Eötvös Loránd Tudományegyetem

Informatikai Kar

Mesterséges Intelligencia

Tanszék

Automatikus annotáció pózbecsléshez és mozgási sebesség méréshez gerincsérült patkányok adatbázisán

Témavezető:
Dr. Gelencsér-Horváth Anna
Tudományos munkatárs, PhD

Szerző:
Dallos Loránd
programtervező informatikus BSc

Budapest, 2025

EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR

SZAKDOLGOZAT TÉMABEJELENTŐ

Hallgató adatai:
Név: Dallos Loránd
Neptun kód: XIKVIF

Képzési adatok:
Szak: programtervező informatikus, alapképzés (BA/BSc/BProf)
Tagozat : Nappali
Belső témavezetővel rendelkezem

Témavezető neve: Dr. Gelencsér-Horváth Anna
munkahelyének neve, tanszéke: **ELTE IK, Mesterséges Intelligencia Tanszék**
munkahelyének címe: **1117, Budapest, Pázmány Péter sétány 1/C.**
beosztás és iskolai végzettsége: **tudományos munkatárs, PhD**

A szakdolgozat címe: Automatikus annotáció póz becsléshez és mozgási sebességének méréséhez gerincsérült egerek adatbázisán

A szakdolgozat témája:
(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását)

A szakdolgozat célja egy alkalmazás elkészítése, amely elősegíti az egerek mozgásának elemzésére épülő kutatást. Az orvosi célú kutatásokban az annotáció hosszadalmas és monoton folyamat. A Semmelweis Egyetem kutatói egerek hátsó lábainak mozgását szeretnék vizsgálni gerincsérülés utáni rehabilitációra irányuló kutatás keretében. A humán annotáció kiváltására mesterséges intelligencia, mély tanulás és a gépi látás módszereivel olyan mély hálót tanítok, amely a teljes adathalmaz legfeljebb 10%-ának annotációja után az adathalmaz maradék részét képes automatikusan felcímkézni. A prediktált kulcspontok alapján olyan módszer implementációját készítem el, amely a képi adatból a releváns kulcspontok mozgási mintázatát/sebességét további elemzésre alkalmas formátumban generálja.
Az algoritmushoz készül egy felhasználóbarát grafikus felület amelyen: betölthető a tanított modell és a videó, amihez az elemzést szeretné a felhasználó elkészíttetni a programmal; generálható a kulcspontokkal kapcsolatos adat további elemzésre.

Budapest, 2024. 10. 11.

Tartalomjegyzék

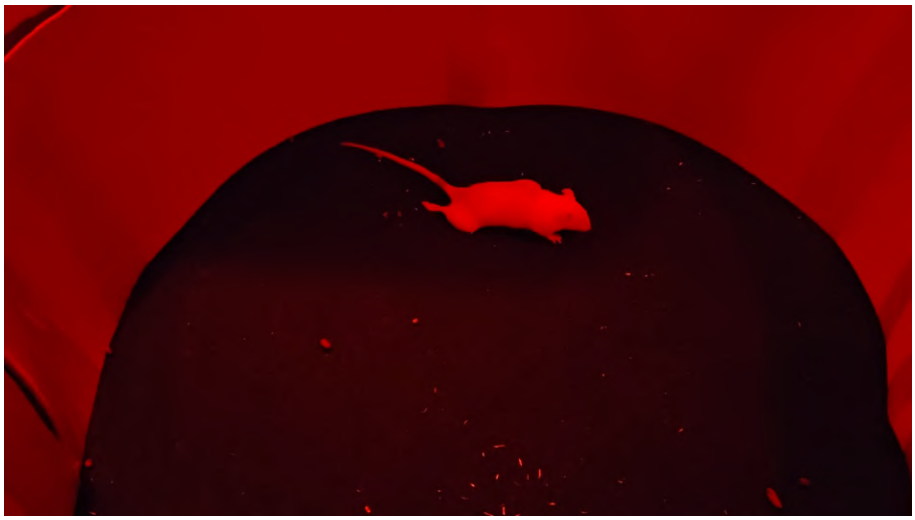
1. fejezet

Bevezetés

1.1. Motiváció

A orvosi és gyógyszerészeti kutatások sok esetben vizsgálják az élő, kezelt és kezeletlen kísérleti állatok viselkedési vagy mozgási mintáit. Jellemzően felvételeket készítenek a vizsgált példányokról, amelyeket aztán humán annotációval látnak el, hogy a szükséges mintázatokat és összefüggéseket kinyerjék az adatból. Ez a folyamat hosszadalmas, szakértelmet kevéssé igénylő, monoton és drága feladat, ezért az automatizálás nagy mértékben tudja segíteni a kutatás hatékonyságát.

A Semmelweis Orvostudományi Egyetem Biofizikai és Sugárbiológiai Intézet kutatói patkányok lábának mozgását vizsgálják. A kísérlet során Sprague-Dawley patkányokon végeztek sebészeti beavatkozást, mely során roncsolták a gerincvelőt a T9-T10-es csigolyák magasságában. Ennek eredményeként az állatok hátsó testtájai érzéketlenné váltak, megbénultak. A különböző csoportok különböző kezelésekben részesültek, ezen kezelések hatásosságát követték nyomon 20 héten keresztül. Az állatok esetleges gyógyulását különböző módszerekkel követték, ezek egyike volt az úgynevezett *Open Field Test* [1, 2]. Ez a teszt az állatok (főként rágcsálók) szorongásos viselkedésének, explorációs aktivitásának és általános mozgékonyságának feltérképezésére szolgál. A tesztet egy felül nyitott, oldalról zárt, kör alakú pályán végezték, teljes sötétségben, infravörös fény mellett. Az állatokról hetente készültek a 4 perc hosszúságú felvételek. Az állatok mozgását 1-től 5-ig tartó skálán pontoztuk, az 1-es jelentette a teljesen béna lábakat és mozgásképtelen hátsó testtáját, az 5-ös pedig a teljesen egészségesen mozgó, aktív állatokat. Az 1.1. ábrán egy minta képkockát mutatok be, ami jól illusztrálja a felvételek jellegét.



1.1. ábra. Minta képkocka, amely jól illusztrálja a felvételek jellegét.

A cél, hogy emberi beavatkozás nélkül vagy minimális emberi időráfordítással lehessen a 20 patkány egyenként 20 videóját kategorizálni az aktuális állapotnak megfelelően. A kategóriákhoz egy-egy klasszifikált videó tartozik referencia adatként. A mozgásminták alapján öt kategóriát különböztetünk meg: az 1-es kategóriába azok a patkányok tartoznak, amelyek szinte egyáltalán nem képesek behúzni a lábukat, míg az 5-ös kategória a teljesen egészséges, gyors lábmozgást végző egyedeknek felel meg. Mivel a részletes annotáció rendkívül hosszú időt venne igénybe, a cél, hogy fejlesztésre kerüljön egy olyan alkalmazás, amely automatikusan kategorizálja a patkány állapotát, kevés mennyiségű annotált adatra építve.

2. fejezet

Felhasználói dokumentáció

Az alábbi fejezetben megtekinthető a programom teljes működése, szakaszaiban bemutatom, hogyan működtethető az alkalmazás egy új számítógépen.

2.1. A program célja

Az alkalmazás célja az, hogy lehetséges legyen egy *ötlépcsős skálán* automatikusan osztályozni egy patkány mozgását megfigyelő videót. A helyes működéshez szükséges egy bemeneti videó, ami a következő formátumok egyikével rendelkezik: '.mp4', '.avi', '.mov', '.mkv', '.flv', '.wmv'. A program a felhasználói felületen egy értékelést ad vissza a bemeneti videóra, emellett szemléltetve a választott osztály adataiból készített grafikonokat a bemeneti videóból generált grafikonokkal együtt. Az említett bemeneti és kimeneti adatokról, illetve a kiegészítő funkciókról bővebben a 2.5. szakaszban értekezem.

2.2. Gépigény

Az alkalmazást Linux operációs rendszerrel rendelkező számítógépeken, illetve távoli szervereken való futtatásra fejlesztettem. Az általam használt gép Ubuntu 22.04 disztribúciót futtat, és egy NVIDIA GeForce RTX 4050 videókártyával, 6GB virtuális memóriával rendelkezik. A futtatás lehetséges dedikált videókártya nélkül is, de ebben az esetben a futási idő jelentős növekedésével érdemes számolni, ezért a megadott konfiguráció az ajánlott minimum követelmény. A programot szerveren, távoli eléréssel is lehet futtatni szerver-kliens formában. A leírt telepítési és futási

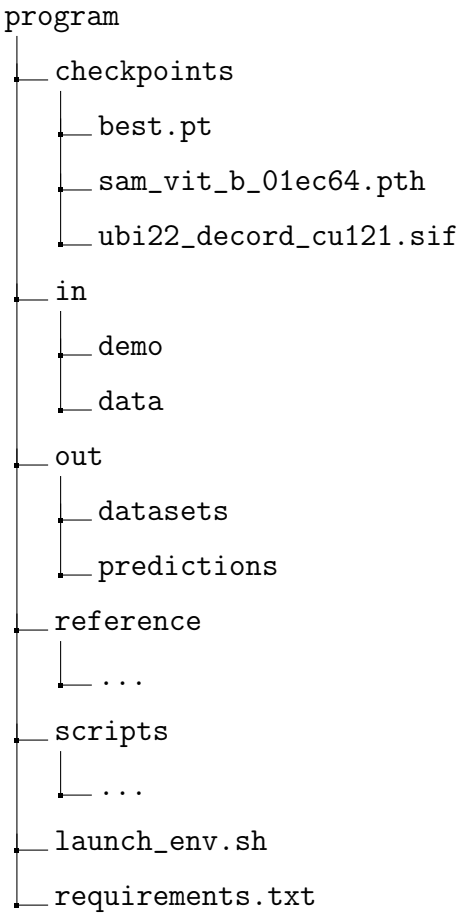
lépéseket olyan szerveren végeztem el, amely két NVIDIA TITAN RTX kártyával rendelkezett, darabonként 24GB virtuális memóriával. Ebből futás alatt körülbelül hat-hét GB-ot igényelt a program.

2.3. Telepítés

A program telepítése során feltételezett, hogy a felhasználó rendelkezésére áll egy felkonfigurált Linuxot futtató gép, vagy egy távoli szerver.

2.3.1. Az alkalmazás fájl-struktúrája

A telepített alkalmazás a következő fájl-hierarchiával rendelkezik (2.1. ábra):



2.1. ábra. A programkönyvtár felépítése.

Az említett könyvtárakat a dokumentáció során részletesebben bemutatom. A `scripts` és `reference` tartalma a fejlesztői dokumentációban (3.4. és 3.14. ábrák)

található kifejtve. Ezekkel felhasználóként nem igényelt az érintkezés.

2.3.2. Lépések

Az alkalmazás fájljainak letöltése utáni lépések a működtetéshez:

(1) Apptainer telepítése:

Az Apptainer szükséges ahhoz, hogy egy virtuális környezetet alakítsunk ki a programnak. Konzolban adjuk ki a következő parancsokat:

```
$ sudo add-apt-repository -y ppa:apptainer/ppa
$ sudo apt update
$ sudo apt install -y apptainer-suid
```

Bővebb információk az Apptainer telepítéséről elérhetőek itt: [3].

(2) Apptainer környezet kialakítása:

A 2.1. ábrán látható `checkpoints/ubi22_decord_cu121.sif` és `launch_env.sh` segítségével létrehozzuk a futtatáshoz szükséges virtuális környezetet, ami arra szolgál, hogy olyan körülményeket alakítson a gépen a programnak, mint amilyenekben azt fejlesztettem. Ezzel garantált a determinisztikus viselkedést. Futtatjuk a `launch_env.sh` fájlt, majd ezt követően kiadjuk a következő parancsokat:

```
./launch_env.sh
apptainer shell instance://yolo_v12
source .envs/venv_yolo_v12/bin/activate
```

A sikert jelzi, ha a következőképp néz ki a parancssor:

```
(venv_yolo_v12) Apptainer>
```

(3) A szükséges python csomagok telepítése a környezetbe:

Ezeket a `requirements.txt` szöveges fájl tartalmazza, a következő paranccsal telepíthetőek a benne szereplő csomagok:

```
pip install -r requirements.txt
```

A program készen áll a használatra, amit a 2.5. szakaszban mutatok be.

2.3.3. Adatok projektkönyvtába letöltése

A bemeneti adatokat a program az `in` mappából tölti be. A `data` mappába kell letölteni vagy átmásolni a tanításhoz szükséges annotált adatokat, figyelve a szerkezeti helyességükre; a `demo` mappába pedig a klasszifikálni kívánt videó fájlokat.

2.4. Felület

Az alábbi alszakaszokban tárgyalom az alkalmazás kinézetét és lehetőségeit, képekkel ellátva az átláthatóság érdekében.

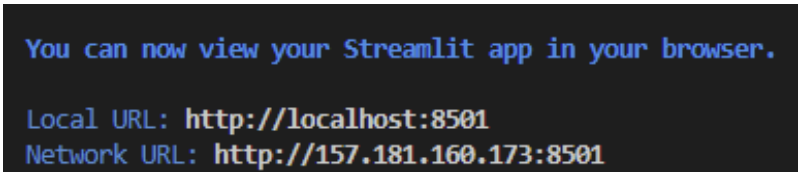
2.5. Használat

2.5.1. Alkalmazás elindítása

Az alkalmazás elindításához egy terminálban a program telepítési könyvtárába, ezen belül a `scripts` mappába kell navigálni, és futtatni a következő parancsot:

```
python -m streamlit run index.py
```

Ezzel futtatjuk a program belépési pontját, a parancs után egy hasonló konzol üzenet kerül kiírásra (2.2. ábra):



2.2. ábra. Az alkalmazást elérő URL-ek.

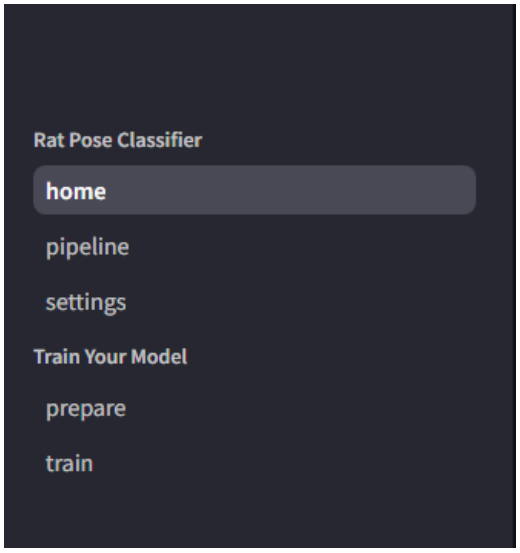
Nyissuk meg a felső URL-t, ami az applikáció főoldalára kísér, ahol újabb összefoglalást kapunk az alkalmazás működéséről (2.3b. ábra).

2.5.2. Munkamenet bemutatása

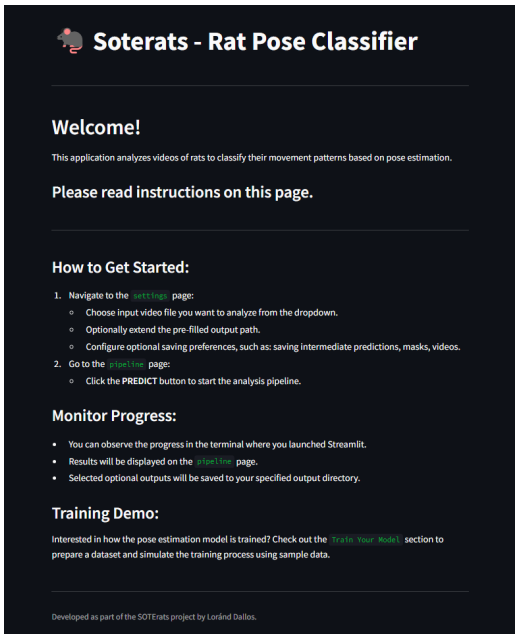
Főoldal

Ez az alkalmazás alapértelmezett kezdő oldala, ahova bedobja a felhasználót indításkor. Az oldalakat a képernyő bal oldalán szereplő menü (2.3a. ábra) segítségével lehetséges navigálni. Itt tudjuk ellenőrizni, hogy a `home` oldal aktív. Az oldalon

útmutató információ szerepel (2.3b. ábra), mely segíthet a következő lépés meghatározásában.



(a) A program navigációs menüje.



(b) A program főoldala.

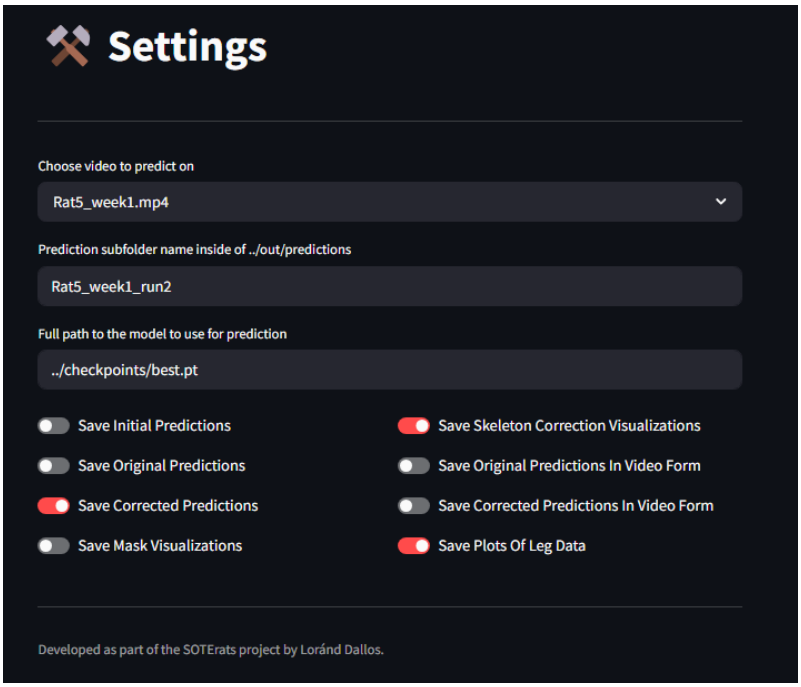
2.3. ábra. A program indításakor szembetűnő felület.

A főoldalon ismertetett információ elolvasása után a `settings` oldalon megadjuk a program futásához szükséges beállításokat.

Beállítások konfigurálása

Az oldal a 2.4. ábrán látható, a programban `settings` néven található meg az oldalsó menüben.

Látható két útvonal bekérő vezérlő, illetve nyolc kapcsoló. A legfelső legördülő menü felkínálja az `in/demo` mappában tárolt videókat, a második vezérlő automatikusan elnevezi a kimeneti könyvtárat a bemeneti videó alapján, lehetőséget hagyva egyéni átnevezésre. Ugyanarra az adatra történő újra-futtatás esetén érdemes egyedi kiegészítő karaktersorral bővíteni a mappa nevét, hogy a korábbi mentést ne írja felül. A kimeneti útvonalaknak szánt gyűjtőmappa a projekt könyvtáron belül az `out/predictions`, ezt a program létrehozza, és ebbe menti a kimenetet egy almappába, melynek neve megegyezik a bemeneti videó nevével. A harmadik mező lehetőséget kínál a használt modell kiválasztására, alapértékül felveszi az általam tanított modell útvonalát. Ezek voltak a kötelező beállítások. A nyolc kapcsoló a feldolgozás során keletkező köztes állapotokat teszi menthetővé. Beállításuk nem kö-



2.4. ábra. A beállítások oldal.

tező és egymástól független, kivéve a videóvá fűzéssel kapcsolatos beállításokat, melyeknek alapfeltétele a videóhoz szükséges képkockák mentése. Amennyiben ez az előfeltétel nem teljesül, a program lefut, de nem kerülnek mentésre a videók és a program felhívja a figyelmet az esetleges malőrre. A beállítások magyarázata a 2.1. táblázatban található összefoglalva:

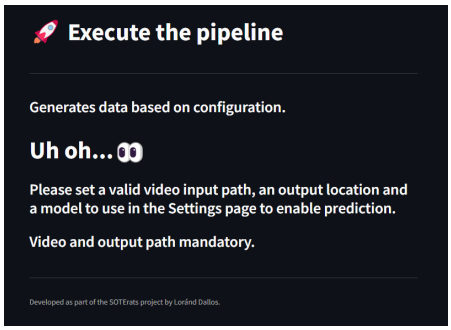
A futtatáshoz adjuk meg a kötelező adatokat, és válasszunk ki tetszőleges köztes mentési állapotokat. A beállítások megadása után a program futtatható a `pipeline` oldalról indítva.

Kapcsoló neve	**Funkció leírása**
`Save Initial Predictions`	Elmenti az elsődleges, nyers YOLO predikciókat (még a feldolgozási lépések előtt, alapértelmezett stílusban kirajzolva a képekre).
`Save Original Predictions`	Elmenti a korrigálatlan, de már feldolgozott predikciókat saját kirajzolási stílussal.
`Save Corrected Predictions`	Elmenti a végleges, lábujjpozíciók alapján korrigált predikciókat saját kirajzolási stílussal.
`Save Mask Visualizations`	Képformátumban elmenti a SAM maszkokat, vizuálisan megjelenítve az objektum körvonalát.
`Save Skeleton Correction Visualizations`	Elmenti a csontvázkorrekció során keletkező képeket, melyeken láthatók a végpontok és az esetleges korrekciók, valamint a keresési sugár.
`Save Original Predictions In Video Form`	Videóvá fűzi az eredeti predikciókat tartalmazó képkockákat.
`Save Corrected Predictions In Video Form`	Videóvá fűzi a kijavított predikciókat tartalmazó képkockákat.
`Save Plots Of Leg Data`	Elmenti a lábmozgásból származtatott idősoros adatokat (amplitúdó, frekvencia, stb.) ábrázoló grafikonokat.

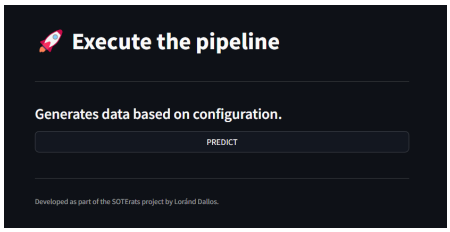
2.1. táblázat. Kimeneti opciók jelentése az alkalmazásban.

Feldolgozás

A `pipeline` oldal egyszerű, egyetlen interaktív elemmel rendelkezik. Ez a `PREDICT` gomb, amivel elindítható a bemenetként megadott videó feldolgozása, feltéve, hogy érvényes bemeneti és kimeneti útvonalat állítottunk be (2.5b. ábra), illetve megadtunk egy használandó modell elérési útvonalát. Amennyiben ez nem így történt, javításra késztető üzenet köszönt az oldalon, illetve a gomb nem elérhető (2.5a. ábra). Lefutás után ezen az oldalon látható a predikció eredménye.



(a) Az oldal beállítások átugrása esetén.



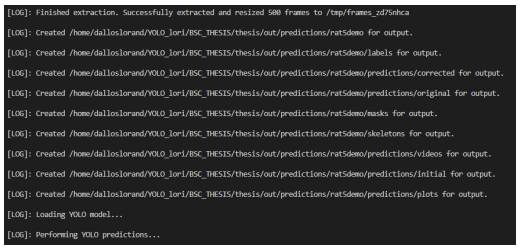
(b) Az oldal helyes beállítások esetén.

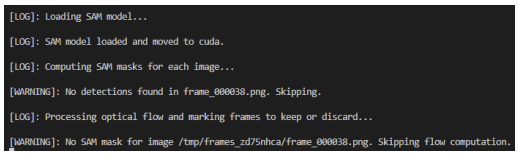
2.5. ábra. A `pipeline` oldal lehetséges kinézetei.

A feldolgozási gombra kattintással elkezdődik a megadott bemeneti adatok feldolgozása, ami teljesen automatikus. Az adattól függően a számítások időigényesek lehetnek, ezért a képernyőn megjelenik egy figyelmeztetés és egy forgó elem, ami a program futását visszajelzi. A felhasználó a folyamat főbb lépéseit abból a konzolból, ahonnan a program felületét indította követni tudja. A futás közbeni menüpontok megnyomása inkonzisztens működéshez vezethet, így egy adott feladat futtatása közben az interakciókat érdemes mellőzni.

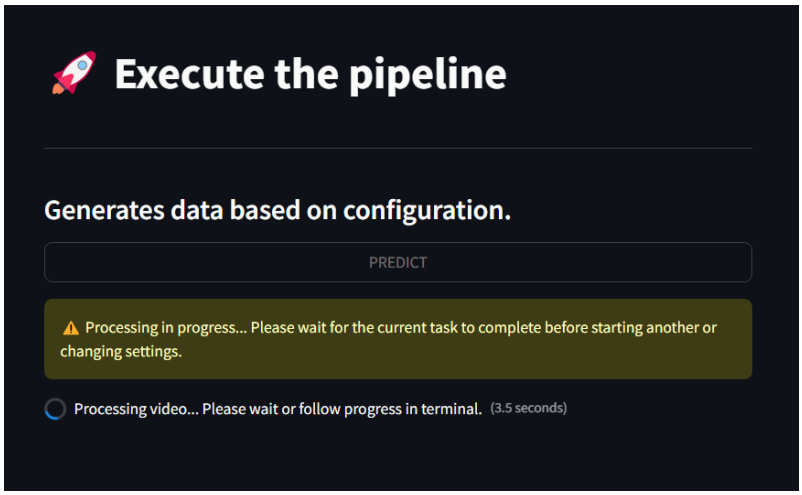
A feldolgozás után a felületen megjelenik a választott mozgási kategória sorszáma, illetve szemléltetésként a bemeneti videóból generált idősor grafikonja a választott kategória idősorának grafikonja mellett. Amennyiben mentettünk, a kimeneti adatok a 3.10. ábrán látható felépítésben jönnek létre és tekinthetők meg. A kimeneti könyvtárban `verdict.txt` szöveges formában kerül mentésre az eredmény.

Szemléltetés képpen megtekinthető az oldal és a konzol futás közben a 2.6. és 2.7. ábrán; lefutás után a 2.8. ábrán:

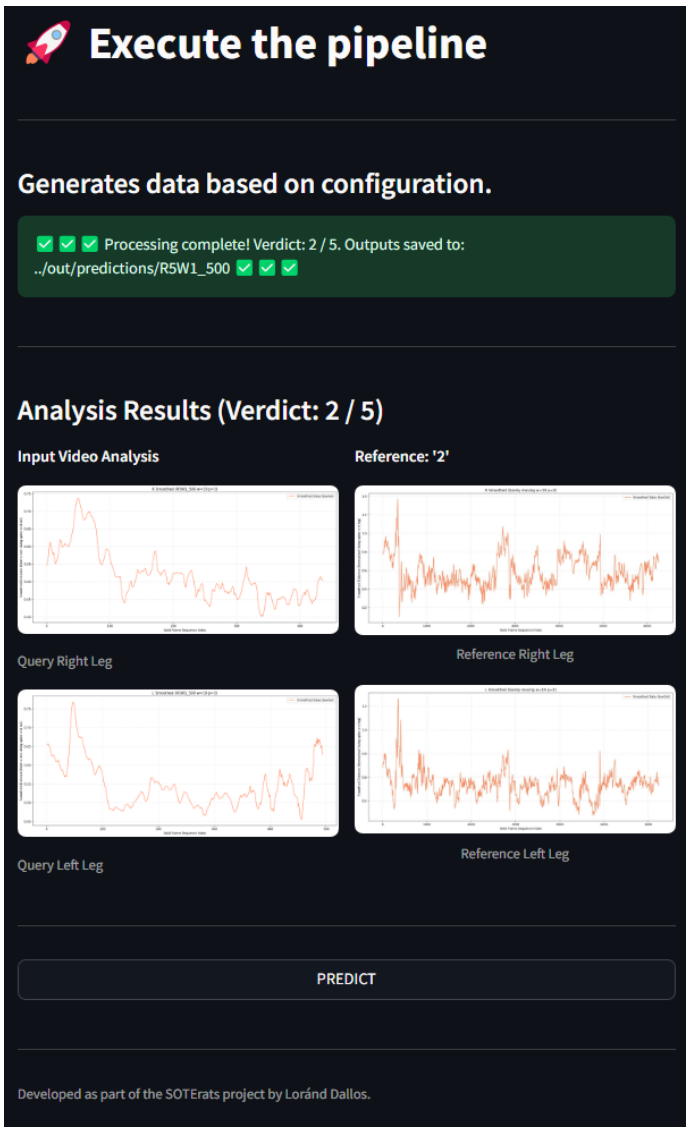




2.6. ábra. Konzolban megjelenő naplózások.



2.7. ábra. Feldolgozás futás közben.



2.8. ábra. Feldolgozás eredménye. A grafikonok kattintással kinagyíthatók.

2.5.3. Modell tanítása

Ez a komponens két oldalt foglal magába: `prepare`, ahol a felhasználó a következőkben bemutatott bemeneti adat alapján tanító adathalmazt generálhat egy saját modell tanításához; `train`, ahol elindítható az adathalmaz alapján a modell tanítása.

Fontos, hogy a felhasználó az alkalmazás ezen részének használata előtt feltöltsön előre meghatározott szerkezetű könyvtárakat (a 2.1. ábrán látható `in/data` mappába), amelyek tartalmazzák a tanító adat alapjául vett képkockákat és annotációkat. Amennyiben a felhasználó további adatot szeretne hozzáadni, ugyanide töltheti fel a megformázott könyvtárakat, melyeket a program felismer.

A kulcspontok annotációja a DeepLabCut [4] szoftverrel készült, ez az alkalmazásom bemenetén elvárt '.csv' fájlokat hozza létre a képi adathoz. A '.csv' fájlokban a következő kulcspontokat annotáltuk minden patkány esetén:

Index	**Kulcspont neve**
0	bal fül
1	jobb fül
2	orr
3	gerinc
4	faroktő
5	farok közép
6	farok vég
7	bal láb comb
8	bal láb térd
9	bal lábfej
10	jobb láb comb
11	jobb láb térd
12	jobb lábfej

2.2. táblázat. Kulcspontok sorszáma és elnevezése.

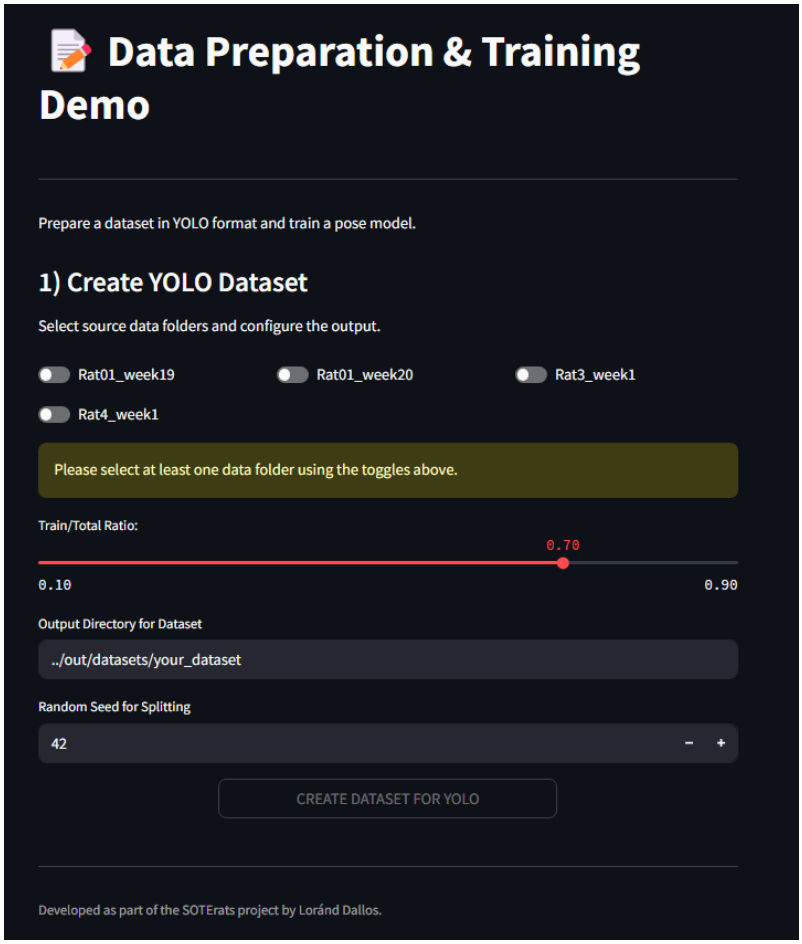
A 2.1. forráskódban megtekinthető egy minta annotáció, ami az annotációs protokollnak megfelelően lett annotálva. Az annotációs protokoll a biológusokkal egyeztetve állítottuk össze a feladathoz illeszkedve. Minden kulcspont egy kétdimenziós (x, y) koordinátarendszer kereteiben annotált.

```
scorer,,,HP,HP,HP, ...
bodyparts,,,left ear,left ear,right ear,right ear,nose,nose,spine,
    spine,tail01,tail01,tail02,tail02,tail03,tail03,left leg,left
    leg,left leg knee,left leg knee,left leg toe,left leg toe,right
    leg,right leg,right leg knee,right leg knee,right leg toe,right
    leg toe
coords,,,x,y,x,y,x,y,x,y,x,y,x,y,x,y,x,y,x,y,x,y,x,y,x,y
labeled-data,Rat01_week19,img0122.png
    ,1090.3079738565527,806.4054245451202, ...
labeled-data,Rat01_week19,img0123.png
    ,1104.091166218707,809.1620630175511, ...
...
```

2.1. forráskód. Kulcspontokat tartalmazó CSV fájl szerkezete.

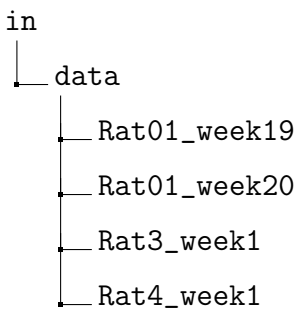
Adatelőkészítés

A `prepare` oldalon található (2.9. ábra), arra szolgál, hogy előkészítsünk egy YOLO pózbecslő modell tanításához szükséges adathalmazt. Kapcsolók láthatóak az `in/data` könyvtárba feltöltött tanítóadatok alapján, ezekből választhatunk.



2.9. ábra. A tanítóadat előállításának oldala.

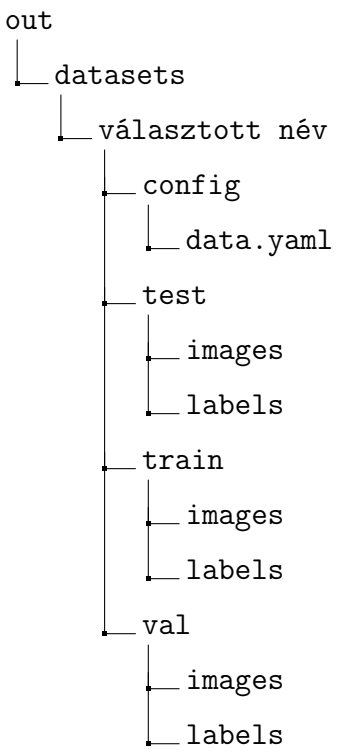
A program felépítésében szereplő `in/data` mappában a fenti ábra alapján a következő könyvtárak szerepelnek:



2.10. ábra. Demó tanító adatok szerkezete a 2.9. ábra alapján.

Ezek alatt egy csúsztatható vezérlő szerepel, amivel kiválaszthatjuk a tanító adatok arányát az összes adathoz képest. Az alapértelmezett beállítás esetén az adatok 70%-át használjuk tanításra. Javasolt legfeljebb az adatok 80%-át tanításra használni, hogy a fennmaradó 20% validációra és tesztelésre szolgáljon, és segítse a modell túltanulásának elkerülését. Továbbá megadható az adathalmaz tárolási útvonala is, melyet ki kell egészíteni az adathalmaz egyedi nevével, például: `out/datasets/ds1`). Ezt a program egy példa névvel előre kitölti, mely szabadon szerkeszthető. Végül egy véletlen szám, amely meghatározza a véletlenszám generálást, és biztosítja, hogy az adatok véletlenszerű felosztása reprodukálható legyen.

A `CREATE DATASET FOR YOLO` gombbal indíthatjuk a generálást, majd szigorúan lefutás után tovább léphetünk. A lefutás hatására az következő struktúrában jön létre adathalmazunk:



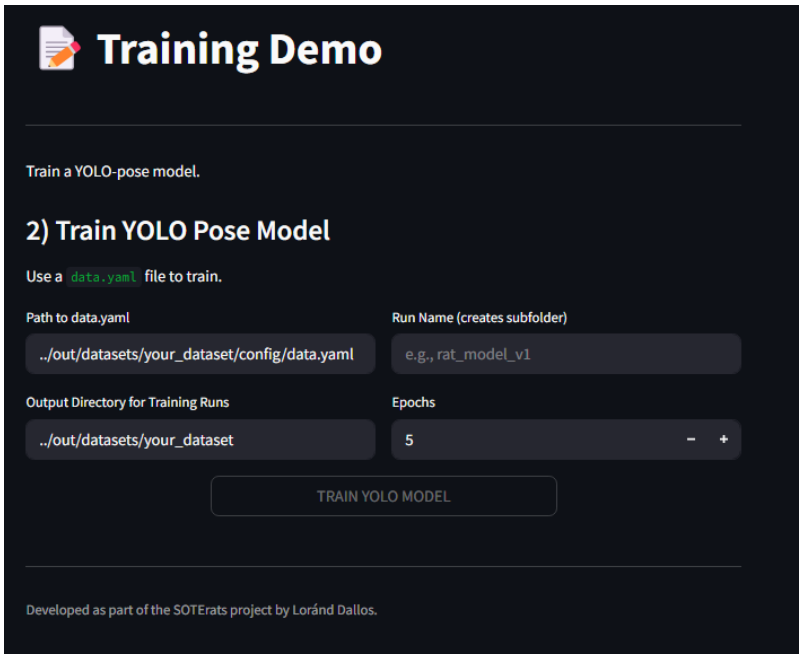
2.11. ábra. Adathalmaz-generálás hatására létrejött felépítés.

A `data.yaml` egy konfigurációs fájl, melynek tartalma megadja, hogy hol található a tanításhoz szükséges adathalmaz.

Tanítás konfigurálása

A `train` oldalon (2.12. ábra) megadjuk a `prepare` oldalon generált konfigurációs fájl útvonalát; a jövőbeli modellünk nevét; a modell tárolási könyvtárának elérési útvonalát (ajánlott az `"out/datasets/előző lépésben generált adathalmaz elérési útvonala"` helyre menteni a modellt, hogy átláthatóan a tanult adathalmaz mellett tároljuk); az edzés ciklusainak számát. Az utóbbi beállítás azt befolyásolja, hogy hányszor iterál végig a tanuló modell az összes tanító adaton. Egy magasabb ciklus szám nagyobb pontosságot eredményezhet, de a tanulási idő jelentősen meghosszabbodhat. A konfigurációs fájl elérési útvonalát, illetve a modell lementési útvonalat a program a `prepare` oldal beállításai alapján kitölti.

A `TRAIN YOLO MODEL` gombbal elindíthatjuk modellünk tanítását.



2.12. ábra. A modell tanító oldal.

2.5.4. Alternatív futtatási lehetőség

Az alkalmazás predikciós része közvetlenül parancssorból is használható (`nogui_pipe.py`). A működés során a videóelemzéshez szükséges bemenet mellett különféle kapcsolókkal lehet szabályozni, milyen kimenetek kerüljenek mentésre. Ezek a kapcsolók megegyeznek a grafikus felületen beállítható mentési opciókkal.

A videófeldolgozás mellett két további szkriptet is készítettem (`nogui_prepare.py` és `nogui_train.py`), amelyek lehetővé teszik az adathalmaz előállítását, valamint a modell tanítását megegyező módon a felhasználói felületes lehetőséggel.

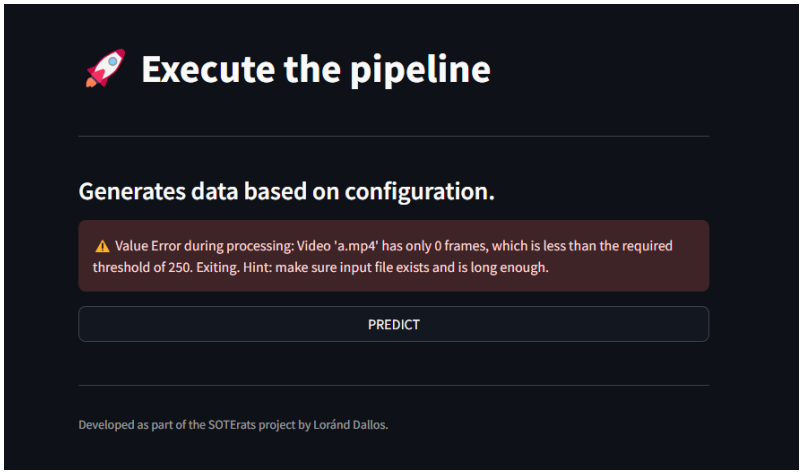
2.6. Helytelen használatból adódó hibák

A program kialakítása segít elkerülni a helytelen használatot. Amennyiben a megadott útvonalon létezik az elvárt formázású adat, a program helyesen lefut. A kimeneti útvonalakon megadott mappákba generálja a kimenetet, szükség szerint létrehozva a mappákat.

A legsúlyosabb hiba, amikor a felhasználó valamelyik feladat futása közben menüpontot vált. Ezt a használt keretrendszerből kifakadóan nem lehet letiltani. A program ezért minden esetben futás alatt egy figyelmeztetést jelenít meg, melyben

felszólítja a felhasználót arra, hogy ne tegye ezt. Ha mégis bekövetkezik az oldalváltás, a program a konzolban továbbra is követhető, ám a felhasználói felület elveszti a kapcsolatot a futási állapottal, ezért egy esetleges visszanavigálásnál a kezdeti beállításokat jeleníti meg. Valamint, ha váltunk egy oldalra egy másik feladat futása közben, majd visszanavigálunk az eredeti oldalra, a felület újra elindítja a folyamatot, megsokszorozva a gépigényt és felülírva az eredeti futtatás kimeneti könyvtárát.

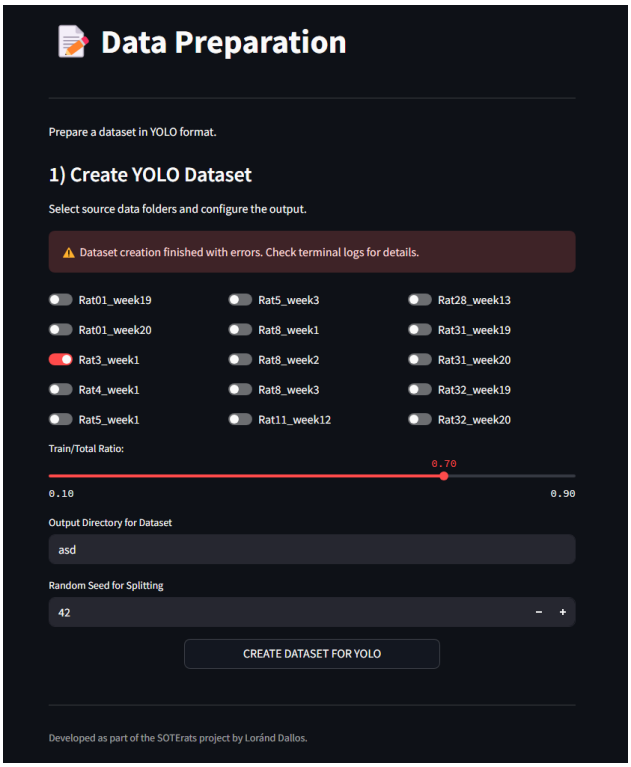
Ha a feldolgozó `pipeline` oldalt videó, vagy kimeneti útvonal megadása nélkül próbáljuk futtatni, a gomb nem jelenik meg ehhez. Ha a feldolgozandó videó túl rövid, a 2.13. ábrán látható módon értesít a program:



2.13. ábra. Hibaüzenet rövid vagy nemlétező videó esetén.

Bármilyen folyamat futása közben az oldalon található összes vezérlő hozzáférése megtagadott a felhasználótól, így nem tudja befolyásolni ezek értékét, ami az oldalról való elnavigáláshoz hasonló eredményhez vezetne.

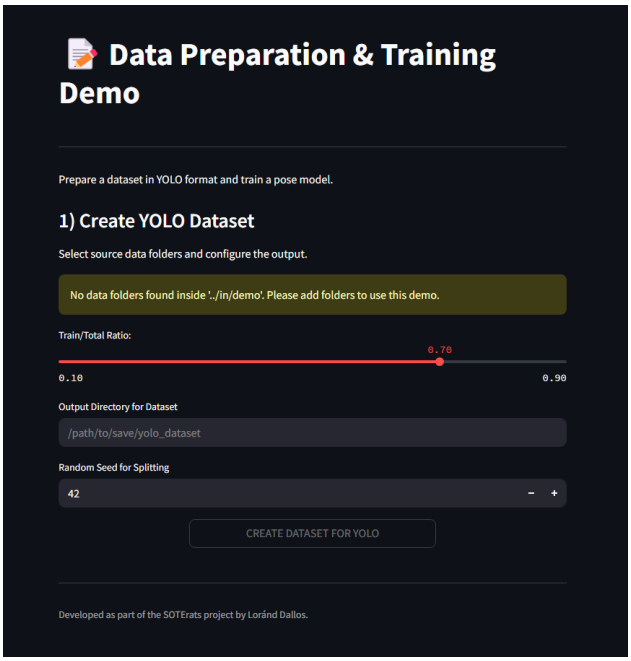
Ha a tanító adathalmaz elkészítésekor kiválasztott alkotóelemek egyike sem tesz eleget a kiszabott szerkezeti megszorításoknak, akkor a 2.14. ábrán látható hibaüzenettel mutatok utat a felhasználónak:



2.14. ábra. Hibaüzenet helytelen szerkezetű tanítóadat esetén.

Ha a megadott könyvtáraknak csak egy része nem elismert szerkezetű, azokat a program átugorja.

Amennyiben a felhasználó még nem töltött fel tanítóadatot, az 2.15. ábrán látható módon figyelmeztet a program.



2.15. ábra. Figyelmeztetés üres tanítóadat könyvtár esetén.

3. fejezet

Fejlesztői dokumentáció

3.1. Tervezés

A programom elkészítéséhez a Python [5] programozási nyelvet választottam. A Python egy jelentősen elterjedt nyelv a tudományos kutatásban, széles körű támogatása kiágazik a képfeldolgozás, mesterséges intelligencia és adatvizualizáció mezeire is. Könnyen olvasható, gyors fejlesztést biztosít.

A szoftver tervezése során arra törekedtem, hogy alegységekre, modulokra bontsam úgy, hogy minden modul az adat feldolgozásának egy fázisáért felelős. Az átláthatóság mellett így több lehetőség van az alkalmazási terület bővítésére, új, akár eltérő jellegű felvételre adaptációra.

A DeepLabCut [4], Sleap [6], MMpose [7], és Ultralytics [8] tanulmányai és alkalmazásai, illetve egy hasonló tematikájú kutatás [9] jelentős kiindulási alapot és útmutatást adtak számomra.

3.2. Fő alkotóelemek és alapfogalmak

A Python nyelvhez alapvetően számos előre megalkotott könyvtár elérhető egy egyszerű `import` utasítással.

Az alábbi szakaszban szeretnék ismertetni pár alapvető elvet, fogalmat, melyek előfordulnak alkalmazásomban.

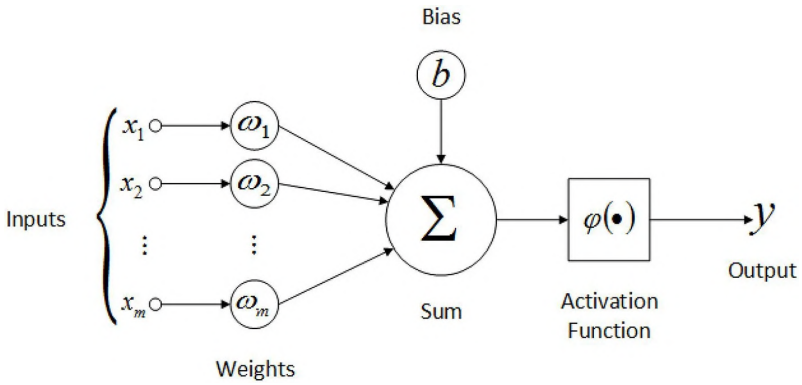
3.2.1. Gépi tanulás

A gépi tanulás a mesterséges intelligencia egyik területe, amelyben a számítógép nem előre definiált algoritmusok alapján old meg feladatokat, hanem az adatokból tanulja meg a megoldásukhoz szükséges mintázatokat és szabályokat.

Főbb típusai összefoglalva a felügyelt tanulás, a felügyeletlen tanulás, és a megerősítéses tanulás. Felügyelt tanulás esetén a tanító adatokhoz címkézés tartozik és célja egy olyan modell létrehozása, amely új, ismeretlen adatok esetén képes megjósolni a címkét. A felügyelet nélküli tanulás esetében nincsenek előre definiált címkék. A modell célja, hogy az adatokból mintázatokat, csoportokat (klasztereket) ismerjen fel. A megerősítéses tanulás esetében egy ágens döntéseket hoz egy környezetben, és visszajelzést (jutalmat vagy büntetést) kap az alapján, hogy döntései mennyire voltak helyesek. Célja a hosszú távon legtöbb jutalom szerzése.

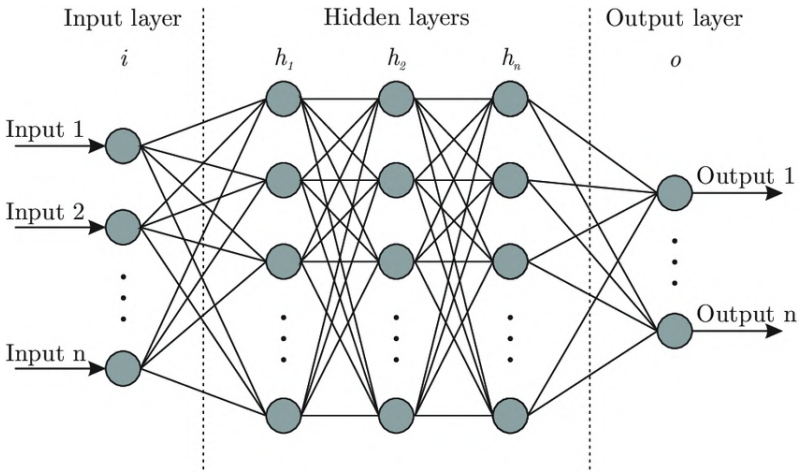
Neurális hálózatok

A neurális hálózatok a gépi tanuláson belül olyan az emberi agy működéséről mintázott matematikai modellek, amely sok, egymással összekapcsolt „neuront” (számítási egységet) tartalmaznak. A mesterséges neuron egy egyszerű számítási egység, amely a biológiai neuron működését modellezi. Minden neuron több bemenetet fogad, amelyekhez súlyok tartoznak. A 3.1. ábra vizuálisan szemlélteti a leírtakat. A neuron először kiszámítja a bemeneti értékek és a hozzájuk tartozó súlyok skalárszorzatát, majd ehhez hozzáad egy eltolást. Az így kapott összeg egy aktivációs függvényen megy keresztül, amely meghatározza a neuron kimenetét. Az aktivációs függvény bevezetése kulcsfontosságú, mivel lehetővé teszi, hogy a hálózat nemlineáris összefüggéseket is megtanuljon, ezzel növelve a modell kifejezőképességét. A nemlinearitás azt jelenti, hogy a bemenetek és a kimenetek közötti kapcsolat nem írható le egy egyszerű lineáris (egyenes vonalú) függvénnyel, hanem komplex mintázatok és összefüggések mentén alakul ki.



3.1. ábra. A mesterséges neuron felépítése [10] alapján.

A neuronok rétegekbe rendezésével, ahol az egyes rétegek kimenetei a következő réteg bemeneteivé válnak, építik fel az ún. neurális hálózatot, biztosítva a hálózaton belüli információáramlást. A tanulási folyamat a hálózatban finomhangolja a súlyokat és eltolásokat, jellemzően a hibavisszaterjesztés (backpropagation) algoritmus és valamilyen optimalizáló eljárás (például gradient descent) segítségével. Ennek eredményeként a hálózat képes egyre pontosabb predikciókat adni a bemenetek alapján. Egy neurális hálózat semantikus ábrját mutatja be a 3.2. ábra.



3.2. ábra. A neurális hálózatok általános felépítése [11] alapján.

Konvolúciós neurális hálózat

A gépi látás olyan tudományterület, amelynek célja, hogy a számítógépek képesek legyenek értelmezni és feldolgozni vizuális információt. Ahogy azt egy informatív weboldalon [12] is megemlítik:

> A gépi látás alapvetően azt szokta jelenteni, amikor képalapú információt (akár videót is) tekintünk bemeneti adatnak, és ezzel az adattal kezdünk

> valamit. Ez a valami lehet pusztán az adatok gyűjtése és kiértékelése, de nagyon gyakran a képet elemezve valamilyen döntési helyzet elé állítunk egy számítógépet, a döntés pedig valamilyen mechanikai megoldást eredményez. Például a parkolóház kijáratában a gép felemeli az autónk előtt a sorompót, ha a rendszámunk kameraképét összevetve a fizetőautomatából érkező információval azt az eredményt kapja, hogy kifizettük a parkolást.

A gépi látásban az objektumdetekció és a pózdetekció gyakori megközelítése a konvolúciós neurális hálók (CNN) alkalmazása.

A CNN-ek alapötlete az emberi látórendszer működésén alapul: ahelyett, hogy minden egyes képpontot külön kezelnének, lokális mintázatokat vizsgálnak, például éleket, sarkokat, textúrákat [13]. Ehhez úgynevezett konvolúciós rétegeket használnak, amelyek során kisebb méretű, tanulható szűrőket csúsztatnak végig a képen, és minden pozícióban kiszámítják a hasonlóságot. A tipikus CNN architektúrák több konvolúciós réteget tartalmaznak, amelyek után aktivációs függvények és lefedési rétegek következnek. Ez a kombináció lehetővé teszi, hogy a hálózat egyre összetettebb jellemzőket ismerjen fel, miközben csökkenti az adat méretét és a számítási igényt. A végső rétegek általában teljesen összekapcsolt neurális rétegek, amelyek az összegyűjtött jellemzőkből előrejelzéseket vagy osztályozást végeznek. A CNN-eket jellemzően felügyelt tanulással és gradiens-alapú hibavisszaterjesztéssel (backpropagation) tanítják, ahol a hálózat előrejelzéseinek hibáját visszavezetik a rétegek súlyainak frissítésére a hiba minimalizálása érdekében.

Az alábbi szakaszban összegyűjtöttem, milyen csomagokat vettem igénybe az alkalmazásom megalkotása érdekében, illetve szeretnék ismertetni pár alapvető elvet, fogalmat, melyek előfordulnak ebben.

3.2.2. Használt csomagok

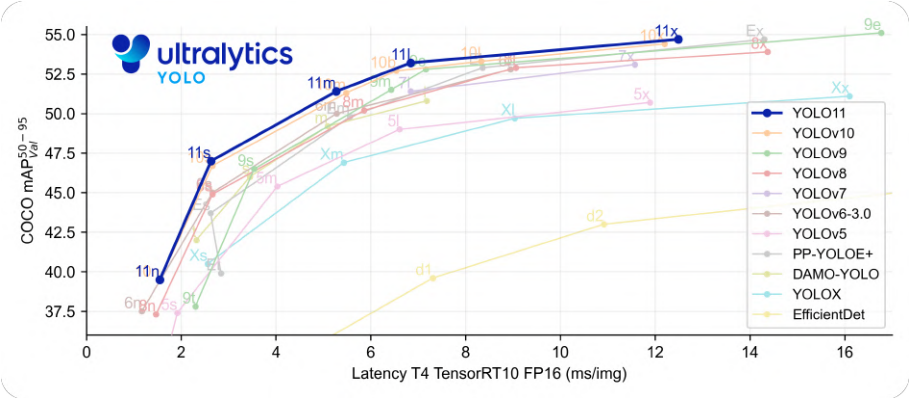
A csomagok a hatékonyság növelését, képfeldolgozást, adatábrázolást, fájlkezelést és matematikai műveletek elvégzését támogatták.
A hatékonyságra kihegyezett csomagok közé tartozik a `numpy` [14], amely tömbalapú numerikus műveletek gyors végrehajtását teszi lehetővé, valamint a `torch` [15], amely a PyTorch mélytanulási keretrendszerrel lehetővé tesz GPU-gyorsított számításokat és neurális hálózatok tanítását.

A hagyományos képfeldolgozási algoritmusokat elsősorban az `opencv` [16] könyvtár segítségével valósítottam meg, amely lehetőséget ad képek beolvasására, feldolgozására és átalakítására. A `PIL` (Pillow) [17] könyvtár szintén képfeldolgozási feladatokat látott el, főként képek betöltése és konvertálása során.
Az eredmények vizualizálására a `matplotlib` [18] szolgált, amellyel grafikonokat és idősorokat ábrázoltam. A saját modellek tanítására az adatot a `pandas` [19] csomaggal elemeztem és kezeltem.
A fájlkezeléshez több modul kombinációját használtam: az `os`, `shutil` és `pathlib` csomagokkal fájlműveleteket és elérési útvonalak kezelését végeztem, míg a `gc` és `tempfile` a memóriakezelést és ideiglenes fájlok létrehozását segítették. A `yaml` és `json` formátumú fájlok beolvasásához és mentéséhez használtam az ugyanolyan nevet viselő könyvtárakat.
A matematikai számításokhoz a `math` csomag biztosította az alapvető függvényeket. Komplexebb műveletekhez, például szűréshez és idősor-transzformációhoz a `scipy` [20], azon belül is a `scipy.signal` almodul nyújtott segítséget. Az `skimage` [21] csomag `morphology` kiterjesztése lehetségessé tette a maszk objektumok középvonalas tengelyeinek kiszámítását. A `random` csomag a véletlenszerűség bevezetésére szolgált, többek között a tanítóadat válogatásakor.
A dolgozat során két modern, mélytanuláson alapuló neurális háló architektúrát alkalmaztam: a YOLOv8 (You Only Look Once) [22] pózbecslés modellt, valamint a Segment Anything Modellt (SAM) [23], mely képalapú szegmentálásra szolgál. A `segment_anything` csomag segítségével a SAM modellt könnyedén integráltam a feldolgozásba. A YOLOv8 verzióját az `Ultralytics` által biztosított Python csomag segítségével használtam, amely magas szintű alkalmazásprogramozási felületet kínál a betanított modellek betöltésére, futtatására és az észlelések eredményeinek kezelésére.

A YOLO modelleknek számos változata létezik. Ezeket két tulajdonság írja le: a méretük (n, s, m, l, vagy x, azaz nano, small, medium, large vagy extra large) és a modell hivatása (pose, seg, cls, stb.). Minél nagyobb egy modell, annál pontosabb eredményt képes adni, viszont annál több erőforrást is igényel a futtatása. A v8 modellcsalád tartalmaz detekciós, szegmentációs, pózdetekciós és klasszifikációs modell architektúrákat. A póz modellek nem csak az objektumokat ismerik fel a képen, hanem a test kulcspontjait is képesek becsülni. Ez különösen hasznos, ha szükséges az információ például egy állat esetében a lokációján kívül, hogy milyen pózban van,

merre néz, vagy hogyan mozog.

A YOLO modell verziók közül a v8 modellt választottam. A hivatalos dokumentációjukban bemutatott referencia mérések és összevetések alapján ígéretesnek látszott (3.1. táblázat), és gyakorlati tapasztalat is ezt igazolta a későbbi, v11 verzióval összevetve: a 3.3. ábrán látható eredmények látványosan feltűntetik, hogy a YOLOv11 túlteljesíti a v8-as kiadást, ám ez a mérés nem pózbecslés feladatán készült. A modellek teljesítményének összehasonlításáról bővebben a 3.5.2. szakaszban található információ.



3.3. ábra. YOLOv11 teljesítményét kiemelő benchmark. [8]

Model	mAP pose 50-95	mAP pose 50
YOLOv8l-pose	67.6	90.0
YOLOv11l-pose	66.1	89.9

3.1. táblázat. YOLOv8 vs YOLOv11 póz becslés teljesítmény.

3.3. Felépítés ismertetése

Az alábbiakban ismertetem az alkalmazás felépítése mögötti gondolatmenetet, először összefoglalva, majd részletesen kifejtve. Az alkalmazásom két fő részre bontható, ezek a következők:

- 3.5 - A modell tanítása
- 3.6 - Az adat feldolgozása

3.3.1. Saját modell tanítása

A modul célja a biológusoktól kapott adatot átalakítani számunkra felhasználható formátumba (3.2. forráskód). Ezt követően az Ultralytics YOLOv8l-pose mo-

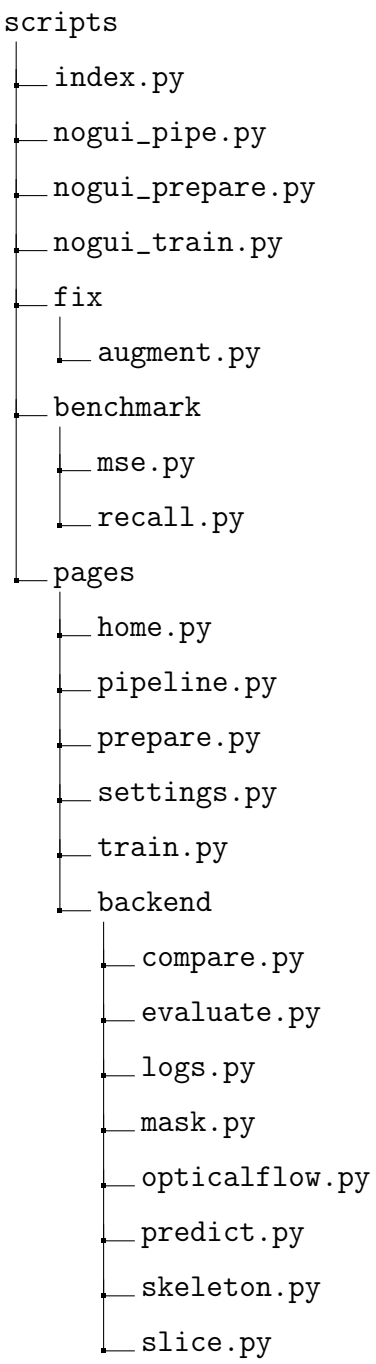
delljét alapul véve a megfelelő formátumba átalakított adattal tovább tanítani ezt. Így lesz egy specializált modell a patkányok kulcspontjainak becslésére.

3.3.2. A feldolgozás

Célja a bemeneti videó filmkockákra darabolása, hogy feldolgozhassam őket a saját YOLO modellemmel; az említett modellel ráilleszteni a kulcspontokat a patkányra; egy SAM modellt felhasználva minden képre egy maszk generálása; a maszkok segítségével azon képek kiszűrése, ahol a patkány nem mozog; szintén a maszkok alapján egy mesterséges csontváz illesztése a maszk alakokra; a lábfej kulcspont becslések javítása az említett csontvázak segítségével; a kijavított adatok összehasonlítása öt létező minta-adatkategóriával, és ennek eredménye alapján az eredeti videóban szereplő egyed osztályozása. Megvalósításom során nevet adok a mozgási kategóriáknak, de a feldolgozás végén számot térítek vissza. A kategóriák: (1) crippled (2) barely moving (3) intermediate (4) almost healthy (5) healthy.

3.4. Szerkezeti felépítés

Ebben a szakaszban szeretném felvázolni az alkalmazás szerkezetét a 3.4. ábra segítségével a későbbi részletezések érthetőbbé tételéért.



3.4. ábra. A projektkönyvtár felépítése.

3.5. A modell tanítása

Alkalmazásom ezen része további két alrészre bomlik.

3.5.1. Az adathalmaz előkészítése

Az alkalmazásom leprogramozása előtt a projekt részeként kipróbáltam hasonló modell-tanító alkalmazásokat (például a bevezetésben említett DeepLabCut, Sleap). Az említettek közül a DeepLabCut (továbbiakban DLC) szoftverét találtam leginkább felhasználóbarátnak. A közreműködő biológusok így az általam ajánlott szoftvert használtak annotálásra, a DeepLabCut annotáló applikációját. Hasznos volt számomra, hogy ez egy '.csv' fájlba menti az elkészített adatokat, így a biológusok könnyen átadhatták nekem a kézzel készített tanítóadatot. Fontos első lépésként az így kapott adatot finomítottam.

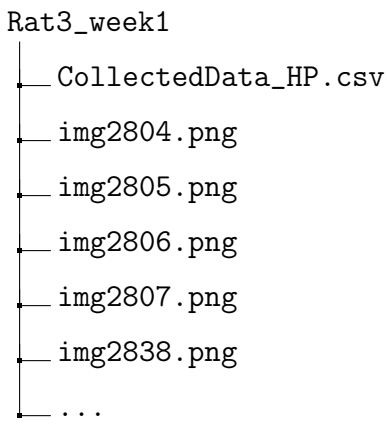
DeepLabCut annotációk

Annotációk készítéséhez telepíteni kell a DLC szoftverét, lehetőleg egy virtuális környezetbe, például egy conda [24] környezetbe. Megnyitás után létre kell hozni egy új projektet tetszőleges beállításokkal, de a felcímkézni kívánt videó útvonalának hozzárendelésével. Következő lépésként kiválasztandó, hogy automatikusan a szoftver által válogatott, vagy manuálisan választott képkockák címkézése a cél. Az utóbbi módszer esetén a projekt `config.yaml` fájljában konfigurálni kell, hogy a videó mekkora részét darabolja fel folytonosan a szoftver.

A beállítások után megkezdhető a képek annotálása, erre külön felületet biztosít a DLC. A felcímkézett adatokat egy átfogó '.csv' fájlba menti. A biológusok ilyen '.csv' összegzéseket, és a fájlokban megemlített képeket biztosították számomra a modell tanítása érdekében. Átfogóbb dokumentáció megtekintése: [25].

Csv beolvasás

A tényleges alkalmazás modell tanító része ennél a fázisnál kezdődik. Az adatot mindig hetekre lebontva kaptam meg. Az adatokat tartalmazó könyvtár formátumát a 3.5. ábrán egy példa-könyvtárral szemléltetem:



3.5. ábra. A patkány-adatot tartalmazó könyvtár.

A 3.5.1. modul egy olyan bemenet mappát vár el, melynek almappái a 3.5. ábrán megfigyelhető szerkezetű mappákból épül fel, illetve az ezekben található '.csv' fájlok a 2.1. forráskódban bemutatott szerkezetet tartják. Amennyiben ettől eltér a bemeneti mappa, a tanító adathalmazt generáló modul nem veszi figyelembe az adott könyvtárat.

A 2.1. forráskód-részletben megfigyelhető az adat eredeti formátuma. Az egységes formájú adat készítése érdekében a bemeneti mappa első '.csv' fájljából kiolvasom az egyedi testrészeket, majd későbbi fájlok esetén ezt alapul véve eldobom azokat, amelyek fejlécei nem egyeznek a megállapított testrészekkel. A kulcspont adat kinyerése alatt folyamatosan figyelem azt is, hogy a testrészek száma által előrejelzett kulcspont mennyiség az adatfájlban be van-e tartva. Ahol nincs annotálva az adott pont, nulla értékkel pótlom. A kulcspont adatokat YOLO formátumúvá téve tárolom, amit a 3.2. forráskódban mutatok be.

A leírtak alapján összegyűjtöm minden '.csv' fájl összes képét és a hozzájuk tartozó kulcspont adatokat egy-egy átfogó kép- illetve adathalmazba. A képeket átnevezem felhasználva a '.csv' fájlban található mappa nevét, így garantáltan minden kép egyedi azonosítóval fog rendelkezni (RatX_WeekY_imgZ.png) és a gyűjtő halmazban nem fogják felülírni egymást.

YOLO adatformátum

A számítógépes látásban, különösen az objektumdetekciós és pózanalízis feladatok során, kulcsfontosságú a képi adatok megfelelő annotálása. Két elterjedt formátum erre az Ultralytics YOLO és a COCO. Dolgozatom keretein belül a YOLO

adatformátumot fogom bemutatni: egy egyszerű, szövegalapú annotációs rendszer, amelyet kifejezetten a YOLO architektúrájú modellek hatékony tanítására optimalizáltak. Minden egyes képhez tartozik egy azonos nevű '.txt' kiterjesztésű fájl.

Ebben a fájlban minden sor egyetlen detektált objektumot reprezentál. A detektált objektum reprezentálása sorfolytonosan: az objektum azonosítója; az objektumot közre záró keret (továbbiakban bounding box) közepének koordinátái, szélessége és magassága. Minden koordináta és távolságot leíró adat a kép méretein normalizált állapotban szerepel, azaz minden érték egy $[0, 1]$ intervallumbeli szám. Ez azért lényeges, mert a modell tanulás alatt újraméretezi a képeket, így a pixelben kimért távolságok és koordináták máshová kerülnének a képen. A normalizálásnak köszönhetően konkrét méret megadása nélkül tárolható egy pozíció a képen. Az adatformátum általános struktúrája megfigyelhető a 3.1. forráskódban:

```
<osztaly_id> <bbox_kozep_x> <bbox_kozep_y> <bbox_szelesseg> <
    bbox_magassag>
```

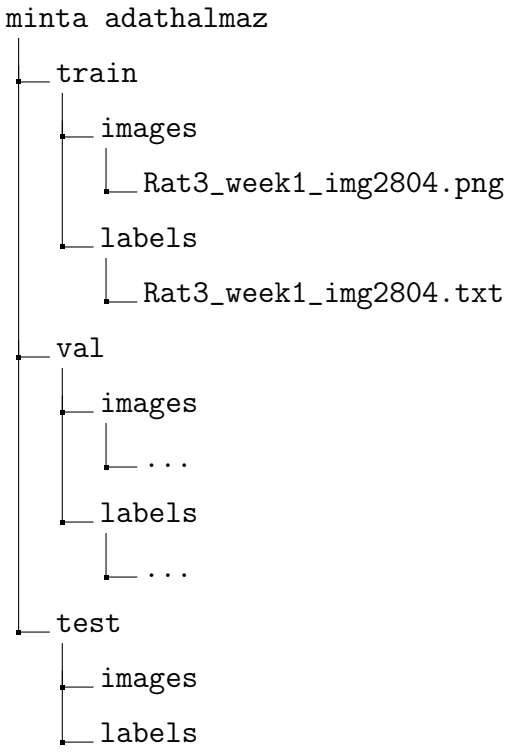
3.1. forráskód. Ultralytics YOLO formátum

A fenti 3.1. forráskód azonban egy általánosabb, objektumfelismerő modellek tanításához szükséges formátum. A testpozíció-követő (pose tracking) modellek formátuma ehhez képest kiegészül az annotálni kívánt kulcspontok koordinátáival, valamint opcionálisan ezek láthatóságával. A láthatóság értéke lehet 0, 1, 2. A nullás érték az jelenti, hogy a kulcspont nincs annotálva, vagy nem látható (pl. takarásban van, rossz minőség). Az egyes értek szerint a kulcspont annotálva van, de nem látható (például emberi becsléssel megadott hely). Végül kettes értékkel jelölik, ha a kulcspont annotálva van és látható a képen (tisztán kivehető, jól látszik). A 3.2. forráskódban tanulmányozható az így kapott felépítés. Programom a láthatóság nélküli megközelítést alkalmazza, mivel a biológusok annotáló eszköze nem támogatja a pontok láthatóságának jelzését.

```
<osztaly_id> <bbox_kozep_x> <bbox_kozep_y> <bbox_szelesseg> <
    bbox_magassag> <px1> <py1> <px2> <py2> ... <pxn> <pyn>
VAGY
<osztaly_id> <bbox_kozep_x> <bbox_kozep_y> <bbox_szelesseg> <
    bbox_magassag> <px1> <py1> <p1-lathatosag> <px2> <py2> <p2-
    lathatosag> ... <pxn> <pyn> <pn-lathatosag>
```

3.2. forráskód. Ultralytics YOLO formátum kulcspontokkal 2 majd 3 dimenziós verzióban.

A fentiekben tárgyaltam az egy képhez tartozó, annotált adatot hordozó '.txt' fájl struktúráját. Szeretném a továbbiakban bemutatni, hogy a rendelkezésünkre álló képeket és szöveges fájlokat milyen elrendezéssel lehetséges tanító adatként átadni egy YOLO-pose modellnek. Egy adathalmazon belül három fő részre osztandók a képek és szövegfájlok, melyeket mindig egy egységként kell kezelni (adott kép és hozzá tartozó szöveges fájl köteles ugyanabban a kategóriában elhelyezkedni). Ezek a részek a `train`, `val` és `test` almappák, melyek további `images` és `labels` almappákban tárolják az adatokat, mint ahogy az a 3.6. ábrán is látható.



3.6. ábra. Adathalmaz felépítése.

A tanító halmaz tartalmazza azokat az adatokat, amelyeken a modell közvetlenül tanul. A tanítási folyamat során a modell ezeken a példákon keresztül ismeri fel a mintázatokat, összefüggéseket a képi információk és a kívánt kimenet között. A hálózat súlyait ezen adatok alapján frissíti iteratív módon. Általában a teljes adathalmaz hetven-nyolcvan százalékát teszi ki.

A validációs halmaz arra szolgál, hogy a tanítási folyamat közben monitorozzuk a modell teljesítményét olyan adatokon, amelyeket a modell nem látott a közvetlen tanulás során. Segítségével észlelhető a túltanulás jelensége: ha a modell a tanító adatokon egyre jobb, de a validációs adatokon romlik a teljesítménye, az túlillesz-

kedésre utal. A tanítást le lehet állítani, amikor a validációs teljesítmény már nem javul tovább egy előre meghatározott korláton belül, ezzel időt és erőforrást spórolva. Ezt nevezik **early stopping**-nak a szakirodalomban. Ez a halmaz a teljes adat tíz-tizenöt százaléka.

A teszt halmazt csak a tanítás befejezése után használjuk, hogy egy végső, objektív értékelést kapjunk a modell általánosító képességéről teljesen ismeretlen adatokon. Ez adja a legreálisabb képet arról, hogyan fog a modell teljesíteni valós körülmények között. Ezt az adatot sem a tanítás, sem a validáció során nem szabad felhasználni, hogy az értékelés torzítatlan legyen. Mérete megegyezik a validációs halmazéval.

Ezen felül egy saját YOLO-pose modell tanításához szükséges egy `data.yaml` konfigurációs fájl is, ami meghatározza a betöltendő adatkészlet elérési útvonalait. A YAML (YAML Ain't Markup Language) formátum egy emberi szemmel jól olvasható adat-szerializációs szabvány, és kulcs-érték párokkal írja le az adatokat. Az általam használt YAML fájl a következőket tartalmazza:

- `path` - Az adathalmaz gyökérkönyvtárának relatív vagy abszolút elérési útja. A train, val, és test utak relatívak ehhez a gyökérhez képest.
- `train` - A tanító mappa elérési útja.
- `val` - A validációs mappa elérési útja.
- `test` - A teszt mappa elérési útja.
- `kpt_shape` - A kulcspontok száma a és a dimenzió mérete b $[a, b]$ formátumban.
- `flip_idx` - Ez egy lista, ami megadja, hogy horizontális tükrözés során melyik kulcspont index melyik másiknak felel meg. Ez lényeges szimmetrikus objektumoknál, mint amilyen egy állat teste. Ha például a 0-ás index a bal szem, az 1-es a jobb szem, akkor a flip_idx listában az nullás helyen 1, az első helyen 0 állna. Ha egy kulcspontnak nincs párja, akkor önmagára hivatkozik az indexe.
- `names` - A különböző felismerhető osztályok neveit tartalmazó szótár.

Ez a felépítés sokkal kevésbé dinamikus esetemben, mint amilyennek elsőre tűnhet. Számos invariáns fennáll a kutatás alatt: A `kpt_shape` helyén $[13, 2]$ áll, ugyanis tizenhárom kulcspontot követek a kísérlet során, és nincs hozzá láthatósággal kapcsolatos adat, ezért kétdimenziós a kulcspont adatom (x,y). A kulcspontok nevei

és index-megfelelői láthatóak a 2.2. táblázatban, melyek szerint a `flip_idx` tartalma is konzisztensen $[1, 0, 2, 3, 4, 5, 6, 10, 11, 12, 7, 8, 9]$, hiszen a fülek és a lábak azok, amiket befolyásolhat egy horizontális tükrözés.

A `names` mező {0: rat} értékkel változatlanul azt képviseli, hogy egyetlen felismerhető osztállyal kell, hogy rendelkezzen a modellem, a patkánnyal. Fontos megemlíteni, hogy ha több osztályom lenne, akkor a sorrendnek meg kellene egyeznie az adatokat hordozó '.txt' fájlokban található sorrenddel. Az egyetlen változó tényező az egész `data.yaml` fájlban a `path`, ugyanis lehetséges más-más hetek adatait összekeverő adathalmazok elkészítése, amit mindig egy új, beállítható útvonalra érdemes menteni. Példának szolgál a 3.3. forráskód, melyen látható egy `data.yaml` teljes tartalma:

```
path: /path/to/datasets/example_dataset
train: train/images
val: val/images
test: test/images
kpt_shape: [13, 2]
flip_idx: [1, 0, 2, 3, 4, 5, 6, 10, 11, 12, 7, 8, 9]
names: {0: rat}
```

3.3. forráskód. YOLO-pose modellt konfiguráló data.yaml.

Az általam írt algoritmus ezeket a szerkezeteket tiszteletben tartva állít elő egy adathalmazt és egy konfigurációs fájlt, feltéve, hogy a bemeneti könyvtár rendelkezésre áll. A komponens teljes kódja megtekinthető a `prepare.py` szkriptben.

3.5.2. Saját modell tanítása

Modellvariánsok

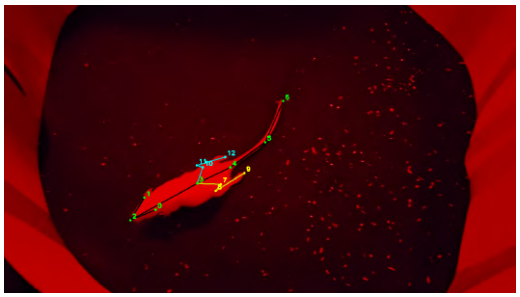
A fejlesztés során a végleges döntésem mellett kipróbáltam a második legmagasabb teljesítménnyel rendelkező modellt pózbecslés feladatán, az újabb kiadású YOLOv11-et.

Továbbá ezen modell típusok pontosságát megpróbáltam fokozni azzal, hogy kivágott képeken tanítom és értékelem ki őket. Ez a módszer lényegesen több elő- és utófeldolgozást igényel. Először felhasználtam egy meglévő és teljes képeken tanult modellt, hogy megbízhatóan megtalálja és kivágja a patkányt az eredeti képről egy Segment Anything Model (SAM, bővebben a 3.6.3. alszakaszban) segítségével. Az új, kisebb képhez kiegészítésképp elmentettem egy tetszőleges adatot, ami segít-

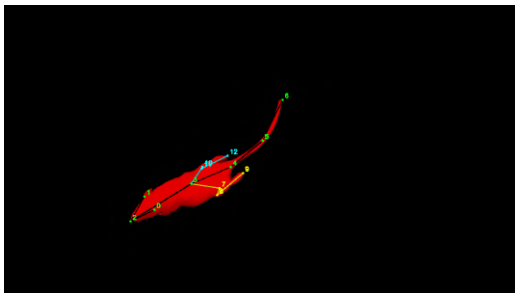
het visszailleszteni a kivágott képdarabot az eredeti képbe. Esetemben ez az adat egy pont koordinátája, amely a kivágás bal felső sarkát jelöli az eredeti képen. A visszaillesztéshez szükséges információk, mint például a kivágott négyszög méretei kinyerhetőek a kivágott kép méreteiből. A kivágott képfeldolgozás utáni visszaillesztése azért szükséges, hogy lehessen videó formátumba összefűzni az előállított adatot, ebben így nem akadályoz a kivágott képek eltérő mérete.

Végső próbálkozásként egy új fajta előfeldolgozás alá vetettem a képeimet, szintén a SAM segítségével. Miután kiszámítok egy maszkot, ami a patkányt jelképezi, az összes ezen kívül álló pixel értékét nullára állítom, ezzel egy sötét képet generálva, ahol csak a patkány látszik. A reményem az volt, hogy a háttér kiszűrésével a modell hatékonyabban megtanulja a patkány sajátos vonásait, pontosabb becsléseket képes majd adni.

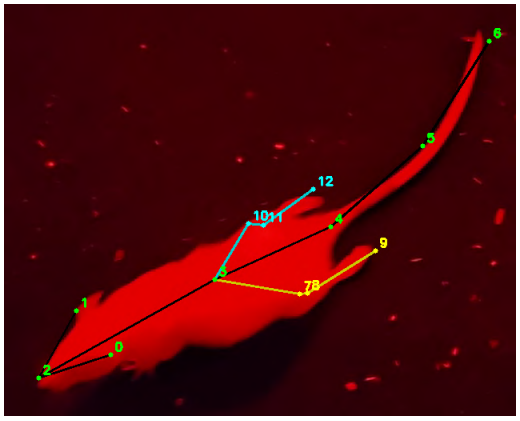
Az említett előfeldolgozásokat ötvöztem is kutatásom során, bár egy meggondolandó hátránya ezen módszereknek, hogy kizárólag olyan adaton képesek működni, mint amilyenen tanultak. Ez azt jelenti, hogy egy sötét képeken tanult modell esetén csak olyan videóra várható el rendes működés, amely előre fel volt dolgozva és minden filmkockája kisötétítve. Egy kivágott képeken tanult modell pedig egy extra lépést igényelt volna: a bemeneti videó feldarabolása után minden képen megkeresni és kivágni a patkányt. A 3.7. ábrán látható egy-egy minta kimenet kép, amelyeken látszik, hogy milyen sajátos eljárásokat igényel a bemenet-párjuk előállítása, de megfigyelhetőek az apró becslési eltérések is.



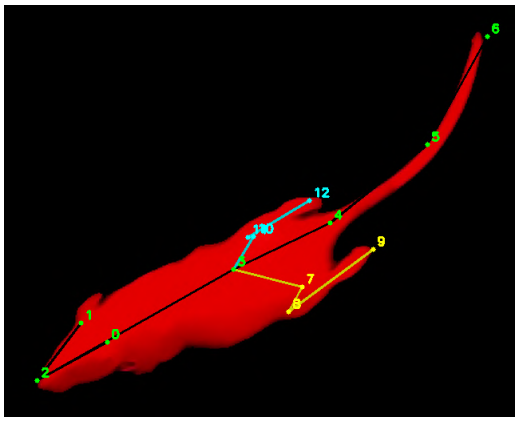
YOLOv8l-pose teljes kép.



YOLOv8l-pose teljes sötétített kép.



YOLOv8l-pose kivágott kép.



YOLOv8l-pose kivágott sötétített kép.

3.7. ábra. YOLOv8l-pose modellvariánsok összehasonlítása.

Augmentációk

A tanítás megkezdése előtt módosítást kell elvégezni az Ultralytics csomagkönyvtárban. A dolgozat írása alatt az általam használt `ultralytics` csomag verziója 8.3.88. A modell tanítását működtető szkriptet kiegészítettem augmentációs módszerekkel, amik alapértelmezetten nem voltak beépítve. A modell általánosító képességét szerettem volna ezáltal is növelni. Ennek érdekében a program fájljai között található **scripts/fix/augment.py** szkripttel felül kell írni a virtuális környezetben található, azonos nevet viselő szriptet. Ennek elérési útvonala: **.envs/venv_yolo_v12/lib/python3.10/site-packages/ultralytics/data/augment.py**. Amennyiben ez a felülírás nem történik meg, előfordulhat, hogy az ugyanazon az adaton tanuló modellek más predikciót fognak elkészíteni az enyémhez képest. A kiegészítésképp bekerülő augmentációk az `ultralytics` által natívan használt `albumentations` csomag további transzformációit foglalják magukba, leírásuk megtekinthető a 3.2. táblázatban, hatásuk pedig

a 3.8. ábrán. Az augmentációk alkalmazása növeli a modell általánosító képességét azáltal, hogy a bemeneti képeket sokféle vizuális variációval látják el. Az augmentációk segítenek, hogy a modell ne ragaszkodjon túl szigorúan a tanító adatok adott jellemzőihez, hanem robusztusan kezelje a természetes varianciát, így pontosabb pozícióbecslést nyújtson ismeretlen környezetekben is (például más típusú kamerával felvett videókon).

Augmentáció	**Leírás**
`A.ChannelDropout`	Egy vagy két színcsatornát véletlenszerűen lenulláz, így a kép bizonyos részei eltűnhetnek, ami növeli a robusztusságot a színinformációk elvesztése ellen.
`A.ColorJitter`	A színeket (fényerő, kontraszt, színtelítettség, árnyalat) véletlenszerűen módosítja, hogy a modell különböző fényviszonyokat jobban kezeljen.
`A.Emboss`	A kép éleit kontúrossá teszi, kiemelve az alakzatokat.
`A.Equalize`	A kép kontrasztját optimalizálja a hisztogram kiegyenlítésével, jobban láthatóvá téve a részleteket.
`A.HueSaturationValue`	Véletlenszerűen módosítja a színárnyalatot, telítettséget és fényességet, utánozva a különböző megvilágítási körülményeket.
`A.ISONoise`	Zajjal terheli meg a képet (ISO-szerű zaj), ami a gyenge fényviszonyokat szimulálja.
`A.InvertImg`	A kép színeit invertálja, teljesen megfordítva a színértékeket.
`A.MultiplicativeNoise`	Véletlenszerű szorzót alkalmaz a pixelekre, csatornánként eltérően, így kis színtorzulásokat hoz létre.
`A.RGBShift`	A piros, zöld és kék csatornák értékeit külön-külön módosítja, finom színtorzításokat okozva.
`A.RandomBrightnessContrast`	Véletlenszerűen változtatja meg a kép fényerejét és kontrasztját, hogy a különféle megvilágítási körülményeket modellezze.
`A.RandomFog`	Köd-effektust ad a képre, csökkentve az élességet és a láthatóságot.
`A.RandomToneCurve`	A kép tónusgörbéjét módosítja, változtatva a színmelegséget és a tónuseloszlást.
`A.Sharpen`	A kép éleit élesebbé teszi, kiemelve a részleteket.
`A.ToSepia`	A képet szépia (meleg barna) árnyalatúvá alakítja, imitálva az öreg vagy meleg tónusú fényképeket.

3.2. táblázat. Alkalmazott képfeldolgozási augmentációk és azok hatásai.

3.8. ábra. Kiegészítő augmentációk egy mintaképen bemutatva.

Tanítás

A pózbecslő modell tanítása egyszerűen megvalósítható az Ultralytics könyvtár beépített függvényével. A 3.4. forráskód példa a YOLO osztály segítségével demonstrálja, hogyan tanítottam a ténylegesen felhasznált modellemet:

```
from ultralytics import YOLO

def train_model(data_yaml_path, epochs, save_path,
    name_of_future_model):
    model = YOLO('yolov8n-pose.pt')

    model.train(
        data=data_yaml_path,
        epochs=epochs,
        imgsz=640,
        save_period=5,
        batch=8,
        degrees=15,
        flipud=0.5,
        augment=True,
        patience=50,
        project=save_path,
        name=name_of_future_model
    )
```

3.4. forráskód. YOLOv8 modell tanítása.

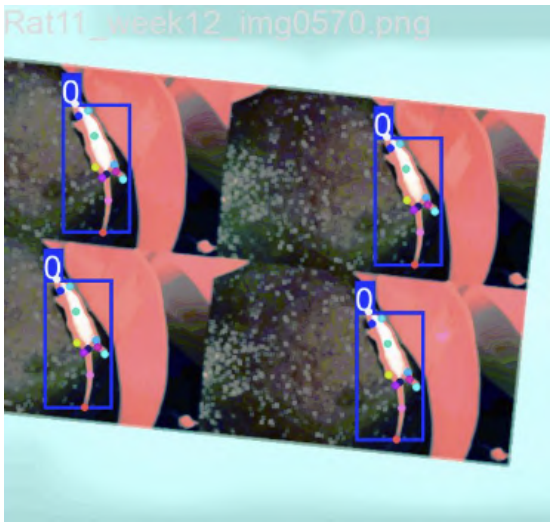
A tanításhoz biztosítani kell a konfigurációs fájlt, amely rámutat a tanulni kívánt adathalmazra. Több fontos paraméter is megadható, amelyekkel finomhangolható a modell teljesítménye:

- `epochs` - az edzési ciklusok száma. Egy ciklus azt jelenti, hogy a modell egyszer végigmegy a teljes tanító adathalmazon. Három ciklus esetén a modell háromszor látja az összes tanító képet. Több epoch általában jobb tanulást jelent, de hosszabb időbe is telik.

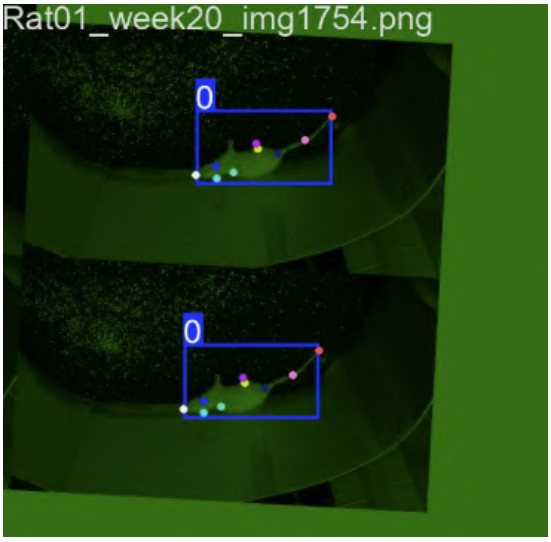
- `imgsz` - a bemeneti képek mérete. A képek egy `imgsz` pixelből álló négyzetté méreteződnek át a tanítás előtt. A YOLO infrastruktúra figyel arra, hogy a képek újraméretezéskor ne torzuljanak, megtartsák eredeti képméret-arányukat. Ha szükséges akkor a képet kipárnázza a megfelelő méret eléréséhez. Ez segíti a

modell egységes feldolgozását, és gyorsítja a számításokat. Nagyobb képméret több részletet jelenthet, de lassabb tanítást.

- `batch` - Ez határozza meg, hogy egyszerre hány képet dolgoz fel a modell egy lépésben. Például egy nyolcas batch azt jelenti, hogy a tanítás minden lépésében nyolc képet vesz figyelembe a súlyfrissítéshez. A nagyobb batch méret gyorsabb lehet, viszont több memóriahasználatot igényel.
- `degrees, flipud, augment` - különféle augmentációs technikák, melyek növelik a modell általánosító képességét.
  - `degrees=15` például maximum 15 fokkal elforgatja a képeket.
  - `flipud=0.5` azt jelenti, hogy 50% eséllyel függőlegesen tükröz egy képet.
  - `augment=True` bekapcsolja az összes elérhető torzítást (pl. fényerő, elforgatás, tükrözés), hogy a modell ne csak az eredeti képeken tanuljon, hanem változatosabb példákon is. Ez segít abban, hogy jobban általánosítson ismeretlen képeken. Szemléltetés a 3.9. ábrán.
- `patience` - Ez az *early stopping* paraméter. Ha ennyi cikluson keresztül nem javul észrevehetően a validációs teljesítmény, akkor a tanítás automatikusan leáll, így időt és erőforrást takarít meg.
- `save_period` - hány epoch-onként mentse el a modell állapotát.



Kontraszt augmentáció.



Színcsatorna augmentáció.

3.9. ábra. Tanító adat augmentációk.

A tanítás során megadott `save_path` mappába számos fájl automatikusan létrejön, az alkalmazásom szempontjából lényegesek a következők:

- `weights` mappa, amely tartalmazza a tanított modell súlyait `.pt` kiterjesztéssel.
- `results.png`, `results.csv`, melyek metrikákat és statisztikákat tartalmazó fájlok.
- `train_batch.jpg` képek, melyeken ellenőrizhető az alkalmazott augmentációk jelentléte.
- `val_batch_labels.jpg` és `val_batch_pred.jpg` képek, ahol összehasonlítható a modell validációs képeken bemutatott teljesítménye a mérési alappal.

A tanítás Streamlit alapú grafikus felületről is vezérelhető, ahol a felhasználó szabadon megadhatja a konfigurációs fájl elérési útvonalát, a tanított modell nevét mint azonosító, a tanítási ciklusok számát, valamint a kimeneti mappát. Egyetlen gombnyomással elindítható a folyamat, amelyet a 3.5. forráskódrészlet-beli függvényhívás reprezentál:

```
1 train.train_model(yaml_input, yolo_epochs, yolo_output, yolo_name)
```

3.5. forráskód. A grafikus felület által hívott függvény saját modell tanítására.

Ez a felhasználóbarát módszer lehetővé teszi, hogy a biológusok is képesek legyenek új YOLOv8-pose alapú modell betanítására, ha azt szükségesnek ítélik és rendelkeznek a korábban meghatározott kiinduló adatszerkezettel.
A komponens teljes kódja megtekinthető a `train.py` szkriptben.

3.6. Az adat feldolgozása

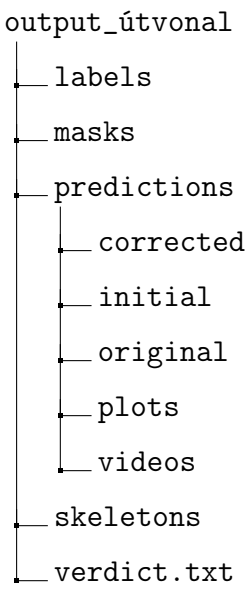
Az alkalmazásom második főkomponense a tényleges bemeneti adat feldolgozásával foglalkozik. Működését magába zárja a `pipeline.py` szkript, melyet a felhasználói felület is alkalmaz. Az ebben szereplő fő végrehajtó függvény neve `execute()`. Ez foglalja magába az alábbi alkomponensek meghívását. A paramétereinek jelentése a 3.3. értelmező táblázatban megtekinthető:

Bemeneti videó, illetve kimeneti könyvtár megadása kötelező; a logikai értékek alapértelmezetten hamisként kerülnek átadásra.

Paraméter név	**Magyarázat**
`input_file`	A bemeneti videó fájl abszolút elérési útvonala.
`output_folder`	A kimeneti könyvtár abszolút elérési útvonala.
`save_initial_predictions`	Logikai érték, amely meghatározza, hogy a YOLO póz modell által becsült eredményeket a program képként elmentse-e az Ultralytics könyvtárban szereplő kirajzolási eljárás szerint.
`save_original_predictions`	Logikai érték, amely meghatározza, hogy a YOLO póz modell által becsült eredményeket a program képként elmentse-e sajátos eljárások szerint.
`save_corrected_predictions`	Logikai érték, amely meghatározza, hogy a YOLO póz modell által becsült eredményeket javítás után a program képként elmentse-e sajátos eljárások szerint.
`save_masks`	Logikai érték, amely meghatározza, hogy a program a SAM modell által becsült patkány maszkokat képként elmentse-e.
`save_skeletons`	Logikai érték, amely meghatározza, hogy a program a patkány maszkján épített csontvázakat képként elmentse-e.
`save_original_vid`	Logikai érték, amely meghatározza, hogy a `save_original_predictions` képeit a program elmentse-e videó formátumban.
`save_corrected_vid`	Logikai érték, amely meghatározza, hogy a `save_corrected_predictions` képeit a program elmentse-e videó formátumban.
`save_plots`	Logikai érték, amely meghatározza, hogy a program elmentse-e a gráfokat, illetve ezek alapjául vett érték listákat, amelyek a patkány mozgásának mértékét szemléltetik és a döntést befolyásolják.

3.3. táblázat. Végrehajtó függvényparaméterek magyarázata.

Kimenetként besorolja a videón szereplő patkányt az ismertetett öt mozgási szint egyikébe, és visszatéríti ezt a döntést, illetve a beállított logikai értékektől függően különböző adatokat ment le mellékhatásként. Attól függően, hogy elég hosszú-e a bemeneti videó (lásd. 3.6.1. szakaszban említett megszorítás), de függetlenül a program lefutásának eredményétől az alábbi fájl-szerkezet létrejön. A benne szereplő mappák tartalma a futtatást megelőző beállításoktól függ.



3.10. ábra. A kimeneti könyvtár felépítése.

A `verdict.txt` kizárólag a sikeres lefutáskor jön létre, ide kerül mentésre a videó végleges osztályozásának neve szöveges formátumban. A továbbiakban alszakaszok keretein belül adok bővebb betekintést a főkomponens alkotóelemeibe.

3.6.1. Bemenet feldarabolása

A feldolgozás első lépése a bemeneti videó képkockákra bontása. Ehhez az OpenCV könyvtárat használom, amely lehetővé teszi a videófájl képkockáinak egyenkénti kiolvasását és azok képként való lementését. A `cv2.VideoCapture` segítségével betöltöm a videót, majd sorban bajárom a képkockákat, és az adatok homogén állapotának megtartása érdekében minden egyes frame-et átméretezve (alapértelmezetten 1280x720-as felbontásra) elmentek PNG formátumban egy ideiglenes mappába. A felbontás beállítása lehetővé teszi a különböző felbontású videók feldolgozását is, ezzel kibővítve azon videók halmazát, amelyekre futtatható az alkalmazásom. A program futási idejét lényegesen befolyásolja a bemeneti képek mérete, így a fent említett eljárás teljesítmény béli növekedést is okozhat.

A képkockák átméretezéséhez a `Lanczos4` interpolációs eljárást alkalmaztam, amely egy nagy pontosságú, szinusz-alapú interpolációs technika. Előnye más elterjedt módszerekkel szemben (`nearest`, `bilinear`, `bicubic`), hogy kiváló élességet és részletmegtartást biztosít az átméretezett képeken. Nagyobb számításigénye ellenére ez a minőség béli többlet különösen előnyös gépi látás feladatoknál, ahol a bemeneti

adatok minősége közvetlen hatással van a modell teljesítményére. A programomban ezt a részletmegtartási előnyt a patkány lábfejei miatt üldöztem.

Bár a YOLO modell képes közvetlenül videófájlokon is futni, az ilyen típusú bemenetek esetén a visszaadott eredmények nem tartalmazzák az eredeti képkockákat. Ezzel szemben, ha képmappán futtatjuk, akkor minden becsléshez társítható az eredeti kép is. Mivel az alkalmazásom további lépései (például maszkolás és vizualizáció) igénylik ezeket az eredeti képeket, ezért a lassabb, de robusztusabb megoldást választottam.

A bemeneti videónak eleget kell tennie a következő megszorításnak: Legalább 250 képkockát kell tartalmaznia, különben a feldolgozás megszakad. A gondolat ezen megszorítás mögött az, hogy egy ennél rövidebb videóból programom egyébként sem képes megbízható becslést adni.

A feldarabolás végeztével a rendszer naplózza a sikeres feldolgozást, és visszaadja az ideiglenes képmappa elérési útját, amelyet a további lépések használnak. A komponens teljes kódja megtekinthető a `slice.py` szkriptben.

3.6.2. Kulcspontok prediktálása

Ez a programrészlet megkapja az előző modulban kinyert képkockákat tartalmazó ideiglenes mappa elérési útvonalát. Az előre betanított modell segítségével becsli a patkány testén elhelyezkedő kulcspontokat. Ez a lépés viszonylag egyszerűen implementálható, mivel az Ultralytics könyvtár egy magas szintű `YOLO` osztályt biztosít.

Predict és Results

Egyetlen `predict` metódus meghívásával futtatható a modell. A dolgom mindössze az eszköz összekötése volt a programom eddigi és ez utáni részeivel a megfelelő paraméterek megadása által:

- `source` - A választott bemeneti adat elérési útvonala. Jelen esetben az eredeti bemeneti videó képkockáit tartalmazó ideiglenes mappa, az előző modul kimenete.
- `conf` - A magabiztossági küszöb (confidence threshold). Csak azokat a detekciókat tartja meg, amelyek bizonyossága meghaladja ezt az értéket. Például `conf=0.5` esetén a modell csak az 50%-nál nagyobb valószínűségű találatokat adja vissza.

- `batch` - Hány képet dolgozzon fel egyszerre. Nagyobb érték gyorsítja a feldolgozást, ha elég memória áll rendelkezésre.
- `device` - A számítási eszköz kiválasztása. Értéke lehet `"cpu"`, `0`, vagy más szám több GPU esetén. `0` esetén a modell a legelső elérhető GPU-t használja. Mivel a programot az NIPG13 szerverén fejlesztettem, rendelkezésemre álltak erős grafikus kártyák, csökkentve a futási időt.
- `verbose` - Logikai érték, igaz érték esetén a futás során részletes naplózási információkat jelenít meg a konzolon, ami hasznos lehet hibakeresésnél vagy fejlesztés közben.

A modell egy `Results` objektumokat tartalmazó listával tér vissza. A `stream` paraméterrel a **predict** metódus működését lehetséges befolyásolni, ugyanis, ha ezt az alapértelmezetten hamis logikai értéket igazként kapja meg, akkor memóriatakarékosan csak egy Results generátorral tér vissza. A listák olyan adatszerkezetek, amelyek az összes elemet előre legenerálják és memóriában tárolják. Ezzel szemben a generátorok lusta kiértékelésűek, vagyis az elemeket csak akkor hozzák létre, amikor ténylegesen szükség van rájuk, például iterálás során. Ez a viselkedés különösen nagy adathalmazok feldolgozásánál előnyös, mivel a generátorok lényegesen kevesebb memóriát igényelnek. Hátrányuk ugyanakkor, hogy nem indexelhetők, és általában csak egyszer járhatók be. Ez a hátrány megakadályozta, hogy a generátorok által nyújtott előnyöket kihasználjam, ugyanis az alkalmazásom moduljai többször igénybe veszik a YOLO póz modell által visszatérített eredményeket. Így az egyszeri bejárás nem volt lehetőség. A `Results` objektumok számos hasznos információt tartalmaznak további elemzéshez. Az alábbi felsorolás bemutatja a programban felhasznált adattagokat:

- `keypoints` – a képen detektált kulcspontok koordinátái és konfidencia értékei.

  Egy `Keypoints` típusú objektum fontos adattagjai:

  - `xy` – egy `(N, K, 2)` méretű `Tensor`, ahol `N` a detektált objektumok száma, `K` a kulcspontok száma. Minden kulcspont (x, y) koordinátákból áll.
  - `cpu()` – a kulcspontokat tartalmazó tenzort átmásolja CPU memóriába. Hasznos, ha GPU-n történt a predikció, de CPU-n szeretnénk feldolgozni az adatot.

  - `numpy()` – a tenzort NumPy tömbbé alakítja, ami kompatibilis sok más feldolgozó könyvtárral (`matplotlib`, `opencv`).

- `boxes` – a patkányokat körülvevő határoló dobozok (bounding boxes).

  Egy `Boxes` típusú objektum fontos adattagjai:

  - `xyxy` – a dobozokat $(x_{min}, y_{min}, x_{max}, y_{max})$ formátumban tartalmazó `Tensor`.
  - `cpu()` – a határoló dobozokat tartalmazó tenzort átmásolja CPU-ra.
  - `numpy()` – a tenzort NumPy tömbbé konvertálja.

- `path` – a bemeneti fájl elérési útvonala, amelyből a predikció készült. Segítségével visszakereshető, melyik képkockához tartozik az adott `Results` objektum.

Az általam használt YOLOv8-pose modell kimenete tehát közvetlenül tartalmazza a kulcspontokat, így nem szükséges további feldolgozás ahhoz, hogy felhasználjuk ezeket a következő lépésekhez. A sajátos modell lecserélése elégedetlenség esetén egyszerű. Új modell tanítása után a 3.6. forráskódban szereplő sorban le kell váltani a `YOLO` osztálynak átadott modell útvonalat.

```
1 model = YOLO("/home/dalloslorand/YOLO_lori/runs/pose/full_v8/
    weights/best.pt")
```

3.6. forráskód. Saját modell specifikálása.

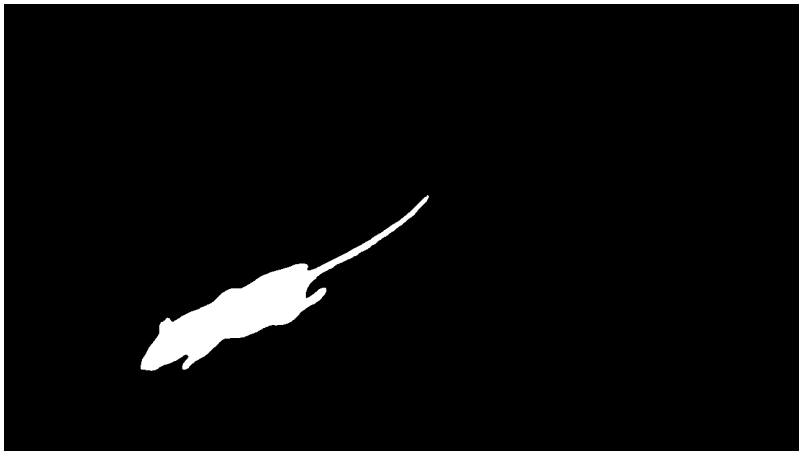
A komponens teljes kódja megtekinthető a `predict.py` szkriptben.

3.6.3. Maszkok előállítása

Maszkok

A modul bevezetése képpen szeretném ismertetni, hogy mi is egy objektum maszkja. Ez egy olyan képi reprezentáció, amely az adott képben található egyes objektumok kontúrjait és terjedelmét jelöli. A maszk minden olyan pixelt tartalmaz, amely egy adott objektumhoz tartozik, miközben a háttér pixeleit nullára állítja (lásd. 3.11. ábra). Ez lehetővé teszi, hogy az objektumok elkülönüljenek a környező területektől. Az objektum maszkokat széles körben használják a képfeldolgozásban

és a számítógépes látásban, például az objektumok felismerésében, szegmentálásában és nyomon követésében. A maszkok segítségével könnyedén izolálhatók és elemezhetők az egyes objektumok, ami elengedhetetlen a pontos képanalízishez és az automatizált döntéshozatalhoz. Programom későbbi fázisaiban a patkány maszkját fogom különféle számításokhoz felhasználni.



3.11. ábra. A patkány maszkja.

Segment Anything Model

Dolgozatomban a patkány sziluettjét képviselő maszkok előállítására a Segment Anything Model (SAM)-t választottam. A SAM egy fejlett képfeldolgozó modell, amelyet a Meta AI szakemberei arra fejlesztettek ki, hogy képes legyen automatikusan szegmentálni bármilyen objektumot a bemeneti képen. A modell célja, hogy minimalizálja az emberi beavatkozás szükségességét, miközben lehetővé teszi a gyors és pontos szegmentálást széleskörű alkalmazásokban. A SAM modell működése egy előre betanított hálózaton alapul, amely képes felismerni és szegmentálni az egyes objektumokat anélkül, hogy különleges annotációra lenne szükség. Más alternatívák, mint a FastSAM vagy az Ultralytics YOLO szegmentáló modelljei ugyan szintén elérhetők és könnyen integrálhatók lettek volna, azonban ezek pontossága, különösen az állatalapú (patkány) képeken, nem bizonyult kellően megbízhatónak a vizsgálatok során.

Hasonlóan az Ultralytics YOLO gazdag modellkészletéhez, SAM modellekből is létezik három verzió, ám ezek közt az eltérés inkább a hálózatok méretére és komplexitására utal. Minél komplexebb egy modell, annál pontosabb eredményeket képes adni, de egyben annál több erőforrást igényel. A modellek nagysági sorrendben:

- SAM-H (huge)
- SAM-L (large)
- SAM-B (base)

A programomat a SAM-H modellel kezdtem el tervezni, ám kénytelen voltam váltani a SAM-B modellre, mert bebizonyosodott, hogy az eredeti választásom impraktikusan lassú. A váltás után feltűnően gyorsabb, de egyben pontatlanabb maszkokat tudtam generálni, ezért utasítási módszeremet is korrigálnom kellett.

Prompt keresés

A modul működése során megkapja a Results objektumok listáját és minden elemre a következő képpen jár el: Felhasználja az előző lépésben becsült gerinc és faroktő pontokat, mint előtér pontok. Egy saját algoritmussal meghatároz egy háttér pontot is, és a három pontot SAM-utasításként felhasználva elkészít egy becsült maszkot a képen látható patkányra.

A háttérpont kiszámításának algoritmusa alapul veszi az adott kép szélességének tíz százalékát, és ilyen távolságra keres egy pontot a becsült gerinc ponttól először jobbra, ha ez a kép határain kívülre esik akkor balra, végül megnézi, hogy ez a pont benne van-e a patkányt közrefogó bounding boxban. Ha benne van, akkor növeli a távolság százalékát öttel, majd újra próbálja. A pontot tíz alkalommal kísérli megtalálni, ha nem sikerül akkor csak a fent említett gerinc és faroktő pontokat használja fel háttér pont nélkül (ezzel rontva a sikeres maszk kiszámítás esélyét). A bemutatott algoritmus megtekinthető az 1. algoritmus ábrán.

A modul a `Results` listán való iterálás közben egy maszkokat tartalmazó listát készít, amit működése végén továbbít a következő programrészletnek. A maszk lista elemei `numpy.ndarray` típusúak, amely egy bináris maszknak felel meg. Mérete megegyezik az eredeti kép méretével, és mindegyik pixel-érték helyén 0 vagy 1 szerepel annak függvényében, hogy az eredeti képen a pozíció alatt ott van-e a patkány, vagy sem. Ha egy képhez nem talál maszkot a SAM, pótlom a listában egy `None` értékkel, amit a további modulok megfelelően kezelnek.

1. algoritmus SAM negatív prompt kiszámítása patkány maszkhoz.

Require: $kp3\_point = (kp3_x, kp3_y)$, $bbox = (x_{min}, y_{min}, x_{max}, y_{max})$, img_h, img_w, $initial\_d\_perc = 10$, $increment\_d\_perc = 5$, $max\_iterations = 10$

$current\_d\_perc \leftarrow initial\_d\_perc$
for $attempt = 1$ to $max\_iterations$ **do**
    $d \leftarrow \text{round}(current\_d\_perc \cdot img\_w)$
    $neg_y \leftarrow kp3_y$
    $neg_x \leftarrow kp3_x + d$
    $point\_valid \leftarrow$ **true**
    if $neg_x < 0$ or $neg_x \geq img_w$ **then**
        $neg_x \leftarrow kp3_x - d$
        if $neg_x < 0$ or $neg_x \geq img_w$ **then**
            $point\_valid \leftarrow$ **false**
        end if
    end if
    if $point\_valid$ **then**
        $inside \leftarrow (x_{min} \leq neg\_x < x_{max})$ **and** $(y_{min} \leq neg\_y < y_{max})$
        if not $inside$ **then**
            return $[\text{round}(neg\_x), \text{round}(neg\_y)]$
        else
            $current\_d\_perc \leftarrow current\_d\_perc + increment\_d\_perc$
        end if
    else
        $current\_d\_perc \leftarrow current\_d\_perc + increment\_d\_perc$
    end if
end for
return *None*

A komponens teljes kódja megtekinthető a `mask.py` szkriptben.

3.6.4. Filmkockák szűrése

A minta bemeneti videók és a programnak szánt jövőbeli éles bemenet videók is sajátos módon kerültek felvételre. A megfigyelés szemszöge nem egy leszögezett, felülnézeti kameráé. A felvételek kézzel fogott kamerával készültek, mely a felülnézetet keveri kis mértékű oldalnézettel is, megfigyelhető remegésekkel. Előfordul, hogy a kamera nézete követi a patkányt, mindig középpontban tartva azt, illetve a beleközelítés és kinagyítás a kamera által látottakba. A felsorolt jelenségek mind olyan tényezők, amelyek zavarhatják a mérés és osztályozás pontosságát, zajt hoznak a rendszerbe. A kamera mozgása általi zajokat a következő alszakaszban bemutatott módszerrel kíséreltem meg kiszűrni. A felvételbe nagyítás által behozott zajt egy későbbi modul során próbáltam mitigálni.

Optikai áramlás

Az optikai áramlás (optical flow) a számítógépes látás egyik alapvető módszere, amely két egymást követő képkocka közötti képpontmozgások becslésére szolgál. A technika célja, hogy meghatározza, hogyan változnak a képen lévő objektumok pozíciói az időben, azaz hogyan "áramlanak" a pixelek egyik képkockáról a másikra.

Az optikai áramlást gyakran alkalmazzák videóalapú elemzések során, különösen olyan esetekben, amikor egy adott objektum (például egy állat vagy jármű, esetemben patkány) mozgását kell elkülöníteni a háttér zajától vagy a kamera mozgásától. Az általam fejlesztett rendszerben az optikai áramlás segít eldönteni, hogy a patkány valóban elmozdult-e a háttérhez képest, így kiszűrhetők azok a képkockák, ahol nincs érdemi mozgás. Ezzel a feldolgozással nem csak az adat minősége és pontossága javul, hanem a soron következő program részletek terhét is könnyítjük, mivel kevesebb filmkockára kell számítást végezniük.

Frame szűrés

A szűrés implementálására egy olyan algoritmust raktam össze, ami végig iterál az összes kép maszkján, közben felépít egy szótár objektumot, melynek kulcsai az éppen vizsgált kép elérési útvonala (ezt a YOLO pózmodell becsléseiből nyerem ki), értékei pedig "discard" vagy "keep", annak függvényében, hogy hogyan döntött az algoritmus.

A szűrés fő feltételéül vett logikai érték összehasonlítja a patkány-maszk optikai áramlását a háttérével. Számolok egy eltérést a kettő között, ami ha nagyobb, mint egy általam definiált küszöbérték, akkor megtartom az adott képkockákat (az áramlás mindig két frame között számolható). Az optikai áramlás számításához a `cv2.calcOpticalFlowFarneback` függvényt használom, amely a Farneback-algoritmust [26] implementálja.

Ezt egy állapotgép megoldással valósítom meg (2. algoritmus), amely két állapot között vált: `rat_moving` és `rat_still`. Az állapotváltásokat egy puffer és egy toleranciaszámláló szabályozza: az algoritmus nem azonnal reagál egyetlen eltérésre, hanem csak akkor, ha a mintázat huzamosabb ideig fennáll. Ez segít kiszűrni a véletlenszerű vagy irreleváns fluktuációkat a mozgásmintákban. A puffer ideiglenesen tárolja az "ellentmondásos" képkockákat, amelyeket aztán visszamenőlegesen megtart vagy elvet, attól függően, hogy milyen irányba dől el az állapotváltozás. Ez

a megközelítés megbízhatóbb és pontosabb döntéshozatalt eredményez, különösen zajos vagy gyenge minőségű videók esetén.

A döntéseket tartalmazó szótár megalkotását követően végig iterálok rajta, és új YOLO-becslés, illetve maszk listákat hozok létre, melyek csak azokat az elemeket tartalmazzák az eredeti listákból, ahol a döntés szótárban az elemhez tartozó kép értéke "keep". Végül ezt a két megszűrt listát téríti vissza a modul, és használja fel a következő, csontvázakat építő modell.
A komponens teljes kódja megtekinthető az `opticalflow.py` szkriptben.

2. algoritmus Állapotgép alapú döntés optikai áramlás alapján.

Require: $Preds$, $Masks$, T_{flow}

$BUF\_SIZE \leftarrow 25$, $TOL \leftarrow 5$, $Decisions \leftarrow \{\}$, $Buffer \leftarrow []$, $TolCount \leftarrow 0$
$State \leftarrow$ `"rat_still"`, $PrevMask \leftarrow None$
for all *path in Preds* ; $i = 0$ **to** $|Preds| - 1$ **do**
    $Mask \leftarrow Masks[i]$
    if $Mask = None$ **then**
        $Decisions[path] \leftarrow$ `"discard"`, puffer törlés, $TolCount \leftarrow 0$, $PrevMask \leftarrow None$
        continue
    end if
    if $PrevMask \neq None$ **then**
        rat flow, *bg flow* kiszámítása
        $Diff \leftarrow$ *rat flow* - *bg flow*
        if $State =$ `"rat_moving"` **then**
            $Decisions[path] \leftarrow$ `"keep"`
            if $Diff \leq T_{flow}$ **then**
                pufferbe ($path$, `"discard"`), $TolCount \leftarrow 0$
            else
                pufferbe ($path$, `"discard"`), $TolCount$++
                if $TolCount > TOL$ **then** puffer törlés, $TolCount \leftarrow 0$
                end if
            end if
            if puffer megtelt **then**
                minden pufferelt kép állapota visszamenőleg legyen `"discard"`, $State \leftarrow$ `"rat_still"`, puffer ürítés
            end if
        else if $State =$ `"rat_still"` **then**
            $Decisions[path] \leftarrow$ `"discard"`
            if $Diff > T_{flow}$ **then**
                pufferbe ($path$, `"keep"`), $TolCount \leftarrow 0$
            else
                pufferbe ($path$, `"keep"`), $TolCount$++
                if $TolCount > TOL$ **then** puffer törlés, $TolCount \leftarrow 0$
                end if
            end if
            if puffer megtelt **then**
                minden pufferelt kép állapota visszamenőleg legyen `"keep"`, $State \leftarrow$ `"rat_moving"`, puffer törlés
            end if
        end if
    else
        $Decisions[path] \leftarrow$ `"discard"`
    end if
    $PrevMask \leftarrow Mask$
end for
if puffer nem üres **then**
    Ha `"rat_moving"` $\Rightarrow$ minden `"discard"`, különben `"keep"`
end if
return *Decisions*

3.6.5. Lábujj kulcspontok javítása szintetikus csontvázakkal

A patkányok lábujjainak detektálása különösen nagy kihívást jelent a gyors mozgásokkal és elmosódásokkal terhelt felvételeken. A YOLO modell kulcspontbecslése az ilyen nehezen kivehető testrészek esetében nem mindig megbízható, gyakran javítható pozíciókat ad vissza. A probléma kezelésére egy csontváz-alapú korrekciós lépést vezettem be, amely a medial axis (középvonalas tengely) alapján próbálja pontosítani a lábujjak helyét.

A medial axis egy bináris alakzat (patkány maszk) belső, topológiai gerincét jelenti: olyan pixelek összessége, amelyek egyenlő távolságra helyezkednek el a forma legalább két különböző határpontjától. A bináris maszkból kiindulva a `skimage.morphology.skeletonize` függvény segítségével állítom elő a medial axist, amely a patkány testének középvonalát reprezentálja egy vékony, egy pixeles szélességű formában.

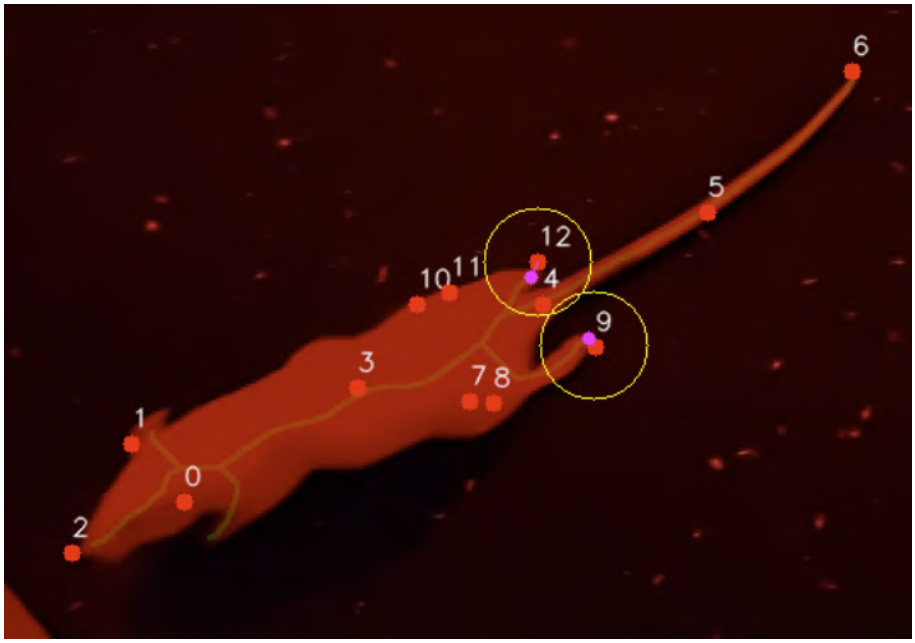
A modell által becsült eredeti 9-es és 12-es indexű lábujj pontok javítása során az algoritmus megkeresi a hozzájuk legközelebb eső végpontokat a generált csontvázon (4. algoritmus). Ehhez először kiszámítok a patkány bounding box méreteiből egy dinamikus keresési rádiuszt, amelyet a doboz átlójának egy rögzített százalékos értékéből származtatok (3. algoritmus). Ez a megközelítés lehetővé teszi, hogy különböző méretű vagy különböző távolságból rögzített patkányok esetében is arányosan alkalmazkodó keresési tartományt használjak.

Ha a lábujjpont középpontú, dinamikusan számított sugarú körön belül található a csontvázon végpont, akkor ezt, vagy több találat esetén ezek átlagát választom ki új kulcspontként. Így az eredeti becslések felülírásra kerülnek egy topológiai alapon valószínűbb pozícióval, ami várhatóan javítja az egész rendszer pontosságát. A folyamat opcionálisan vizualizálható is, ahol a különböző kulcspontok, csontvázak és döntési körök különböző színekkel jelölve kerülnek rá az eredeti képekre. Egy példa vizualizáció megtekinthető a 3.12. ábrán.

Ebben a fázisban tehát az előző modul által megszűrt maszk listából először kiszámítok egy csontvázakat tartalmazó listát, majd ezerre futtatom a javító algoritmusomat, végül további elemzésre előállítok egy szótárat, amiben a kulcsok az eredeti képek elérési útvonalai, az értékek pedig az összes YOLO póz által becsült kulcspont koordinátája az adott képre, de a javított lábfej koordinátákkal. A módszer célja nem a teljes kulcspont-rendszer újrateremtése, hanem egy célzott, kis léptékű

korrekció, amely a lábujjakon segít pontosabb eredményeket elérni, ezzel támogatva a későbbi mozgáselemző szakaszokat.

A komponens teljes kódja megtekinthető a `skeleton.py` szkriptben.



3.12. ábra. A patkány bináris maszkjából származtatott csontváz és kulcspont javítási vizsgálat vizualizációja.

3. algoritmus Keresési sugár meghatározása a határoló doboz alapján.

Require: P_{result} (predikciós eredmény objektum), $R_{default}$ (alapértelmezett sugár), S_{factor} (skálázási faktor)

$bbox \leftarrow P_{result}$.boxes.xywh[0] ▷ Határoló doboz [cx, cy, w, h] lekérése
if P_{result} tartalmaz határoló dobozt **és** $bbox$ nem üres **then**
  $w \leftarrow bbox[2]$ ▷ Szélesség
  $h \leftarrow bbox[3]$ ▷ Magasság
  $diagonal \leftarrow \sqrt{w^2 + h^2}$
  $calculated\_radius \leftarrow \text{round}(diagonal \cdot S_{factor})$
  $final\_radius \leftarrow \max(5, calculated\_radius)$ ▷ Minimum 5 pixel sugár biztosítása
  return $final\_radius$
else
  return $R_{default}$
end if

4. algoritmus Lábujj korrekció közeli csontváz végpontok alapján.

Require: $ToeCoord = (t_x, t_y)$ (aktuális lábujj koordináta), $SearchRadius = R$, $Endpoints = \{e_1, e_2, ..., e_m\}$ (csontváz végpontok listája)

if $t_x = 0$ **and** $t_y = 0$ **then** ▷ Érvénytelen bemeneti lábujj koordináta
  return $None$, [] ▷ Nincs korrekció, üres lista
end if
if $Endpoints$ üres **then** ▷ Nincsenek csontváz végpontok
  return $None$, [] ▷ Nincs korrekció, üres lista
end if
$NearbyPoints \leftarrow []$ ▷ Lista a közeli pontok tárolására
for all végpont $e = (e_x, e_y)$ **in** $Endpoints$ **do**
  $distance \leftarrow \sqrt{(e_x - t_x)^2 + (e_y - t_y)^2}$ ▷ Euklideszi távolság
  if $distance \leq R$ **then**
    Berak e a $NearbyPoints$ listába
  end if
end for
$ChosenPoint \leftarrow None$
if $NearbyPoints$ üres **then** ▷ Nem található közeli végpont
  return $None$, $NearbyPoints$ ▷ NearbyPoints itt üres
else if $SizeOf(NearbyPoints) = 1$ **then** ▷ Pontosan egy közeli végpont
  $ChosenPoint \leftarrow NearbyPoints[0]$ ▷ Az egyetlen pont kiválasztása
else ▷ Több közeli végpont
  $sum\_x \leftarrow 0$, $sum\_y \leftarrow 0$
  for all pont $p = (p_x, p_y)$ **in** $NearbyPoints$ **do**
    $sum\_x \leftarrow sum\_x + p_x$
    $sum\_y \leftarrow sum\_y + p_y$
  end for
  $avg\_x \leftarrow \text{round}(sum\_x / SizeOf(NearbyPoints))$
  $avg\_y \leftarrow \text{round}(sum\_y / SizeOf(NearbyPoints))$
  $ChosenPoint \leftarrow (avg\_x, avg\_y)$ ▷ Az átlagpont kiválasztása
end if
return $ChosenPoint$, $NearbyPoints$

3.6.6. Adat analízis

Mivel ez egy viszonylag összetettebb modul, ezért működését alszakaszokra bontva fogom szemléltetni. A szokásos rend szerint az előző programrészlet kimenetét kapja meg, mint bemenet (kijavított koordinátákat tartalmazó szótár). Elemzés után kategorizálja a videót a dolgozat bevezetésében ismertetett skálán.

Kijavított címkék

A bemeneti szótár által tartalmazott kulcspontokat a program menti egy köztes állapotba, ami a fejlesztés során hasznos kiindulási állapot a további elemzéseknek. A

lementés folyamata úgy történik, hogy végig iterál a szótáron és a kép nevét használva létrehoz egy azonos névvel rendelkező szöveges fájlt, melybe YOLO formátumban írja az adatokat, figyelve arra, hogy az adatok semmi alapján nem normalizáltak, pixel értékeket jelképeznek. A szöveges fájlokból visszatölti csak a kulcspont adatokat egy listába, melynek eleme egy `(x, y)` formátumban pixel-koordinátákat tartalmazó allista.

Idősorok

Az idősor olyan adatsorozat, amelyben az adatok időrendben egymást követve jelennek meg. Az idősoros adatelemzés célja, hogy feltárja az időben zajló változásokat, mintázatokat.

A koordinátákat tartalmazó listából kiolvasom a két láb adatait, majd minden lábra két-két idősort hozok létre. Dolgozatomban az idősorokat a patkány hátsó lábainak kulcspontjai alapján definiálom. Minden egyes képkockához kiszámítom a lábfej és a comb közötti távolságot, így az idősor egy-egy eleme eme távolság változását szemlélteti az idő telésével. Ez a mérőszám jól jellemzi a lábak mozgását, illetve annak amplitúdóját, és alapjául szolgál a viselkedés kategorizálásának is. Egy adott lábhoz létrehozom tehát ezt a távolságot szemléltető idősort pixel értékekben, illetve ugyanezt a távolságot normalizált állapotban szemléltető idősort is. A viselkedés elemzéséhez a normalizált idősort használjuk fel, a normalizálás folyamatát a későbbi alszakaszban kifejtem.

Az idősorokat `ndarray` típussal generálom, `double` értékeket tartalmaznak. Hasonlóan a javított címkékhez, itt is lementem, majd visszaolvasom őket a memóriába. A lementett fájl `.npy` kiterjesztésű, ami a `NumPy` standard bináris fájl formátuma és lehetővé teszi a későbbi gyors visszatöltést.

Normalizálás

A nyers (pixel) idősorozatok a lábak comb és lábujjak között mért euklideszi távolságokat tartalmazzák minden egyes képkockára. Mivel ezek az értékek abszolút távolságot jelentenek a képen belüli koordinátarendszerben, ezért befolyásolja a távolságot az, ha például közelebbről látszik a patkány a felvételen (a távolság egyértelműen nagyobb). A jelenség kiszűrése érdekében szükséges az adatok test-

méret alapján történő normalizálása, hogy összehasonlíthatóvá váljanak különböző távolságból felvett patkányok is.

A normalizálás referenciahosszként a gerinc és a faroktő közötti távolságot használja. Ez a testhossz egy viszonylag stabil érték, ami arányos a testmérettel. Az egyes képkockákban mért lábtávolságokat ezzel az értékkel osztjuk el, így kapjuk meg az arányos távolságokat.

Ez a módszer biztosítja, hogy az elemzés független legyen a kamera nagyítástól vagy a patkány testméretétől, és az így kapott idősorozat a mozgás mintázatát reprezentálja, nem az abszolút méreteket.

Z-score normalizálás és DTW

A **Z-score normalizálás** egy olyan skálázási eljárás, amely a nyers értékeket a saját eloszlásuk átlagától és szórásától függőleges viszonyszámokká alakítja. Minden egyes adatpontból kivonjuk az adott változó átlagát, majd ezt elosztjuk annak szórásával, így az eredmény nulla átlagú és egységnyi szórású eloszlás lesz. Ezáltal az egyes jellemzők összevethetővé válnak, függetlenül az eredeti mértékegységüktől vagy nagyságrendüktől.

A **DTW** (Dynamic Time Warping) pedig egy dinamikus programozáson alapuló algoritmus, amely két időfüggvény (jelen esetben lábmozgás-idősorok) közötti leghatékonyabb „illesztést” keresi. A DTW rugalmasan kezeli az időeltolódásokat és sebességkülönbségeket: a görbék hasonlóságát úgy méri, hogy lokálisan nyújtja vagy tömöríti a sorokat, hogy a lehető legkisebb torzulási költséget érje el.

Az én kontextusomban azonban a rágcsálók lábmozgásának jellegzetességei túl finom és zajos mintázatot mutattak: a különböző felvételekben a mozgás amplitúdója és frekvenciája erősen variált, és a zajos adatpontok a DTW útvonalillesztés során nem biztosítottak elég nagy differenciát a különböző egészségi állapotú patkányok között. Még a Z-score normalizálás sem csökkentette eléggé a belső szórást, így a DTW alapján kapott távolságmátrix nem vezetett megbízható klaszterezéshez vagy osztályozáshoz.

Továbbá az előre elkészített, “black-box” jellegű csomagok belső működését nem tekintettem eléggé átláthatónak, ezért olyan feldolgozási eljárást terveztem és valósítottam meg, amelynek minden lépését saját kezűleg építettem fel. Ez lehetővé tette, hogy biztos legyek abban, hogy minden számítási fázis megfelel a kísérleti követelményeknek.

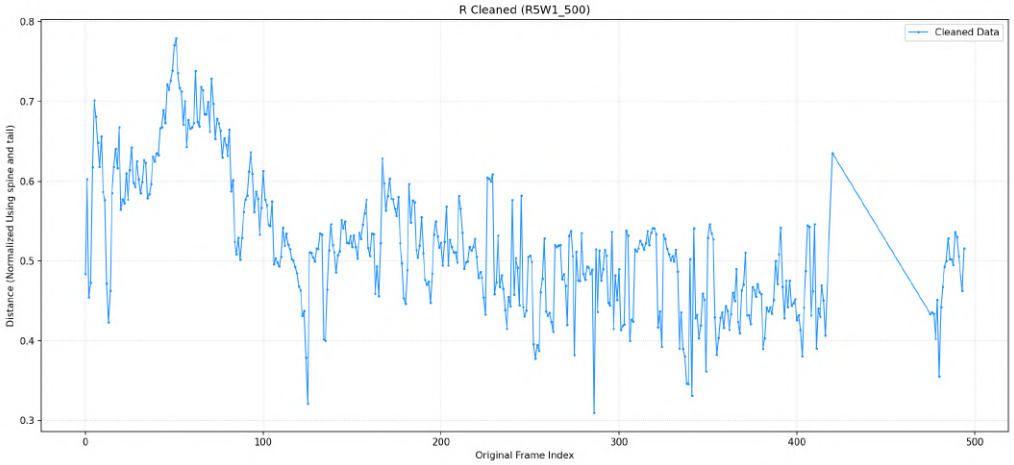
Idősor jellemzők

Az osztályozás megvalósításához szükségem van az idősorokban megfigyelt tulajdonságokra (features), ami alapján elkülöníthető egy egészséges patkány mozgása által generált idősor egy béna patkányétól. Logikus gondolat az amplitúdó alapú megkülönböztetés, hiszen egy egészséges patkány lábait elemezve szinte biztos, hogy a távolság változása a comb és a lábfej között nagyobb tartományban történik, mint egy béna patkány esetében.

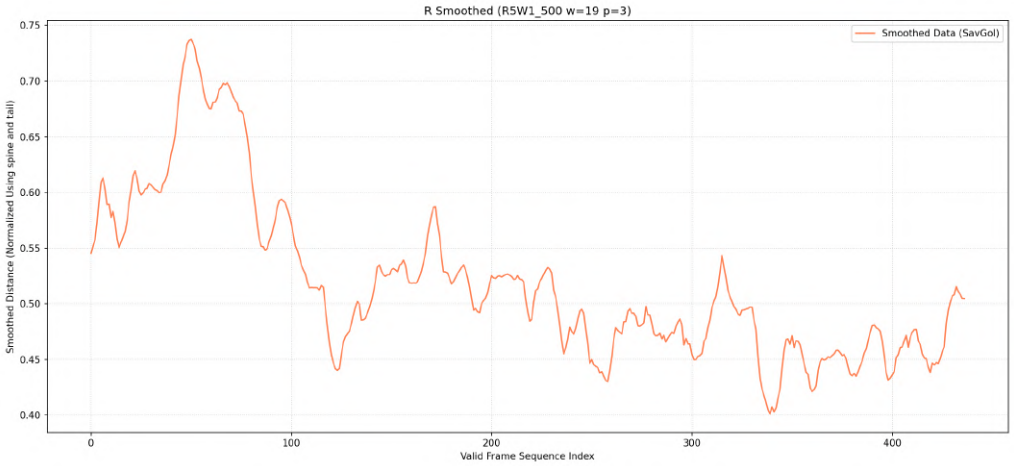
A jellemzők kinyerése előtt az idősorok átesnek egy előfeldolgozáson. Első lépésként eldobom az érvénytelen értékeket (`np.nan`), melyek azt jelképezik, hogy a normalizált érték valamilyen oknál fogva nem került kiszámolásra az adott képkockában, például hiányzó kulcspontok miatt. Második lépésként a sűrű értékeket tartalmazó idősorokra alkalmazok egy matematikai transzformációt [27], hogy tompább, simább idősorokat kapjunk. Harmadik lépésként megkeresem a simított idősorokban szereplő csúcsok és völgyek indexeit, melyek lokális maximum, illetve minimum értékeket képviselnek. Negyedik lépésként az említett indexekkel kiszámolom az idősorok átlagos amplitúdóját és ezek frekvenciáit. Utolsó lépésként az egy videóhoz tartozó két láb jellemzőit átlagolom, így egy videót mindig két jellemző ír le, nem négy.

Grafikonok

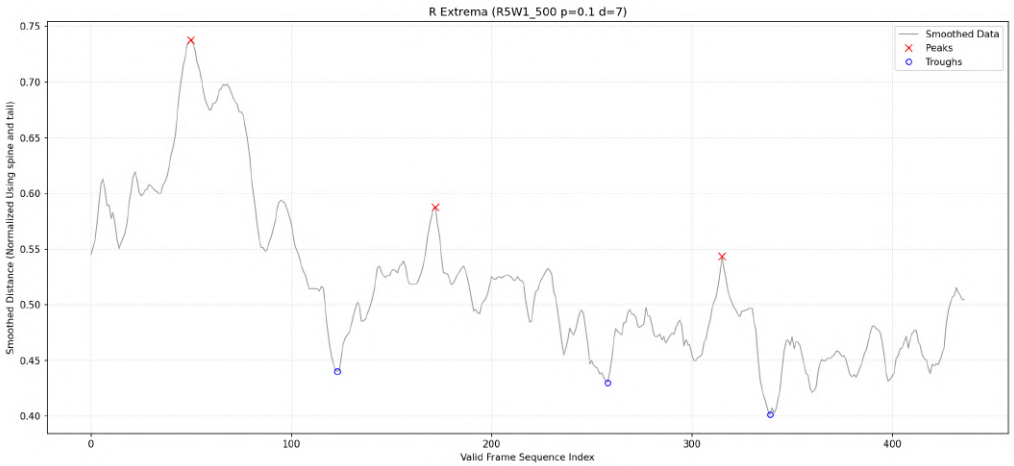
Az idősorok előbb felsorolt sűrített, simított és szélsőérték-jelölt állapotokban tetszőlegesen lementhetőek, amennyiben a program futása után a felhasználó elemezné a folyamatot. A felhasználói felületen a végső osztályozás mellett kirajzolásra kerül a bemeneti videó által generált sima idősor mindkét lábra, valamint ugyanezek a program által választott kategória mintavideójára is. Szemléltetés céljából az 3.13a., 3.13b. és 3.13c. ábrákon megfigyelhető egy idősor a különféle állapotokban.



(a) Szűrt idősor.



(b) Simított idősor.



(c) Lokális szélsőértékekkel.

3.13. ábra. Egy mintavideó jobb hátsó láb mozgásának idősorai különböző feldolgozási lépések után.

Összehasonlítás

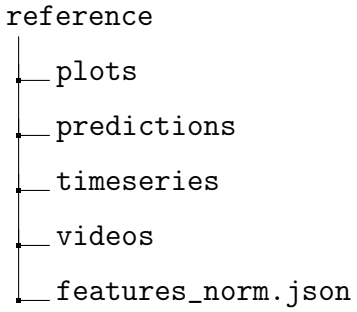
Az éles alkalmazás mindig csak a bemeneti videóra számítja ki az idősorokat és az ezekhez tartozó ábrákat. A referencia osztályok adatait lementett állapotból tölti vissza az alkalmazás futtatáskor. Ezek a 3.14. ábrán mellékelt fájlstruktúrában találhatóak. Json fájlokban tároljuk a fő jellemzőit egy adott osztálynak, tartalmuk az 3.7. forráskódban szemléltetett. A lementett állapotok kötelezően a legfrissebb módszer szerinti idősor adatokat tartalmazzák, ami fejlesztés alatt változhatott, vagy továbbfejlesztésnél ismét változhat.

```
    "crippled": {
        "category": "crippled",
        "amplitude": 0.26510135904167714,
        "frequency": 0.007281553398058253
    },
    "barely moving": {
        "category": "barely moving",
        "amplitude": 0.22515096615898295,
        "frequency": 0.010095389507154214
    },
    "intermediate": {
        "category": "intermediate",
        "amplitude": 0.2697913547147352,
        "frequency": 0.009966777408637873
    },
    "almost healthy": {
        "category": "almost healthy",
        "amplitude": 0.24277992233672452,
        "frequency": 0.011629546095291796
    },
    "healthy": {
        "category": "healthy",
        "amplitude": 0.24066798243544263,
        "frequency": 0.0066566794710244195
    }
```

3.7. forráskód. Referencia kategóriák amplitúdó- és frekvenciaértékeinek JSON struktúrája.

Referencia adatok cserélése

Ha változik az adatok elemzésének módja, kötelező módon le kell generálni a referencia adatokat az új eljárás szerint az igazságos összehasonlítás érdekében. Ezt a fejlesztő megteheti a következő módon:
A 3.14. ábrán látható `plots`, `predictions`, `timeseries` könyvtárak tartalmát és `features_norm.json` fájlt készenléti tartalék készítése után törölje. Ha a referencia alapjául vett videókat is cserélné, a `videos` mappában teheti ezt, de videó fájlok elnevezései továbbra is feleljenek meg a 3.3.2. alszakaszban felállított rendszernek (adott kategóriára vonatkozó videó a kategória nevét viselje). Ezek után futtassa az `scripts/pages/pipeline.py` szkript `execute_reference()` függvényét. Ez lehetséges konzolból interaktív python segítségével, vagy a szkript átírásával. Ez létrehozza a szükséges grafikon adatokat és '.json' fájlt.
A komponens teljes kódja megtekinthető az `evaluate.py`, illetve `compare.py` szkriptekben.



3.14. ábra. A referencia adatok felépítése.

3.6.7. Kimenet összegzése

A program által visszaadott tényleges kimenet egy szám, ami felveszi az értékét annak az osztálynak, amibe besoroltuk a bemeneti videóban szereplő patkányt (1-5). A videó osztályozását mentem szöveges formában a jövőbeli visszakeresés megkönnyítése érdekében. Ezt a számot a 3.6.6. szakasz adja vissza, de a program futtatása során megadott beállításoktól függően mellékhatásként a következőkben bemutatott adatok is mentésre kerülhetnek mindazok mellett, amiket a program leírásában korábban ismertettem:

Kirajzolások

A becsült kulcspontok egy saját stílusban szemléltethetőek a képeken, ilyen módon is ellenőrizhető a modell pontossága; kikutatható egy észlelt anomália oka. A kirajzolás stílusa látható a 3.7. ábrán mellékelt képeken, de szóban is összefoglalva: A patkányra felkerülnek a becsült kulcspontok eredeti vagy javított koordinátával (mindkettő verzió kirajzolására van lehetőség), a sorszámuk kíséri őket fölöttük jobbra eltolva. A szomszédos kulcspontokat saját választás alapján összekötöm, így látványosabb a kulcspontok közötti távolság változása. A csontváz (nem összekeverendő a 3.6.5. szakaszban kiszámított középvonalas tengellyel) eredeti színe fekete, a kulcspontoké pedig zöld, de ezt befolyásolhatja az, hogy melyik testrészhez tartozik az adott kulcspont. A jobb láb pontjai, és az ezeket összekötő vonalak világos kék, a bal láb pedig sárga színű, hogy kiemelje ezeket a testrészeket, mint a projekt vizsgálatának tárgyait.

Videó generálás

Az előbb körülírt képeket lehetséges videó formátumba is összefűzni, ahol megadható saját FPS (frames per second, avagy képkockák másodpercenként) mérték. Ehhez azonban szükséges, hogy megfelelő kategóriájú képek is mentésre kerüljenek. Azért terveztem így a programomat, mert a videón megfigyelt tények mélyebb elemzéséhez feltételeztem, hogy szükség lesz az azt alkotó képekre is.

Grafikonok

A 3.6.6. szakaszban említett idősorozatok, és azok finomított állapotai lementhetőek a program döntésének megértése, vizsgálása érdekében.
A kirajzolások teljes kódja megtekinthető szintén az `evaluate.py` szkriptben.

3.7. Futtatási lehetőségek

3.7.1. A felhasználói felület

A projekt felhasználói felületének megvalósításához a `Streamlit` nevű Python-alapú webes keretrendszert használtam. A választásom elsősorban annak köszönhető, hogy a Streamlit lehetővé teszi egyszerű, gyors és interaktív felhasználói felületek fejlesztését minimális frontend ismeretek mellett. Mindent Pythonból lehet vezérelni,

így nem kellett külön HTML-t vagy JavaScriptet írnom. Emellett rengeteg beépített elemet kínál, például csúszkákat, szövegbeviteli mezőket, gombokat, felugró szöveges ablakokat, amelyekkel néhány sor kóddal könnyedén interaktívvá tehető az alkalmazás. A vizualizációk megjelenítéséhez használhatóak a `matplotlib`, `OpenCV` és `PIL` könyvtárak, így a képeket és grafikonokat közvetlenül az oldalon tudtam megjeleníteni.

Mivel a kutatás alatt a hangsúly a képfeldolgozási és adatvizualizációs algoritmusok fejlesztésén volt, fontos szempont volt, hogy a UI (user interface) létrehozása ne vegyen el aránytalanul sok időt a projektből. Ugyanakkor az egyszerűségnek ára is van: a felület testreszabhatósága korlátozottabb, mint például egy teljes értékű frontend keretrendszernél. Így a dizájn és az elrendezés szempontjából kompromisszumot kellett kötnöm. Elfogadtam, hogy nem formázhatok mindent teljesen szabadon, cserébe viszont nagyon gyorsan működő alkalmazást kaptam.

A projekt során egy távoli szerveren dolgoztam, a `Streamlit` segítségével a felhasználói felület egy webes alkalmazásként működött. Így nem volt szükségem külön grafikus felületre a szerveren, böngészőből elértem a programot. Fontosnak tartottam, hogy ez lehetséges legyen, ugyanis a programom jelentősen számítógépteljesítmény igényes. Úgy gondoltam, hogy ritka az az asztali számítógép, amelyik rendelkezik elegendő virtuális memóriával a programom futtatásához, ezért a felhasználói is valószínűleg egy hozzám hasonló környezetből fogják futtatni.

A felhasználói felületet egy oldalra szegélyezett menüvel tettem bejárhatóvá, ahol a program két fő komponense (3.5. és 3.6. szakaszok) alapján csoportosítottam az oldalakat. A program futtatása során a `Streamlit` az `index.py` fájlt futtatja, amely egyből átirányít a `home.py` oldalra, ahonnan további szándék szerint felfedezhető az applikáció.

Felhasználói hibák elleni védelem

A fejlesztés során sokat kísérleteztem módszerekkel, amivel megvédhető az applikációm rendeltetés szerű működése a felhasználó kíváncsisága ellen. Az előbbi sorokban tárgyalt egyszerűség ára itt is érezhető volt, ugyanis csak nyakatekert módon lehet a felhasználó vezérlőkhöz való hozzáférését korlátozni. A végső, és szerintem legátfedőbb megoldás, hogy ha bármilyen feladat fut éppen az oldalon, akkor az összes látható vezérlőt inaktívvá teszem, hogy ne lehessen értéküket változtatni futás közben. Ezen felül a felhasználót egy figyelmeztetéssel arra kérem, hogy várja

meg a lefutás végét. Az ellen, ha a felhasználó elnavigál az akutális oldalról futás közben, sajnos a keretrendszeren belül nem tudtam intézkedni. Ilyenkor a háttérben fut tovább a programom a felületről elindított paraméterekkel, de a felületben ez már nem nyomon követhető például oldal visszatöltésekor.

3.7.2. Parancssori futtatás

Videóelemzés

A `Streamlit` alapú grafikus felhasználói felület mellett a program parancssorból is futtatható. Ennek célja, hogy fejlettebb felhasználók is kényelmesen használhassák az alkalmazást, UI nélkül.

A parancssoros verzió az `argparse` könyvtár segítségével valósul meg, amely lehetővé teszi, hogy a felhasználó különféle kapcsolókkal szabályozza, mit mentsen el a feldolgozás során. A kötelező paraméterek az input videó elérési útvonala, valamint a kimeneti mappa, ahová az eredmények kerülnek. Emellett számos opcionális kapcsoló is elérhető a dolgozatban előzőleg megemlített mentési lehetőségek szabályozására. A teljes kód megtekinthető a `nogui_pipe.py` szkriptben.

Adatelőkészítés és modell tanító demó

Annak érdekében, hogy minden alapfunkciót lefedjek felhasználói felület nélküli futtathatósággal, elkészítettem a `nogui_prepare.py` és `nogui_train.py` szkripteket is, melyek segítségével hasonlóan állítható elő adathalmaz vagy tanítható modell, mint a felhasználói felületből. A megvalósítás szintén az `argparse` könyvtárral történt.

3.8. Tesztelés

3.8.1. Backend

A különböző alkotóelemek működését a szakmai dolgozat jellegéből adódóan alapvetően csak szemmel lehet jóváhagyni, vagy kritizálni. A tényleges pontosság méréséhez szükséges lenne azon címkézések elkészítése a bemeneti adatokra, amit az applikációm pont, hogy elkerülni próbál. A 3.5.2. szakaszban bemutatott teljesít-

mény összehasonlító táblázatok például objektív tesztek, hiszen ezekhez az adatokhoz rendelkeztem a felcímkézett mérési alappal.

A programban szereplő megszorítások (például a bemeneti videó hossza) kipróbálásán túl nem volt lehetőség további tesztelésre.

3.8.2. Frontend

A felhasználói felületet a 2. fejezetben leírt felhasználói dokumentáció alapján, illetve a 3.7.1. szakaszban bemutatottak alapján tesztelhető. Alapvető tesztek:

- A **pipeline** menüben csak akkor jelenik meg a prediktáló gomb, ha megadtunk helyes bemeneti videót és kimeneti könyvtár-útvonalat.
- Bármilyen gombra nyomás után a feladat lefutásáig nem lehetséges az adott oldalon szereplő vezérlőkhöz nyúlni. Megjelenik egy nyomatékos figyelmeztető, ami elbátortalanítja a felhasználót a programhoz piszkálásától, ám ezt nem lehetséges letiltani.
- Feladat lefutás után megjelenik egy összesítés a feladat eredményeiről, sikeréről.

3.9. Eredmények

A modellek tanításához 6000 annotált képet használtam, amelyet a Semmelweis Orvostudományi Egyetem Biofizikai és Sugárbiológiai Intézet munkatársai készítettek DeepLabCut használatával. Ezt használtam mérési alapnak (továbbiakban GT, mint ground truth). Az annotált adatokat YOLO formátumra alakítottam (3.2. forráskód), hogy formailag megegyezzenek azzal az adattal, amit egy modell prediktáláskor hoz létre.

A pontosság meghatározásához az átlagos négyzetes hibát (MSE) értékeltem ki, a következők szerint. Kizárólag a releváns, azaz a $[7, 9, 10, 12]$ kulcspontokat (2.2. tábla) figyelembe véve kiszámolom a GT-beli és a modell által becsült pont négyzetes Euklideszi távolságát, de csak akkor, ha a GT-beli pont eredetileg nem nulla koordinátájú. Adott képre a végső MSE az ezen négyzetes hibák átlaga az adott képen található érvényes kulcspontokra nézve. Az alacsonyabb MSE jobb pontosságot jelez. Egy modellre nézve a végső MSE az összes teszt-képre kiszámolt MSE-k átlaga.

Ez a szkript (`mse.py`) a projekt könyvtáron belül a `scripts/benchmarks` mappában található. Az eredményeket a 3.4. és 3.5. táblázatok mutatják be. Ezekhez a jelmagyarázat itt megtekinthető:

- full - a modell teljes méretű képekre tanult
- cropped - a modell kivágott méretű képekre tanult
- dark - a patkány maszkját kivéve az egész kép sötét
- v8 - a modell alapja a YOLOv8l-pose
- v11 - a modell alapja a YOLOv11l-pose

3.4. táblázat. MSE az összes kulcspont alapján

Model name	**# entries**	**Total MSE**	**Average MSE**
full_v8	4521	19.397779	0.004291
full_v11	4522	24.590980	0.005438
darkfull_v8	4510	24.532054	0.005439
darkfull_v11	4518	28.378929	0.006281
cropped_v8	4521	109.731735	0.024272
cropped_v11	4521	132.152753	0.029231
darkcropped_v8	4521	107.613112	0.023803
darkcropped_v11	4521	127.968257	0.028305

3.5. táblázat. MSE comb és lábfej alapján

Model name	**# entries**	**Total MSE**	**Average MSE**
full_v8	4500	30.386261	0.006753
full_v11	4500	54.308615	0.012069
darkfull_v8	4492	45.833419	0.010203
darkfull_v11	4496	57.527402	0.012795
cropped_v8	4499	147.725306	0.032835
cropped_v11	4500	160.793024	0.035732
darkcropped_v8	4500	130.749963	0.029056
darkcropped_v11	4500	130.684783	0.029041

A variánsokat lemértem érzékenység (recall) szempontjából is, ami azt méri, hogy a modell mennyire képes megtalálni az összes kulcspontot. Ezt a következő képlettel számoltam ki:

$$\text{Recall} = \frac{\text{Helyesen megtalált kulcspontok}}{\text{Helyesen megtalált kulcspontok} + \text{Kihagyott, de valójában jelen lévő kulcspontok}}$$

Két fajta elfogadási módszerrel számoltam: az első mérésnél elfogadtam, ha a becsült kulcspont közelebb van a GT-beli ponthoz, mint a kép magasságának 5%-a, a másodiknál pedig a bouding box nagyobbik dimenziójának 10%-a volt a kritérium. A végső recall értéket itt is átlagolva számítottam ki az összes bemenetre. A teljes folyamat kipróbálható szintén a `scripts/benchmarks/recall.py` szkripttel. Az eredmények a 3.6. és 3.7. táblázatban figyelhetőek meg:

3.6. táblázat. Recall (kép magasság 5%)

Model name	**# entries**	**Average Recall %**
full_v8	4500	97.97
full_v11	4500	96.78
darkfull_v8	4492	96.75
darkfull_v11	4496	96.24
cropped_v8	4500	56.74
cropped_v11	4500	54.91
darkcropped_v8	4500	56.05
darkcropped_v11	4500	55.56

3.7. táblázat. Recall (bounding box 10%)

Model name	**# entries**	**Average Recall %**
full_v8	4500	97.69
full_v11	4500	96.27
darkfull_v8	4492	96.09
darkfull_v11	4496	95.57
cropped_v8	4500	56.25
cropped_v11	4500	54.44
darkcropped_v8	4500	55.54
darkcropped_v11	4500	55.16

Az eljárások szkriptjeinek futtatásához szükséges egy könyvtárba összegyűjteni az összehasonlítani kívánt modellek predikcióit, illetve a GT-t YOLO formátumban. A szkriptek ezen a könyvtáron iterálnak végig, és összegzik a modellek eredményeit.

A legjobban teljesítő modell a YOLOv8l-póz, amely a teljes képen tanult, a háttér maszkolása nélkül.

4. fejezet

Összegzés

A dolgozatban bemutatott alkalmazás egy komplex és modulárisan felépített rendszert valósít meg, amely a videóalapú viselkedésanalízist szolgálja patkányok mozgásmintáinak vizsgálatán keresztül. A fejlesztés során a Python programozási nyelvet és számos modern könyvtárat használtam. Egy felhasználói felülettel rendelkező alkalmazást fejlesztettem, melynek két fő része van. A póz becslő modell tanítása és a modell felhasználása predikcióhoz. Az alkalmazás futtatható lokálisan vagy távoli eléréssel szerveren. Saját adat elrendezése és feltöltése a program könyvtárába szükséges használat előtt. Az annotált adatok előállításához a DeepLabCut annotáló szoftverét használhatjuk.

A gépi látás és mélytanulás eszközei közül a YOLOv8-pose és a Segment Anything Model (SAM) modelleket kombináltam. A modell tanító komponenssel saját `yolov8l-pose` modell tanítható, így újra felhasználható a program, akár más adatokra is.

A rendszer lépésről lépésre feldolgozza a bemeneti videót, kulcspontokat és maszkokat állít elő, szintetikus csontvázzal javítja a hibás kulcspontokat, majd az így kinyert idősoros jellemzők alapján végzi el a besorolást egy ötfokú mozgáskategória-skálán.

A megvalósítás során külön figyelmet fordítottam a megbízhatóságot és rugalmasságot biztosító megoldásokra, így a program minden fő komponense elkülöníthetően futtatható, konfigurálható, és grafikus felhasználói felület is készült hozzá. A fejlesztés egyik legnagyobb erőssége az, hogy a különböző zajforrások kiszűrésére több szinten történik adatfinomítás. Az videóadat feldolgozásáért felelős programrészlet egy bemeneti videóra képes megmondani, hogy egy 1-től 5-ig terjedő skálán

mennyire mozog egészségesen a megfigyelt patkány. Ezentúl a feldolgozás lépései során létrejövő adatot lehetséges kapcsolók segítségével köztes állapotokban lementeni.

A grafikus felhasználói felületen keresztül az informatikában kevésbé jártas felhasználók is elvégezhetik egy videó elemzését annotációk készítése nélkül. A fő továbbfejlesztési lehetőség az osztályozás pontossága.

A teljes program letölthető az alábbi OneDrive hivatkozáson keresztül, ELTE Informatikai Kar szervezeti fiókkal történő bejelentkezés után: `https://ikelte-my.sharepoint.com/:f:/g/personal/xikvif_inf_elte_hu/EpSsI66NMklFtpFJMDKVeS4BcsI9F2Dc7kjZR9uZmVZYHw?e=6dMjkA`

4.1. Nehézségek

A programomat teljes egészében az NIPG (Neural Information Processing Group) távoli szerverén fejlesztettem, ami betekintést nyújtott az ssh titkosítás, az Apptainer környezetek kialakítása és a Jupyter Notebookok fogalmaiba. Emiatt viszont kellett pár sajátos megoldást alkalmaznom, például felugró ablakok megtekintése nem volt lehetséges, ezért az IO műveletek szolgáltak az adatok megtekintésére. Ez a körülmény inspirálta a Streamlit használatát. A Streamlit által biztosított felület nem teljesen testre szabható, magamat hátráltattam azzal, hogy jobban testre akartam szabni mint azt lehetséges.

Az általam használt csomagok verzióinak összehangolása kihívásnak bizonyult, ugyanis gyakran egy csomag befolyásolta a másik verziószámát telepítéskor. Az általam használt `ultralytics` könyvtár lokális fájljait módosítanom kellett: a témavezetőm útmutatásával kiegészítenem a kódot további kép-torzító lehetőségekkel, amelyek a pózbecslő modell általánosító képességét növelhetik.

A viselkedés-megfigyelésre szánt videófelvételek nem egy stabil felül- vagy alulnézeti kamerával készültek, hanem kézzel fogottal. Emiatt a videófelvételen kivehető egy alapszintű remegés, illetve a felvételt készítő személy követi a patkányt mozgása közben, esetenként bele is nagyít a felvételbe. Az említett tények mind zajt hoznak be a rendszerbe kulcspontanalízis szempontjából. A pontos osztályozás céljaként ezek kiküszöbölése kutatást igényelt. Különböző zajtípusok kiszűrésére használt eljárásokat alkalmaztam, mint az optikai áramlás mozgó kamera esetén, és a test-normalizálás annak érdekében, hogy egy adott távolság pixeltől független legyen.

4.2. Továbbfejlesztési lehetőségek

A fejlesztői dokumentációban szereplő, kivágott képeken tanított és ezen működő modellek potenciálja elsősorban speciális esetek kezelésében érvényesülhet, azonban ezek működésének finomhangolása további vizsgálatot igényel. A szélsőséges esetek változatossága arra utal, hogy az ilyen modellek alkalmazása robusztusabb adatfeldolgozási módszerekkel válhat eredményesebbé. A teljes képeken tanult modellekhez viszonyított alacsonyabb teljesítmény okának feltárása hozzájárulhat a rendszer megbízhatóságának és általánosítási képességének javításához.

A különböző szegmentáló modellek teljesítményének összehasonlítása a fejlesztési időkorlátok miatt jelenleg korlátozott mértékben történt meg. A dokumentált eredmények (4.1. táblázat) alapján léteznek olyan alternatívák, mint a FastSAM, SAM2 vagy a YOLO szegmentálásra optimalizált modelljei, amelyek potenciálisan gyorsabb működést kínálnak, bár integrációjuk komplexitása, illetve esetenkénti újratanítási igényük mérlegelendő. A jövőbeli fejlesztések során érdemes lehet ezek összevetése a jelenleg alkalmazott megoldással.

Modell	**Méret (MB)**	**Paraméterek (M)**	**Sebesség (CPU) (ms/kép)**
Meta SAM-b	375	93.7	49401
Meta SAM2-b	162	80.8	31901
Meta SAM2-t	78.1	38.9	25997
MobileSAM	40.7	10.1	25381
FastSAM-s egy YOLOv8 backbone-al	23.7	11.8	55.9
Ultralytics YOLOv8n-seg	**6.7 (11.7x kisebb)**	**3.4 (11.4x kevesebb)**	**24.5 (1061x gyorsabb)**
Ultralytics YOLO11n-seg	**5.9 (13.2x kisebb)**	**2.9 (13.4x kevesebb)**	**30.1 (864x gyorsabb)**

4.1. táblázat. Modellek összehasonlítása méret, paraméterszám és CPU sebesség alapján.

Az Ultralytics éppen a dolgozat elkészítése alatt jelenítette meg a legújabb generációs YOLOv12 modelljeit, melyek ekkor még nem támogatták a pózbecslést. Az architektúra tanításával illetve az előtanított modell súlyaiból indított tanítással további javulás válhat elérhetővé a pontosságban, így a futási időn az utólagos javítási lépések csökkentésével gyorsítani lehetne.

A legmagasabb prioritást élvező továbbfejlesztési lehetőség a bemeneti videókból kinyert adatok osztályozási logikája. A biológiai kutatási kérdések és ehhez tartozó statisztikai mintázatok meghatározása túlmutat a szakdolgozat keretein, de a kutatási eredményekhez szükségesek.

Köszönetnyilvánítás

Szeretnék köszönetet nyílvánítani tudományos vezetőmnek, Dr. Gelencsér-Horváth Annának a folyamatos támogatásáért, szakmai ötleteiért, rugalmasságáért amit az egész eddigi kutatás alatt nyújtott jóval a szakmai dolgozat megalkotása előtt, nem csak alatt.

Hálás vagyok Édesapámnak, Dallos Lorandnak, amiért esetenként helyettem is bízott bennem, és szintén segített az ötletelésben; illetve az egész családomnak a támogatásukért, megértésükért.

Külön meg szeretném köszönni Molnár Róbertnének (leánykori nevén Baranyai Erzsébet), hogy a dolgozat elkészítése alatt szüntelenül gondoskodott arról, hogy ezt ne éhgyomorral tegyem. Érdeme képpen jelentősen hatékonyabban végezhettem tanulmányaimat.

Irodalomjegyzék

[1] C.S. Hall. „Emotional behavior in the rat. I. Defecation and urination as measures of individual differences in emotionality". *Journal of Comparative Psychology* 18.3 (1934), 385–403. old. DOI: 10.1037/h0071444.

[2] R.N. Walsh és R.A. Cummins. „The open-field test: a critical review". *Psychological Bulletin* 83.3 (1976), 482–504. old. DOI: 10.1037/0033-2909.83.3.482.

[3] Apptainer Contributors. *Installation on Linux — Apptainer Documentation*. Accessed: 2025-04-16. 2024. URL: https://apptainer.org/docs/admin/main/installation.html#installation-on-linux.

[4] Alexander Mathis és tsai. „DeepLabCut: markerless pose estimation of user-defined body parts with deep learning". *Nature neuroscience* 21.9 (2018), 1281–1289. old.

[5] Guido Van Rossum és Fred L Drake Jr. *Python tutorial*. 620. köt. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[6] Talmo D Pereira és tsai. „SLEAP: A deep learning system for multi-animal pose tracking". *Nature methods* 19.4 (2022), 486–495. old.

[7] Arindam Sengupta és Siyang Cao. „mmpose-nlp: A natural language processing approach to precise skeletal pose estimation using mmwave radars". *IEEE Transactions on Neural Networks and Learning Systems* 34.11 (2022), 8418–8429. old.

[8] Ultralytics. *Ultralytics YOLO Documentation*. 2025. URL: https://docs.ultralytics.com (elérés dátuma 2025. 04. 16.).

[9] Yuxiang Yang és tsai. „An Effective Yak Behavior Classification Model with Improved YOLO-Pose Network Using Yak Skeleton Key Points Images." *Agriculture; Basel* 14.10 (2024).

[10] Zehra. *Neural network components*. 2019. URL: https://zerzavot.medium.com/neural-networks-components-a28c03d9dec (elérés dátuma 2025. 04. 19.).

[11] Facundo Bre, Juan Gimenez és Víctor Fachinotti. „Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks". *Energy and Buildings* 158 (2017. nov.). DOI: 10.1016/j.enbuild.2017.11.045.

[12] Benedek Gergő. *Computer Vision / Machine Vision: mit jelent a Gépi Látás, és mire lehet felhasználni?* 2020. URL: https://lexunit.hu/blog/mit-jelent-a-gepi-latas-es-mire-lehet-felhasznalni/ (elérés dátuma 2025. 04. 13.).

[13] Xia Zhao és tsai. „A review of convolutional neural networks in computer vision". *Artificial Intelligence Review* 57.4 (2024), 99. old.

[14] Charles R Harris és tsai. „Array programming with NumPy". *Nature* 585.7825 (2020), 357–362. old.

[15] Nikhil Ketkar és tsai. „Introduction to pytorch". *Deep learning with python: learn best practices of deep learning models with PyTorch* (2021), 27–91. old.

[16] Gary Bradski. „The opencv library." *Dr. Dobb's Journal: Software Tools for the Professional Programmer* 25.11 (2000), 120–123. old.

[17] Alex Clark és tsai. „Pillow (pil fork) documentation". *readthedocs* (2015).

[18] Sandro Tosi. *Matplotlib for Python developers*. Packt Publishing Ltd, 2009.

[19] Jeff Reback és tsai. „pandas-dev/pandas: Pandas 1.0. 5". *Zenodo* (2020).

[20] Pauli Virtanen és tsai. „SciPy 1.0: fundamental algorithms for scientific computing in Python". *Nature methods* 17.3 (2020), 261–272. old.

[21] Stefan Van der Walt és tsai. „scikit-image: image processing in Python". *PeerJ* 2 (2014), e453.

[22] Glenn Jocher, Ayush Chaurasia és Jing Qiu. *Ultralytics YOLOv8*. 8.0.0. verzió. 2023. URL: https://github.com/ultralytics/ultralytics.

[23] Alexander Kirillov és tsai. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV]. URL: https://arxiv.org/abs/2304.02643.

[24] Anaconda, Inc. *Anaconda Documentation*. https://www.anaconda.com/docs/main. Accessed: 2025-04-17. 2024.

[25] Mathis Lab. *DeepLabCut Standard User Guide*. `https://deeplabcut.github.io/DeepLabCut/docs/standardDeepLabCut_UserGuide.html`. Accessed: 2025-04-17. 2024.

[26] Gunnar Farnebäck. „Two-frame motion estimation based on polynomial expansion". *Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, June 29–July 2, 2003 Proceedings 13*. Springer. 2003, 363–370. old.

[27] Ronald W Schafer. „What is a savitzky-golay filter?[lecture notes]". *IEEE Signal processing magazine* 28.4 (2011), 111–117. old.

Ábrák jegyzéke

Táblázatok jegyzéke

Algoritmusjegyzék

Forráskódjegyzék