



**SZÉCHENYI
EGYETEM**
UNIVERSITY OF GYŐR



Széchenyi István Egyetem
Gépészmérnöki, Informatikai és Villamosmérnöki Kar

TMDK Dolgozat

**KLASK társasjáték menetének naplózása mesterséges intelligen-
cia használatával**

Kopcsándi Barnabás

Mérnökinformatikus BSc, VII. félév

Bordás Gréta

Gazdaságinformatikus BSc, III. félév

Konzulens: Dr. Hollósi János, egyetemi adjunktus

Széchenyi István Egyetem

2024/25. tanév I. félév

Győr

Tartalomjegyzék

1. Absztrakt	4
2. Bevezetés	5
3. Alapfogalmak	6
3.1. Mesterséges intelligencia	6
3.2. Gépi látás	6
3.3. Gépi tanulás	7
3.4. A gépi tanulás alapfogalmai	7
3.5. Mesterséges neurális hálózatok	7
4. KLASK	9
4.1. Mi az a KLASK?	9
4.2. A játék elemei	10
4.3. Szabályok	11
5. Felhasznált hardverek	12
5.1. StereoLabs ZED 2 sztereókamera	12
5.2. Feldolgozó egység	14
5.3. Kameraállvány	15
6. Felhasznált szoftverek	16
6.1. PyCharm	16
6.2. Fotók a videóból	17
6.3. Roboflow	18
6.4. YOLO	18
6.5. Google Colab	19
7. Kódok összehasonlítása	20
7.1. Objektum pozíciók kezelése	20
7.2. Naplófájl feltöltése	22
7.3. Pozíciók jelzése	23
7.4. Felhasznált Python könyvtárak	24
8. Annotáció	25
9. Tanítási folyamat	27
9.1. Előkészítés	27
9.2. A tanítás	28
10. Eredmények	31

10.1. Az első teszt eredménye	31
10.2. A második teszt eredménye	37
11. Összegzés és konklúzió	39
12. Irodalomjegyzék.....	40
13. Ábrajegyzék	43
14. Forráskód jegyzék	45

1. Absztrakt

A dolgozatunk egy több félévet felölelő projekt második lépését tartalmazza, amely a Neumann János Informatikai Szakkollégium Mesterséges Intelligencia munkacsoportja által kitűzött feladat része. Hosszútávú tervünk a KLASK nevezetű társasjáték automatizálása, egy mesterséges intelligencia alapú program létrehozásával, mely képes megmérkőzni egy ember (vagy akár másik gép) ellen egy virtuális KLASK társasjátékban. Miután a mesterséges intelligencia sikeresen megtanulja a KLASK szabályait egy virtuális környezetben, a következő lépés egy robot építése, mely képes a valóságban is játszani, a program döntései alapján fizikailag is cselekedve. Miután helyesen működő játék detektáló szoftvert és ehhez készített játékot játszani képes hardvert hoztunk létre, célunk egy országos szintű egyetemi verseny szervezése, melyben különböző magyarországi egyetemek megoldásai képesek egymással a KLASK-ban versenyezni. Amennyiben ezen tervünk sikerül, akár nemzetközi verseny szervezése is előfordulhat. A témát alaposan áttanulmányoztuk és arra a megállapításra jutottunk, hogy ilyen jellegű projekt még nem került megvalósításra, a játék automatizálására eddig senki sem tett kísérletet. Társasjáték automatizálásra elenyésző számú megvalósítás történt, KLASK társasjáték esetében azonban még semmilyen hasonló publikáció nem létezik.

A projekt kezdeti fázisában a cél, hogy sikeres legyen a játékmenet nyomon követése és naplózása. Első lépésként tanuló társaink a fizikai KLASK játéktáblát videókamerával rögzítették, képfeldolgozási módszereken alapuló játékmenet detektálást végeztek. Most, a második fázisban a nyomon követést mesterséges intelligencia (gépi látás alapú módszerek) segítségével valósítjuk meg, aminek feladata a táblán lévő objektumok (játékos, labda, játéktér) felismerése, folyamatos nyomon követése.

Összefoglalva a dolgozat célja a játékmenet sikeres detektálása, mesterséges intelligencia fejlesztéssel kivitelezve – majd a kapott mérési eredményeinket összevetve a hallgató társaink 2023/24-es tanév tavaszi eredményeivel (képfeldolgozás alapú módszerek) dönteni a projekt folytatásának módszeréről.

2. Bevezetés

A mesterséges intelligencia, és azon belül a gépi látás, gépi tanulás, és neurális hálózatok ágai nagy népszerűségnek örvendnek napjainkban egyre nagyobb precizitásuk, hatékonyságuk és újra felhasználhatóságuk miatt. Manapság már az egyes megoldások implementálása is nagyon leegyszerűsödött – így egy kis túlzással már egy átlagos számítógép felhasználó kis kutakodás után maga is képes egyszerűbb megoldások létrehozására. Mára ezek felhasználhatósága nagyon magas, mivel egy-egy apró változtatással a kódban egy teljesen eltérő problémára tudjuk formálni a modellünket/megoldásunkat.

Jelen dolgozatunkban is ezen megoldások igénybevételével alkottunk meg egy szoftvert, amely egy társasjáték automatizálásának egy folyamatát hivatott véghez vinni. A Neumann János Informatikai Szakkollégium Mesterséges Intelligencia munkacsoportja egy olyan projektet tűzött ki maga elé, hogy a KLASK nevezetű társasjátékot felfogni/értelmezni tudó, a szabályoknak megfelelően cselekedni képes automatizált ágenszt hozzon létre.

A mi dolgozatunk tehát a teljes projekt második lépése: a 2023/24-es tanév tavaszi félévében két hallgatótársunk a KLASK játékot játszva, felvételtől az alakzatok felismerését (labda, játékosok) végezte el, csupán képfeldolgozási algoritmusok és megoldások segítségével (Python nyelven az OpenCV könyvtárat alkalmazva). Ennek következtében jött az ötlet, hogy fejlesszünk erre a problémára egy tanuló ágenszt, ami elé ugyanezt a feladatot adjuk – és a kapott eredményeket összevetve egymással eldöntjük, hogy melyik megoldással célszerű a jövőben a munkacsoportnak tovább dolgoznia. Dolgozatunk célja így nem az, hogy minden áron a mesterséges intelligencia alkalmazások felé vegyük az irányt, hanem hogy a kapott eredményeket kiértékeljük és összevessük a korábbi eredményekkel – elképzelhető, hogy a hagyományos képfeldolgozási módszerek pontosabb és jobb eredménnyel szolgálnak jelen projekt keretein belül, mint az élet minden területén felkapott mesterséges intelligencia megoldások.

A fejlesztés Python nyelven történt, a YOLO11 architektúra igénybevételével, azon belül is (GPU kapacitásunk végett) a nano (n) verzió használatával. Az architektúra objektum detektálási megoldását alkalmaztuk.

3. Alapfogalmak

3.1. Mesterséges intelligencia

A mesterséges intelligencia (mint fogalom és mint terület), nem rendelkezik egy jól definiált általános, minden kontextusban jól elhelyezhető megfogalmazással, fogalommal. Számos területe és felhasználási módja van, melyre kivétel nélkül külön-külön megfogalmazásokat lehet tenni. Ennek következtében összegyűjtöttünk pár megfogalmazást a fogalommal kapcsolatban, különböző szerzőket idézve:

„A mesterséges intelligencia (MI) a gépek emberhez hasonló képességeit jelenti, mint például az érvelés, a tanulás, a tervezés és a kreativitás. Lehetővé teszi a technika számára, hogy érzékelje környezetét, foglalkozzon azzal, amit észlel, problémákat oldjon meg, és konkrét cél elérése érdekében tervezze meg lépéseit. A számítógép nemcsak adatokat fogad (már előkészített vagy összegyűjtött adatokat érzékelőin, például kameráján keresztül), hanem fel is dolgozza azokat és reagál rájuk.” – *Az Európai Parlament megfogalmazása szerint* [1]

„A mesterséges intelligencia az intelligens viselkedés megértésével és számítógépes rendszerekben való megvalósításával foglalkozó tudományág.” – *S. Russell és P. Norvig megfogalmazása szerint* [2]

„Olyan szoftverek, amelyeket olyan feladatok elvégzésére vagy kimenetek előállítására használnak, amelyekről korábban úgy gondolták, hogy emberi intelligenciát igényelnek, különösen a gépi tanulás alkalmazásával, nagy adatgyűjteményekből történő extrapolációval.” – *Az Oxford angol szótár megfogalmazása szerint* [3]

3.2. Gépi látás

„A gépi látás (Computer Vision) egy olyan interdiszciplináris tudományterület, mely a számítógépek számára teszi lehetővé álló- vagy mozgóképek olyan magasszintű értelmezését, „látását”, mellyel eddig csak emberek rendelkeztek. Fontos hangsúlyozni a magas szintet, mert a használt technológiáktól a háromdimenziós világunk egyfajta megismerését, feltárását várjuk el, úgy, hogy a rendszer bemenete gyakran csak kétdimenziós képek, illetve képek sorozata.” – *A mesterséges intelligencia felhasználási lehetőségei az objektumfelismerésben – Kollár Csaba, Nagy Barna (125. oldal, 3. bekezdés)* [4]

3.3. Gépi tanulás

A gépi tanulás a mesterséges intelligenciának azon ága, amely azzal foglalkozik, hogy racionális ágenseket hozzon létre. Racionális ágensnek tekintünk egy olyan rendszert, amely az elé állított feladatokat elvárt szinten képes végrehajtani/elvégezni. A racionalitás így az elvárt teljesítményt maximalizálja. Olyan rendszerekről beszélünk, amelyek tanulni képesek, és a tapasztalataikból tudást tudnak generálni a későbbi (lehetőleg eredményesebb) végrehajtás érdekében.

3.4. A gépi tanulás alapfogalmai

Adathalmaz: adatsorozat, amelyet a tanuló ágens értelmez, és ennek egy bemeneti részhalma-zára egy adott kontextusban értelmezhető kimenetet ad.

Tanítás: Az a folyamat, amely során a tanuló ágens folyamatos bemeneti adattal látjuk el, és az ágens belső paramétereit hangoljuk.

Tesztelés/validáció: Az a folyamat, amely során az ágens teljesítményét valamilyen teljesítménymértékkel szemben ellenőrizzük.

Hiba/veszteség: Annak jelzése, hogy a tanuló ágens mennyire hoz helyes következtetéseket egy adott bemenetre - egy hasznosságfüggvény iterációkon keresztül mért értéke. Ennél a lépésnél kerülnek be az optimalizáló algoritmusok (Gradient Descent, Stochastic Gradient Descent, Adam, Adagrad, momentum alapú megoldások, stb.).

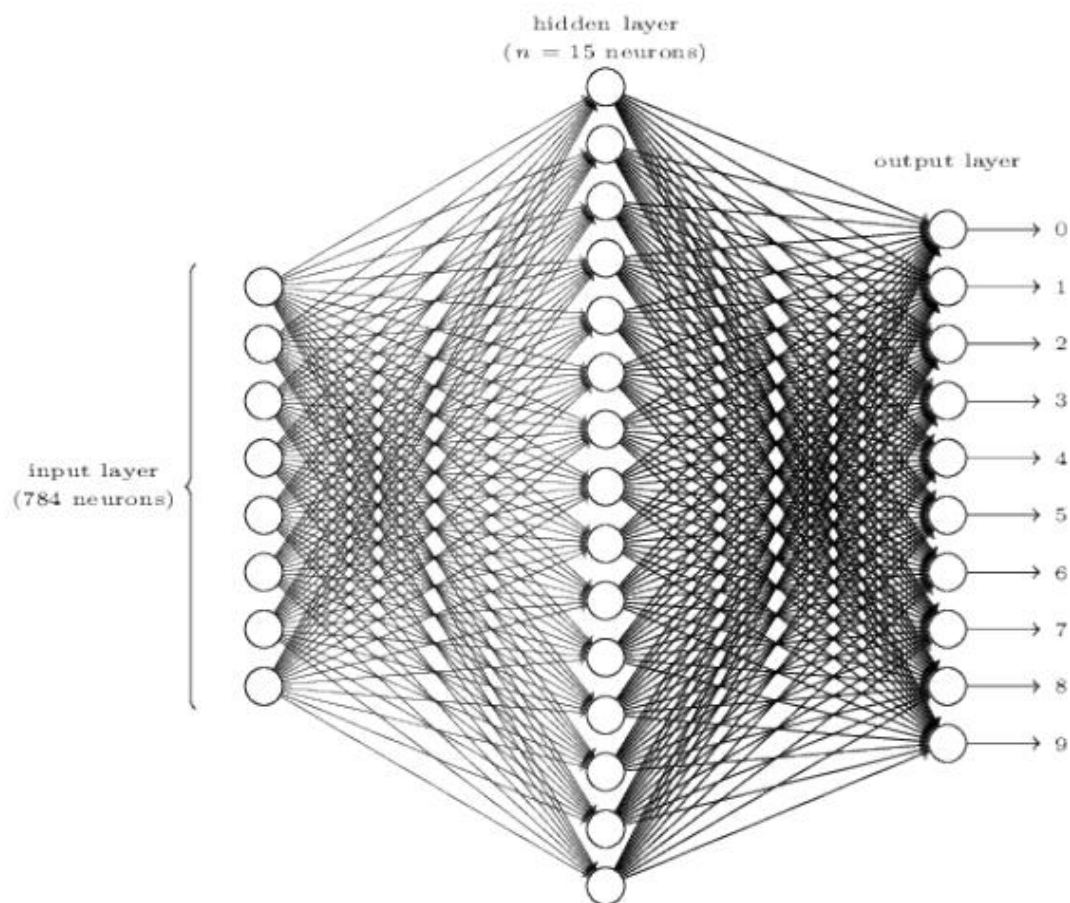
Hiperparaméter: A tanuló ágens elemeinek olyan paramétere, amely tanítás során nem változik – pl.: tanulási ráta, rétegek száma, stb.).

A tanítás során mi is ezekkel az adatokkal fogjuk beállítani/tanítani a modellünket, így a későbbi „9. Tanítási folyamat” részben részletezzük az általunk használt értékeket.

3.5. Mesterséges neurális hálózatok

„A mesterséges neurális hálók a biológiai neurális rendszerek elvére felépített számítógépes rendszerek (Elsberry, 1998). A hálók több, egymáshoz kapcsolódó és párhuzamosan dolgozó processzorból (neuronból) állnak, és ily módon próbálják utánozni a biológiai idegrendszer információfelvételének és -feldolgozásának módját. A mesterséges neurális hálók a

hagyományos algoritmikus eljárások helyett más módon, tanulással nyerik el azt a képességüket, hogy bizonyos feladatokat meg tudjanak oldani. A nagy számításigény és a nagyszámú megoldási lehetőség áttanulmányozása megkívánja a párhuzamos feldolgozást, hogy egyidejűleg több különböző lehetőséget lehessen megvizsgálni. A tanulás az információfeldolgozás módját határozza meg.” – *A mesterséges neurális hálók a jövő kutatás szolgálatában* – Kristóf Tamás (6. oldal, 2. bekezdés) [5]



1. ábra - Egy egyszerű mesterséges neurális hálózat [6]

4. KLASK

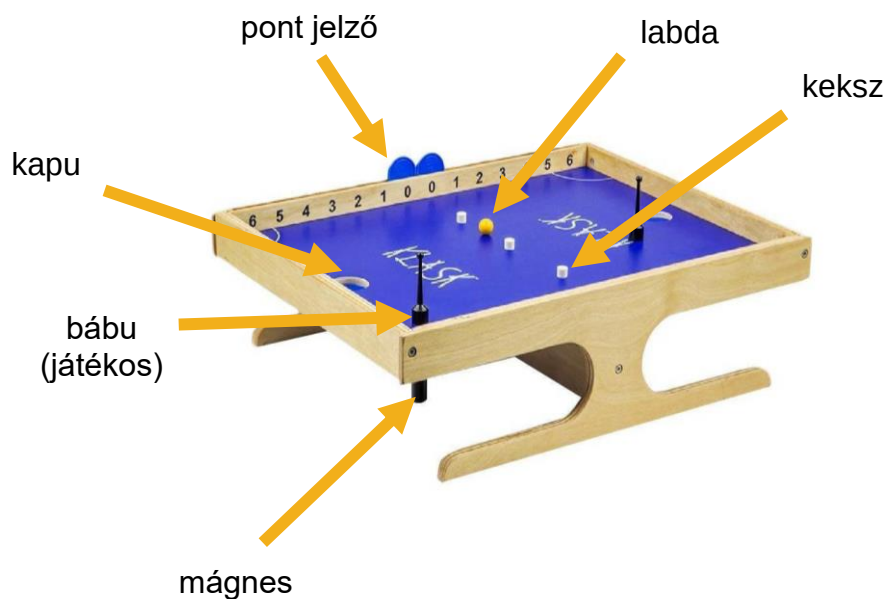
4.1. Mi az a KLASK?

A KLASK egy kétszemélyes ügyességi, stratégiai, interaktív társasjáték, mely a léghoki és a csocsó tulajdonságait ötvözi. Előbbiekhez képest egyedi vonása, hogy a játék alapja a mágnesesség. Feltalálása Mikkel Bertelsen, dán asztalos nevéhez fűződik, aki 2013-ban, műhelyében készítette el az első játéktáblát. Az klask szó jelentése a dán nyelvben „ütés”, ami a koppánás hangjára utal, melyet akkor hallunk, mikor a bábu beleesik a kapuként funkcionáló mélyedésbe. Napjainkban 16 különböző országban rendeznek KLASK bajnokságokat, amiket a Finnországban (Helsinki) záruló világbajnoksággal zárnak le. Viszont a hivatalos versenyek mellett számtalan háztartásban, közösségi helyszínen is megtalálható a közkedvelt, világhírű játék. [7]



3. ábra – KLASK társasjáték

4.2. A játék elemei



3. ábra – KLASK elemei



4. ábra – Labdák, kekszek, bábu és a mágnes

Pálya: A játéktérület egy $42 \times 31,7$ centiméteres, fából készült téglatest, körülötte szegéllyel, hogy a labda visszapattanhasson a pálya széléről. Ezzel együtt a tábla teljes kerülete $44,5 \times 33,7$ centiméter. Létezik 4 személyes verzió, ahol a pálya kör alakú, viszont mivel a dolgozat a 2 személyes változatra épül, a továbbiakban csak arról fogunk beszélni.

Bábu: A bábu reprezentálja a játékost a pályán. A felszínen található, úgynevezett antenna aljában mágnes található, és a pálya alatt található irányító (az ábrán mágnes) is mágneses. Így a játékos úgy mozgatja bábuját, hogy nem nyúl a játéktérbe, kizárólag a mágnest használja irányításra. A játékszabály tiltja a bábu mágnes nélküli, kézzel történő irányítását.

Labda: A játék célja, hogy a műanyaggyóty, azaz a labdát, az ellenfél kapujába juttassuk, ezzel pontot szerezve. Színe szándékosan élénk narancssárga, a sötétkék felületen feltűnő színű. A kontrasztos színek a képfeldolgozást is nagyban segítik.

Kapu: A tábla két végén 45 milliméter átmérőjű mélyedések.

Kekszt: A pálya közepén lévő 3 fehér körbe elhelyezendő, 3 darab mágneses elem neve.

Pontjelző: A korong eltolásával jelezzük, hogyha pontszerzés történt. Ha a korong valamely térfélen eléri a hatos jelzést, a játékos győzött. [8]

4.3. Szabályok

Előkészületek:

1. A játék elején a pontjelző korongokat nullára állítjuk.
2. Minden kör kezdetén a 3 kijelölt körbe helyezzük a kekszeket, a pálya közepén.
3. A csatárok bárholnan indíthatják a kört, nincs erre vonatkozó szabály.
4. Az első körben a fiatalabb játékos kezdi a kört, utána pedig mindig az a fél, aki sikeresen pontot szerzett.

Pontszerzés:

1. Gól, azaz a labda sikeres betalálása az ellenfél kapujába.
2. Az ellenfél minimum 2 kekszet szerez – az antennájára tapadnak.
3. Az ellenfél bábuja „szétesik”, nem képes tovább a mágnessel mozgatni antennáját, mivel a mágneses vonzás megszűnt a 2 részelem közt.
4. Az ellenfél beleesik a saját kapujába.

+1: Mindig csak egy pont szerezhető egyszerre, hiába teljesül több feltétel is a fentiekből.

Ha egy játékos pontot szerzett, a következő kört ő kezdi. Ilyenkor a saját térfelén bármely kiinduló pontba helyezheti a labdát a kettő közül. A kiinduló pontok a játéktábla 4 sarkán elhelyezkedő, fehér negyedkörök. A további szabályok közé tartozik, hogy amennyiben a labda véletlenül elhagyja a pályát, az elhagyás pontjához legközelebbi kiinduló pontba kell visszahelyezni a kör folytatásához.

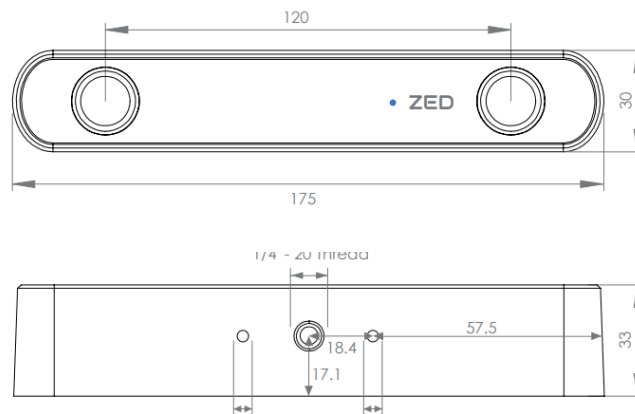
5. Felhasznált hardverek

5.1. StereoLabs ZED 2 sztereókamera

A ZED 2 egy fejlett, mesterséges intelligencia alapú rendszer, mely 2 kamerája felvételei által neurális mélységérzékelést és térbeli tájékozódást tesz lehetővé. Kifejezetten AI és gépi látás projektekhez készült, melyek a valós idejű térképezést, objektumkövetést, és képfeldolgozást igényelnek.

Fizikai adatok:

- méret: 175×30×33 mm
- súly: 166 gramm
- áramellátás: 380 mA / 5V, USB-ről táplálva



5. ábra – ZED 2 kamera dimenziói, mm-ben kifejezve

Felbontás:

A kamera különböző felbontási és képráfrítási lehetőségeket kínál. A felbontás növekedésével a képráfrítás sebessége (fps) csökken. Lehetőségünk van, 15, 30, 60, vagy akár 120 fps elérésére is. Esetünkben a 30 fps ideális, mellyel 1920×1080 HD felbontás megvalósítható. Egyensúlyt biztosít a képminőség és a sebesség között, optimális gyors mozgásérzékeléshez és valós idejű alkalmazásokhoz tehát megfelelő.

Mélységérzékelés:

Az egyes kamerák által rögzített képek közötti eltérés alapján a rendszer képes meghatározni az objektumok távolságát, neurális hálózatokat alkalmazva az emberi látás reprodukálására. A ZED 2 sztereókamera érzékelési tartománya 0,3-tól 20 méterig terjed. A hibahatár 3 méteren belül kevesebb, mint 1%, 15 méteren belül pedig 5% alatti.

Látószög:

- vízszintes síkban 110°-os látószög lefedésére képes;
- függőlegesen 70°-os látószög lefedésére
- átlós látószög, azaz a függőleges és vízszintes kombinációja 120°

Rendszerkövetelmények:

- processzor: minimum 2,4 GHz, legalább 2 db processzormag
- memória: minimum 4 GB RAM memória
- operációs rendszer: Windows 10 (64 bit) [9]

5.2. Feldolgozó egység

A projekt során használt feldolgozó egység egy HP VICTUS 16 laptop, mely számos paraméterével teszi lehetővé a számítási igényes feladatokat, mint a mélytanulás és gépi látás.

Processzor:

- megnevezés: Intel Core i5 13500H Raptor Lake (13. generáció)
- működési frekvencia: 2,60 GHz
- magok száma: 14, melyből 6 db teljesítményi, 8 db hatékonysági

Memória:

- 16 GB DDR5, 512 GB SSD

Kijelző:

- méret: 16,1"
- felbontás: 1920×1080

Videókártya: NVIDIA GeForce RTX 3050

A videokártya 2048 mag lehetővé teszi a GPU-k (grafikus feldolgozó egység) használatát általános számítási feladatokhoz, mint a gépi tanulás, tudományos számítások, képfeldolgozás és más párhuzamosan végrehajtható feladatok.

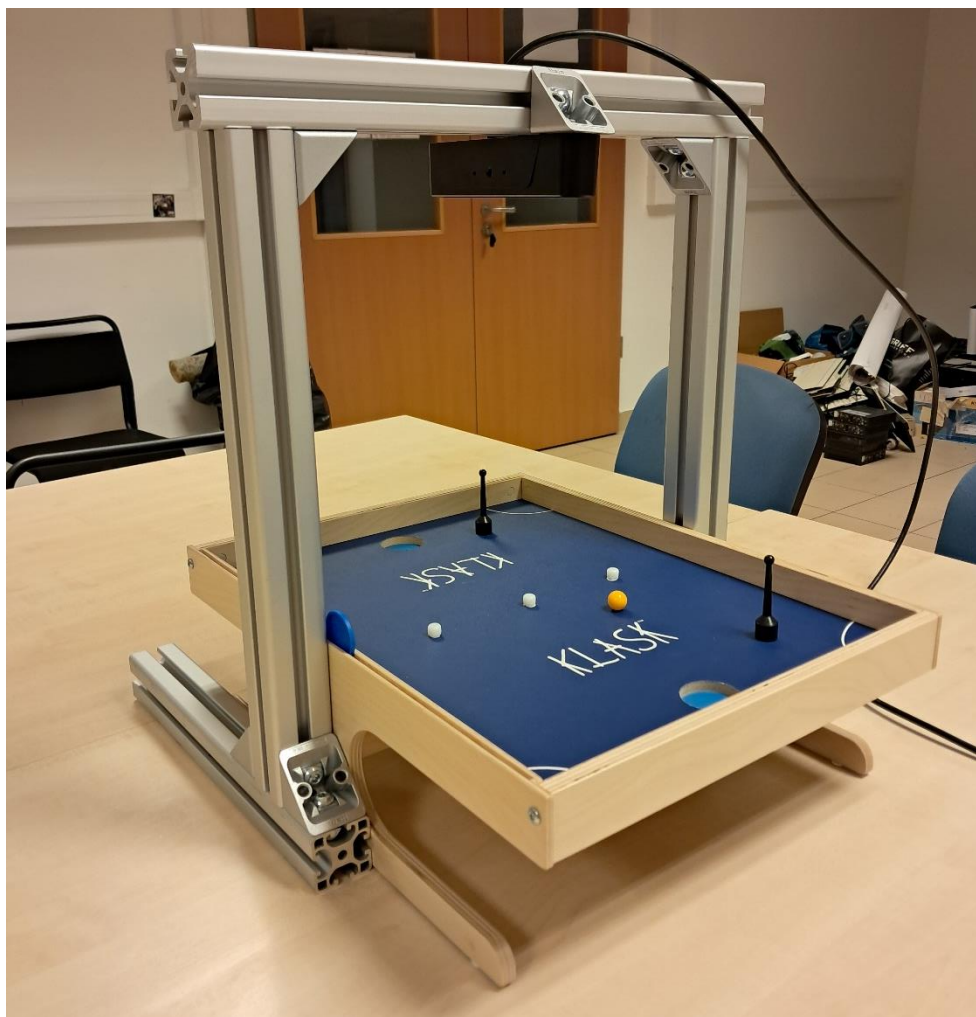
Operációs rendszer: Windows 11 Home (64 bit) [10]



6. ábra – HP VICTUS 16 Gaming Laptop

5.3. Kameraállvány

A kamerát tartó állványt, melyre a videófelvevételek elkészítésére szolgáló kamerát tudjuk helyezni, egy egyetemi oktató készítette kérésünkre, a szakkollégium részére. A szerkezet (ahogy az a 7. ábrán látható) alumínium elemekből épül fel, és kifejezetten a KLASK pálya szélességére és méretére lett szabva. A szerkezetet a pályára felülről tudjuk felhelyezni, mivel a rácsúsztatás (a pályát tartó csavarok enyhe kidudorodása végett) nem lehetséges. A szerkezet belső tűrése 1-2 mm tűréssel megegyezik a társasjáték pálya szélességével, ezért stabilan körbe fogja azt.



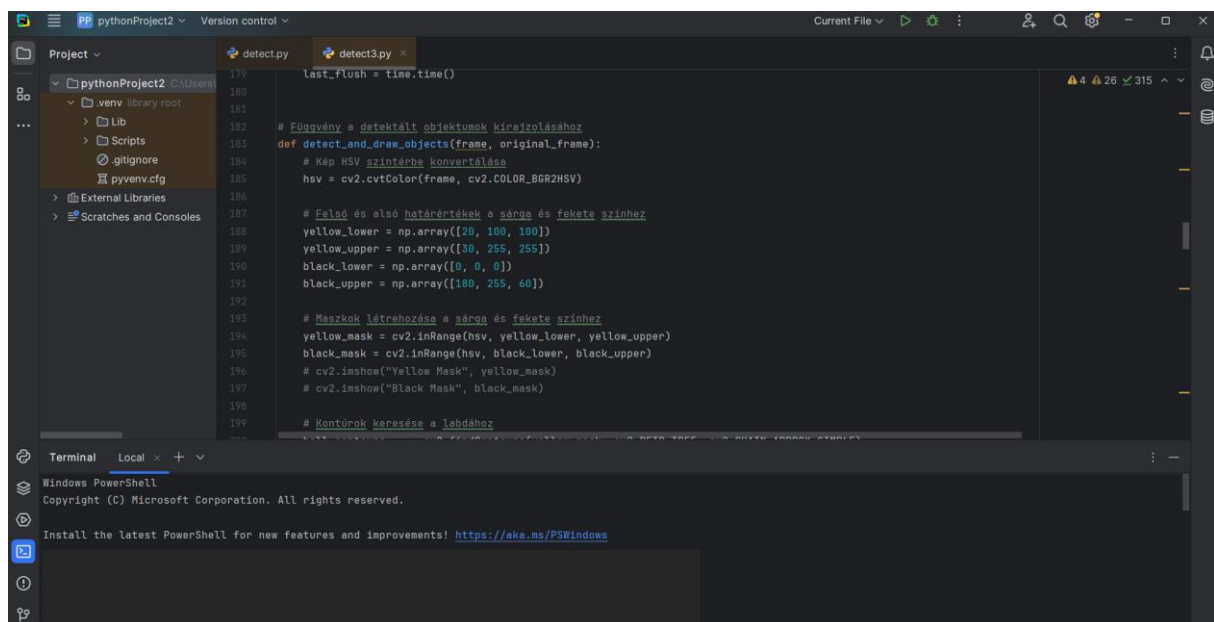
7. ábra – Kamera állvány, kamera és a KLASK

6. Felhasznált szoftverek

6.1. PyCharm



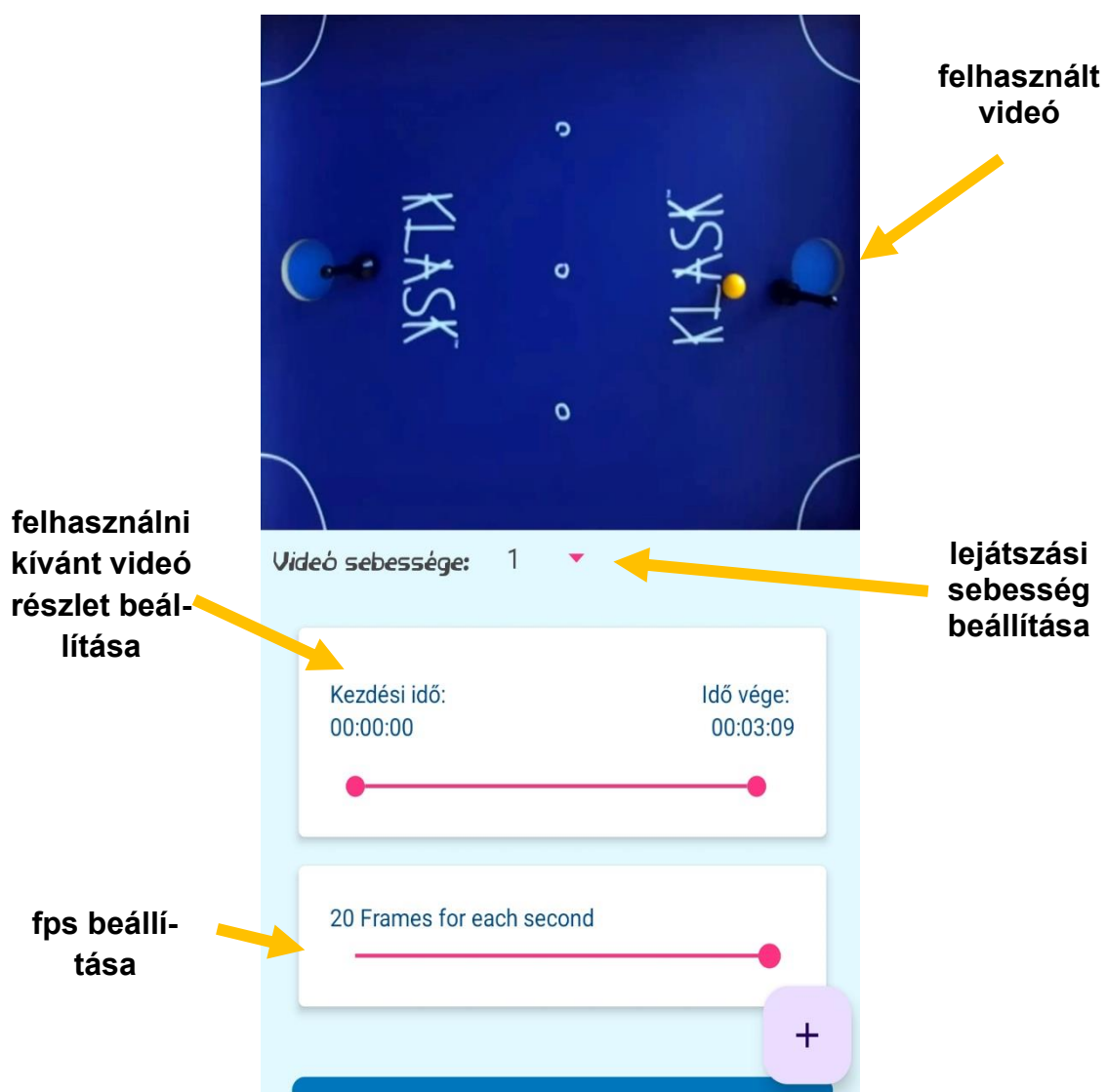
A feladat megvalósításához python programozási nyelvet használtunk, tekintve, hogy rendelkezik a YOLO futtatásához szükséges gépi látás és mélytanulási könyvtárakkal. A kódoláshoz többek között a PyCharm szoftver került felhasználásra, azon belül a PyCharm Professional verzió. A program szándékosan python környezet fejlesztéséhez készült, viszont adatbázis kezelő eszközökkel is rendelkezik, illetve webfejlesztési keretrendszert is támogat.



8. ábra – A PyCharm felülete

6.2. Fotók a videóból

Az egyszerű, Play áruházban is megtalálható „Fotók a videóból” nevű alkalmazás szükségessége két ponton is megjelenik. Dolgozatunkban a 9. fejezet tartalmazza az annotációt, melyet kézzel végeztünk. Az annotáláshoz a játékmenetről készült videót képkockákra kellett felbontanunk, hogy kijelölhessük rajta a különböző objektumokat. A modellünk ezen képek adatai alapján volt képes tanulni. Később, a modell tanításának befejeztével is használtuk a programot, ellenőrzésre. A tanítások végén előállt, detektált videókat szintén képkockákra bontottuk, majd ezek helyességét manuálisan is ellenőriztük a gépi elemzések mellett.



9. ábra – "Fotók a videóból" alkalmazási felülete

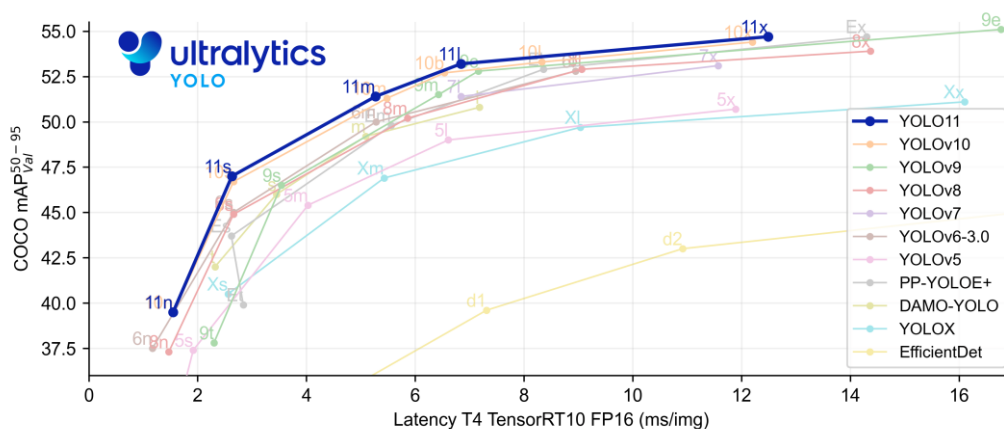
6.3. Roboflow

A Roboflow egy kifejezetten gépi látási projektek fejlesztéséhez készült szoftver. Célja, hogy megkönnyítse a kép alapú AI-modellek fejlesztését, segítve a felhasználókat az annotálásában és előkészítésében, valamint a modellek telepítésében és monitorozásában. Használatához először fel kell tölteni az annotálni kívánt adatkészletet, mely állhat képekből, videókból, vagy objektumfelismerési annotációs fájlokból. Ezután van lehetőségünk objektumok címkéinek hozzáadására, azaz annak meghatározására, hogy milyen objektumokat szándékozunk detektálni munkánk során. Folytatásként 3 lehetőséget kínál: annotálhatunk manuálisa, felkérhetünk embert a feladatra, vagy használhatjuk az automatikus detektálást. [12]

6.4. YOLO

A tanításhoz használt szoftverkörnyezet az Ultralytics YOLO (You Only Look Once) volt, mely egy mélytanulás alapú objektum felismerő modell. A gépi tanulást teszi lehetővé, gyors, pontos, valós idejű detektálásra fókuszálva, így videófelvételben szereplő objektumok nyomon követésére tökéletesen alkalmas. A modell továbbá előre betanított súlyokkal érkezik, és a kód is dokumentált, így könnyen implementálható Python alapú környezetben, melyben mi is dolgoztunk. [13]

Munkánk során a YOLO (You Only Look Once) 11 modellt, azon belül a nano (11n) verziót használtuk, kis teljesítmény igénye miatt. A KLASK társasjáték gyors reakciót és pontos érzékelést igényel, a YOLO pedig egyetlen "pillanat alatt" képes feldolgozni a képet és egy teljes képkockán belül egyszerre azonosítani az objektumokat, így minimálisra csökkenti a késleltetést, és lehetővé teszi a valós idejű megfigyelést.



10. ábra – A YOLO architektúra különböző verziói összehasonlítva [14]

6.5. Google Colab

A tanításhoz a Google Colabot felületét választottuk. A szoftver különlegessége, hogy ingyenes hozzáférést biztosít számítási erőforrásokhoz, beleértve a GPU-kat és TPU-kat is. Használatához nincs szükség telepítésre, és gépi tanuláshoz tökéletesen alkalmas. A Colab erőforrások nem garantáltak és nem korlátlanok, valamint a felhasználási határok időnként változhatnak. Lehetővé teszi, hogy Jupyter notebookokat használjunk anélkül, hogy bármit le kellene töltenünk, telepítenünk vagy futtatnunk. [15]

A felület lehetőséget biztosít Google Drive-hoz történő csatlakoztatásra, így minden szükséges videó, fotó importálható a kódba, a Drive-ból történő behívással.

7. Kódok összehasonlítása

Kutatásunk alapjául a 2023/24 tavaszi félévben szakkollégiumi társaink által publikált kutatás kódját vettük alapul, melyet utána átalakítottunk olyan módon, hogy az általa generált adatok felhasználhatóak legyenek neurális hálózat tanításával történő összehasonlításra.

Az eredeti, *detect1.py* nevezetű kód eredményeként előállt videó annotált játékmenetet tartalmaz, melyben megjeleníti a játékosok és a labda pozícióját, jelölőnégyzetet rajzolva köréjük, illetve az ezekhez tartozó korrekciós tényezős középpontokat. Az ő kódjuk esetén a kimeneti CSV fájl az objektumokhoz tartozó korrekciós tényezős középpont X és Y koordinátáját tartalmazza. Mi ezen változtattunk, esetünkben az előállt videó és a naplófájl sem tartalmazza a korrekciós középpontokat. A videó kizárólag a jelölőnégyzeteket tartalmazza, a naplófájl pedig a jelölőnégyzetek bal felső és jobb alsó sarkának X és Y koordinátáját. A kódunk *detect2.py* névre hallgat. Munkánk végén a, *detect2.py* kód sikerességét vetjük össze a YOLO segítségével feltanított modell detektálásának sikerességével, ezzel meghatározva, hogy melyik megoldás alkalmasabb projektünk folytatásához. [11]

7.1. Objektum pozíciók kezelése

```
def log_positions(players, ball, cm):  
    ...  
    x, y = convert_to_cm(player[1], player[2])  
    positions_cache.append((x, y))  
    else:  
        positions_cache.append((player[1], player[2]))  
    ...  
    x, y = convert_to_cm(ball[0], ball[1])  
    positions_cache.append((x, y))  
    positions_cache.append((ball[0], ball[1]))  
    else:  
        positions_cache.append(("N/A", "N/A"))  
    ...
```

1. forráskód – *log_positions* függvény a *detect1.py*-ban

```
def log_positions(players, ball, cm):
    ...
    top_left_x, top_left_y = convert_to_cm(player[1][0], player[1][1])
    bottom_right_x, bottom_right_y = convert_to_cm(player[2][0], player[2][1])
    positions_cache.append((top_left_x, top_left_y, bottom_right_x, bottom_right_y))
    else:
        positions_cache.append((player[1][0], player[1][1], player[2][0], player[2][1]))
    ...
    top_left_x, top_left_y = convert_to_cm(ball[0][0], ball[0][1])
    bottom_right_x, bottom_right_y = convert_to_cm(ball[1][0], ball[1][1])
    positions_cache.append((top_left_x, top_left_y, bottom_right_x, bottom_right_y))
    else:
        positions_cache.append((ball[0][0], ball[0][1], ball[1][0], ball[1][1]))
    else:
        positions_cache.append(("N/A", "N/A", "N/A", "N/A"))
    ...
```

2. forráskód – *log_positions* függvény a *detect2.py*-ban

Az 1. illetve a 2. forráskódban egyaránt az objektum pozíciók naplózásának kivitelezése látható, azonban eltérő megközelítéssel. Az 1. forráskód tanulótársaink kódját mutatja, melyben a játékosok korrekciós középpontjának X és Y koordinátáját (*player[1]*, *player[2]*) centiméterekre konvertálja a *convert_to_cm* függvény, majd ezek hozzáadódnak a *positions_cache* listához. A labda pozíciója hasonlóan kerül mentésre. Ha nincs érvényes adat, azaz az aktuális képkocka nem használható (emberi kéz belenyúl a képbe, ezáltal a detektálás abbamarad), az „N/A” értéket tároljuk. Az első kódban tehát pontként kezeljük az objektumokat, mivel a korrekciós tényező közepontot használjuk pozíciójuk meghatározására, így ennek X és Y koordinátája 2 db „N/A” értéket jelent.

Hasonló felépítésű a 2. forráskód, viszont minden játékosra és a labdára vonatkozóan két koordinátapárt (*top_left_x*, *top_left_y* és *bottom_right_x*, *bottom_right_y*) ment a *positions_cache* listába, a jelölőnégyzet bal felső és jobb alsó sarkának x és y koordinátáját. Mivel ebben az esetben 2 darab pontunk van, a sarkok, 4 db „N/A” értéket kerül tárolásra a naplófájlban. A naplófájl feltöltésének működését a 3. és 4. forráskód mutatja, ahol szembe-tűnő, hogy mennyivel több adat tárolódik a *detect2.py* esetében.

7.2. Naplófájl feltöltése

```
def flush_positions():
    ...
    f.write("Player1_X,Player1_Y,Player2_X,Player2_Y,Ball_X,Ball_Y\n")
    for i in range(0, len(positions_cache), 3):
        f.write(f"{positions_cache[i][0]},{positions_cache[i][1]},"
                f"{positions_cache[i + 1][0]},{positions_cache[i + 1][1]},"
                f"{positions_cache[i + 2][0]},{positions_cache[i + 2][1]}\n")
    ...
```

3. forráskód – *flush_positions* függvény a *detect1.py*-ban

```
def flush_positions():
    ...
    f.write("Player1_TopLeft_X,Player1_TopLeft_Y,Player1_BottomRight_X,Player1_BottomRight_Y,"
            "Player2_TopLeft_X,Player2_TopLeft_Y,Player2_BottomRight_X,Player2_BottomRight_Y,"
            "Ball_TopLeft_X,Ball_TopLeft_Y,Ball_BottomRight_X,Ball_BottomRight_Y\n")
    for i in range(0, len(positions_cache), 3):
        if i + 2 < len(positions_cache):
            player1_data = positions_cache[i] if len(positions_cache[i]) == 4 else ("N/A", "N/A", "N/A", "N/A")
            player2_data = positions_cache[i + 1] if len(positions_cache[i + 1]) == 4 else (
                "N/A", "N/A", "N/A", "N/A")
            ball_data = positions_cache[i + 2] if len(positions_cache[i + 2]) == 4 else ("N/A", "N/A", "N/A",
"N/A")
            f.write(f"{player1_data[0]},{player1_data[1]},{player1_data[2]},{player1_data[3]},"
                    f"{player2_data[0]},{player2_data[1]},{player2_data[2]},{player2_data[3]},"
                    f"{ball_data[0]},{ball_data[1]},{ball_data[2]},{ball_data[3]}\n")
    ...
```

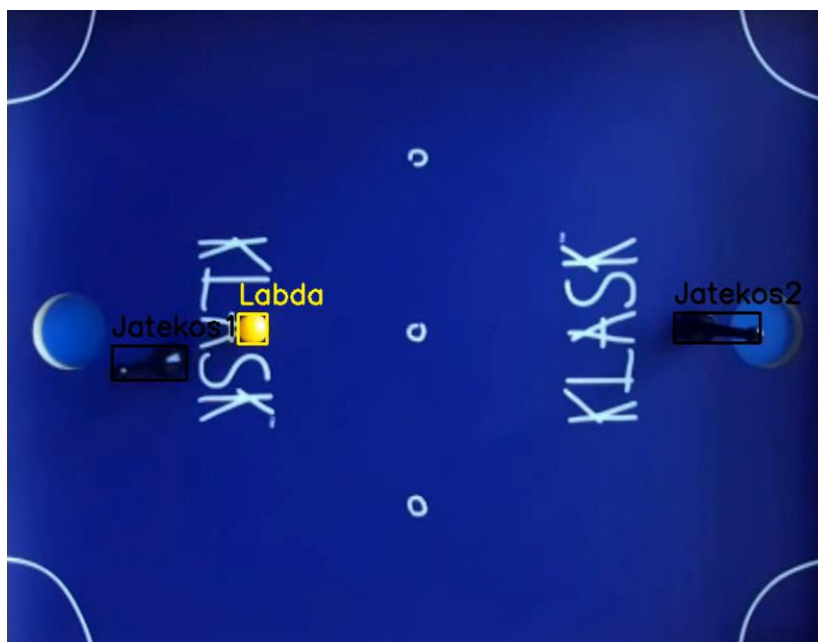
4. forráskód – *flush_positions* függvény a *detect2.py*-ban

Az objektumok pozíciójának naplózását a *flush_positions* függvény végzi. A *positions_cache* listában találhatóak az objektumok koordinátái (a lista feltöltése az 1. és 2. forráskódban látható), a *flush_positions* pedig ennek elemeit írja ki a naplófájlba. A fájl első sorába a címeket menti, majd három elemenként (két játékos és egy labda) kiírja a hozzájuk tartozó adatokat a fájlba. A 3. és 4. forráskód annyiban tér el, hogy míg a 3. kód 6 oszlopot generál (*Player1_X,Player1_Y,Player2_X,Player2_Y,Ball_X,Ball_Y*), majd tölt fel adatokkal, a 4. kód 12 oszlopot (mind a 3 objektumhoz tartozó 2 sarok X és Y koordinátája)

7.3. Pozíciók jelzése

A `detect1.py` és `detect2.py` része továbbá, hogy a sikeresen detektált játékelemek köré egy befoglaló téglalapot rajzol, valamint feliratozza is azokat. A detektálás itt már megtörtént, csupán a felhasználó tájékoztatására van szükség, tehát a befoglaló téglalapok kirajzolására. A téglalap méretét az objektum szélessége (w) és magassága (h) határozza meg. A koordináták (x és y) a jelölőnégyzet bal felső sarkának helyzetét adják meg. A bal oldali térfélen lévő csatár a `Jatekos1`, a jobb oldali a `Jatekos2` nevet kapta, a labda pedig természetesen `Labda` névre hallgat. [11]

A `detect1.py` kódban korrekciós középpontjuk koordinátái által van meghatározva a játékelemek pozíciója, így ezen pont segítségével találja meg a program, hogy mely játékelem köré szükséges befoglaló téglalapot rajzolni. A középpont kiszámítása az $x + w // 2$ és $y + h // 2$ képletekkel történik a labda és a játékosok esetében is. A `detect2.py` esetében a befoglaló téglalap bal alsó és jobb alsó sarkának koordinátái által találja meg a program, hogy hová szükséges befoglaló téglalapot rajzolni. A téglalap rajzolásának képlete a jobb alsó sarok esetén $x + w$ és $y + h$, tehát a kettővel történő osztás elmarad, mivel nem középpontot számolunk. A bal felső sarok koordinátái egyszerűen az x és y értékekkel azonosíthatók, vagyis az első koordináta közvetlenül megegyezik a detektált objektum bal felső sarokpontjának helyével. A `detect2.py` teljesebb információt nyújt a téglalap méretéről (bal felső és jobb alsó sarok koordinátái).



11. ábra – Képkocka a `detect2.py` által előállt videóból

7.4. Felhasznált Python könyvtárak



A detektáláshoz az alábbi Python könyvtárak kerültek felhasználásra:

OpenCV: Ez a könyvtár a képfeldolgozáshoz és számítógépes látás algoritmusokhoz nyújt eszközöket. Használatával került megvalósításra a videó beolvasása (*cv2.VideoCapture*), színterek konvertálása (*cv2.cvtColor*), objektumok kontúrjainak keresése (*cv2.findContours*) és kirajzolása, a perspektíva transzformáció (*cv2.warpPerspective*), továbbá a detektált objektumok pozíciójának jelölése, és videó kimenet mentése. [16]

Numpy: többdimenziós tömbökkel és mátrixokkal való műveletek végrehajtására alkalmas. Munkánk során is ezzel kezeltük a tömböket, matematikai műveleteket. A maszkolt képkockák feldolgozásához az *opencv* és a *numpy* együttes használatára volt szükség. [17]

Datetime: Ezen modul a dátumok és időpontok kezelését könnyíti meg. Esetünkben feladata a fájlnevek egyedi időbélyegzővel való ellátása volt, `((datetime.datetime.now().strftime("%Y%m%d_%H%M%S")))`, így minden futtatás egyedi fájlba menti az objektumok pozícióját. [18]

Time: Idómérésekhez, időzítók használatához specializált könyvtár. A képkockák pozícióinak időzített fájlba írását segíti. Másodpercenként menti az adatokat, hogy csökkentse a fájl műveletek számát.

OS: Az operációs rendszerrel való interakciót segíti. A kódunkban ez ellenőrzi a pozíciókat tároló CSV fájl meglétét (*os.path.isfile*), és listázza a fájlokat a képfájlokat tartalmazó könyvtárba (*os.listdir*). [19]

Argparse: Ezzel a modullal parancssori argumentumokat kezelhetünk. Használata által a kódunkban a felhasználó meghatározhatja, hogy videót vagy validációs képeket dolgozzon fel, illetve, hogy centiméterben vagy pixelben szeretné-e az eredményeket megkapni. [20]

Re: A regular expressions szövegminták kereséséért felel. Feladata volt, hogy a fájlnevekből kivonja a numerikus értékeket, ezzel segítve a fájlok rendezését (*re.findall*) [21]

Rbf: Interpolációra szolgál, két koordináta-rendszer közti pontos átváltásra használható. A kódban a pixel- és a centiméter koordináták között interpolál, így pontos centiméter értékeket tudunk számolni a játékobjektumok pozícióihoz. [22]

8. Annotáció

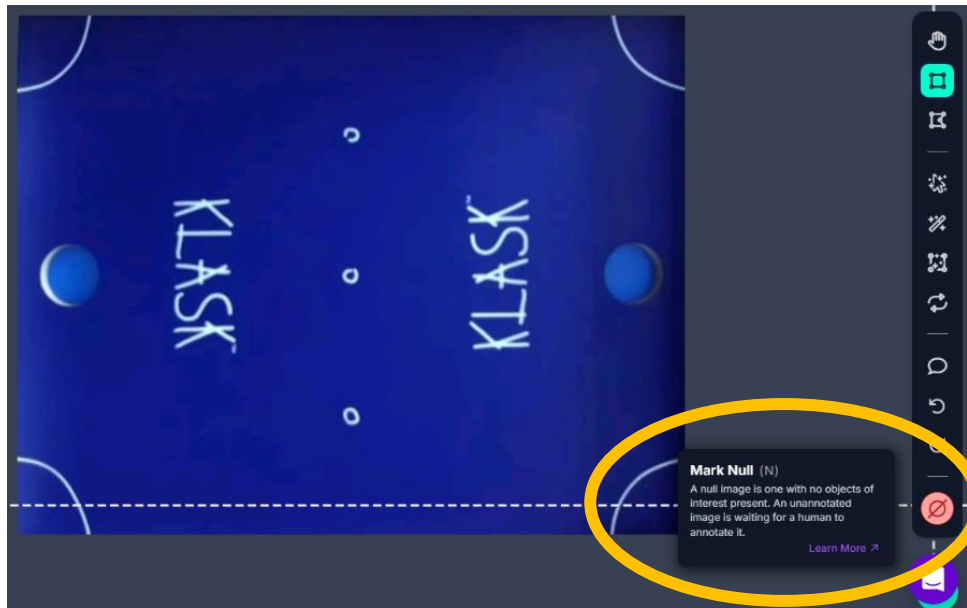
A 3 perc 9 másodperces videónkból, mely a játékmenetet rögzíti, 20 fps használatával 3795 képkocka keletkezett, mely a hibásak eltávolítása után is elegendő a tanításhoz. Miután videót sikeresen képkockákra bontottuk, kézi annotációra volt szükség, melyhez a 6.3. fejezetben említett Roboflow szoftvert használtuk. Projektet hoztunk létre, a képeket feltöltöttük, majd létrehoztuk a 3 objektum címkét, melyek a „Labda”, „Jatekos1”, „Jatekos2”. Az annotációt kézzel végeztük el, hogy a YOLO-nak már a megfelelő formátumban legyen lehetőségünk adatokat adni.

COLOR	COUNT ↻	CLASS NAME
	2113	Jatekos1
	2024	Jatekos2
	2009	Labda

12. ábra – Objektumok meghatározása a Roboflow-ban



13. ábra – Objektumkijelölés felülete a Roboflow-ban



14. ábra – „Üres kép” megjelölése



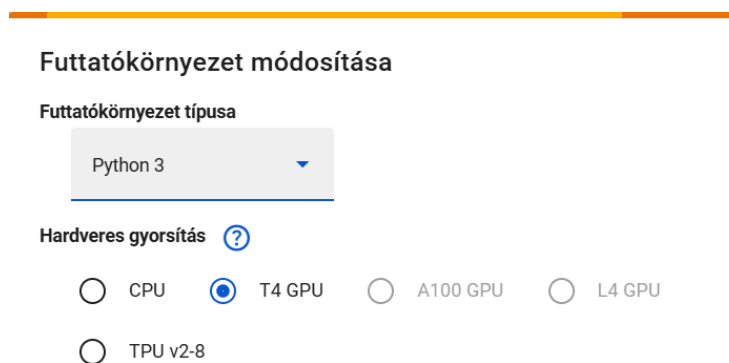
15. ábra – Példa egy sikeresen annotált képkockára

9. Tanítási folyamat

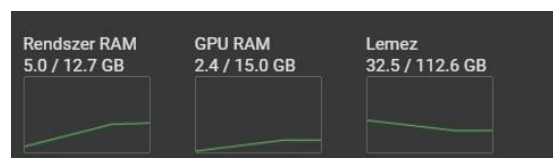
9.1. Előkészítés

A tanítás megkezdése előtt gondoskodnunk kellett a megfelelő környezet beállításáról, hogy a program a lehető legpontosabban és leggyorsabban, megfelelő adatokkal dolgozhasson. Első lépésként hozzárendeltük a szükséges erőforrásokat. Kezdetben CPU-t választottunk számítási egységként, azonban hamar kiderült, hogy ezzel a módszerrel a folyamat túlságosan lassú lenne. Emiatt áttértünk a GPU használatára, amellyel a tanítás kevesebb, mint egy óra alatt be is fejeződött.

A GPU-k különleges képessége a párhuzamos feldolgozás, amely lehetővé teszi számukra, hogy egyszerre számos számítást hajtsanak végre, így felgyorsítva a nagy adatállományokkal való számításokat. A mesterséges intelligencia modellek, különösen a mély neurális hálózatok esetében, a GPU-k kiváló teljesítménynövekedést biztosítanak a párhuzamos feldolgozási képességük révén, így jelentősen lerövidítik a tanítási időt és növelik a hatékonyságot.



16. ábra – Futtatókörnyezet beállítása Colab-ban



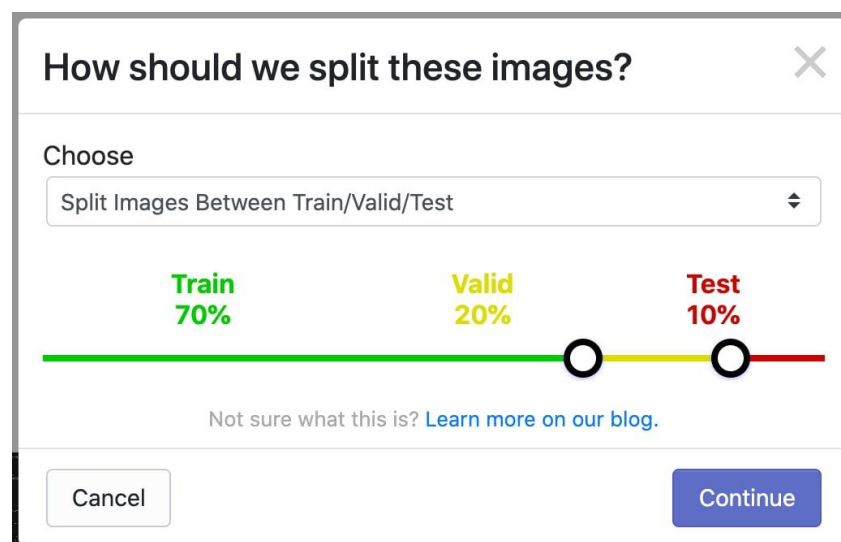
17. ábra – A Colab-ban használt erőforrások

A következő lépésben telepítettük az Ultralytics és PyTorch könyvtárakat, majd importáltuk a YOLO-t, és összekapcsoltuk a Roboflow projektet a Colab környezetünkkel, hogy a program az annotált képek alapján tanulhasson. A PyTorchot fontos megemlíteni, mivel kulcs-szerepet játszik ebben a folyamatban, mivel biztosítja a szükséges eszközöket a modellek tanításához, optimalizálásához és finomhangolásához, és különösen erős a GPU-kkal való kompatibilitásban, ami jelentős teljesítménynövekedést eredményez a komplex számítási feladatok során.

9.2. A tanítás

A dolgozatunk „3.4. Gépi tanulás alapfogalmai” részének végén megjegyeztük, hogy ugyanezeket a tulajdonságokat fogjuk használni mi is a tanítás folyamán, így az egyes értékeket kifejtjük ebben a részben:

Adathalmaz: A videónkat összesen 2936 fényképet tartalmazó adathalmazra konvertáltuk, amelyeket annotáltunk. Ezután „train -tanító”, „valid - validációs” és „test - teszt” halmazokra bontottuk az adathalmazt (a felsorolásnak megfelelően 70%-20%-10% arányban). Ez számszerűen a következő értékeket tartalmazza: 2074 tanító adat, 593 validációs adat, 269 teszt adat.



18. ábra – Az adathalmaz felrészekre bontása

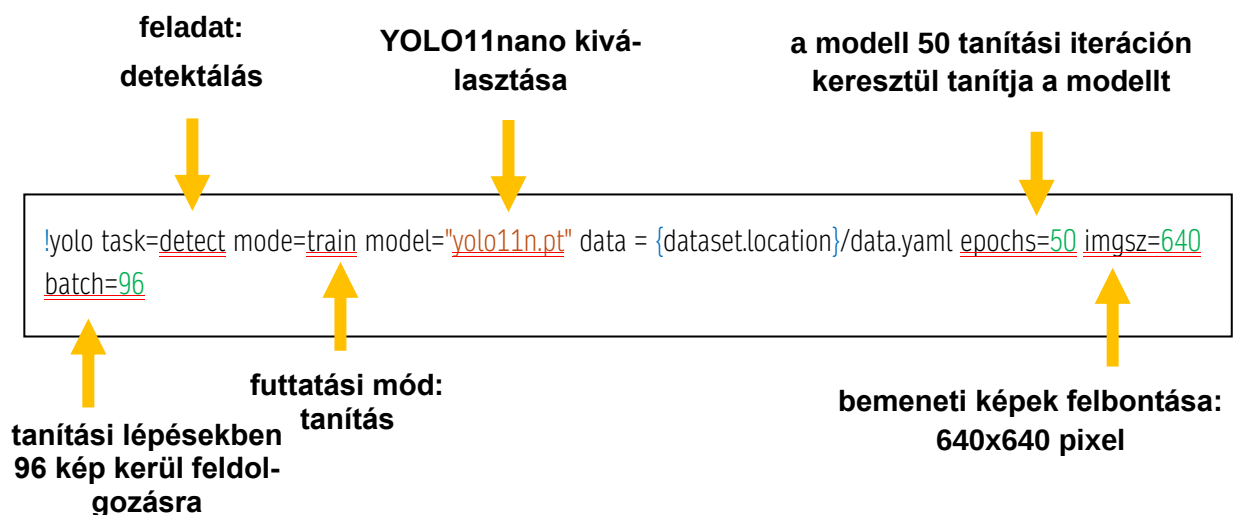
Tanítás: Ez az a folyamat, mely keretén belül az adathalmaz 2074 adatával tanítjuk a model-lünket meghatározott hiperparaméterek mellett.

Tesztelés/validáció: Az a folyamat, amely során az adathalmaz validációs és tesztelési adatain futtatjuk a modellt és az eredményeket kiértékeljük. Ez a folyamat adja meg, hogy a modellünk milyen hatékonysággal ismeri fel az általa még nem ismert adatokat.

Hiba/veszteség: Ez az érték adja meg, hogy a modellünk milyen hibaértékek mellett ismeri fel az általa nem ismert adatokat.

Hiperparaméter: Ezek azok az értékek, amik a tanítás folyamán nem változnak. Jelen esetben:

- Iteráció/epoch: ez az érték adja meg, hogy hány tanulási iteráción keresztül tanuljon a modellünk – a mi esetünkben ez 50, és az „epoch” paraméterrel lehet beállítani.
- Bemeneti képek felbontása: a tanító adatok milyen felbontásúak legyenek. A mi esetünkben 640×640 pixel felbontásúak, és az „imgsz” paraméterrel lehet az értékét állítani.
- Batch: ez az érték adja meg, hogy az egyes epoch-ok során mennyi adat/kép legyen a modellnek „beadva” tanításra. A mi esetünkben ez 96, és a „batch” paraméterrel lehet az értékét beállítani



19. ábra – Tanítást indító utasítás

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
5/50	13.2G	1.338	1.48	1.086	198	640: 100% 22/22 [00:36<00:00, 1.64s/it]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 4/4 [00:10<00:00, 2.68s/it]
	all	593	1427	0.649	0.697	0.704 0.329
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
6/50	12.9G	1.316	1.382	1.068	214	640: 100% 22/22 [00:39<00:00, 1.79s/it]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 4/4 [00:06<00:00, 1.65s/it]
	all	593	1427	0.737	0.849	0.823 0.392
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
7/50	13G	1.291	1.309	1.061	211	640: 100% 22/22 [00:36<00:00, 1.64s/it]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 4/4 [00:10<00:00, 2.66s/it]
	all	593	1427	0.608	0.805	0.731 0.361
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
8/50	13G	1.282	1.249	1.053	198	640: 100% 22/22 [00:37<00:00, 1.71s/it]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 4/4 [00:08<00:00, 2.19s/it]
	all	593	1427	0.611	0.797	0.722 0.375

20. ábra – Részlet a tanulási folyamatból

10. Eredmények

10.1. Az első teszt eredménye

Az első tesztelés során – ahogy a fenti ábrákon is látható – a modell 50 epoch-on keresztül tanulta az adatokat, azaz 50 ciklusban elemezte a rendelkezésére bocsátott adathalmazt. Ennek eredményeképpen előállt egy detektált videó, melyet a *Fotók a videóból* alkalmazás újbóli használatával 3300 képkockára bontottunk fel. Folytatásként ezeket egyesével átnéztük, hogy lehetőségünk legyen a hibaarány meghatározására.

A tesztelésből kinyert adatok:

- összes képkocka: 3500
- helyesen detektált képkockák száma: 3462
- hibás képkockák száma: 38

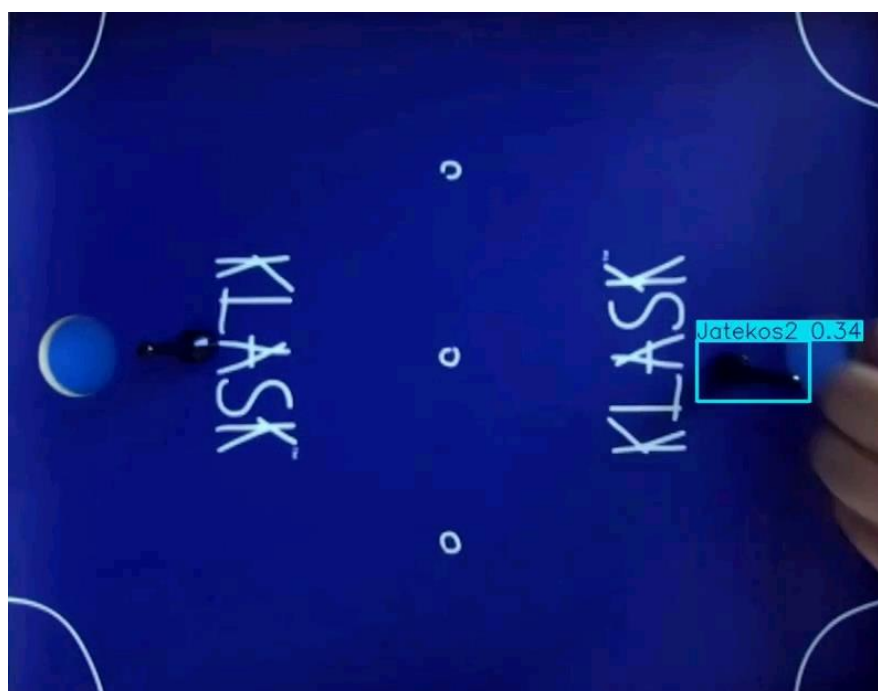
Fellelhető hibák:

1. a gyors objektum mozgás elmosódott képet eredményezett, így a modell képtelen volt annak felismerésére – 25. ábra
2. a detektálás folytatódott olyan esetekben, amikor helyes működés esetén le kellett volna állnia (amikor emberi kéz kerül a felvételbe) – 26. ábra
3. a modell játékosnak vagy labdának érzékelt valamit, ami valójában nem az

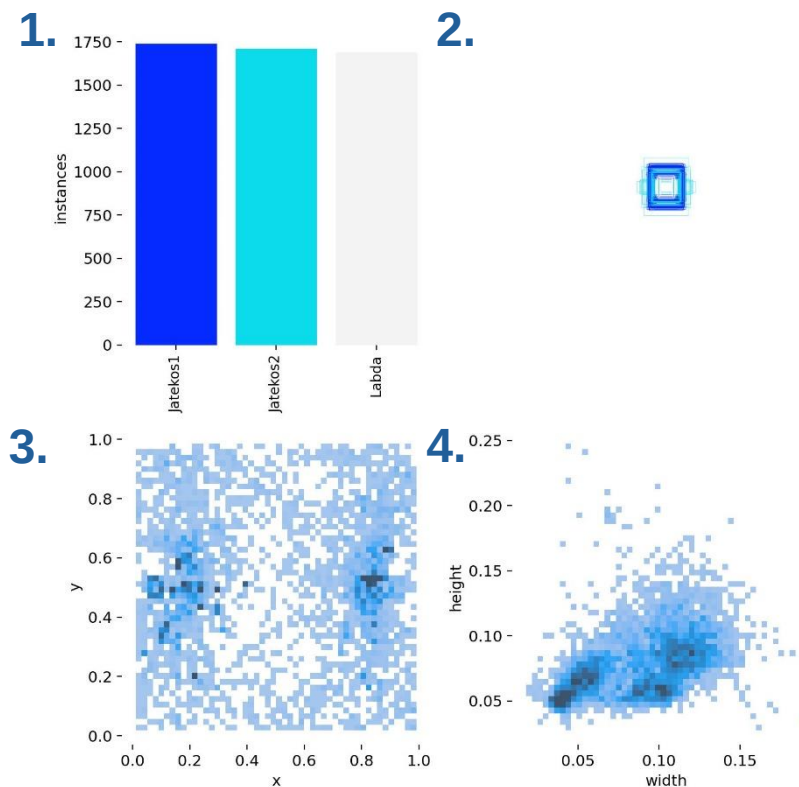
Ezen információkból megállapítható, hogy a modell sikeressége 98,9%-os, a hibaarány így 1,1%. A modell javíthatósága az epoch számban rejlik. Két jelenséget különböztetünk meg ezen témakörben, az alul tanítást, illetve a túl tanítást. Előbbi esetben a modell nem tanult meg elég mintázatot az adathalmazból, így felismerései pontatlanok. Gyenge teljesítményt nyújt új, nem látott adatokon, mivel a tanulási ciklusok száma nem volt elegendő ahhoz, hogy alaposan megértse a releváns jellemzőket. Túltanításról ellenkező esetben akkor beszélünk, hogyha a modell úgymond „megtanulta” a konkrét képeket, ahelyett, hogy általános jellemzőket sajátított volna el. A túltanított modell ezáltal jól teljesít az ismert adatokon, de új adatokra rosszul reagál, mivel túl szorosan követte a tanulási adatok sajátosságait. [23]



21. ábra – Példa hibás képkockára



22. ábra – Példa hibás képkockára



23. ábra – Eredmény vizualizáció

1. Osztályeloszlás oszlopdiagramja

A diagram a 3 osztályhoz („Jatekos1”, „Jatekos2” és „Labda”) tartozó előfordulások számát mutatja. A „Jatekos1” valamivel több példányt tartalmaz, mint a „Jatekos2”, míg a „Labda” előfordulása ennél kicsit kevesebb. Ez azért történik, mert a labda többször elhagyja a játékteret, illetve van néhány másodperc, ahol a Jatekos2 felborul és eltávolításra kerül.

2. IoU – Intersection over Union („Metszet per unió”)

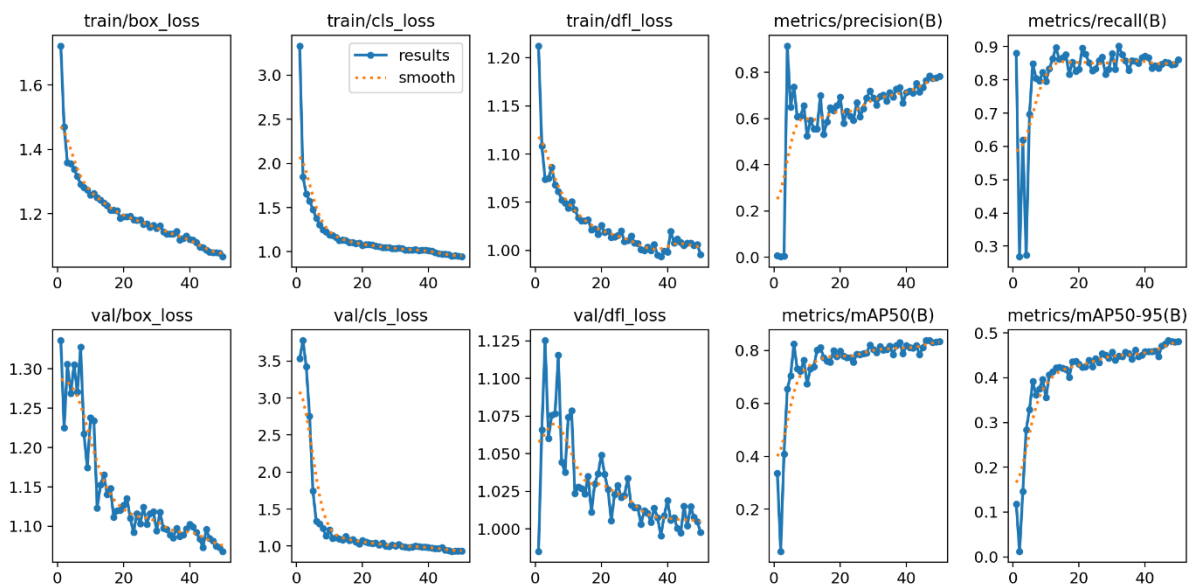
Az ábra megadja, hogy mennyire fedt át egymást egy jósolt, illetve a valós határolókeret. A határolókeretek jelentős átfedésben vannak, ami magas IoU értékeket jelez, azaz a modell jól lokalizálta az objektumokat. [24]

3. X és Y koordináták szórásdiagramja

A szórásdiagram az objektumközéppontok 2D-s eloszlását mutatja az „x” és „y” dimenziókban. A sötét pixelek a gyakori előfordulási helyeket mutatják, és ahogy világosodik a színárnyalat, annál ritkább előfordulásról beszélünk az adott ponton. Az ábra mutatja, hogy az objektumok leggyakrabban a kapu közelében jelentek meg.

4. Objektum szélesség és magasság szórásdiagrammja

A 7. ábra negyedik diagramja a határolókeretek méretének 2D-s eloszlását mutatja. Alacsony számokat ábrázol, melynek jelentése, hogy az előforduló objektumok nem méretesek. A modell tehát kis játékelemek felismerését tanulta.



24. ábra – Veszteségek és pontosság

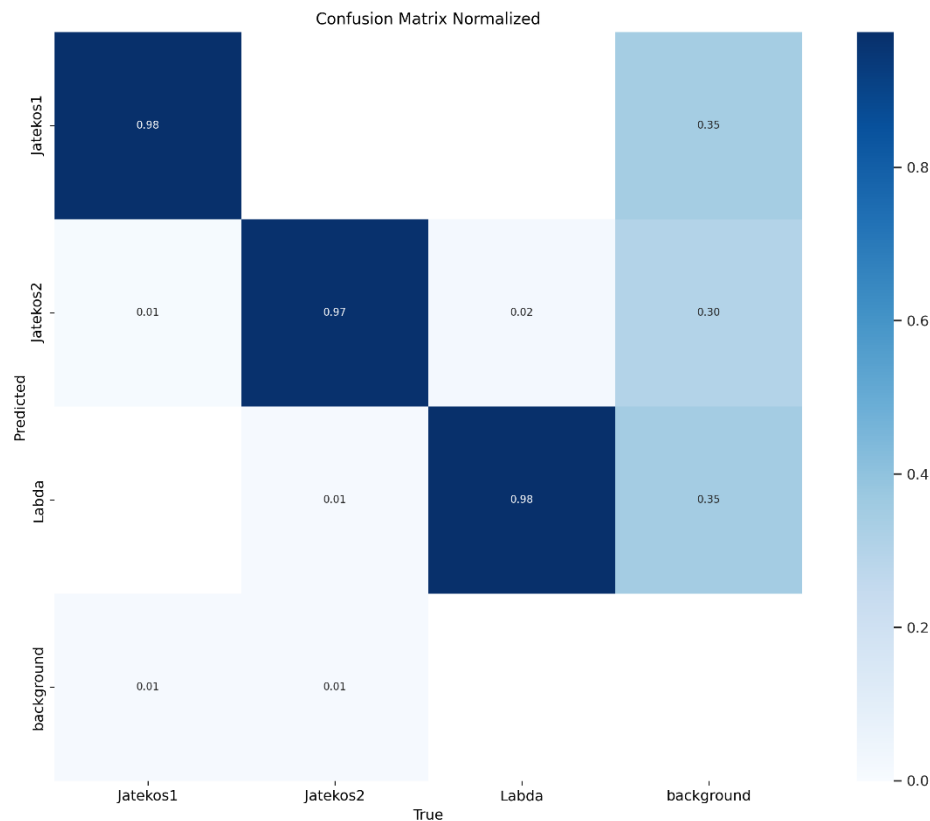
1. Veszteségek:

Megkülönböztetünk tanítási, és validációs veszteséget. Előbbi kategóriába tartozik a *train/box_loss* (határolókeretek, dobozok vesztesége), *train/cl_loss* (osztályveszteség) és a *train/dfl_loss* (diszkrét jellemzők vesztesége). Az ábra alapján a veszteségek az első néhány epoch után gyorsan, nagyjából fokozatosan csökkennek, azaz a tanulás fokozatosan egyre pontosabb lett. A validációs veszteségekhez soroljuk a *val/box_loss*-t, *val/cls_loss*-t és *val/dfl_loss*-t ábrázoló grafikonokat. A veszteségek itt is gyorsan csökkennek, csupán kisebb ingadozások figyelhetők meg a doboz- illetve a diszkrét jellemzők vesztesége esetén. [24]

2. Pontosság és visszahívás:

Pontosság mutatóink közé tartozik többek közt a *metrics/precision(B)*, melynek növekedése arra utal, hogy a modell egyre pontosabbá válik az objektumok helyes azonosításában. A görbe kisimul, ami azt jelzi, hogy a pontosság stabilizálódott. A *metrics/mAP50(B)* hasonló, a modell pontosságát méri 50%-os átfedési küszöbnél, míg a *metrics/mAP50-95(B)* 50% és 95% közötti átfedési küszöbnél teszi ugyanezt, azaz szigorúbb átfedési szinteken mér. Végül a

$metrics/recall(B)$ pedig a visszahívás mutató, melynek növekedése arra utal, hogy a modell sikeresen felismeri az objektumok nagy részét.



25. ábra – Zavarmátrix

Zavarmátrixok:

Elemzésre használt táblázat, mely mutatja, hogy a gépi tanulási osztályozó hogyan teljesít az adatkészleten. A tényleges elhelyezkedést hasonlítják össze a modell előre jelzett értékeivel. Pontosságot, specifikusságot, visszahívást jelenítenek meg. A sorok és oszlopok az egyes osztályokat, azok felismerési helyességét jelenítik meg. A főátlón szereplő értékek a helyes besorolásokat jelzik, míg a mellékátlón szereplők a tévesztések számát. [25]

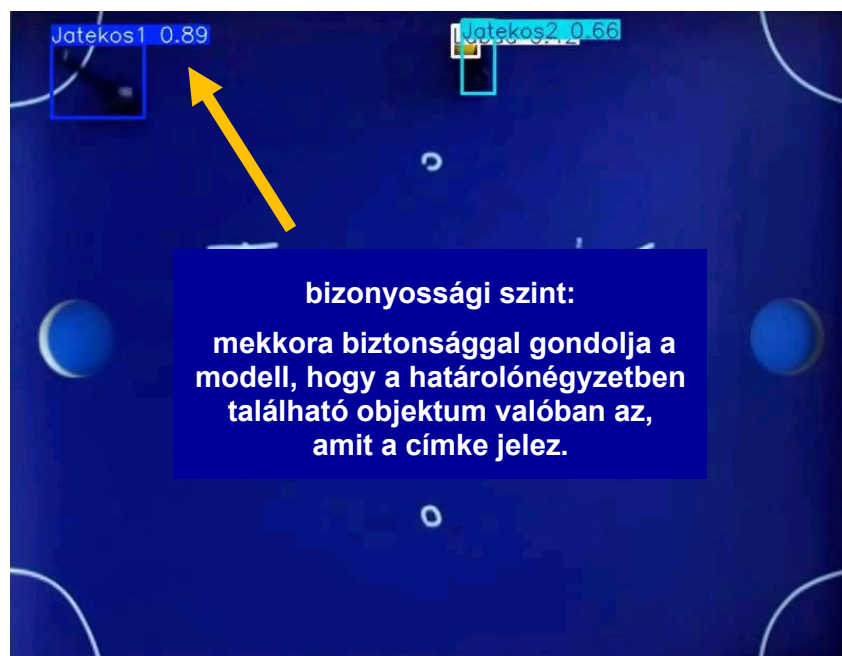
A tanítás zavarmátrixának magyarázata:

A mátrix által megtudhatjuk, hogy a sikeres azonosítás aránya a Jatekos1 és a Labda esetében 98%, míg a Jatekos2 esetében 97%. Ami a tévesztéseket illeti, a modell a Jatekos1-et az esetek 1%-ában Jatekos2-nek, 35%-ában háttérnek osztályozta. A Jatekos2 az esetek 1%-ában Jatekos1-nek és 2%-ában Labdának lett tévesztve, illetve 30%-ban itt is háttérként került besorolásra. A labda az esetek 1%-ában hibásan volt Jatekos1-nek gondolva, és a háttér közötti tévesztés 35%, mely szintén jelentős. Összességében azt állapíthatjuk meg, hogy a modell

számára az objektumok néha nehezen különíthetők el a háttértől, felismerésük viszont szinte tökéletes, közel 100%-os.



26. ábra – Példa helyesen detektált képkockára



27. ábra – Bizonyossági szint

10.2. A második teszt eredménye

A második tesztelési kísérlet során számos problémába ütköztünk. A Google Colab egyik hátránya, hogy a GPU hozzáférés erősen korlátozott és nehezen elérhető, így jelentős mennyiségű próbálkozás eredményeképpen sem sikerült megemelt epoch számmal futtatnunk a tanítási folyamatot. Megoldásként a Pycharm-ot igyekeztünk felhasználni a célra, azonban itt a GPU kihasználás valamilyen okból kifolyólag nem működött, a program hibásan ismerte fel a CUDA illetve a PyTorch verziót. Emiatt CPU alapú futtatást próbáltunk végrehajtani, mely egy egész éjszakán át futott, végül sikertelen eredményt produkálva. További próbálkozásunk során a HP Victus laptop PowerShell terminálját használtuk, ahol a megfelelő parancsok megadása után sikerült GPU alapú tanítást indítanunk. Ennek azért van kiemelkedő szerepe, mert $16\times$ gyorsabb folyamatról beszélünk a CPU alapúhoz tanításhoz képest.

Mindezen erőfeszítések eredményeképp sikerült a 100 epoch-al történő futtatás, az alábbiakban pedig részletezzük a tesztelés eredményeit.

A tesztelésből kinyert adatok:

- összes képkocka: 3500
- helyesen detektált képkockák száma: 3423
- hibás képkockák száma: 77

Fellelhető hibák:

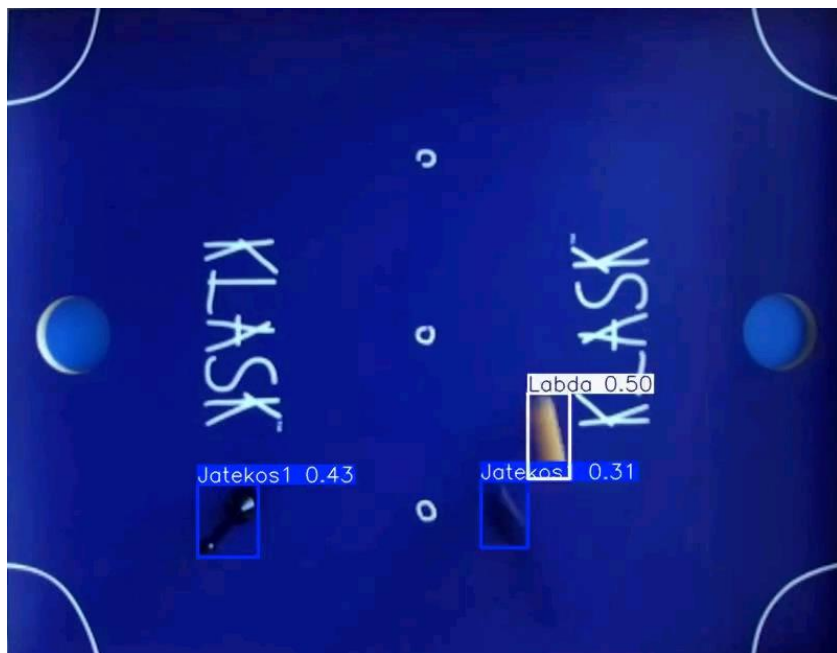
1. a gyors objektum mozgás elmosódott képet eredményezett, így a modell képtelen volt annak felismerésére
2. nem történik gyors mozgás a képkockán, mégsem sikeres a felismerés
3. a detektálás folytatódott olyan esetekben, amikor helyes működés esetén le kellett volna állnia (amikor emberi kéz kerül a felvételbe) – 32. ábra
4. a modell játékosnak vagy labdának érzékelt valamit, ami valójában nem az
5. hibás címkézés, objektumok összetévesztése – 33. ábra

A fenti információkból kiderül, hogy a modell számottevően rosszabbul teljesített az első teszteléshez képest, ugyanis a sikeresen detektált képkockák aránya 97,8%, majdnem egy egész százalékkal gyengébb teljesítményt produkál. A hibatípusok is szélesebb skálán mozognak, ugyanis az első teszt esetében nem fordult elő hibás objektum címkézés, itt viszont sajnos igen. Hangsúlyos hibának tekinthető, hogy a hibák 59%-át az teszi ki, hogy a detektálás emberi kéz megjelenése ellenére is folytatódott, az objektumok érzékelése tehát sok esetben nem fejeződött be. 50 epoch esetén is előfordult, hogy a detektálás későn fejeződött be, azonban jelen

helyzetben minden objektum észlelése fennmaradt, tehát itt hibáról beszélünk a késői reakció-idő helyett. Ezzel megcáfoltuk azon hipotézisünket, hogy a modell alul tanítást szenvedett el 50 epoch esetén, hiszen, ha igazunk lett volna, 100 epoch-nál javuló tendenciát kellett volna mutatnia.



28. ábra – Példa hibás képkockára



29. ábra – Példa hibás képkockára

11. Összegzés és konklúzió

Dolgozatunk lezárásával összegeztük az eredményeket, illetve levontuk a szükséges következtetéseket. 2 tanítás futtatása után megállapítottuk, hogy további próbálkozásaink során valahol 50 és 100 epoch között szükséges tanítanunk, hogy feltehetőleg jobb, pontosabb eredményt tudunk elérni. Ezt a kijelentést a 10.1. fejezetben található magyarázattal indokoljuk. Arra következtetünk, hogy a modell túl tanult a 100 ciklus esetén, 50 ciklussal pedig alul tanítás történt. Mivel 50 epochnál a modell elérte a legjobb pontot (99,1%), ehhez közeli ciklus számmal érdemes folytatnunk a tesztelést, az optimális tanulás elérése érdekében. Továbbá javíthatjuk a sikerességet azzal is, hogy minél több eltérő, szélsőséges adathalmazzal végezzük a tanítást, például eltérő fényviszonyok mellett. A dolgozatunk végeredményének az 50 epochos tanítást tekintjük, így ennek hatékonysága a munkánk végleges eredménye: 99,1%.

A szakkollégiumi társaink által [11] a 2023/24-es tanév tavaszi Tudományos Diákköri Konferencián bemutatott módszerével a modell hatékonysága 99,2%-os volt, ezáltal 0,3%-l pontosabb eredményeket sikerült elérniük. Fontos kiemelnünk azonban, hogy a célunk nem a jobb eredmény volt, csupán annak kiderítése, hogy melyik megoldás képes pontosabb detektálásra, illetve annak vizsgálata, hogy a YOLO-t egy ilyen sajátos környezetben mennyire lehetséges hatékonyan használni. Mivel a hatékonysága magas, az általunk készített modellt hasonló módszerekkel kívánjuk fejleszteni a továbbiakban, eltérő adathalmazzal és epoch számokkal tesztelve, hogy végül a lehető legpontosabb eredményt kaphassuk. Előfordulhat, hogy valahol az 50 epoch közelében találjuk meg a lehető legideálisabb működést. Feltehetőleg a gépi látás megoldásai pontosabb eredményekhez vezethetnek, csak a megfelelő tanítási ciklus mennyiséget kell megtalálnunk. Miután a modellünk képes egy 100%-tól minimálisan eltérő pontosságra, megkezdjük a következő lépést, a virtuális KLASK játékos fejlesztését a Neumann János Informatikai Szakkollégiumban, mely végeztével a fizikai játékost is meg tervezzük építeni. Kísérleteinknek és munkáknak így még közel sincs vége.

12. Irodalomjegyzék

- [1] – *Európai Parlament, „Mi az a mesterséges intelligencia, és mire használják?”* 2020.09.04. [Online]. Elérhető: <https://www.europarl.europa.eu/topics/hu/article/20200827STO85804/mi-az-a-mesterseges-intelligencia-es-mire-hasznaljak>.

[Hozzáférés dátuma: 2024.11.01.]

- [2] – *S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Pearson, 2020.*

[Hozzáférés dátuma: 2024.11.01.]

- [3] – *Oxford English Dictionary, s.v. „artificial intelligence (n.),” 2023 december,* <https://doi.org/10.1093/OED/7359280480>.

[Hozzáférés dátuma: 2024.11.01.]

- [4] – *C. Kollár & B. Nagy, „A mesterséges intelligencia felhasználási lehetőségei az objektumfelismerésben (első rész),” Biztonságtudományi Szemle, vol. 3, no. 1, pp. 123-140, 2021.*

[Hozzáférés dátuma: 2024.11.01.]

- [5] – *T. Kristóf, „A mesterséges neurális hálók a jövő kutatás szolgálatában [Artificial neural networks in Futures Studies],” Jövőelméletek, no. 9, Budapest, Hungary: Budapesti Közgazdaságtudományi és Államigazgatási Egyetem, 2002. Elérhető: <http://dx.doi.org/10.13140/RG.2.1.2835.6562>*

[Hozzáférés dátuma: 2024.10.12.]

- [6] – *M. A. Nielsen, Neural Networks and Deep Learning. Determination Press, 2015.*

[Hozzáférés dátuma: 2024.10.12.]

- [7] – *"Our Story," KLASK, 2013. [Online]. Elérhető: <https://www.klaskgame.com/pages/about-us>.*

[Hozzáférés dátuma: 2024.10.12.]

-
- [8] – **„KLASK: A Detailed Review,”** *BoardGameGeek*, 2020. [Online]. Elérhető: <https://boardgamegeek.com/thread/2566483/klask-detailed-review>.
[Hozzáférés dátuma: 2024.10.12.]
- [9] – **Stereolabs, „ZED 2 Camera and SDK Overview,”** 2019. [Online]. Elérhető: <https://www.generationrobots.com/media/zed2-camera-datasheet.pdf>.
[Hozzáférés dátuma: 2024.10.14.]
- [10] – **„Victus Gaming Laptop 16-r0002nh,”** *HP Magyarország*, 2023. [Online]. Elérhető: <https://www.hp.com/hu-hu/products/laptops/product-details/2101826640>.
[Hozzáférés dátuma: 2024.10.08.]
- [11] – **M. Horváth & Z. Zahuczki, „Képfeldolgozás alapú játékmélet naplózás megvalósítása a Klask társasjátékhoz,”** *Tudományos és Művészeti Diákköri Konferencia, Széchenyi István Egyetem, Győr, Magyarország*, 2024.04.24.
[Hozzáférés dátuma: 2024.10.08.]
- [12] – **B. Dwyer, J. Nelson, T. Hansen, et al., „Roboflow (Version 1.0) [Software],”** 2024. [Online]. Elérhető: <https://roboflow.com>.
[Hozzáférés dátuma: 2024.10.18.]
- [13] – **G. Jocher & J. Qiu, Ultralytics YOLO11, version 11.0.0,** 2024. [Online]. Elérhető: <https://github.com/ultralytics/ultralytics>.
[Hozzáférés dátuma: 2024.10.14.]
- [14] – **R. Khanam & M. Hussain, „YOLOv11: An Overview of the Key Architectural Enhancements,”** *arXiv preprint arXiv:2410.17725*, 2024.
[Hozzáférés dátuma: 2024.10.14.]
- [15] – **Google, „Google Colaboratory,”** [Online]. Elérhető: <https://colab.research.google.com/>.
[Hozzáférés dátuma: 2024.10.15.]
- [16] – **OpenCV Team. Open Source Computer Vision Library.** <https://github.com/opencv/opencv>, 2024
[Hozzáférés dátuma: 2024.10.19.]

-
- [17] – **C. R. Harris et al.**, „*Array programming with NumPy*,” *Nature*, vol. 585, pp. 357–362, 2020. <https://doi.org/10.1038/s41586-020-2649-2>
- [Hozzáférés dátuma: 2024.10.19.]
- [18] – **Python Software Foundation**. *Python Standard Library: datetime*. <https://docs.python.org/3/library/datetime.html>, 2024
- [Hozzáférés dátuma: 2024.10.19.]
- [19] – **Python Software Foundation**. *Python Standard Library: os*. <https://docs.python.org/3/library/os.html>, 2024
- [Hozzáférés dátuma: 2024.10.19.]
- [20] – **Python Software Foundation**. *Python Standard Library: argparse*. <https://docs.python.org/3/library/argparse.html>, 2024
- [Hozzáférés dátuma: 2024.10.19.]
- [21] – **Python Software Foundation**. *Python Standard Library: re*. <https://docs.python.org/3/library/re.html>, 2024
- [Hozzáférés dátuma: 2024.10.19.]
- [22] – **P. Virtanen et al.**, „*SciPy 1.0: Fundamental algorithms for scientific computing in Python*,” *Nature Methods*, vol. 17, pp. 261–272, 2020. <https://doi.org/10.1038/s41592-019-0686-2>
- [Hozzáférés dátuma: 2024.10.19.]
- [23] – **D. Bashir, R. G. Baraniuk, & L. Carin**, „*An Information-Theoretic Perspective on Overfitting and Underfitting*,” *arXiv preprint arXiv:2010.06076*, 2020.
- [Hozzáférés dátuma: 2024.10.30.]
- [24] – **H. Rezatofighi et al.**, „*Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*,” *arXiv preprint arXiv:1902.09630*, 2019.
- [Hozzáférés dátuma: 2024.11.01.]
- [25] – **I. Düntsch & G. Gediga**, „*Confusion Matrices and Rough Set Data Analysis*,” *arXiv preprint arXiv:1902.01487*, 2019.
- [Hozzáférés dátuma: 2024.11.01.]

13. Ábrajegyzék

1. ábra - Egy egyszerű mesterséges neurális hálózat	8
2. ábra - KLASK társasjáték	9
3. ábra - KLASK elemei	10
4. ábra - Labdák, kekszek, bábu és a mágnes	Hiba! A könyvjelző nem létezik.
5. ábra - ZED 2 kamera dimenziói, mm-ben kifejezve.....	12
6. ábra - HP VICTUS 16 Gaming Laptop.....	144
7. ábra - Kamera állvány, kamera és a KLASK.....	155
8. ábra - A PyCharm felülete	166
9. ábra - "Fotók a videóból" alkalmazási felülete	177
10. ábra - A YOLO architektúra különböző verziói összehasonlítva	188
11. ábra - Képkocka a detect2.py által előállt videóból.....	233
12. ábra - Objektumok meghatározása a Roboflow-ban.....	255
13. ábra - Objektumkijelölés felülete a Roboflow-ban.....	Hiba! A könyvjelző nem létezik.5
14. ábra - „Üres kép” megjelölése	266
15. ábra - Példa egy sikeresen annotált képkockára ...	Hiba! A könyvjelző nem létezik.6
16. ábra - Futtatókörnyezet beállítása Colab-ban	277
17. ábra: A Colab-ban használt erőforrások	267
18. ábra - Az adathalmaz felrészekre bontása.....	28
19. ábra - Tanítást indító utasítás	229
20. ábra - Részlet a tanulási folyamatból	300
21. ábra – Példa hibás képkockára	Hiba! A könyvjelző nem létezik.2
22. ábra – Példa hibás képkockára	322
23. ábra – Eredmény vizualizáció	Hiba! A könyvjelző nem létezik.3
24. ábra – Veszteségek és pontosság	344
25. ábra – Zavarmátrix	353

26. ábra – Példa helyesen detektált képkockára.....	366
27. ábra – Bizonyossági szint	366
28. ábra – Példa hibás képkockára.....	38
29. ábra – Példa hibás képkockára.....	38

14. Forráskód jegyzék

1. forráskód – log_positions függvény a detect1.py-ban	20
2. forráskód – log_positions függvény a detect2.py-ban	21
3. forráskód – flush_positions függvény a detect1.py-ban	22
4. forráskód – flush_positions függvény a detect2.py-ban	22