### UNIVERSITATEA DIN BUCUREȘTI

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA INFORMATICĂ**

**Lucrare de licență**

# TITLUL LUCRĂRII DE LICENȚĂ

### Absolvent Numele studentului

**Coordonator științific**

**Titlul și numele profesorului coordonatorului**

**București, iunie 2021**

### UNIVERSITATEA DIN BUCUREȘTI

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**SPECIALIZAREA INFORMATICĂ**

**Lucrare de licență**

# TITLUL LUCRĂRII DE LICENȚĂ

### Absolvent Numele studentului

**Coordonator științific**

**Titlul și numele profesorului coordonatorului**

**București, iunie 2021**

**Rezumat**

Statisticile miscarilor dintr-un meci de tenis pot oferi un mare avantaj in pregatirea unui jucator. O cunoastere detaliata a unui oponent poate fi benefica in pregatirea unui jucator, oferind un avantaj in intelegerea punctelor forte si a celor slabe ale unui adversar, ceea ce aduce un plus pentru desfasurarea antrenamentelor.

Pentru aceasta vom crea o aplicatie ce va oferi predictii asupra miscarilor unui jucator dintr-un videoclip folosind limbajul python si biblioteca PyTorch pentru a implementa modelele.

**Abstract**

The statistics of the movements in a tennis match can offer a great advantage in the preparation of a player. A detailes knowledge of an opponent can be beneficial in the preparation of a player, offering an advantage in understanding the strengths and weakness of opponent, which brings a plus for training.

For this we will create an application that will provide predictions on the movements of a player from a video using python and the PyTorch library to implement the models.

Contents

[1. Introducere 5](#_Toc133672565)

[1.1. Motivatie 5](#_Toc133672566)

[1.2 Structura lucrarii 5](#_Toc133672567)

[1.2.1: Detectia jucatorului 6](#_Toc133672568)

[1.2.2: Predictia actiunii 6](#_Toc133672569)

[2. Concepte teoretice 7](#_Toc133672570)

[2.1: LSTM 7](#_Toc133672571)

[2.2 YOLO 8](#_Toc133672572)

[2.3 YOLO pose 8](#_Toc133672573)

[3. Abordari recente 9](#_Toc133672574)

[4. Tehnologii folosite 11](#_Toc133672575)

[4.1 Python 11](#_Toc133672576)

[4.2 PyTorch 11](#_Toc133672577)

[4.3 Matplotlib 11](#_Toc133672578)

[4.4 OpenCV 11](#_Toc133672579)

[4.5 YOLOv7 12](#_Toc133672580)

[5. Abordarea propusa 13](#_Toc133672581)

[5.1 Descrierea setului de date 13](#_Toc133672582)

[5.2 Configurarea modelului YOLO 14](#_Toc133672583)

[5.3 Transformarea setului de date 15](#_Toc133672584)

[5.4 Normalizarea datelor 16](#_Toc133672585)

[5.5 Incarcarea datelor si crearea secventelor 17](#_Toc133672586)

[5.6 Crearea modelului 19](#_Toc133672587)

[5.7 Antrenarea modelului 21](#_Toc133672588)

[6. Experimente si rezultate 24](#_Toc133672589)

[6.1. Prezentarea metricilor de acuratete si primele rezultate 24](#_Toc133672590)

[6.2 Interpretarea rezultatelor 27](#_Toc133672591)

[7. Descrierea aplicatiei 29](#_Toc133672592)

[7.1 Front-end 29](#_Toc133672593)

[7.2 Back-end 30](#_Toc133672594)

[8. Bibliografie 33](#_Toc133672595)

# 

# Introducere

### 1.1. Motivatie

Analiza video a sporturilor este o solutie pentru a extrage diferite informatii care pot fi folosite de profesionisti cat si de telespectatori pentru a avea o privire de ansamblu analitica a ceea ce se intampla pe parcursul jocului.

O astfel de tehnologie este sistemul „Hawk-Eye Live” care poate semnala daca mingea a fost data afara suprafetei de joc. Insa instalarea acestui sistem este dificila si costisitoare, conform [Espn, 2023] costul pentru instalarea acestui sistem pe un teren este intre $60.000 si $70.000, facand folosirea acesteia imposibila pentru antrenamente sau pentru turneele mici.

Scopul acestei lucrari este de a oferi o solutie accesibila din punct de vedere computational, fiind usor de rulat pe un GPU.

Lucrarea propune o modalitate de a vedea ce miscari precise efectueaza un jucator de tenis in cadrul unei filmari, pentru a putea fi analizata. Prin intermediul acestei abordari se poate analiza atent jocul jucatorului, antrenorii putand imbunatati strategiile, tehnica de joc, precum si pregatirea fizica inaintea meciurilor.

Abordarea aleasa va folosi limbajul pyhon pentru implementare si tehnici inovatoare pentru a detecta jucatorul si coordonatele pozitiilor membrelor. Astfel pentru detectia jucatorului vom folosi YOLOv7 care conform [Wang et al., 2022] intrece toti detectorii in privinta vitezei si a acuratetei, oferind o acuratete de 56.8% pe setul de date MS COCO. Pentru obtinerea coordonatelor membrelor vom folosi un detector de pose estimation antrenat pe YOLOv7 avand la baza tehnica prezentata in [Maji et al., 2022] care ofera o acuratete buna, de 68.5 pe setul de date COCO 2017 si are o performanta computationala foarte buna.

### 1.2 Structura lucrarii

* **Detectia jucatorului**: Selectarea jucatorului care are dimensinile cele mai mari din fiecare cadru, extragerea coordonatelor membrelor si normalizarea datelor pentru a putea fi prelucrate.
* **Predictia actiunii**: Antrenarea, proiectarea si inferenta intr-un videoclip bazandu-se pe coordonatele membrelor pentru a oferi predictia miscarii efectuate, pe un segment de video, avand numarul de date prestabilit ca hiperparametru.

### 1.2.1: Detectia jucatorului

Pentru a detecta jucatorul vom folosi modelul YOLOv7 preantrenat pentru detectia pozitiilor membrelor, pentru a detecta toate persoanele dintr-o imagine. Dupa aceasta detectie vom selecta numai persoana care are dimensiunea cea mai mare din cadru (daca ar fi feed-ul unei camere din transmisia TV, acest model va selecta numai jucatorul din partea de jos). Pentru fiecare membru din „cutia” dominanta ca si dimensiuni, coordonatele acestuia se vor normaliza in intervalul [0, 1]. Aceasta operatiune se va repeta pentru fiecare cadru dintr-o filmare

### 1.2.2: Predictia actiunii

Vom lua datele de la punctul anterior, ne vom folosi de un set de date adnotat cu videoclipuri cu lovituri din tenis pentru a antrena un model care sa poata da o predictie asupra miscarii efectuate de catre un jucator, oferind ca iesire una din 12 miscari, elementare jocului de tenis.

# Concepte teoretice

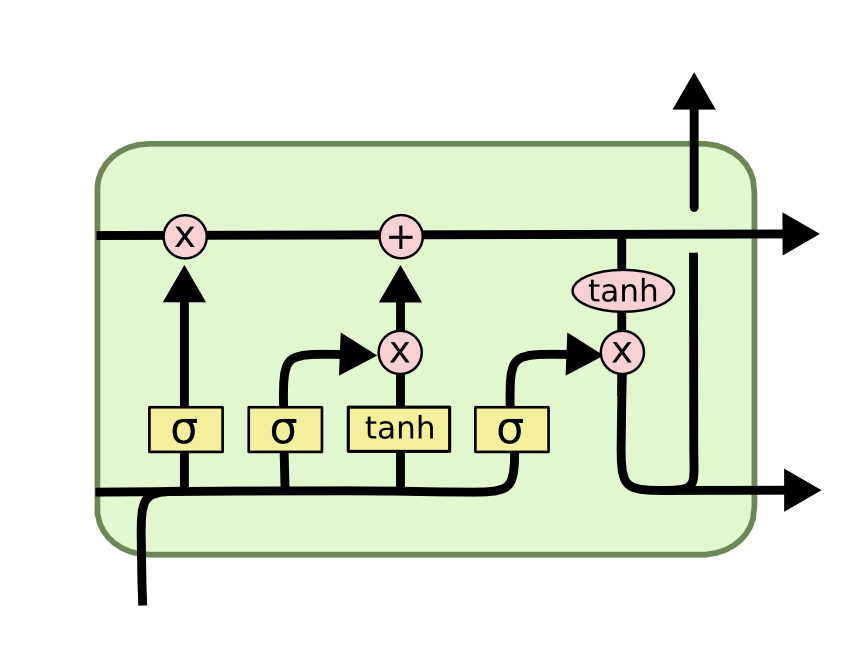
### 2.1: LSTM

Retelele de tip LSTM (Long short term memory) vin ca o evolutie a retelelor RNN (Recurrent neural networks), care conform [Staudemeyer et al., 2019.] nu pot depasi mai mult de 5-10 pasi de timp intrucat gradientii se pot micsora foarte tare (dispar) sau pot avea valori foarte mari (explodeaza). In cazul in care gradientii explodeaza, valoarea „w” oscileaza, iar in cazul in care gradientii dispar, antrenarea dureaza foarte mult timp sau chiar nu functioneaza.

Solutia acestei probleme este folosirea LSTM care poate avea conform [Staudemeyer et al., 2019.] si mai mult de 1000 de pasi de timp. Retelele LSTM exceleaza pe sarcinile care necesita ca un set de date secvential sa fie tinut minte pentru o perioada lunga de timp. Aceasta propietate este atribuita blocurilor de memorie, care pot accesa intrarea si produc o iesire, prevenind astfel ca informatiile irelevante sa intre sau sa iasa din acest bloc. De asemenea aceste blocuri mai au si o poarta de „uitare” astfel incat daca o caracteristica devine irelevanta, aceasta nu mai este luata in calcul [Staudemeyer et al., 2019.].

Aceste retele se pot suprapune pentru a forma o retea deep LSTM, astfel iesirea unei astfel de retele, reprezinta intrarea urmatoarei. Aceasta abordare a adus o imbunatatire dramatica fata de o retea cu un singur strat, in domeniul transcrierii mesajelor audio in text [Graves et al., 2013].

Pentru a accelera antrenarea putem folosi „Gradient clipping” care mentine gradientii intr-un interval stabilit ca si hiperparametru. De exemplu daca am setat valoarea de 0.25 ca hiperparametru si gradientul nostru are valoarea de 0.7, acesta va fi setat la valoarea de 0.25. Conform [Zhang et al., 2019] folosind aceasta tehnica, au testat pe o sarcina de prelucrarea limbajului natural, folosind modelul AWD-LSTM, au putut observa o crestere a vitezei cu care modelul este antrenat.



Imagine descriptiva a unui bloc LSTM [colah.github.io, 2023]

### 2.2 YOLO

YOLO (You only look once) este o retea capabila sa detecteze locatiile obiectelor in timp real din imagini. In aceasta lucrare vom folosi YOLOv7 care imbunatateste acuratetea si viteza. Pentru antreneare acesta foloseste setul de date MS COCO, antrenarea facandu-se de la 0, nefolosindu-se alte seturi de date sau retele preantrenate. Comparativ cu alte metode, de exemplu YOLOv4, acest model are cu 75% mai putini parametrii si aduce o imbunatatire a preciziei medii de 1.5%. Comparativ cu modelul state-of-the-art YOLOR-CSP, autorii ne prezinta ca modelul are cu YOLOv7 are cu 43% mai putini parametrii si o precizie medie cu 0.4% mai mare [Wang et al., 2022].

### 2.3 YOLO pose

Sarcina acestui model este de a detecta coordonatele articulatiilor tuturor persoanelor dintr-p imagine. Pentru antrenarea acestui model, autorii au folosit setul de date COCO Keypoint Detection, unde au augumentat datele folosind redimensionare, translatare, inversare, augumentari mozaic si diferite auggumentari ale culorilor. Pe setul de date COCO val2017 precizia medie este de 68.5% iar cel mai bun model este HigherHRNet cu 70.5%. Conform autorilor la rezolutii mai mici, YOLOv5s6-pose are o performanta mult mai buna decat alte modele de complexitate mica, de exemplu EfficientHRNet, pe setul de date COCO val2017 [Maji et al., 2022].

# Abordari recente

O abordare propusa in estimarea actiunii este folosindu-ne de caracteristici calculate de mana precum in lucrarea [Wang et al., 2013] care se foloseste de caracteristici calculate de mana pentru a estima miscarea. Pentru aceasta, autorii calculeaza homografia folosind algoritmul RANSAC pentru a stabiliza miscarea camerei, intre 2 cadre. Dupa aceea se folosesc de un detector uman pentru a detecta oamenii din imagine, folosindu-se de acele detectii pentru a imbunatati estimarea optical flow-ului eliminand oamenii care pot avea o alta miscare fata de cea a camerei, astfel stricand estimarea miscarii camerei. In urmatorul pas sunt calculati descriptorii HOF si MBH optical flow, iar descriptorul HOG sunt aplicati pe imaginea originala. Pentru clasificare autorii se folosesc de un SVM liniar pentru a face predictiile, datele fiind codificate intr-un vector Fisher.

La momentul aparitiei aceasta abordare s-a dovedit a fi cea mai buna pe seturile de dateHMDB51 (57.2% acuratete), UCF50 (91.2% acuratete).

O alta abordare este cea [Varol et al., 2017] care se foloseste de retele convolutionale adanci pentru a estima actiunea. Ca si date de intrare autorii experimenteaza cu diferite tehnici, de exemplu de a oferi imaginile ca RGB sau ca un optical flow. Rezultatele cele mai bune provin din folosirea metodei Brox de a obtine optical flow-ul, aceasta fiind furnizata retelei. Alte experimente au fost efectuate in gasirea numarului optim de cadre care sa fie furnizat retelei.

Cand a aparut rezultatele pe seturile de date UCF101 (92.7% acuratete) si pe HMDB1 (67.2% acuratete) au fost cele mai bune.

O alta abordare este facuta in [Luvizon et al., 2018] care pentru o imagine se foloseste de Inception-V4 pentru a extrage caracteristicile vizuale precum si estimarea locatiilor articulatiilor. Aceste 2 rezultate sunt agregate pentru a furniza predictia finala. Autorii folosesc aceste 2 informatii deoarece unele actiuni pot fi dificil de interpretat folosind doar una dintre metode. Ei dau ca exemplu actiunea de a bea apa si actiunea de a da un telefon care poate sa fie dificil de interpretat doar din locatiile articulatiilor. Ei antreneaza reteaua folosind 16 cadre continue, obtinand cele mai bune rezultate pe setul de date Penn Action for 2D action recognition (98.6% acuratete).

Autorii setului de date ales pentru aceasta lucrare [Gourgari et al., 2013] denumit THETIS, au obtinut pe filmarile care ofera informatii despre profunzime o acuratete de 60.23% folosind 12 retele SVM (pentru fiecare clasa), folosind caracteristici oferite de HOG (histogram of oriented gradients) si HOF (histogram of optical flow).

Pe setul de date ales pentru aceasta lucrare (THETIS) de tenis o abordare prezentata in [Vinyes et al., 2017] care se foloseste de setul de date RGB presupune in a extrage caracteristicile dintr-o imagine folosind reteaua Inception preantrenata pe ImageNet pentru a extrage caracteristicile fiecarei imagini. Aceste caracteristici sunt transmise unei retele LSTM care este pe 3 straturi, fiecare strat avand 90 de celule ascunse, lungimea unei secvente folosite fiind de 100 de cadre. Acuratetea obtinuta folosind aceasta tehnica este de 47.22%.

O alta abordare efectuata pe acest set de date este prezentata de [Vainstein et al., 2014] in care autorii se folosesc de silueta jucatorului dintr-un anumit cadru si imaginea de unde provine aceasta pentru a segmenta jucatorul, si a putea decupa din aceasta doar jucatorul. Aceste imagini decupate sunt grupate 2 cate 2 si formeaza optical flow-ul. Acest optical flow este apoi clasificat folosind CRF (conditional random fields) obtinand o acuratete de 86.44% conform autorilor.

# Tehnologii folosite

### 4.1 Python

Pentru realizarea aplicatiei am folosit limbajul Python care ofera o flexibilitate mare in privinta alegerii unei biblioteci de deep learning, oferind cele mai populare 2 optiuni: TensorFlow si Pytorch. De asemenea ofera biblioteca Open-CV care este de mare ajutor pentru extrage cadrele din videoclipuri.

### 4.2PyTorch

Pytorch este o biblioteca de deep learming care se ocupa de operatii cu tensori si cu o diferentiere automata, utilizandu-se de accelerarea placii grafice. Principiile de proiectare sunt de a fi cat mai usor de utilizat pentru un dezvoltator familiarizat cu limbajul Python, sa puna oamenii din cercetare pe primul loc, oferind un mijloc usor de a dezvolta modele, optimizatoare si un mijloc de incarcare al datelor usor de folosit. Complexitatea operatiunilor din model trebuie sa fie ascunsa de dezvoltator, el trebuind sa foloseasca doar API-uri intuitive. De asemenea trebuie sa aiba o performanta pragrmatica, implementarea acceptand putina complexitate in plus pentru a oferi o experienta mai intutiva. Aceasta biblioteca este scrisa in Python cu cod optimizat de C++ [Paszke et al., 2018]. Aceasta biblioteca a fost folosita pentru a construi modelul de LSTM folosit pentru recunoasterea actiunii.

### 4.3 Matplotlib

Matplotlib este o biblioteca de Python care are rolul de a crea grafice statice, animate si interactive. In cadrul proiectului aceasta a fost folosita pentru a afisa metricele de performanta, precum acuratetea modelului, si functiile de pierdere.

### 4.4 OpenCV

Este o biblioteca open-source care are o functionalitate potrivita pentru sarcinile de vedere artificiala in timp real. Este scrisa in principal in C++ si este oferita pentru Python, Java si MATLAB/Octave. Incepand cu anul 2010 aceasta biblioteca ofera accelerare de pe placa grafica. In cadrul acestui proiect aceasta biblioteca a fost folosita pentru a citi videoclipurile cadru cu cadru pentru a putea fi prelucrate de modelul YOLO.

### 4.5 YOLOv7

Pentru a prelua informatiile despre coordonatele persoanelor, respectiv a coordonatelor articulatiilor m-am folosit de implementarea oficiala a [Wang et al., 2022] care se poate regasi la [github.com/yolov7, 2023]. Din implementarile oferite am folosit modelul preantrenat pe gasirea coordonatelor articulatiilor, folosindu-ma de functionaliteatea oferita de biblioteca. De exemplu o functie folosita a fost cea care imi translateaza coordonatele detectiei de la formatul in care se ofera coordonatele detectiei, aceasta fiind oferita in locatia x si y a punctului din mijloc, in lungimea si inaltimea detectiei in pixeli, in cea care reprezinta punctul din stanga sus si cel din dreapta jos.

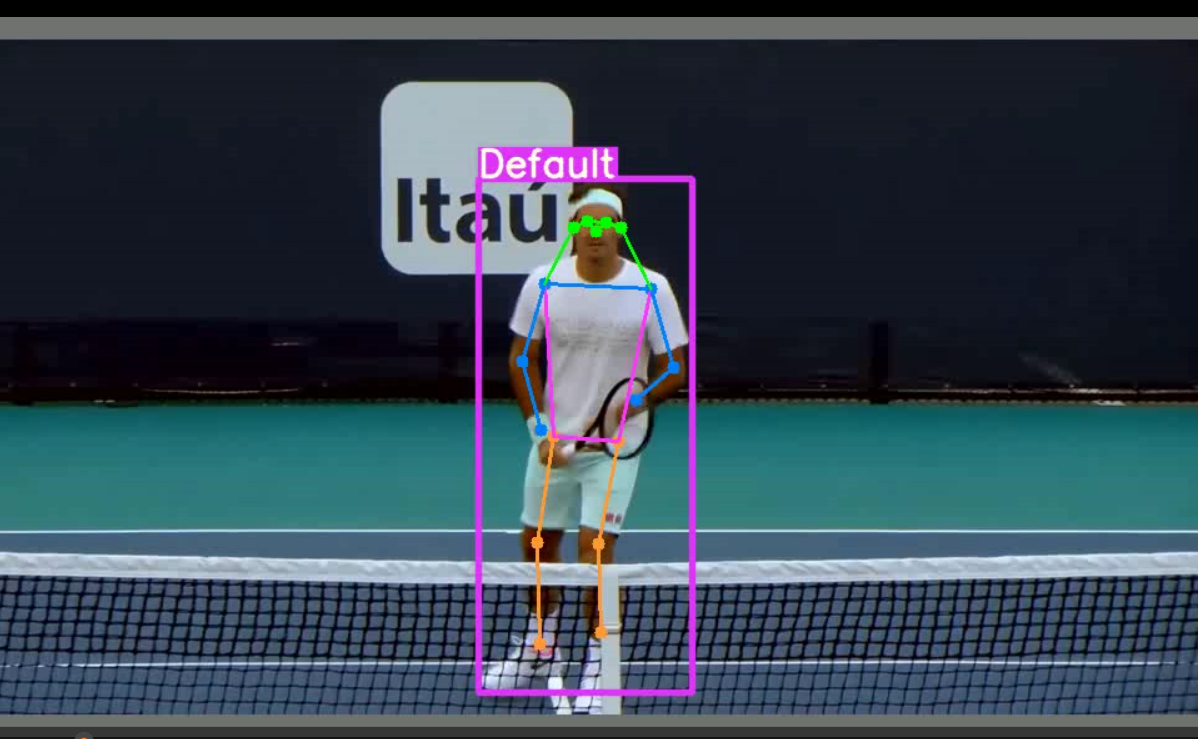


Figura 4.1

# Abordarea propusa

### 5.1 Descrierea setului de date

Setul de date folosit pentru acest proiect este THETIS [Gourgari et al., 2013] care este alcatuit din 8734 din videoclipuri, 1980 fiind in formatul RGB, executate de 31 de amatori si 24 de jucatori experimentati de tenis. Durata totala a videoclipurilor este de 7 ore si 15 minute. Fiecare persoana repeta cate o lovitura de 3-4 ori. Acest set de date contine mai multe tipuri de videoclipuri, fiind captate cu o camera care are senzor de profunzime. Pentru aceasta lucrare am folosit videoclipurile RGB, cu scopul ca mai apoi modelul sa fie rulat pe videoclipuri cu tenis captate cu o camera video obisnuita. Setul contine 12 lovituri din tenis care sunt adnotate. Loviturile sunt:

* Backhand with two hands
* Backhand
* Backhand Slice
* Backhand volley
* Forehand open stands
* Forehand slice
* Forehand volley
* Service flat
* Service kick
* Service slice
* Smash

Aceste 12 lovituri sunt executate in lipsa unei mingi de tenis.

Pentru dezvoltarea modelului din am ales la intamplare aproximativ 10% din videoclipurile din fiecare clasa rezultand in 192 de videoclipuri. Restul de 1788 au fost folosite pentru antrenament.



Figura 5.1

### 5.2 Configurarea modelului YOLO

Implementarea modelului YOLOv7 este cea oficiala a lucrarii [Wang et al., 2022] care a fost adaptata pentru a fi compatibila cu cerintele mele. Am definit o clasa intitulata „YoloModel” care se ocupa cu extragerea coordnatelor persoanelor si a articulatiilor. Pentru aceasta am definit functia „inference\_on\_image” care primeste ca parametru o imagine si are ca iesire coordonatele fiecarei persoane cat si pozitiile fiecarui membru. Astfel iesirea acestei functii are dimensiunea numar\_persoane x 58. Aceasta functie este apelata in functia principala „read\_from\_video” care primeste ca parametru calea catre un videoclip. Aceasta functie proceseaza videoclipul cadru cu cadru, luand in considerare persoana care are dimensiunile maxime. Aceasta limitare este facuta deoarece in setul de date sunt prezente si alte persoane, persoana care trebuie sa fie luata in calcul este doar cea de dimensiune cea mai mare. De asemenea in cadrul videoclipurilor reale, neavand informatii despre teren va fi luat in considerare doar persoana cea mai apropiata de camera. Pentru a transforma coordonatele detectiilor intr-un format mai usor de procesat am creat fisierul „utils\_detection.py” care contine mai multe functii ajutoatoare. Deoarece algoritmul ne furnizeaza rezultatul coordonatelor unde se afla persoana alipit de rezultatul coordonatelor membrelor, am creat functia „get\_detection\_box\_yolo” care primeste ca parametru detectia intreaga. Din aceasta extrag coordonatele aferente acestui tip de detectie si il transform din formatul {x, y, w, h} care reprezinta coordonatele centrului, lungimea si latimea dreptungiului, in formatul {x1, y1, x2, y2} care semnifica coordonatele coltului din stanga sus, respectiv dreapta jos. Pentru formatarea coordonatelor membrelor, care vin intr-un vector care are dimensiunea 1, fiecare articulatie din cele 17 fiind de forma {x, y, incredere} x, y reprezentand punctul unde se afla membrul, iar „incredere” reprezentand increderea modelului ca membrul respectiv sa fie in acea locatie. Pentru fiecare detectie de persoana am ales sa schimb dimensiunea acestui vector astfel acesta sa aiba dimensiunea 17x2, 17 reprezentand numarul de articulatii detectate, iar 2 fiind punctele respective. Pentru fiecare articulatie care are valoarea „incredere” < 0.5 am decis sa fac coordonatele nule. Folosind aceasta formatare pentru a fi mai usor de a normaliza datele.

### 5.3 Transformarea setului de date

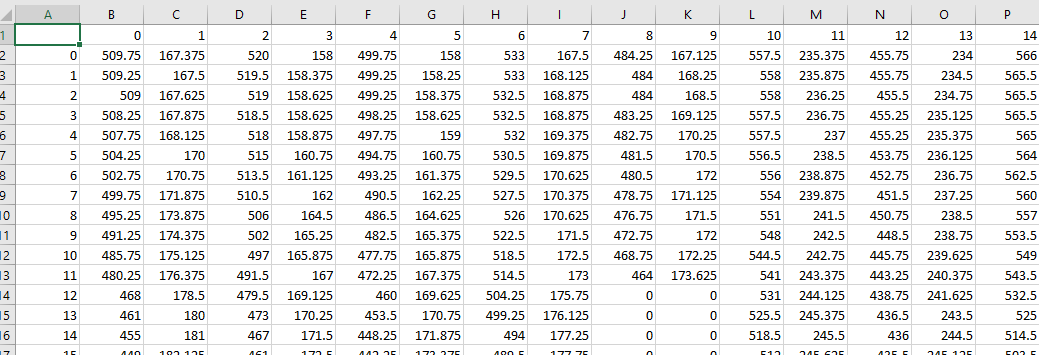
Pentru a transforma setul de date din videoclipuri in secvente de numere care pot fi procesate am creat fisierul „create\_dataset.py” care pentru fiecare videoclip din setul de date aplica modelul YOLO si salveaza coordonatele articulatiilor si al dreptunghiului unde se afla jucatorul in cate 2 fisiere csv pentru fiecare videoclip.

Cele doua fisiere sunt denumite dupa formatul:

* „**nume\_original\_limbs.csv**” pentru coordonatele articulatiilor
* „**nume\_original\_yolo.csv**” pentru salvarea coordonatele dreptunghiurilor

Astfel fiecare fisier, numarul de linii va fi egal cu numarul de cadre al clipului. Coordonatele articulatiilor vor fi in ordinea formatului COCO, grupate 2 cate doua in functie de ce articulatie reprezinta. Pentru formatul YOLO se salveaza cate 5 numere care reprezinta, coordonatele punctului din stanga sus al dreptunghiului, coordonatele dreapta jos, si increderea pe care o are modelul in predictia facuta. Fisierele sunt grupate in foldere in functie de ce clasa reprezinta, astfel o sa avem 12 foldere care contin fisierele csv dupa formatul enuntat mai sus. In aceste fisiere datele nu sunt normalizate. Crearea acestor fisiere csv este benefica deoarece evita rularea algoritmului YOLO de fiecare data cand dorim sa antrenam algoritmul de recunoastere a actiunii, algoritmul YOLO fiind o operatiune dificila din punct de vedere computational.

Crearea acestui set de date a durat aproximativ 7 ore pentru toate cele 1980 de videoclipuri. In figura 5.2 este exemplificat un astfel de fisier.

Figura 5.2

### 5.4 Normalizarea datelor

O etapa importanta in incarcarea datelor este preprocesarea lor, intrucat dorim ca toate caracteristicile sa folosesasca o scala similara. Intrucat pentru antrenarea modelului vrem sa folosim doar coordonatele articulatiilor, dorim sa le aducem in intervalul [0, 1] pe fiecare. Pentru aceasta, le vom normaliza in functie de pozitia lor relativa din dreptunghiul de detectie, folosind normalizarea min-max. De exemplu daca avem dreptunghiul de detectie de coordonate (10, 10), (100, 100) si punctul care reprezinta una din articulatii ca fiind (55, 55), dorim ca acesta sa aiba coordonatele normalizate ca fiind (0.5, 0.5).

In fisierul „utils\_detection.py” am creat functia „normalize\_detection\_limbs” care se ocupa cu acest lucru:

def normalize\_detection\_limbs(yolo\_boxes, detection\_limbs):

    detection\_for\_limbs = detection\_limbs.copy()

    for idx in range(yolo\_boxes.shape[0]):

        x\_std\_normalization = yolo\_boxes[idx, 2] - yolo\_boxes[idx, 0]

        y\_std\_normalization = yolo\_boxes[idx, 3] - yolo\_boxes[idx, 1]

        detection\_for\_limbs[idx, :, 0] = (detection\_for\_limbs[idx, :, 0] - yolo\_boxes[idx, 0]) / x\_std\_normalization

        detection\_for\_limbs[idx, :, 1] = (detection\_for\_limbs[idx, :, 1] - yolo\_boxes[idx, 1]) / y\_std\_normalization

    for idx in range(detection\_for\_limbs.shape[0]):

        for jdx in range(detection\_for\_limbs.shape[1]):

            if detection\_for\_limbs[idx,jdx,0] < 0 or detection\_for\_limbs[idx,jdx,1] < 0:

                detection\_for\_limbs[idx,jdx,0] = 0

                detection\_for\_limbs[idx,jdx,1] = 0

    return detection\_for\_limbs

Aceasta primeste ca parametru dreptunghiurile de detectie si coordonatele articulatiilor. Cele doua sunt denumite intuitiv, ele fiind de tipul „numpy array” pentru a putea face operatiile intr-un timp cat mai scurt. Astfel pentru fiecare dreptunghi dintr-un videoclip, se calculeaza pentru fiecare dimensiune diferenta intre punctul maxim si punctul minim. Intrucat variabila „detection\_for\_limbs” are dimensiunile numar\_cadre x 17 x 2 ne este usor sa normalizam, putand cu usurinta sa scadem de pe coordonata x, respectiv y punctul de coordonate minim, ca apoi sa putem sa normalizam folosind diferenta dintre maxim si minim. Ultimele 2 parcurgeri sunt in cazul in care coordonatele normalizate sunt negative, caz in care le aducem la valoarea 0.

### 5.5 Incarcarea datelor si crearea secventelor

Pentru a mentine datele in memorie am creat clasa „DatasetLoader” care are ca parinte clasa „torch.utils.data.Dataset” care va facilita antrenarea modelului. Aceasta clasa se ocupa cu incarcarea tuturor datelor din toate fisierele csv si de crearea a secventelor de date. Pentru a crea secventele in fiserul „utils\_detection.py” am creat functia „get\_all\_sequences\_from\_2D\_format(matrix, sequence\_length, y\_value, step = 1)” care primeste ca parametru coordonatele membrelor, lungimea pe care sa o reprezinta o secventa si valoarea codificata a clasei care este reprezentata in videoclip.

Crearea secventelor poate sa fie reprezentata in modul urmator:

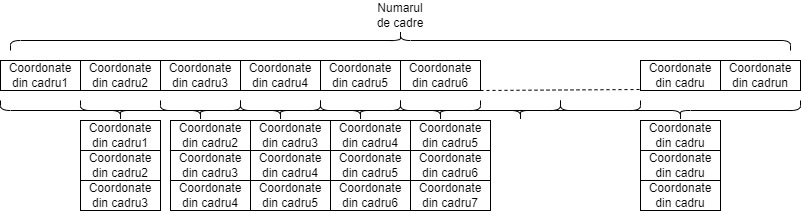


Figura 5.3

In acest exemplu lungimea unei secvente este de 3, iar pasul este 1. Acest lucru inseamna ca intre fiecare 2 rezultate consecutive, numarul de coordonate intercalate este de lungime\_secventa – 1, in exemplul de mai sus, numarul de cadre intercalate fiind de 2. Pentru reprezentarea datelor in cadrul proiectului, pentru fiecare videoclip am ales sa mentin pasul cu valoarea 1, pentru a avea un numar de date cat mai mare, stabilirea lungimii optime a unei secvente fiind un hiper-parametru. Toate secventele de cadre vor fi adunate intr-o singura variabila, ea avand dimensiunile numar\_secvente x lungime\_secventa x 34. De exemplu pentru lungimea unei secvente de 20, vom avea 95915 de secvente in total pentru antrenare. Intrucat parintele acestei clase este „torch.utils.data.Dataset” am supraincarcat metodele \_\_len\_\_(self) care returneaza numarul de secvente in total si \_\_getitem\_\_(self, index) care returneaza secventa indicata de index si valoarea codificata a actiunii care este reprezentata. La final, datele sunt transformate in tensori, care vor fi incarcati pe GPU, in cazul in care acesta exista, in caz contrar fiind incarcat pe CPU.

Pentru ca incarcarea in model sa fie cat mai facila aceasta clasa este intializata pentru setul de date de antrenare cat si pentru setul de date de testare. Apoi ne folosim de clasa DataLoader din Pytorch care are ca scop „impachetarea” datelor avand o lungime specificata ca hiper-parametrul „BATCH-SIZE”. Pentru antrenare datele sunt amestecate pentru a oferi o diversitate in cadrul unei actualizari a ponderilor modelului.

### 5.6 Crearea modelului

Pentru crearea modelului am creat o clasa care sa fie usor de personalizat, primind ca parametrii in constructor numarul de caracteristici al unei intrari, in cazul de fata 34, numarul de straturi al numarul de unitati al LSTM precum si lungimea secventei pe care o va primi ca input. Codul prin care este definita reteaua este:

class LSTM(nn.Module):

    def \_\_init\_\_(self, num\_features, hidden\_units, seq\_length):

        super(LSTM, self).\_\_init\_\_()

        self.num\_features = num\_features

        self.num\_classes = 12

        self.hidden\_units = hidden\_units

        self.seq\_length = seq\_length

        self.num\_layers = 3

        self.lstm = nn.LSTM(input\_size = self.num\_features,

                            hidden\_size = self.hidden\_units, num\_layers = self.num\_layers,

                            batch\_first = True,

                            dropout = 0.7)

        self.fc\_1  = nn.Linear(self.hidden\_units, 256)

        self.fc\_final = nn.Linear(256, self.num\_classes)

        self.dropout = nn.Dropout(0.7)

        self.relu = nn.ReLU()

    def forward(self, x):

        batch\_size = x.shape[0]

        h0 = torch.zeros(self.num\_layers, batch\_size, self.hidden\_units).to("cuda:0").requires\_grad\_()

        c0 = torch.zeros(self.num\_layers, batch\_size, self.hidden\_units).to("cuda:0").requires\_grad\_()

        outputs, (hn, \_) = self.lstm(x, (h0, c0))

        outputs = outputs[:, -1, :]

        outputs = self.dropout(outputs)

        out = self.relu(outputs)

        out = self.fc\_1(out)

        out = self.dropout(out)

        out = self.relu(out)

        out = self.fc\_final(out)

        return out

    def return\_train\_data():

        SEQUENCE\_LENGTH = 23

        INPUT\_SIZE = 34

        HIDDEN\_SIZE = 512

        return SEQUENCE\_LENGTH, INPUT\_SIZE, HIDDEN\_SIZE

Aceasta clasa mosteneste torch.nn.Module pentru a permite antrenarea si constructia modelului. In constructor se initializeaza parametrii modelului.

Aceasta este compusa din:

* **Reteaua LSTM**: input\_size reprezinta numarul de caracteristicial unei intrari (34), hidden\_size numarul de unitati de LSTM, num\_layers numarul de straturi LSTM
* **Reteaua MLP**: este alcatuita din 2 unitati, prima avand dimensiunea numarul\_unitati\_LSTM x 256, cea de-a doua avand 256 x 12 (12 fiind numarul de clase care trebuie prezis)

Aceste 2 componente au un probabilitate de „dropout” in cadrul antrenarii de 0.7. Conform [Srivastava et al., 2014] „dropout” presupune in dezactivarea anumitor neuroni pentru a reduce rata de suprainvatare a modelului, cea in care modelul se descurca foarte bine pe datele de antrenare dar esueaza in a face predictii corecte pe alte date. In aceasta lucrare, folosind aceasta tehnica autorii ne prezinta ca aplicand aceasta tehnica eroare pe toate seturile de date testate aceasta scade. De exemplu, pe setul de date CIFAR-100, cu rata de dropout aplicata pe toate componentele retelei eroarea a scazut de la 43.48% la 37.20% fara sa aduca alte imbunatatiri asupra setului de date sau sa augumenteze datele.

Functia „return\_train\_data()” care este statica are rolul de a facilita initializarea modelului in timpul inferentei, pentru fiecare model antrenat, am definit cate o functie separata.

Functia „forward(self, x)” este apelata pentru a oferi predictiile pentru fiecare intrare. Variabila „x” reprezinta datele care urmeaza a fi parcuse, avand forma batch\_size x lungime\_secventa\_34, astfel oferindu-se „batch\_size” predictii.

In aceasta functie se mai initializeaza si „starea ascunsa” (variabila h0) care retine informatiile importante din intrarile anterioare ale secventei. Aceasta poate „uita” sau „retine” informatiile. O alta variabila necesara care este initializata este „starea celulei” (variabila c0), care retine informatiile pe termen lung din intrarile secventei.

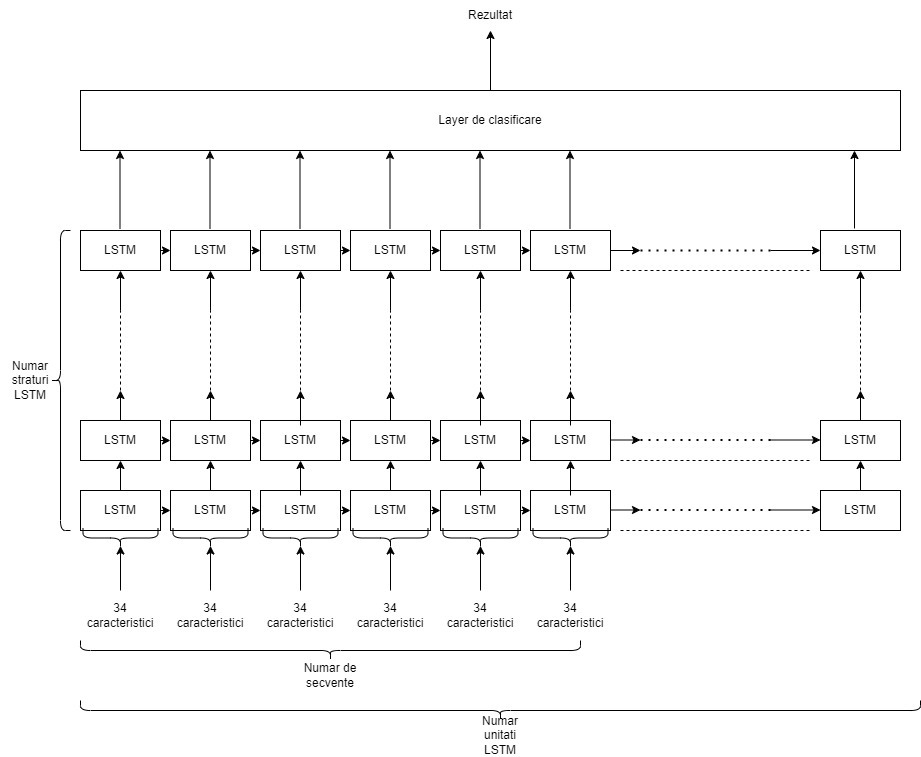


Figura 5.3

In figura 5.3 este prezentata structura modelului si ce reprezinta fiecare notatie din model.

### 5.7 Antrenarea modelului

Pentru antrenarea modelului ma folosesc de functionalitatea PyTorch pentru antrenarea sa. Pentru antrenare ma folosesc de optimizatorul Adam si functia de pierdere „CrossEntropyLoss” care are formula: . Din aceasta ecuatie simbolizeaza valoarea prezisa de modelul nostru, iar reprezinta valoarea reala a acelei clase.

Codul pentru antrenarea modelului:

for epoch in range(num\_epochs):

    time\_begin = time.perf\_counter()

    model.train()

    total\_loss = 0

    epochs.append(epoch)

    for X, y in train\_loader:

        output = model(X)

        loss = loss\_function(output, y)

        optimizer.zero\_grad()

        loss.backward()

        torch.nn.utils.clip\_grad\_norm\_(model.parameters(), 0.5)

        optimizer.step()

        total\_loss += loss.item()

    time\_end = time.perf\_counter()

    avg\_loss = total\_loss / num\_batches

    print(f"Train loss for epoch {epoch}: {avg\_loss}, duration {time\_end-time\_begin} seconds")

    train\_loss.append(avg\_loss)

    test\_loss, precision, precision2 = test\_model(test\_loader, model, loss\_function)

    if test\_loss < minimum\_testing\_error:

        minimum\_testing\_error = test\_loss

        minimum\_model = copy.deepcopy(model.state\_dict())

        minimum\_epoch = epoch

    torch.save(model.state\_dict(), f"intermediary\_results\\saved\_checkpoint\_{model.\_\_class\_\_.\_\_name\_\_}\_{epoch}\_epoch.pth")

    validation\_loss.append(test\_loss)

    precision\_validation.append(precision)

    precision2\_validation.append(precision2)

Modelul este antrenat pentru un numar de epoci stabilit ca si parametru. Pentru a incepe antrenamentul este necesar sa punem modelul in modul „train”, acest pas fiind facut la inceputul fiecarei epoci deoarece functia „test\_model(test\_loader, model, loss\_function)” trece modelul in modul de test. Rolul acestei functii este sa faca predictii pe setul de date destinat testarii, care nu este inclus in antrenare si de a returna eroarea pe acesta precum si precizia modelului. Variabilele „train\_loss”, „validation\_loss”, „precision\_validation” si „precision2\_validation” sunt de tip lista si au scopul de a retine rezultatele pe care modelul le obtine de-a lungul antrenarii. Variabilele „X”si „y” reprezinta date extrase din setul de date de antrenament, fiind extrase sub forma de batch. De exemplu variabila X o sa aiba dimensiunile batch\_size x lungime\_secventa x 34, iar pentru y o sa fie de forma batch\_size x 1. Apelul functiei „optimizer.zero\_grad()” are rolul de a reseta toti gradientii modelului. Daca nu am folosi aceasta functie comportamentul implicit al PyTorch este de a acumula gradientul curent cu cei de la pasii anteriori.

Apelul functiei „loss.backward()” face ca gradientii modelului sa fie calculati in functie de eroarea obtinuta pe setul de date. Dupa acest pas, pentru a a actualiza parametrii modelului trebuie apelata functia „optimizer.step()”, care trece prin fiecare parametru al modelului si il actualizeaza folosindu-se de valoarea gradientului calculata anterior. Pentru a limita gradientii care depasesc o anumita valoare, folosesc functia „torch.nn.utils.clip\_grad\_norm(model.parameters(), valoare)”. In acest exemplu am limitat parametrii la valoarea 0.5, asta simbolizand faptul ca in momentul in care aceasta valoare este depasita, gradientul folosit va avea valoarea 0.5.

In cadrul antrenarii, dupa fiecare epoca afisez statisticile aferente setului de date de antrenare precum si a celui de test (acest lucru facandu-se in functia „test\_model”). De asemenea la finalul fiecarei epoci se salveaza modelul in memoria calculatorului, precum si modelul care a avut cea mai mica eroare intr-o variabila.

# Experimente si rezultate

### Prezentarea metricilor de acuratete si primele rezultate

Pentru antrenarea modelului am folosit optimizatorul Adam avand o rata de invatare de 1e-4. Am realizat modelul de clasificare ca in capitolul 5.6, rezultatele optime fiind obtinute cu aceasta retea.

Pentru a evalua modelele am folosit doua metrici de acuratete:

* **Metrica 1**: pe fiecare secventa de date este facuta predictia, acuratetea finala fiind reprezentat de formula unde reprezinta numarul de secvente prezise corect, reprezentatnd numarul de secvente care au fost prezise incorect
* **Metrica 2**: pentru ca rezultatele sa poate fi comparate in mod direct cu [Vinyes et al., 2017], pentru fiecare videoclip din setul de testare, am facut predictiile necesare pe fiecare secventa, iar apoi predictia finala pentru acest videoclip a fost reprezentata de predictia majoritara. Aceasta predictie finala este comparata cu valoarea de adevar pe care o reprezinta videoclipul, iar apoi se face calculul prezentat in formula de la Metrica 1.

Experimentele au fost realizate folosind lungimele de secventa urmatoare: 16, 18, 20, 23. Pe parcursul antrenarii graficul functiei de pierdere a aratat asemanator pentru oricare din lungimile de secventa. Un exemplu este prezentat in figura 6.1

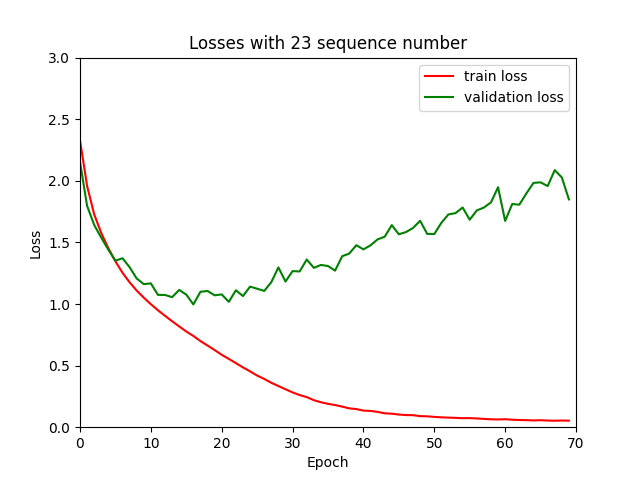


Figura 6.1

Pentru doua din aceste lungimi de secventa graficele de imbunatatire a acuratetii sunt:

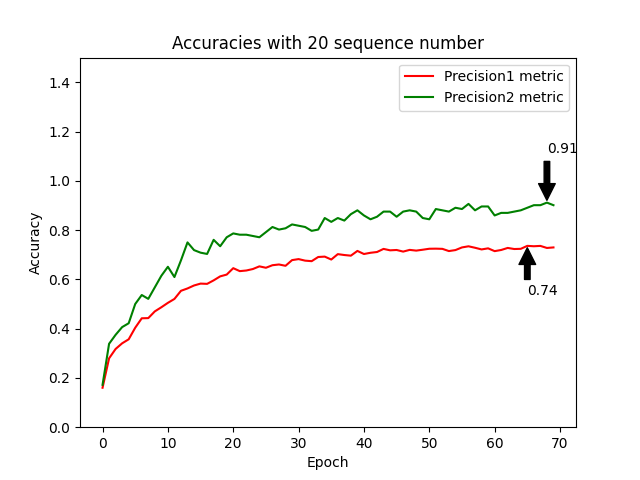


Figura 6.2

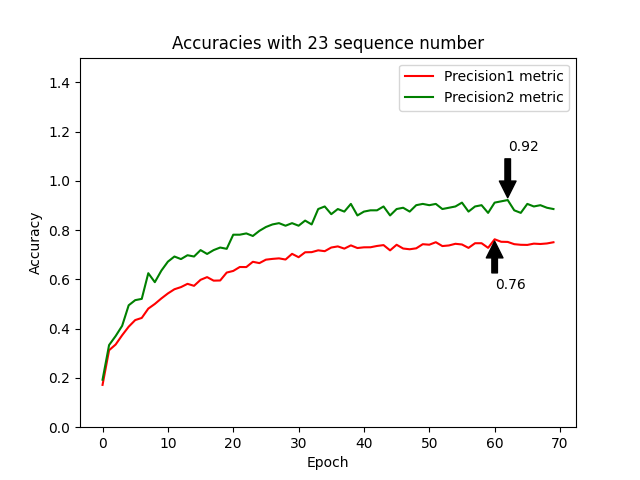


Figura 6.3

Putem observa din figura 6.1 ca functia de pierdere pentru setul de testare incepe sa creasca in jurul epocii 10, simbolizand aparitia fenimenului de overfit. Din figurile 6.2 si 6.3 putem observa ca acuratetea pentru setul de date de testare continua sa creasca asemenator pentru cele 2 metrice. Imbunatatirile asupra acuratetii par sa atinga un platou in jurul epocii 40, valorile maxime fiind atinse in jurul epocii 60. Mai putem observa ca perfirmanta metricii 1 este putin imbunatatita in momentul in care avem o lungime de secventa mai mare. Acest lucru devine mai aparent in momentul in care ne uitam la figura 6.4 care prezinta acuratetile folosind mai multe lungimi de secventa.

Modelul ales are ca si hiperparametrii:

* **LSTM**: 3 straturi ale retelei cu 512 hidden units cu o rata de dropout de 0.7
* **Layer de clasificare**: 2 straturi de MLP, primul avand dimensiunea de 512 x 256, iar cel de-al doilea de 256 x 12. Se aplica o rata de dropout de 0.7

|  |  |  |
| --- | --- | --- |
| Numar secvente | Acuratete maxima metrica 1 | Acuratete maxima metrica 2 |
| 23 | 0.76 | 0.92 |
| 20 | 0.74 | 0.91 |
| 18 | 0.73 | 0.92 |
| 16 | 0.72 | 0.93 |

Figura 6.4

In cadrul acestui proiect metrica cea mai importanta este metrica 1 intrucat pentru inferenta din cadrul unui meci real de tenis vom dori sa rulam predictia pe secvente de cadre si sa afisam miscarea efectuata. Putem observa din tabel ca metrica 1 creste progresiv in functie de lungimea secventei pe care modelul o primeste, valoarea maxima fiind de 0.76.

Comparativ cu [Vinyes et al., 2017], aceasta abordare pare sa aduca o imbunatatire a acuratetei, acuratetea prezentata de autor fiind de 0.47.

In lucrarea [Gourgari et al., 2013] autorii au executat experimente similare folosind metrica de acuratete 2, folosind videoclipuri cu date despre profunzime si coordonatele articulatiilor in trei dimensiuni comparativ cu metoda prezentata care se foloseste doar de videoclipuri RGB. Astfel ei au obtinut rezultate de 0.60, respectiv 0.54.

O alta abordare se poate gasi in [Vainstein et al., 2014] unde autorii s-au folosit de optical flow-ul oferit de siluetele jucatorilor, care apoi a fost clasificat folosind CRF. Acuratetea obtinuta de acestia a fost de 0.86.

Alte configuratii ale retelei nu au avut rezultate la fel de bune. De exemplu alte experimente au constat intr-o retea in care numarul de straturi LSTM este 1, rezultatul maxim al metricii 1 a fost de 0.70, metrica 2 avand acuratetea de 0.88. Alt experiment a fost sa micsorez numarul de hidden units a retelei LSTM la valoarea de 256, neschimband numarul de straturi, rezultatul maxim pentru metrica 1 fiind 0.70, iar pentru metrica 2 de 0.84. Pentru aceste 2 experimente numarul de secvente a fost pastrat constant si anume la valoarea de 23. Astfel pentru a interpreta rezultatele am pastrat modelul cu cea ma buna performanta si anume cel care are numarul de hidden units pentru reteaua LSTM de 512 si numarul de straturi al retelei LSTM a fost pastrat la 3.

### 6.2 Interpretarea rezultatelor



Figura 6.5

In figura 6.5 este prezentata matricea de confuzie a predictiilor pentru modelul standard care foloseste 23 de cadre pentru efectuarea predictiilor. Vizualizarea matricei de confuzie este importanta deoarece ne ofera o vizualizare a performantei algoritmului per clasa. Axa orizontala reprezinta valoarea prezisa de model, iar axa verticala reprezinta valoarea reala a clasei. Aceasta matrice corespunde metricii 1.

Putem observa ca rezultatele pot fi interpretate. De exemplu diferenta intre „kick service” si „slice service” este minora, ele diferind doar in directia data mingii. Clasa „smash” difera de toate celelalte tipuri de serve prin faptul ca nu este prima lovitura in cadrul unui joc. Modelul tinde sa prezica smash cand clasele reale erau unele de serviciu. Videoclipurile din setul de date THETIS nu contin mingea de tenis, acest lucru ar putea explica de ce aceste confuzii sunt facute.

Pentru miscarile care sunt mai distincte putem observa ca modelul are o acuratete buna, 0.94 pentru „forehand openstands”, 0.88 pentru clasa „backhand2hands” sau 0.87 pentru „backhand”.

# Descrierea aplicatiei

Scopul acestei aplicatii este de a oferi unui utilizator o interfata grafica intuitiva, pe care o poate folosi pentru a afla statisticile referitoare la loviturile pe care un jucator le-a facut in cadrul unui meci de tenis. Aplicatia are ca scop sa ruleze pe desktop.

Aplicatia este impartita in 2 parti:

* **Front-end**: interfata grafica a aplicatiei care a fost dezvoltata cu libraria tkinter din python
* **Back-end**: se ocupa cu prelucrarea videoclipurilor, gestiunea unei baze de date, si oferirea rezultatelor

### 7.1 Front-end

Interfata grafica a fost construita folosind biblioteca tkinter din python, fiind singura biblioteca de interfata grafica care vine preinstalata cu python. Functionalitatea principala consta in facilitarea selectarii unui videoclip de catre utilizator. Intentia utilizatorului de a selecta un videoclip este solicitata prin apasarea butonului „Select file” care va deschide o alta fereastra, in care acesta va selecta videoclipul pe care doreste sa-l analizeze. Interfata grafica va afisa doar fisierele care au formatul „avi” si „mp4”, formatul compatibil cu acest program. In momentul in care detectia persoanei este in progress, butonul „Select file” este dezactivat, astfel utilizatorul poate selecta un alt videoclip in momentul in care cel curent s-a terminat de analizat. De asemenea pe parcursul analizei, utilizatorul primeste informatii despre ceea ce este executat, starile acesteia fiind:

* „**The file is found at ‚location’**” – in cazul in care fisierul video a fost deja analizat, „location” reprezentand locatia fisierului video deja analizat.
* „**Finding the person in the video ‚location’**” – simbolizeaza faptul ca detectia YOLO este in curs de desfasurare, „location” reprezentand locatia fisierului video care este in curs de analiza.
* „**Predicting the action from ‚location’**” – simbolizeaza faptul ca predictia miscarilor executate este in curs de desfasurare, „location” reprezentand acelasi lucru ca mai sus
* „**Creating the zip file**” – salvarea fisierelor este in curs de executare.
* „**Finished, zip file at location ‚location’**” – programul a terminat de analizat videoclipul, locatia unde a fost salvat fisierul zip fiind indicata.

De asemenea finalul detectiei, programul va deschide arhiva „zip”, care contine fisierele oferite de acest program, folosind aplicatia prestabilita a sistemului de operare. In figura 7.1 este prezentata pagina principala a programului.

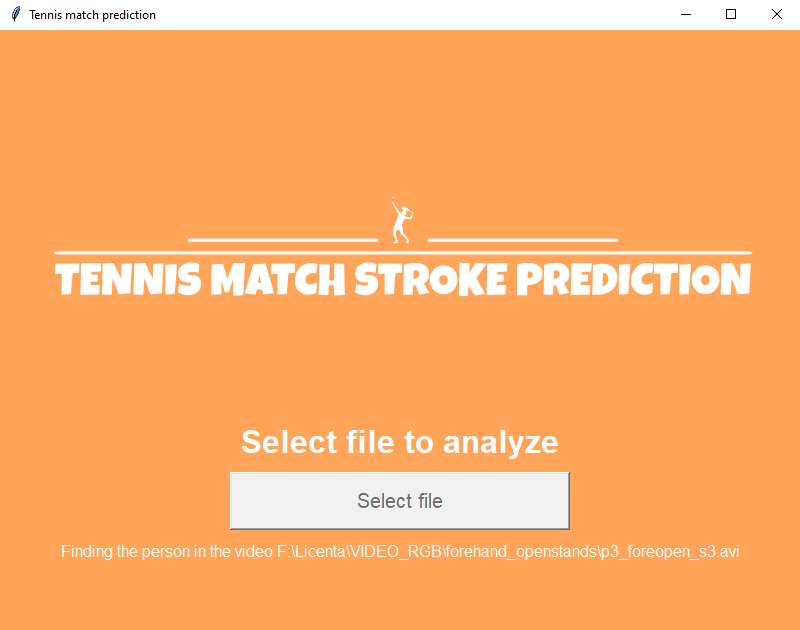


Figura 7.1

### 7.2 Back-end

Back-end-ul este alcatuit din urmatoarele componente:

* **Baza de date**
* **Semnatura digitala a unui fisier video**
* **Generarea videoclipului ce contine loviturile**
* **Generarea fisierelor de statistici**

### 7.2.1 Baza de date

In cadrul acestui proiect baza de date folosita este „Microsoft SQL Server” si are rolul de a retine fisierele video care au fost deja analizate de catre utilizator. Legatura dintre server si python a fost facuta utilizand biblioteca „pyodbc”. Baza de date contine o singura tabela, care are ca propietati:

* **HASH** (PK) – semnatura hash a fisierului, generat folosind algoritmul md5
* **VIDEOPATH** – contine calea catre fisierul zip generat

Pentru fiecare executie a algoritmului, programul adauga in aceasta tabela datele corespunzatoare. Valoarea HASH este calculata folosind continutul videoclipului, astfel fiind independenta de numele si locatia acestui fisier. In cazul in care utilizatorul introduce din nou pentru analiza un videoclip care deja a fost analizat, programul il va impiedica si i se va indica locatia rezultatelor deja efectuate.Acest pas are ca rol impiedicarea executiei de doua ori a aceluiasi fisier. Daca intre timp fisierul zip a fost sters si utilizatorul doreste sa reanalizeze acelasi videoclip, in baza de date se va reactualiza locatia si numele fisierului zip oferit de program. In figura 7.2 este un exemplu de cum arata datele in baza de date.

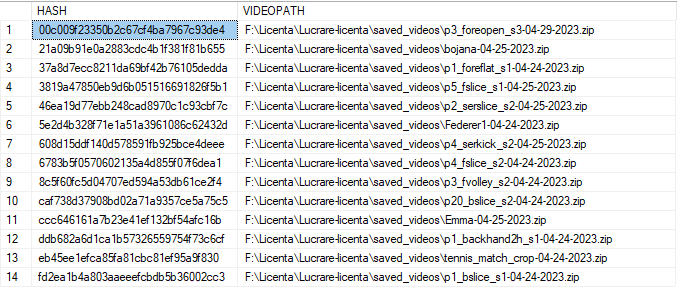


Figura 7.2

### 7.2.2 Semnatura digitala a unui fisier video

Pentru a genera semnatura digitala a unui fisier video, am calculat valoarea hash MD5. Pentru aceasta am folosit biblioteca hashlib. Codul care efectueaza aceasta operatie este:

def get\_hash\_of\_file(file\_path):

    hash\_md5 = hashlib.md5()

    with open(file\_path, "rb") as f:

        for chunk in iter(lambda: f.read(4096), b""):

            hash\_md5.update(chunk)

    return hash\_md5.hexdigest()

Acesta primeste ca parametru calea catre un fisier video, care este deschis folosind modul „rb” care permite citirea fisierelor binare. Fisierul este citit in blocuri de cate 4096 de octeti pentru a nu umple memoria calculatorului in cazul in care fisierul este de dimensiuni foarte mari. Valoarea hash este actualizata pentru fiecare bloc citit, iar apoi functia va returna valoarea hash-ului in format de sir de caractere in baza 16. Aceasta codificare este unica pentru fiecare videoclip, fiind bazata doar pe continutul acestuia.

### 7.2.3 Generarea videoclipului ce contine loviturile

Pentru generarea predictiilor fiecarui videoclip primul pas a fost ca pentru fiecare cadru din clip sa fie detectata persoana care are campul de detectie cel mai mare. Coordonatele membrelor oferite de algoritmul YOLO sunt apoi structurate in grupuri consecutive de cate 23 de cadre, care nu se intersecteaza, facandu-se pe fiecare din acesta predictia miscarii efectuate.

In pasul urmator este creat un videoclip pornind de la cel original, care pentru fiecare cadru va avea adnotat dreptunghiul de detectie si miscarea efectuata de jucator. In figurile 7.3 si 7.4 sunt prezentate cadre care au fost generate folosind aceasta aplicatie.

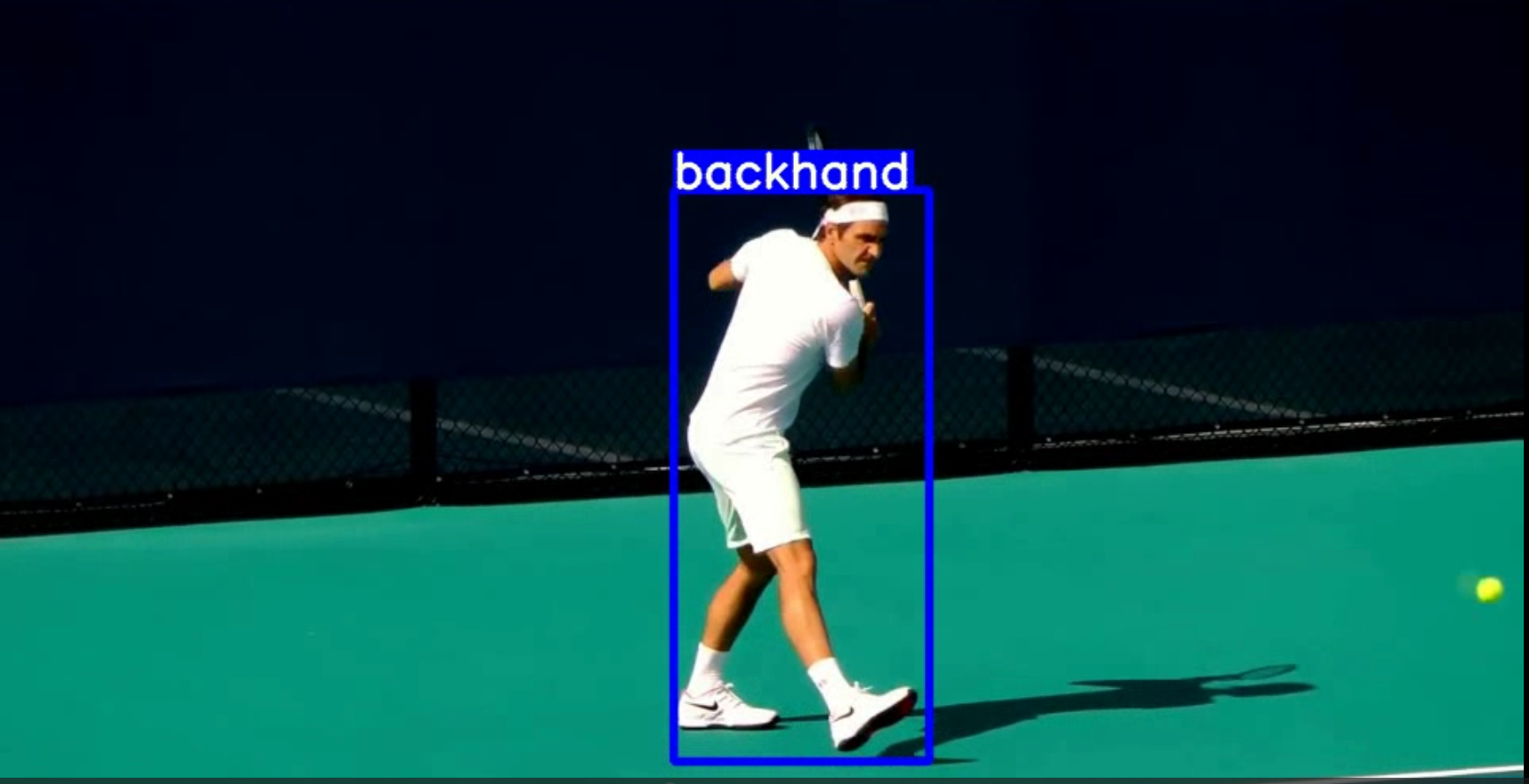


Figura 7.3

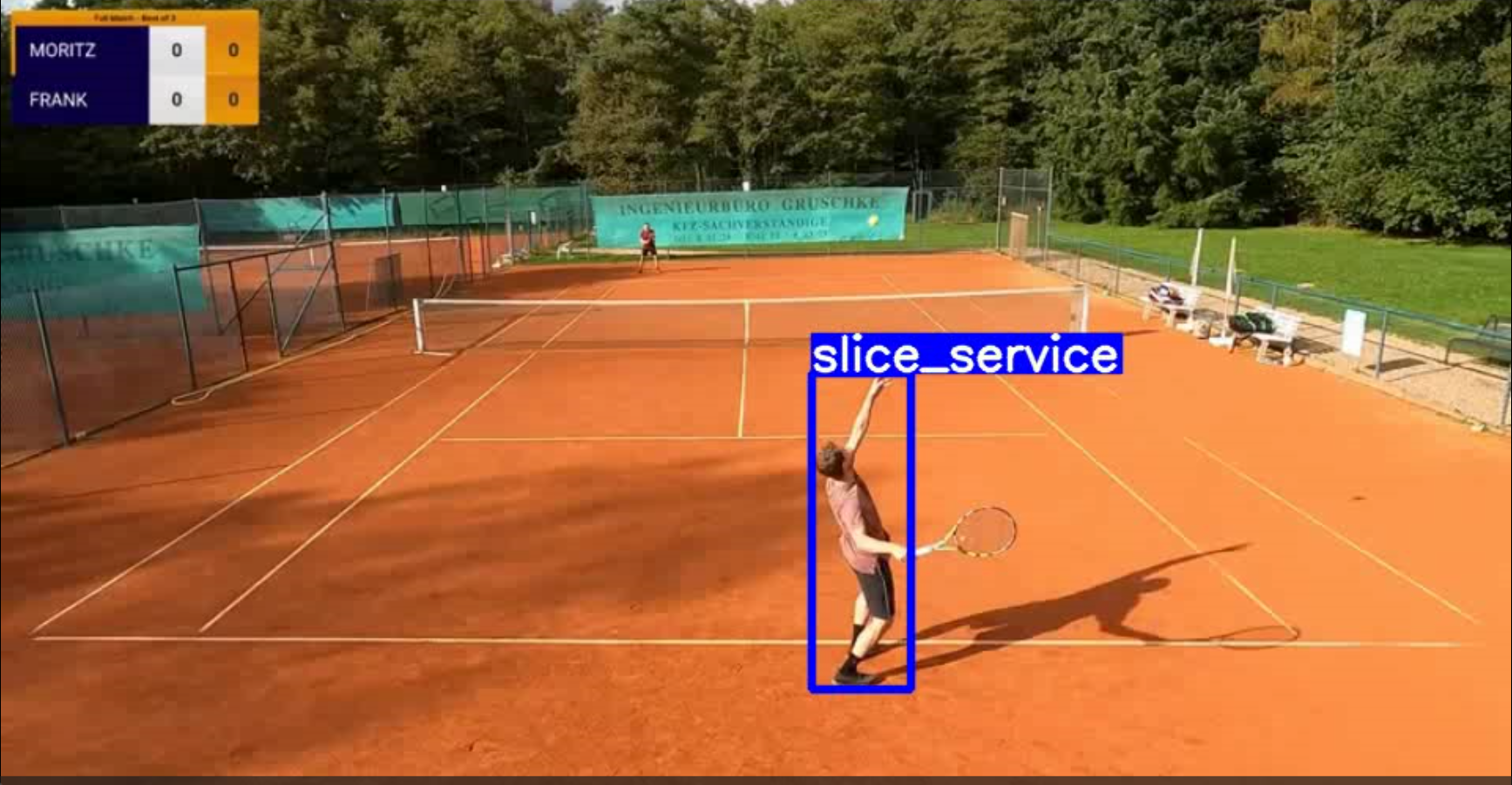


Figura 7.4

### 7.2.4 Generarea fisierelor de statistici

La inceputul programului, va fi verificat daca exista doua foldere denumite „saved-videos” si „.tmp”. Acestea vor fi folosite pentru scrierea rezultatelor. In folderul „.tmp” vor fi puse rezultatele inainte sa fie arhivate, dupa care vor fi sterse din acest folder. Arhiva cu rezultate va fi salvata in folderul „saved-videos”, formatul numelui dat va fi dat de numele clipului original caruia i se va adauga data la care detectia a fost facuta. Fisierul arhivat va contine 3 alte fisiere.

O image in formatul „.png” care va contine un grafic cu numarul de lovituri din fiecare clasa. Un fisier csv care va contine numarul de lovituri din fiecare clasa, oferit cu scopul in care utilizatorul doreste sa prelucreze aceste informatii intr-o maniera specifica. Ultimul fisier este videoclipul generat care are loviturile generate. In figura 7.5 este prezentat graficul generat pentru un fisier aleator.

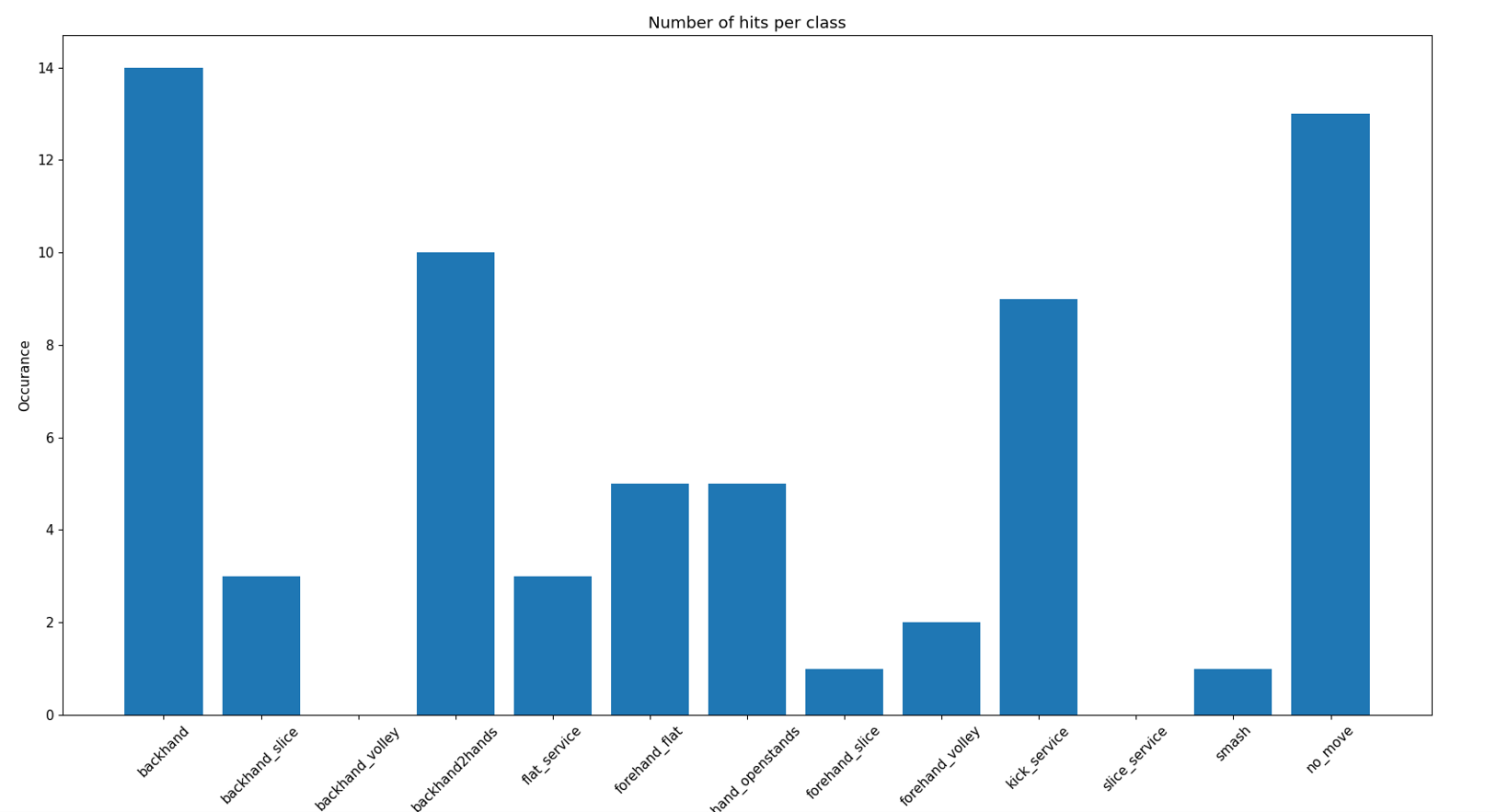


Figura 7.5

# Concluzii si dezvoltari ulterioare

# 9.Bibliografie

[Espn, 2023] URL: <https://www.espn.com/tennis/story/_/id/30877297/hawk-eye-live-gains-more-support-australian-open> (accesat la data 17.04.2023)

[Wang et al., 2022] Wang, Chien-Yao, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors." *arXiv preprint arXiv:2207.02696* (2022).

[Maji et al., 2022] Maji, Debapriya, Soyeb Nagori, Manu Mathew, and Deepak Poddar. "YOLO-Pose: Enhancing YOLO for Multi Person Pose Estimation Using Object Keypoint Similarity Loss." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2637-2646. 2022.

[Staudemeyer et al., 2019.] Staudemeyer, Ralf C., and Eric Rothstein Morris. "Understanding LSTM--a tutorial into long short-term memory recurrent neural networks." *arXiv preprint arXiv:1909.09586* (2019).

[Graves et al., 2013] Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks." In *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6645-6649. Ieee, 2013.

[Zhang et al., 2019] Zhang, Jingzhao, Tianxing He, Suvrit Sra, and Ali Jadbabaie. "Why gradient clipping accelerates training: A theoretical justification for adaptivity." *arXiv preprint arXiv:1905.11881* (2019).

[colah.github.io, 2023] URL: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (accesat la data 17.04.2023)

[Wang et al., 2013] Wang, Heng, and Cordelia Schmid. "Action recognition with improved trajectories." In *Proceedings of the IEEE international conference on computer vision*, pp. 3551-3558. 2013.

[Varol et al., 2017] Varol, G., Laptev, I. and Schmid, C., 2017. Long-term temporal convolutions for action recognition. *IEEE transactions on pattern analysis and machine intelligence*, *40*(6), pp.1510-1517.

[Luvizon et al., 2018] Luvizon, Diogo C., David Picard, and Hedi Tabia. "2d/3d pose estimation and action recognition using multitask deep learning." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5137-5146. 2018.

[Gourgari et al., 2013] Gourgari, Sofia, Georgios Goudelis, Konstantinos Karpouzis, and Stefanos Kollias. "Thetis: Three dimensional tennis shots a human action dataset." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 676-681. 2013.

[Vinyes et al., 2017] Vinyes Mora, Silvia, and William J. Knottenbelt. "Deep learning for domain-specific action recognition in tennis." In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 114-122. 2017.

[Vainstein et al., 2014] Vainstein, Jonathan, José F. Manera, Pablo Negri, Claudio Delrieux, and Ana Maguitman. "Modeling video activity with dynamic phrases and its application to action recognition in tennis videos." In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 19th Iberoamerican Congress, CIARP 2014, Puerto Vallarta, Mexico, November 2-5, 2014. Proceedings 19*, pp. 909-916. Springer International Publishing, 2014.

[Paszke et al., 2018] Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen et al. "Pytorch: An imperative style, high-performance deep learning library." *Advances in neural information processing systems* 32 (2019).

[wikipedia.com, 2023] URL: <https://en.wikipedia.org/wiki/OpenCV> (accesat la data 18.04.2023)

[github.com/yolov7, 2023] URL: <https://github.com/WongKinYiu/yolov7> (accesat la data 18.04.2023)

[Srivastava et al., 2014] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." *The journal of machine learning research* 15, no. 1 (2014): 1929-1958.