eng. Golf Swing)
5. Aruncarea cu ciocanul (eng. Hammer Throw)
6. Curse de cai (eng. Horse Race)
7. Canotaj (eng. Rowing)
8. Schi (eng. Skiing)
9. Lovitura de pedeapsa in fotbal (eng. Soccer Penalty)
10. Lovitura de tenis (eng. Tennis Swing)

3.2 Algoritmul de preprocesare

Modelul are ca input doua cadre, sau mai bine spus doua reprezentari a doua cadre. Multimea de date este formata doar din video-uri, deci apare necesitatea unui algoritm de preprocesare a video-urilor in cadre separate pentru a putea fi inroduse in model pentru antrenare. Sunt doua metode de abordare a preprocesarii: preprocesarea in timpul antrenarii si preprocesarea independenta de antrenare.

Preprocesarea in timpul antrenarii consinsta din procesarea datele doar atunci cand am nevoie de ele. Astfel parcurg lista de video-uri pe care trebuie sa le ofer modelului, extrag cadrele din video, le introduc in model, apoi continui cu urmatorul video. Aceasta abordarea ar fi potrivita daca ar exista o limita de memorie de stocare, astfel in memorie vor fi doar cadrele unui singur video. Partea negativa a acestei abordari este ca daca modelul esueaza, trebuie sa reia toata preprocesarea din nou. Dar pe langa asta, precum a fost mentionat si in **Capitolul 2. Arhitectura si implementare**, modelul necesita perechi de cadre atat pozitive, din acelasi video, cat si perechi de cadre negative, din cadre diferite. Asta ar insemna ca pentru fiecare cadru negativ ar trebui sa preprocesam un video aleator in plus pe langa cel care se antreneaza. Si precum la fiecare a 5-a pereche una trebuie sa fie negativa, vom preprocesa multe video-uri doar pentru un singur cadru aleator. Ceea ce aduce o

anumita ingreunarea a antrenarii modelului. Aceasta poate fi evitata folosind cealalta metoda de preprocesare.

Prerpocesarea independenta de antrenare consista din preprocesarea a tuturor video-urilor inainte de antrenare si apoi parcurgand lista cu video-uri pe care trebuie sa le invate modelul, ofer modelului doar calea aboluta ca cadrele strict necesare invatarii. Astfel, separam actiunea de invatare a modelului cu cea de preprocesare a input-ului. Abordarea aceasta mai aduce un beneficiu. O data ce video-urile au fost procesate, toate cadrele raman acolo, si daca avem nevoia sa rerulam modelul, nu avem decat sa rerulam doar modelul, nu si preprocesarea. Astfel am salvat si timp.

Odata ce am ales modul de executie a preprocesarii am rulat niste teste sa vad cata memorie este necesara pentru toate cadrele procesate. Dimensiunea dataset-ului original este de 6.69GB, dezarhivat. Am rulat preprocesarea pe laptop-ul meu si am avut aceste rezultate. Pentru 30 de video-uri din care am extras toate cadrele, dimensiunea folder-ului a crescut la 73.9MB in 19.4 secunde. Considerand ca avem peste 9500 de video-uri, dimensiunea ar fi crescut la aproximativ 23.5GB, iar timpul, pe masina mea locala, iar in timp ar fi 1.7 ore. Timpul rularii este prezentat doar ca un punct de reper, in realitate preprocesarea a fost rulata pe masina virtuala, unde aveam mult mai multa putere computationala.

Realizand ca mi-ar fi luat prea mult timp si spatiu pentru preprocesare, am dorit sa caut o solutie mai eificienta. Fiecare video contine intre 100 si 200 de cadre, dar nu avem nevoie de fiecare cadru pentru antrenare. Continuitatea intre cadre nu asigura informatie noua. Pixelii a doua cadre succesive nu difera foarte mult, si astfel nu contribuie la invatarea noilor trasaturi (eng. features). De aceea am procesat din 10 cadre unul singur. Acest numar a fost ales in urma observarii empirice atat si consultand codul sursa confuz din articolul original. Schimband algoritmul inital de preprocesare, sa salveze doar al 10-lea cadru din video, am obtinut rezultate mult mai bune. Ruland acelasi test de 30 de video-uri am obtinut, 9.05MB dimensiunea folder-ului si timp de 4 secunde. Ceea ce reprezinta aproximativ o imbunatatire de 5 ori in timp si aproximativ 8 ori in spatiu. Experimentul de fata reprezinta doar un punct de reper pentru algoritmul initial.

Pentru a citi cadrele dintr-un video am folosit libraria OpenCV[8] si anume obiectul `VideoCapture`. Initial am creat un algoritm care avea nevoie de numarul total de cadre dintr-un video. Pentru aceasta extrageam din obiectul `VideoCapture` creat proprietatea `cv2.CAP_PROP_FRAME_COUNT`. Dar dupa ce am rulat modelul in masina virtuala, am depistat o eroare, mai degraba o consecinta nedorita. Proprietatea mentionata anterior nu ofera numarul exact

[8] https://opencv.org/

de cadre, ci mai degraba ofera doar o aproximare a acestui numar. Avand nevoie de numarul exact, trebuia sa caut o alta solutie.

Precum input-ul in model cere doua cadre, unul negativ si unul pozitiv, trebuia sa iau si asta in calcul cand cream algoritmul. Dupa consultarea din nou a codului sursa original, am decis sa iau 4 din 5 perechi de cadre sa fie pozitive, adica din acelasi video, si a 5-a sa fie negativa, adica un frame aleator dintr-un video aleator dintr-o categorie aleatoare. Avand in vedere acest lucru am ajustat algoritmul inital sa salveze si cadrele imediat urmatoare cand este cazul si sa nu le salveze cand va fi perechea negativa. Astfel optimizand timpul de preprocesare. In loc sa folosesc numarul de frame-uri totale din video, am folosit o asa numaita fereastra de doua frame-uri. Cat timp ambele frame-uri erau prezente in fereastra, faceam verificarea, anume daca suntem la a 10-lea cadru din video, si daca este necesar cadrul imediat urmator sau unul negativ.

Cadrele sunt salvate in felul urmator. Mai intai fiecare capitol isi are propriul folder. Apoi in acest folder fiecare video initial isi are propriul folder in care se contin toate cadrele extrase din acest video. Cadrele care vor fi procesate au urmatoarea denumire: `query_frame_xxxxx.jpg`, unde `xxxxx` reprezinta numarul cadrului procesat. Cadrul imediat urmator din video, care este necesar pentru perechea poztiva din model se afla in acelasi folder sub denumirea de `next_frame_xxxxx.jpg`, unde `xxxxx` reprezinta numarul de cadru care il reprezinta.

Modelul are ca si forma de intrare o dimensiune fixa, si anume 227 pe 227 pixeli. Fiecare video are dimensiuni diferite, deci fiecare cadru din fiecare video trebuie redimensionat. Preferabil aceasta trebuie facuta in timpul preprocesarii, ca in final sa avem din toata multimea de video-uri doar cadrele strict necesare pentru invatarea modelului. Astfel in algoritm-ul de preprocesare inaintea salvarii cadrului, il redimensionez folosind `cv2.resize`, iar apoi il salvez ca fisier `jpg` folosind `cv2.imwrite`.

Capitolul 4. Invatarea modelului si masina virtuala

4.1 Google Cloud Platform

Pentru invatarea a oricarei retele neuronale este necesara putere computationala. Pentru o rularea rapida, si o invatare relativ rapida a modelului, am decurs la solutii cloud, si in anume Google Cloud Platform. La inregistrarea unui cont ei iti ofera $300 credit valabili 12 luni. Cu acest credit poti face anumite lucruri pe platforma lor, care de altfel ar fi trebuit sa platesti pentru ele. Unul din ele, si ceea ce am folosit eu a fost Computing Engine, adica o masina virtuala.

La inceput doream sa configurez o masina virtuala pe cat mai mult timp posibil, adica cat mai ieftin dar care o pot folosi. Cu aceasta in gand am configurat o masina virtuala foarte slaba, avand doar 2 procesoare, fara placa video si o memorie RAM de 6GB la aproximativ $75 pe luna, cee a ce mi-ar fi oferit in jur de 3 luni. Cand am intrat pe masina virtuala, folosind tool-ul integrat in Windows: Remote Desktop Connection, rapid mi-am data seama ca nu o sa fie deajuns, si m-am gandit la alta solutie.

In loc sa maximizez timpul folosit de masina virtuala, ar fi mai bine sa maximizez resursele oferite de Google Cloud Platform. Pentru aceasta am dorit sa configurez o masina virtuala cu o placa video. La inceput doream iar sa maximizez timpul, dar am observat rapid ca pretul pentru masinile virtuale cu placi video sunt considerabil mai scumpe decat cele fara, si pe langa asta platforma ofera doar placi de video super performante.

Platforma nu iti permite sa creezi o masina virtuala cu placa video asa simplu. Daca vrei sa creezi una trebuie sa explici motivul pentru care vrei sa folosesti o placa video. In urma cererii ei iti trimit un email in care iti explica ca cererea este procesata in timp de 2 zile lucratoare, si daca e urgent poti sa le trimiti un reply sa explici de ce e urgent. Am depus cererea seara si a doua zi dimineata a fost aprobata.

Dupa o analiza a posibilitatilor de configurare a noii masini virtuale, am ajuns la o concluzie decisiva. Masina pe care am rulat codul arata asa: 4 procesoare, 15 GB RAM, placa video NVIDIA TESLA V100 si rula Windows Server 2016. Toate la pretul de aproximativ $1200 pe luna sau $2.04 pe ora. Partea buna din asta este ca esti taxat doar pe ceea ce consumi. Adica aveam oportunitatea de a calcula cate iteratii posibile pot face in masina mea virtuala pana cand nu o sa mai am credit. Si exact asta am facut.

4.2 Google APIs

Ruland codul pe o masina virtuala asupra careia nu am acces, am vrut sa fiu sigur ca pot vedea progresul avand acces doar la internet si sa salvez modelul la un anumit interval. Pentru a asigura aceste doua functionalitati am folosit Google Spreadsheet API si Google Drive API.

Google Spreadsheet API, l-am folosit pentru scrierea informatiilor relevante intr-o foaie de stil (eng. spreadsheet). In timpul invatarii scriam cate o linie din tabel la fiecare iteratie. Informatia relevanta includea 4 lucruri: cand a fost scris lina in tabel (eng. timestamp-ul), cate iteratii au trecut, cate ore au trecut de la inceputul antrenarii (ajustate la fusul orar din Romania, masina virtuala fiind in Iowa, SUA), si daca a fost salvat modelul sau nu. Inserand aceste date intr-un spreadsheet ce poate fi accesat de mine oricand, puteam vedea progresul invatarii modelului de oriunde aveam acces la internet.

Pe langa liniile logate in timpul antrenarii, am setat ca fiecare eroare intampinata sa fie afisata. Astfel in caz ca apare o eroare in timpul rularii, o voi vedea si o sa pot sa intru pe masina virtuala si sa o corectez. Precautia aceasta s-a dovedit a fi folositoare.

La fel, in acelasi spreadsheet, am afisat si progresul de testare a modelului si rezultatul final. Astfel in acest spreadsheet, poate fi observata tot procesul invatarii si evaluarii modelului.

Google Drive API, l-am folosit pentru salvarea fisierului `HDF5` ce reprezenta modelul salvat, pe platforma de stocare a datelor in cloud Drive. La fiecare a 10-a iteratie salvam modelul si il trimiteam in contul meu Drive. Am creat modulul relativ generic astfel incat sa pot trimite orice tip de fisier as avea nevoie. A fost folositoare aceasta abordare, astfel incat am trimis atat modelul salvat cat si imaginile ce reprezentau vizualizarea datelor in format `jpeg`.

4.3 CUDA

Odata ce am decis sa folosesc placa video pentru invatarea modelului, am configurat masina virtuala cu placa video si am pornit-o, urmatorul pas era sa instalez driver-ele necesare pentru a permite rularea codului pe placa video. Pe site-ul oficial Tensorflow[9] sunt afisate resursele necesare pentru instalarea driverelor necesare permiterii rularii framework-ului pe placa video. Desi link-ul oficial ofera resuresele potrivite pentru ce ai nevoie, nu ofera si versiunile care chiar functioneaza. Natural astepti sa mearga ultima versiune cel mai bine, dar din pacate nu e cazul cand e vorba de programare pe placa video. Dupa instalarea destul de lunga a resurselor Nvidia si CUDA, cand verifici incerci sa verifici prin Tensorflow ce placa video e valabila, primesti erori foarte neintuitive, erori direct in cod pentru ca nu gaseste anumite resurse. Dupa mai mult timp in StackOverflow si Github

[9] https://www.tensorflow.org/install/gpu

issues am gasit o configuratie care la cineva mergea. Am instalat-o la mine si a mers. E foarte neintuitiv pentru ca sunt mai multe componente fiecare cu versiunile sale, si daca nu selectezi versiunile compatibile, nu o sa functioneze si nici nu o sa ai eroare clara care ar indica sursa problemei.

Configuratia care a functionat pentru masina virtuala folosita a fost urmatoarea:

- CUDA 9.0
- Tensorflow 1.12.0
- CUDNN 7.4.1.5

Kears-ul daca detecteaza o placa video pe masina pe care este rulat, ruleaza automat in spate fara sa setezi tu ceva anume explicit. Confirmarea vine la intializarea modelului.

4.4 Antrenarea modelului

Precum am spus aveam $300 credit in Google Cloud Platform. Dupa instalarea driver-elor necesare programarii pe placa video, dupa preprocesarea video-urilor am rulat modelul o singura data sa monotorizez timpul rularii unei iteratii. La inceput voiam sa refac acelasi experiment facut in articolul original, dar rapid mi-am dat seama ca nu e posibil.

M-am logat in conturile Google pentru autentificarea API-urilor, si am rulat o iteratie plina pentru a verifica daca nu apar erori. Odata ce m-am asigurat ca iteratia merge bine am pornit antrenarea efectiva.

Dupa 58 de ore de invatare continua, o eroare de conexiune a API-ului Google SpreadSheet a fost aruncata seara. Am observat-o in spreadsheet dimineata, si in tot timpul parcurs masina virtuala continua executia si am pierdut cativa dolari din creditul ramas degeaba. Pentru momente din astea am implementat salvarea modelului. Am intrat inapoi in masina virtuala, am schimbat numarul de iteratii, calculand de la cea la care s-a salvat modelul, am incarcat modelul salvat si am rerulat invatarea. Eroarea asta m-a costat 250 de iteratii scazand numarul de iteratii totale la 500. In total modelul a fost antrenat pentru aproximativ 68 de ore continue, adica aproximativ 2.8 zile.

Capitolul 5. Vizualizarea rezultatelor

Dupa antrenarea modelului, natural ar trebui sa il evaluam. Pentru asta am introdus video-urile dedicate testelor in model, si am extras toate trasaturile din acestea. Pe baza vectorului de trasaturi, am rulat algoritmul Kmeans. Pe baza rezultatelor Kmeans-ului si etichetele video-urilor, am determinat entropia conditionala a predictiei modelului. Entropia conditionata a modelului antrenat din articol este de **3.91**, pentru modelul antrenat de mine am obtinut obtinut **2.60**, cu cat mai mica cu atat mai bine. Diferenta dintre aceste doua rezultate poate fi explicata in multe moduri. Unul ar fi diferenta enorma intre setul de date folosit de mine si setul de date folosit in articolul original. Am folosit un set de date de 10 ori mai mic, si am rulat mult mai putine iteratii.

Pentru vizualizarea mai buna a rezultatelor am folosit libraria `matplotlib`. Am inserat rezultatele in doua imagini, una contine clasificarea cadrelor in urma aplicarii Kmeans-ului iar alta reprezinta insasi cadrele in plan euclidian. Ambele imagini au avut nevoie de preprocesare pentru a putea fi create. Fiecare cadru procesat prin model returneaza o lista de trasaturi (eng. feature space) de dimensiunea 4096. Pentru a putea vizualiza mai bine aceste trasaturi le transformam intr-o reprezentarea de 2 dimensiuni, adica in plan euclidian. Pentru aceasta am folosit algoritmul Barnes-Hut tSNE din libraria `sklearn.manifold`.

Astfel in **Figura 1** am reprezentat rezultatul Kmeans-ului. Fiecare culoare reprezentand un cluster. Desi lucrarea se bazeaza pe invatarea nesupravegheata a video-urilor, pentru simplificarea procesului de evaluare si crearea a rezultatelor am ales, la fel ca si autorii lucrarii originale, sa folosesc numarul de clustere cunoscut deja.

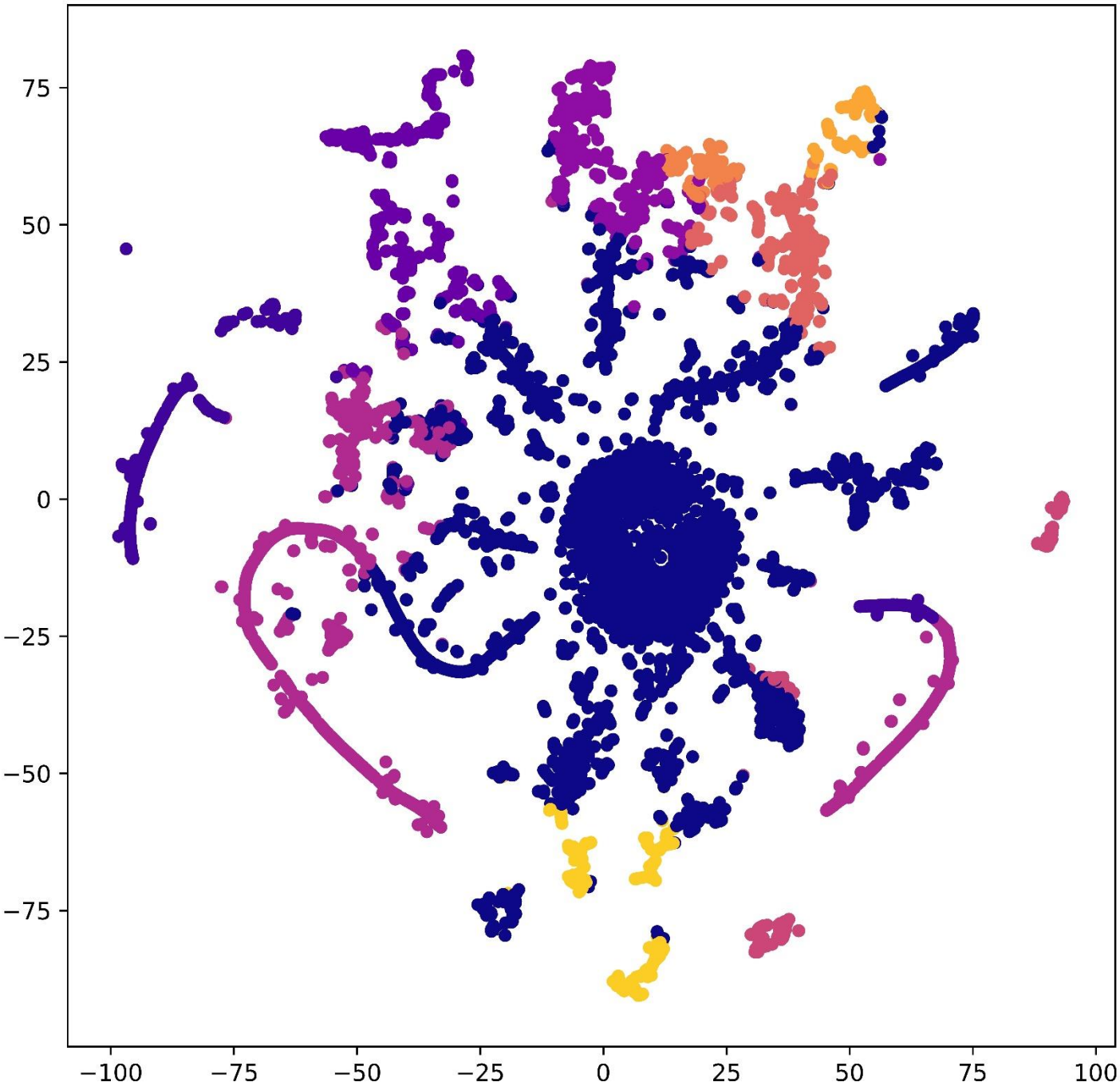

Figura 1: Clasificarea algoritmului Kmeans-ului a cadrelor din video-urile de antrenare

In a **Figura 2** este afisata reprezentarea fiecarui cadru procesat, in plan euclidian. Aici se poate vedea cel mai bine rezultatele invatarii modelului.

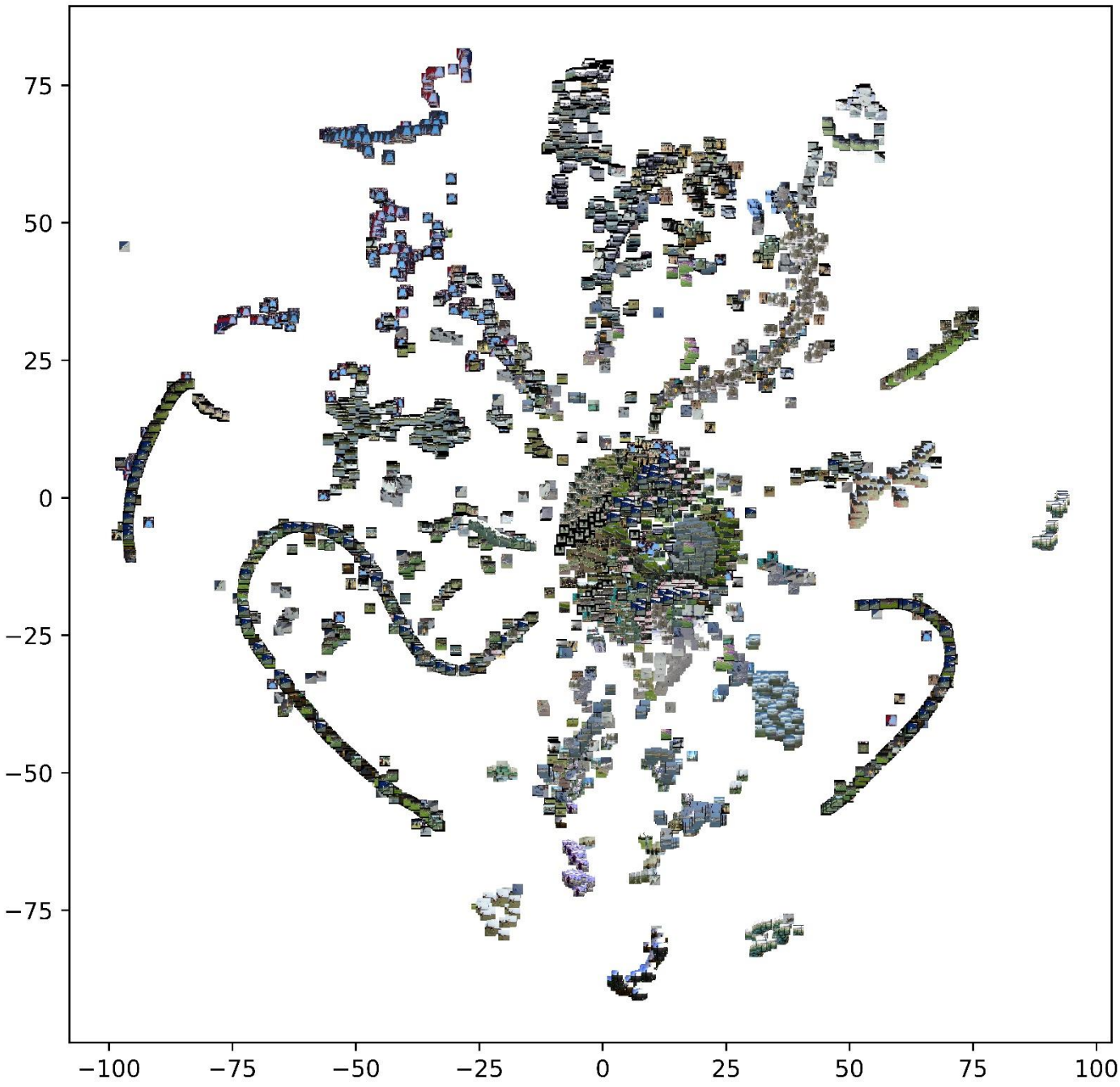

Figura 2: Reprezentarea in plan euclidian a trasaturilor fiecarui cadru procesat.
Cel mai bine vizualizat cu zoom

Concluzii

Rezultatele lucrarii prezinta doar un mod de executare a antrenarii modelului. Orice modificare a anumitor pasi intermediari ar putea aduce rezultate diferite.

Avand mai mult timp pe masina virtuala, modelul ar fi invatat mai bine. Astfel am fi putut avea distante mai mici intre cadrele din aceleasi categorii, si distante mai mare intre cadrele video-urilor din categorii diferite. Eventual am fi putut avea o distingere clara intre categorii. Extindearea antrenarii modelului este posibila pentru ca modelul a fost salvat.

O extindere a acestui experiment ar fi sa folosim greutatile (eng. weights) modelului antrenat, si sa sa continuam antrenarea dar deja supervizata. Astfel incepand antrenarea cu un model deja partial antrenat.

Bibliografie

[1] 160 YouTube Statistics and Facts (2019) | By the Numbers
https://expandedramblings.com/index.php/youtube-statistics/
[2] Unsupervised learning from videos using temporal coherency deep networks
https://www.sciencedirect.com/science/article/pii/S1077314218301772
[3] Convolutional Neural Networks – Coursera Course
https://www.coursera.org/learn/convolutional-neural-networks/home/welcome
[4] Keras
https://keras.io/
[5] Keras – Tensorflow Guide
https://www.tensorflow.org/guide/keras
[6] Unsupervised learning from videos using temporal coherency deep networks: GitHub repository
https://github.com/gramuah/unsupervised
[7] UCF101 - Action Recognition Data Set
https://www.crcv.ucf.edu/data/UCF101.php
[8] OpenCV
https://opencv.org/
[9] Tensorflow GPU support
https://www.tensorflow.org/install/gpu