[Külső fekete borítólap formátuma]

Széchenyi István Egyetem

Gépészmérnöki, Informatikai és Villamosmérnöki Kar

Informatika Tanszék

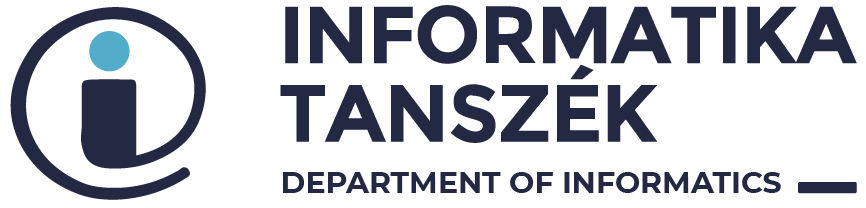
**SZAKDOLGOZAT**

**Angyal Ádám Szilárd**

**Mérnök Informatikus BSc szak**

[beadás éve]

|  |
| --- |
| [Gerincen:] Hallgató Neve, Évszám {Titkosított} |



**SZAKDOLGOZ****AT**

**Állatok detektálása felvételről mesterséges intelligencia segítségével**

**Angyal Ádám Szilárd**

**Mérnök Informatikus BSc szak**

**[beadás éve]**

**[Ennek a bekezdésnek a helyére szerkessze be az aláírásokkal ellátott feladatkiírási lap szkennelt változatát.]**

# Nyilatkozat

Alulírott, [Hallgató Neve (Neptun-kód), Hallgató szakja, képzési szintje] szakos hallgató kijelentem, hogy a [Szakdolgozat Címe] című szakdolgozat feladat kidolgozása a saját munkám, abban csak a megjelölt forrásokat, és a megjelölt mértékben használtam fel, az idézés szabályainak megfelelően, a hivatkozások pontos megjelölésével.

Eredményeim saját munkán, számításokon, kutatáson, valós méréseken alapulnak, és a legjobb tudásom szerint hitelesek.

Győr, [beadás dátuma]

hallgató

# Kivonat

Állatok detektálása felvételről mesterséges intelligencia segítségével

Egy gazdaság területén előforduló állatok videófelvételről való detektálására alkalmas, neurális hálózatot használó programok bemutatása. A programok mögött meghúzódó elméleti felépítés leírása. A programok technológiai és működési részének részletezése. Mindhárom program megvalósítása és tesztelése az ajánlott és saját adatkészlettel. Az eddig bemutatott programok összehasonlítása (pl. hatékonyság, pontosság, erőforrás felhasználás, stb) alapján. Az összehasonlítás után egy végső konklúzió levonása a kapott eredmények alapján, melyik módszer a leghatékonyabb.

# Abstract

Detecting animals from recordings using artificial intelligence

Demonstration of neural network programs for detecting animals on a farm from video footage. Describe the theoretical architecture behind the programs. Details of the technological and operational aspects of the programmes. Implementation and testing of all three programs with the recommended and own datasets. Comparison of the programs presented so far (e.g. efficiency, accuracy, resource usage, etc.). After the comparison, a final conclusion will be drawn based on the results obtained as to which method is the most efficient.

# Tartalomjegyzék

[Nyilatkozat 5](#_Toc165236569)

[Kivonat 6](#_Toc165236570)

[Abstract 7](#_Toc165236571)

[Tartalomjegyzék 8](#_Toc165236572)

[1.Bevezetés 1](#_Toc165236573)

[2.Háttérelmélet 2](#_Toc165236574)

[2.1.Neurális hálózat 2](#_Toc165236575)

[2.2.Neuronális réteg 4](#_Toc165236576)

[2.3.Aktiválási függvények 5](#_Toc165236577)

[2.4.Szabályozási technikák 7](#_Toc165236578)

[2.5.Optimalizálók 7](#_Toc165236579)

[3.1.YOLOv8 8](#_Toc165236580)

[3.1.1.Architektúra 8](#_Toc165236581)

[3.1.2.Tanítás 9](#_Toc165236582)

[3.1.3.Validáció 10](#_Toc165236583)

[3.1.4.Predikció 10](#_Toc165236584)

[3.2.Zamba 12](#_Toc165236585)

[3.2.1.Architektúra 12](#_Toc165236586)

[3.2.1.Modell 13](#_Toc165236587)

[3.2.2.Predikció 14](#_Toc165236588)

[3.3.Yolov7 15](#_Toc165236589)

[3.3.1.Architektúra 15](#_Toc165236590)

[3.3.2.Adatkészlet 16](#_Toc165236591)

[3.3.3.Predikció 17](#_Toc165236592)

[4.Programok megvalósítása 18](#_Toc165236593)

[4.1.Általánosan felhasznált eszközök 18](#_Toc165236594)

[4.2.Saját adatkészlet készítése 18](#_Toc165236595)

[4.3.Yolov8 telepítése 19](#_Toc165236596)

[4.3.1*.*Tanítás és detektálás 20](#_Toc165236597)

[4.4.Zamba telepítése 21](#_Toc165236598)

[4.4.1.Zamba általános használata 22](#_Toc165236599)

[4.4.2.Zamba tanítása 23](#_Toc165236600)

[4.5.Yolov7 használata a Google colab-ban 23](#_Toc165236601)

[4.5.1.Yolov7 telepítése 24](#_Toc165236602)

[4.5.2.Tanítás 24](#_Toc165236603)

[4.5.3.Detektálás 24](#_Toc165236604)

[5.Összehasonlítás és összegzés 26](#_Toc165236605)

[5.1. Erőforrás használat 26](#_Toc165236606)

[5.2. Eredmények 26](#_Toc165236607)

[5.3. Konklúzió 28](#_Toc165236608)

[6.Irodalomjegyzék 30](#_Toc165236609)

[7.Mellékletek 32](#_Toc165236610)

# 1.Bevezetés

A tárgyfelismerés az a módszer, amely egy adott arc vagy tárgy felismerésére szolgál egy nagyobb képből vagy videóból. A képek megkülönböztetése, felismerése és osztályozása az eredetétől nem fontos vagy fárasztó feladat az emberi szem számára. Azonban egy gép számára nehéz feladat a valós körülmények között lévő tárgyak megértése, mivel azokat különböző tényezők, például formák, méret, szín és textúra alapján veszik fel. A legújabb fejlesztések a mesterséges intelligencia és a számítógépes látás területén a tárgyak észlelésére és követésére szolgáló technológia széleskörű megoldásaival trendek.

A mezőgazdaságban egyre nagyobb népszerűségnek örvend a biztosángi kamerák által felvett videók állat felismerési szintű kielemzése. Erre azért van szükség, mivel fontos nyomon követni a haszon állatok mozgását, viselkedését, egészségügyi állapotát. Ezeken felül fontos a károk elkerülése vagy kielemzése végett az egyéb (kóbor kutya, menyét, stb…), nem a gazdasághoz tartozó állatok felismerése is.

A szakdolgozatom témája a fentebb említettekre próbál néhány elterjedtebb példával (neurális hálózatot használó programok) megoldást mutatni. Ezen módszereket részletesen bemutatom és számszerűsítve is összehasonlítom (pl. hatékonyság, pontosság, erőforrás felhasználás alapján), majd a kapott eredmények alapján levonok egy végső konklúziót.

A programok elérhetőek a github oldalamon: https://github.com/AASZ40/Allatok-detektalasa-felvetelrol-mesterseges-intelligencia-segitsegevel

….

# 2.Háttérelmélet

Ahhoz, hogy megértsük, hogyan is működik egy detektáló program, a mögötte meghúzódó elméletet is érteni kell, hiszen a leg elemibb része segíthet egy probléma leküzdésében vagy egy forradalmi fejlesztésben.

## 2.1.Neurális hálózat

A neurális hálózat összekapcsolt biológiai neuronok csoportja. A modern használatban a szó alatt mesterséges idegsejtekből álló mesterséges ideghálózatot értünk. Ezért a neurális hálózat kifejezés 2 különböző fogalmat is jelent: A biológiai neurális hálózat a fókuszok vagy funkcionálisan kapcsolódó neuronok csomópontja a perifériás vagy a központi idegrendszerben. Az idegtudomány területén leggyakrabban a laboratóriumi elemzésre alkalmas idegrendszerek csoportjaként azonosítják.A mesterséges neurális hálózatok biológiailag motivált gépek/programok, amelyek modellezik a biológiai neurális hálózatok néhány jellemzőjét. Az alkalmazások többsége technikai, nem kognitív modell. A mesterséges neurális hálózatok nemcsak a biológia, hanem más tudományágak (Matematika, Fizika, Pszichológia) eredményeit is használják. A származástól függetlenül a hálózat hatásmechanizmusa többé-kevésbé azonos: ezeknek a hálózatoknak az az elve, hogy a számítást egymással összekapcsolt kis feldolgozó egységek játsszák, az idegsejtek közötti összekötő rendszer fontos szerepet játszik az idegsejtek kiszámításában, ezért az ideghálózatot összekötő hálózatnak is nevezik, és a dolgozó szakembereket konnektológusoknak is nevezik.

A neurális hálózat egyszerű egységekből áll abban az értelemben, hogy belső állapotát számokkal lehet leírni, ezek aktiválási értékek. Minden egység létrehoz egy kimeneti értéket (jelet), amely az aktiválási értéktől függ. Az egységek egymáshoz vannak csatlakoztatva, és mindegyik kapcsolat egyedi súlyokat tartalmaz. Minden egység kimeneti értéket küld a kimenő kapcsolat összes többi egységének. A "rendszer" bemenete lehet szenzoros vagy mesterséges érzékelő, az érzékelőtől származó adatok, kimenete pedig művelet, jel a kimeneti neuronon, vagy mesterségesen megjelenített válasz egy kérdésre (neuronhálózat esetén természetesen minta). Ezen csatlakozások miatt az egység kimenete befolyásolja más egységek aktiválását. A kapcsolat bemeneti oldalán lévő egység megkapja az értékeket, és kiszámítja azok aktiválási értékeit súlyozással (a bemeneti jelet megszorozzuk a megfelelő bemeneti súlyokkal, és figyelembe vesszük azok összegét) a kimenetet az aktiválási funkció határozza meg az aktiválásra adott válaszként (például, ha az aktiválás meghaladja a határértéket, az egység aktiválódik). Az, hogy ezek a jelek (értékek) elektrokémiai, elektromos vagy szimbolikusak-e, a "megvalósítás" módszerétől függ (biológiai, hardver, szoftver), de ez nem a működés elve.

Fontos megjegyezni, hogy a neuronok számításokat végeznek, de nem processzorokat. A fő különbség az, hogy az idegsejteket tanítják (a súlymátrix értékének beállításával), miközben a processzort programozzák (folyamatos utasítássorozatot kapnak nekik). Minden hálózati tanulási módszer vezérelhető. Ellenőrizetlen típusú tanulás. Ellenőrzött tanulási N-net esetén a rendszer számos előre meghatározott példából tanul. Speciális algoritmust használ az idegsejtek közötti kapcsolat megváltoztatásához, amíg a megadott bemenet mindig "okozza" a megadott kimenetet. A legtöbb esetben a hálózat a kapcsolat súlyának beállításával tanul. A súlyok beállításakor figyelembe veszik az úgynevezett hibafunkció eredményeit. A hibafunkció értékének kiszámításához sokféleképpen lehet kiszámítani, de a legegyszerűbb eset a helyes kimeneti érték kivonása a kimeneti értékből.

A nem ellenőrzött hálók esetében a Kohonen önszerveződő térképeket használják leggyakrabban, valamint azokat a hálózatokat, amelyek abból a feltételezésből működnek, hogy a hálózat képes azonosítani a teljes bemeneti mintakészlet közös jellemzőit.

Mesterséges neurális hálózatok vagy ANN (mesterséges neurális hálózatok) biológiailag inspirált szimulációk. Fő alkalmazási területe a gépi tanulás, amelynek célja, hogy ezeket a hálókat valóban tanulási rendszerként alkalmazza. Gráf alapú modell, amelyben a rétegekbe helyezett mesterséges neuronok nemlineáris aktivációs funkciókon keresztül kommunikálnak egymással. A nemlineáris aktivációs funkciók legalább három rétegével rendelkező neurális hálózat univerzális funkcionális közelítés, a visszacsatolással és rekurzióval rendelkező architektúra pedig Turing-integritással rendelkezik, így bármilyen problémát meg tud oldani, vagy bármilyen algoritmust végrehajthat, amelyet egy számítógép megoldhat vagy végrehajthat.

Általában gradiens ereszkedéssel tanítják őket a hibák visszaterjedésével kombinálva, de számos alternatív algoritmus is rendelkezésre áll.

A konvolúciós neurális hálózatok (CNN-ek vagy ConvNets) mély és előrecsatolt neurális hálózatok, amelyeket általában képelemzésre és klasszifikációra használnak. 1962-ben Hubel és munkatársai macskák és majmok látókéregét tanulmányozták, és azt állapították meg, hogy léteznek olyan struktúrák ezen állatok szemében, amelyet receptív mezőnek neveznek. A vizuális ingerek a látótér kis régióiban csak egy neuron gyűrűjét váltják ki a receptív térben.

Ennek a neuronnak a szomszédjai hasonló receptív mezővel rendelkeznek. Tanulmányuk is a vizuális sejtek két típusának létezését javasolta: egyszerű sejtek és összetett sejtek.Az egyszerű sejtek egyenes éleket érzékelnek meghatározott irányokkal, míg az összetett sejtek nem veszik figyelembe az élek helyzetét.E megfigyelések alapján Fukushima és Kunihiko bevezette a Neocognitron-t. A Neocognitron volt az első megvalósított konvolúciós neurális hálózat és a receptív mező elméletének első alkalmazása művészi neurális hálózatokra. A Neocognitron egy vizuális mintát sok részmintára (feature) bont. Ezután az összes jellemző a jellemzősíkok kaszkádján mennek keresztül. A Neocognitron modellezi a biologikus vizuális rendszereket, így lehetővé teszi a felismerést még akkor is, ha a tárgyak eltolódnak vagy kissé el vannak csavarodva.

LeCun egy 7 szintű konvolúciós neurális hálózatot épített ki, ennek neve LeNet-5 és a kézzel írt számjegyek felismerésére alkalmas. Ebben a neurális hálózatban háromféle réteg van: konvolúciós réteg, almintavételező réteg és teljesen összekapcsolt réteg. Az összes konvolúciós réteg 5-5 méretű kernellel végeznek konvolúciót, és az összes egység egy jellemzőtérképben osztozik ugyanazon kernelen. A magok egy képponttal vannak eltolva. A mintavételező rétegek 2-2 kernelt használnak, ahol a kernel két pixellel eltolódik, hogy a bemeneti mezők ne fedjék egymást.

A modern konvolúciós hálózatokat sikeresen használták a képklasszifikációs problémákban, mivel a hagyományos neurális hálózatok nehezen tudnak ilyen problémákat kezelni. Például egy olyan hálózat, mint a CIFAR-10, amely bemeneti képeket fogad be, amelyek mérete 32×32×3, 3,072 súlyt igényel a teljesen összekapcsolt rétegének dene-hez. Az ilyen hálózatok teljesen összekapcsolt rétegekkel történő képzése költséges eljárás lehet. Egy konvolúciós neurális hálózat minden egyes rejtett rétege úgy absztrahálható, hogy három dimenzióval rendelkezik: szélesség, magasság és mélység. A szélesség és a magasság a térbeli információ dimenziója, míg a mélység a kinyert szemantikai információ mennyiségének tekinthető. Konvolúciós hálózatok általában háromféle rétegből állnak: konvolúciós rétegek, pooling rétegek és teljesen összekapcsolt rétegek. Például a CIFAR-10 hálózat, amelyet képalkotáshoz, képklasszifikációhoz használnak, a következő architektúrával rendelkezik: bemeneti réteg, konvolúciós réteg, ReLU réteg, pooling réteg, teljesen összekapcsolt réteg.

## 2.2.Neuronális réteg

A mesterséges neuronok a neurális hálózatok kiszámításának alapegysége, a biológiai neuronok rendkívül egyszerűsített modellje. Az idegsejteket általában nem modellezik maguk, de sok azonos idegsejtet 1 rétegként értelmeznek, és ezen a ponton a legtöbb idegi számítás mátrixműveletként írható le. Az adott réteg által végrehajtott pontos művelet megadja a réteg típusát, a hálózatra alkalmazott rétegek számát, típusát és sorrendjét, és meghatározza az adott neurális hálózat architektúráját.

Tökéletesen összekapcsolt, sűrű, lineáris kombináció: a bemeneti és a tárolt tömegmátrix lineáris kombinációjának előállítása:

(2.1)

ahol X a bemeneti mátrix, W A súlymátrix, b pedig az opcionális eltolási súlyvektor.

Egyszerű relapszus réteg( relapszus): a saját kimenetét adja vissza bemenetként (az adatokból generált bemenet mellett):

(2.2)

ahol X a bemeneti mátrix, Wx és bx a releváns súlyok, Ht - 1 az előző kimenet, Wh és bh pedig a releváns súlyok.

Konvolúciós réteg (Convxd, ahol X jelzi a bemenet méretét, például Conv2D kép): nincs teljes váltás, és a neuron csak a bemenet egy részét fogadja. Alapvetően keresztkorrelál a bemeneti mátrix neuronjainak súlymátrixával:

(2.3)

ahol \* keresztkorrelációt jelez.

Aktiválási réteg: az aktiválási funkció elemet a kapott mátrixon vagy tenzoron elemenként hívja bemenetként:

(2.4)

A neurális hálózatok általában legalább 3 funkcionálisan és szerkezetileg különböző részből állnak. A bemeneti rétegben bemeneti adatokat küld a hálózat többi részére, amely nem változott. Ha egy neurális hálózat ágakat tartalmaz, akkor több bemeneti réteg is lehet. A neuronok számát A bemeneti adatok mérete és alakja határozza meg. A rejtett rétegekben a bemenet és a kimenet között helyezkednek el, feladatuk az információk átalakítása és kódolása, absztrakciók és köztes reprezentációk létrehozása. Számuk, típusuk, a kapcsolatok sorrendje és a neuronok száma a hálózat változó paraméterei.

A kimeneti rétegben a kimeneti funkciók és a kimeneti neuronok számát A probléma jellege határozza meg. Osztályozás esetén általában ugyanannyi kimeneti neuron van, mint a rendelkezésre álló kategóriák, és a kimeneti függvény célja, hogy a kategóriák között egy adott osztályba tartozás valószínűségét reprezentálja. Az ilyen típusú kimenetekhez a Sigmoid és a SoftMax funkciók alkalmasak. Regressziós problémák esetén a kimeneti függvényt nem alkalmazzák, és kiszámítják az utolsó rejtett réteg kimeneteinek lineáris kombinációját. A neurális hálózatnak elméletileg több kimenete lehet, ebben az esetben a veszteségfüggvényt minden kimenethez társítani kell.

A kernel trükköket használó tanulási algoritmushoz hasonlóan egy neurális hálózatban kijelenthetjük, hogy a rejtett réteg konfigurálható és tanulható kernelként működik, és hogy a kimeneti réteg osztályozó.

## 2.3.Aktiválási függvények

Különböző aktiválási függvények alkalmazhatók a neuronális rétegek között. Ezeket általában a bemeneti mátrix minden elemére értékelik, de különleges esetekben a bemenet más elemeit is megszámolják.

Sigmoid vagy logisztikai függvények:

(2.5)

Extrém negatív vagy pozitív bemenet esetén a gradiens nagyon kicsi és csökkenti az oktatás hatékonyságát (gradiens nekrózist okozva), így a rejtett réteg aktiválási függvényeként a bemenetet 0 és 1 közé szorítja, így továbbra is felhasználható annak valószínűségének kifejezésére, hogy egy adott kategóriába tartozik, két kategóriájú osztályozás és több kategóriájú, többcímkés a kimeneti rétegben használják a fájlok osztályozásához.

Hiperbolikus érintő:

(2.6)

Ezenkívül a szigmoidokhoz hasonló aktiválási függvényeket általában a rejtett rétegekre alkalmazzák. Ma a modernebb függvényeket nagyrészt felváltották. A bemenet -1 és +1 között van rögzítve.

ReLU (korrigált lineáris egység) :

(2.7)

Talán a leggyakrabban használt aktiválási függvény a rejtett rétegben. A negatív bemenetet nullára állítja, a pozitív bemenetet pedig változatlanul hagyja. A 0-nak nincs deriváltja, de a lebegőpontos számításokban nagyon kevés 0 bemeneti érték van elrejtve a rétegek között, és ez valójában nem okoz problémát. Nem számítási szempontból intenzív, és nem okoz gradiens nekrózist.

Leaky ReLU (szivárgó ReLU):

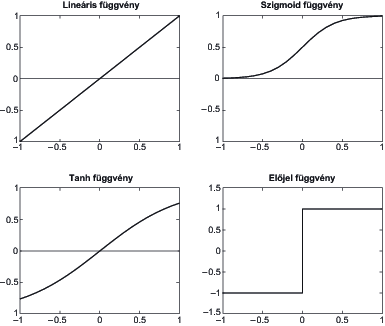
(2.8)

Úgy találták ki, hogy megszüntesse a ReLU esetében előforduló "halott ReLU" jelenséget. Valamilyen oknál fogva, ha egy ReLU minden bemenetnél 0 eltolási súlyt tanul, akkor a gradiense mindig 0 lesz, így a ReLU működésképtelenné válik. A szivárgási ReLU-ba beépített szivárgási tényező (bőség) egy megtanulható paraméter, ideális esetben 0 és 1 közötti szám.

Softmax:

(2.9)

A kimeneti aktiválási függvény több kategóriájú osztályozási problémákhoz használható. A Softmax egy vektor bemeneti függvény, amelyet logisztikai regresszióban használnak. Először kiszámítjuk a nevező exponenciális összegét, majd az egyes elemek exponenciális összegét elosztjuk ezzel az összeggel. Mivel a kimenet olyan vektort ad vissza, amelynek elemei 0 és 1 közötti értékek, és a vektorok összege 1, A kimenetet valószínűségi eloszlásként értelmezzük, amely megadja az egyes kategóriákhoz való tartozás valószínűségét, és a maximális index a legnagyobb valószínűséggel rendelkező kategória.



2.3.1.ábra: Aktivációs függvények(forrás: https://gyires.inf.unideb.hu/KMITT/a04/ch05s04.html)

## 2.4.Szabályozási technikák

A neurális hálózatok, különösen a mély architektúrák, nagyon érzékenyek a statisztikai túlillesztés problémájára. Ennek elkerülésére regularizációs technikák alkalmazhatók. Az alapelv az, hogy a regularizációs tanulás a képzett modellt általánosabbá teszi, mint a nem regularizált modellt. Lp regularizáció: a súlyok P normája hozzáadódik a veszteségfüggvényhez, általában az L1 vagy L2 norma segítségével. Ezeket a regularizációs feltételeket egyszerűen hozzáadjuk a veszteségfüggvényhez, így kapunk egy együtthatót, amely az osztályozási pontosságból és a regularizációs feltételek súlyozásából (Lagrange-koefficiensek) származó hibaértéket képviseli. A kieső regularizáció esetében a neuronok egy véletlenszerűen kiválasztott halmazát eltávolítjuk a képzési halmazból. A tételes regularizáció esetében a hálózat rögzíti a bemenetek futó átlagát és futó szórását, és ezeket használja az újonnan kapott bemenetek regularizálására. Ez jelentősen csökkenti a konvergencia sebességét és a túlillesztés valószínűségét.

## 2.5.Optimalizálók

A képzési fázisban az egyes súlyok hozzájárulását ehhez a hibához úgy becsüljük meg, hogy egy differenciálható hibafüggvény segítségével meghatározzuk a neurális hálózat kimenetének hibáját, majd meghatározzuk a hibafüggvény gradiensét a súlyok függvényében. Egyszerű esetben ezt a gradienst kivonjuk a súlyokból, ezáltal a súlyokat olyan irányba módosítjuk, amely garantáltan csökkenti a veszteségfüggvényt. Bizonyos esetekben azonban a gradiens önmagában lassan konvergálhat. Ilyen esetekben a gradiens ereszkedési algoritmus különböző kiterjesztései felgyorsíthatják a konvergenciát. Ezeket és más módszereket, amelyek nem veszik figyelembe a gradienst, optimalizálóknak nevezzük.

A gradiensillesztés a súlyokat a veszteségfüggvény gradienséből vonjuk le.

A lendület módszerek a gradiens mozgó átlagát (a "súrlódással" csökkentett sebességet) rögzítik, és ezt vonják ki a súlyokból, mintha egy labdát gurítanának lefelé a lejtőn. A Nesterov lendület hasonló a lendülethez, de először a tárolt lendület mentén tesz egy lépést, majd kiszámítja a gradienst, visszalép, ha a pontosság csökken. Az Adagrad egy adaptív gradiens. Minden egyes súlyhoz egyedi tanulási sebességet tárol, amely az adott súlyra eddig kiszámított összes gradiens négyzetösszegéből áll. Az aktuális tanulási rátával számított gradiens osztva van ennek az értéknek a gyökével, így a hosszú ideig alacsonyabb gradiensű súlyok nagyobb lépéseket tehetnek. Az RMSprop (root mean square propagation) Jeffrey Hinton adatkutató (a mélytanulás keresztapja) nem publikált módszere, amely a gradiens négyzetének mozgó átlagát veszi, és azt használja a súlyonkénti tanulási sebesség beállítására. A Resilient Propagation (Rprop) tanulási módszer adaptációja a részminták tanulására. Az Adam egy adaptív momentum módszer, az Adagrad és az RMSprop technikákat kombinálja, hogy a gradiens mozgó átlagát (momentum) és a gradiens négyzetének mozgó átlagát (memória) regisztrálja, és a súlyokat a memóriával módosított egyéni tanulási rátákkal megszorzott momentum alapján csökkenti. Ezt az optimalizálót általában új problémákra ajánlott először alkalmazni.

**3.Technológiák architektúrája és általános működése**

Az általam tesztelt technológiák architektúráját és általános működését szeretném bemutatni ebben a fejezetben.

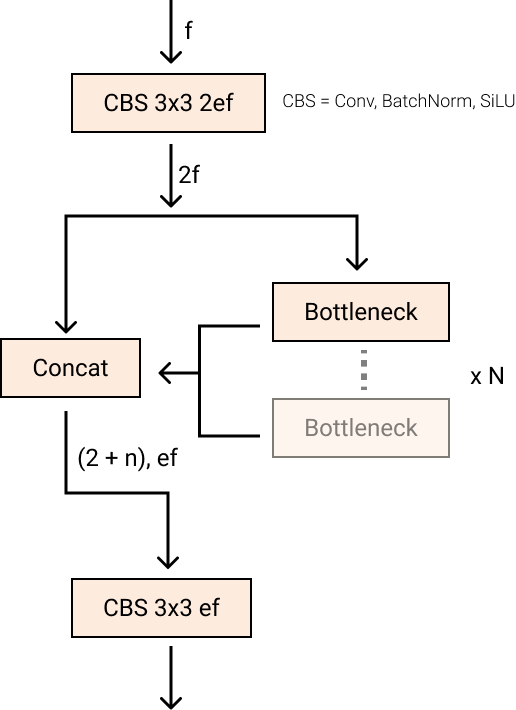
## 3.1.YOLOv8

Az Ultralytics csapata 2022-ben mutatta be YOLO modelljük legújabb verzióját (<https://github.com/ultralytics/ultralytics>; https://docs.ultralytics.com/).

A YOLOv8 a YOLO valós idejű objektumdetektorok sorozatának legújabb tagja, amely élvonalbeli teljesítményt nyújt a pontosság és a sebesség tekintetében. A YOLOv8 a korábbi YOLO-verziók fejlesztéseire építve új funkciókat és optimalizálásokat vezet be, amelyek ideális választássá teszik a különböző objektumérési feladatokhoz az alkalmazások széles körében.

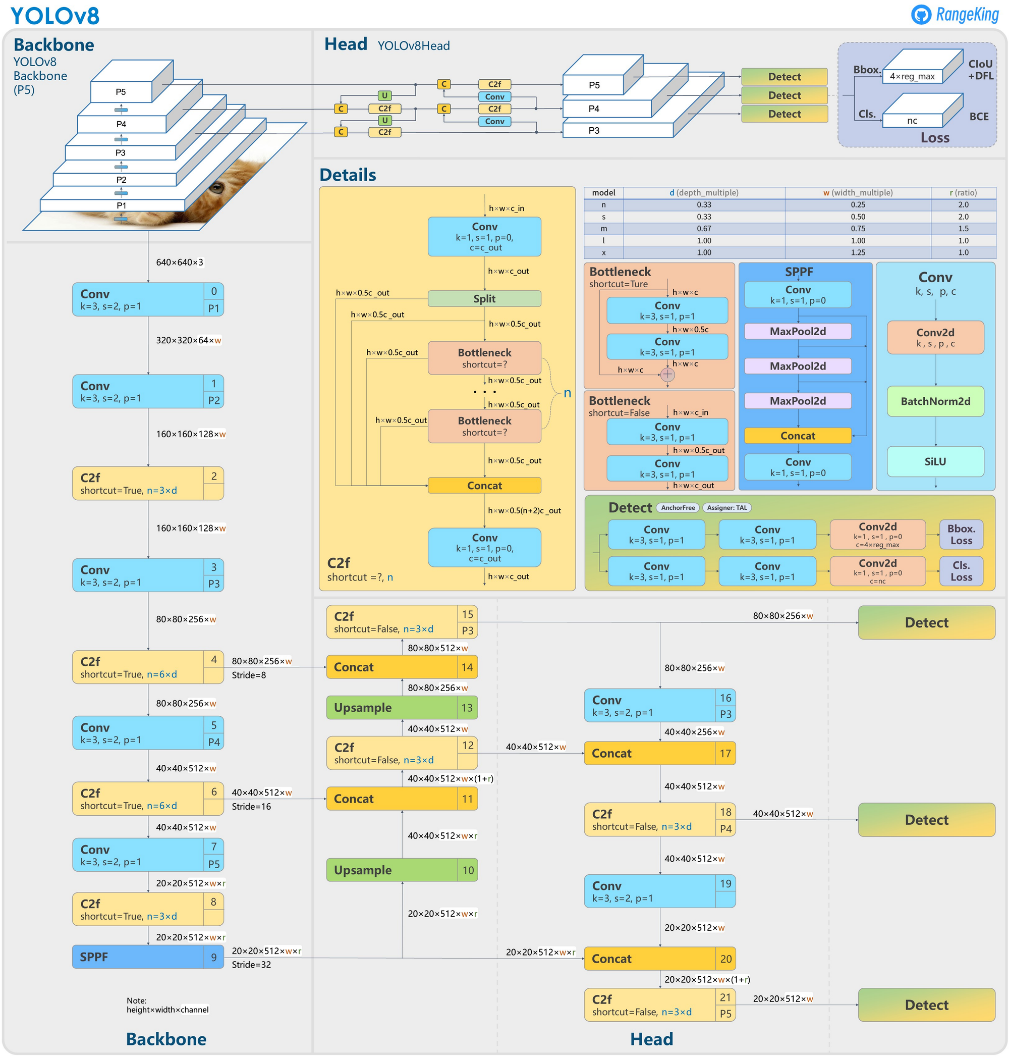
### 3.1.1.Architektúra

A YOLOv8 egy horgonymentes modell. Ez azt jelenti, hogy közvetlenül az objektum középpontját jósolja meg az ismert horgonydoboztól való eltolás helyett. A horgonydobozok a korábbi YOLO-modellek köztudottan trükkös részei voltak, mivel reprezentálhatják a cél benchmark dobozainak eloszlását, de nem az egyedi eloszlását.az adatkészletnek .A horgonymentes észlelés csökkenti a doboz-predikciók számát, ami felgyorsítja a nem maximális elnyomást (NMS), egy bonyolult utófeldolgozási lépést, amely a következtetések levonása után átszitálja a jelöltek észleléseit. A szár első 6x6 konv-ját egy 3x3-as váltja fel, a fő felépítési blokk megváltozott, és a C2f a C3 helyére került. A modult az alábbi kép foglalja össze, ahol "f" a szolgáltatások száma, "e" a bővítési sebesség, a CBS pedig egy Conv, BatchNorm és egy SiLU későbbi blokkja.



3.1.1.1.ábra: YOLOv8 cf2 modul (forrás: https://blog.roboflow.com/whats-new-in-yolov8/)

A C2f-ben a Bottleneck összes kimenete (fantasztikus neve két 3x3-as konvziónak maradék kapcsolatokkal) össze van fűzve. Míg a C3-ban csak az utolsó Bottleneck kimenetét használták. A szűk keresztmetszet ugyanaz, mint a YOLOv5-ben, de az első konvolúció kernelmérete 1x1-ről 3x3-ra módosult. Ebből az információból láthatjuk, hogy a YOLOv8 kezd visszatérni a 2015-ben definiált ResNet blokkhoz. A nyakban a funkciók közvetlenül vannak összefűzve anélkül, hogy ugyanazokat a csatornaméreteket kényszerítenék ki. Ez csökkenti a paraméterek számát és a tenzorok teljes méretét.YOLOv8 növeli a képeket az online tanítás során. Minden korszakban a modell egy kicsit más-más variációt lát a számára biztosított képeken. Az egyik ilyen augmentáció az úgynevezett mozaiknövelés. Ez magában foglalja négy kép összeillesztését, ami arra kényszeríti a modellt, hogy új helyeken, részleges elzáródásban és különböző környező pixelekkel szemben tanulja meg az objektumokat. Ez a kiegészítés azonban empirikusan kimutatta, hogy rontja a teljesítményt, ha a teljes képzési rutin során végrehajtják. Előnyös, ha kikapcsolva van az utolsó tíz tanítási időszakra.​



3.1.1.2.ábra: YOLOv8 architektúra (forrás: https://blog.roboflow.com/whats-new-in-yolov8/)

### 3.1.2.Tanítás

Az első használat során automatikusan letöltődnek az olyan szabványos adatkészletek, mint a COCO, a VOC és az ImageNet. A folyamat felgyorsítása érdekében a képzési erőfeszítéseket zökkenőmentesen skálázhatja több GPU-ra. Lehetőséget biztosít a hiperparaméterek módosítására YAML konfigurációs fájlokon vagy CLI-argumentumokon keresztül. A képzési metrikák valós idejű nyomon követése és a tanulási folyamat vizualizálása is elérthető a jobb betekintés érdekében.

A YOLOv8n képzése a COCO128 adathalmazon 100 epochán keresztül 640 képméret mellett. A tréningeszköz az eszköz argumentummal adható meg. Ha nem adunk meg argumentumot, akkor a GPU device=0-t fogja használni, ha van, egyébként device=cpu-t fog használni. A képzési argumentumok teljes listáját lásd az Arguments szakaszban.

from ultralytics import YOLO

# Load a model

model = YOLO('yolov8n.yaml') # build a new model from YAML

model = YOLO('yolov8n.pt') # load a pretrained model (recommended for training)

model = YOLO('yolov8n.yaml').load('yolov8n.pt') # build from YAML and transfer weights

# Train the model

results = model.train(data='coco128.yaml', epochs=100, imgsz=640)

3.1.2.1.forráskód: Single-GPU és CPU tanítás

### 3.1.3.Validáció

A modellek emlékeznek a képzési konfigurációkra az egyszerű validálás érdekében. CLI vagy a Python API közül is lehet választani a validálással kapcsolatos preferenciák alapján. Zökkenőmentesen működik a képzési fázis során használt adatkészletekkel, valamint az egyéni adatkészletekkel. A YOLO modellek validálási beállításai a különböző hiperparaméterekre és konfigurációkra vonatkoznak, amelyeket a modell teljesítményének validálási adathalmazon történő értékeléséhez használnak. Ezek a beállítások befolyásolhatják a modell teljesítményét, sebességét és pontosságát. Néhány gyakori YOLO validálási beállítás közé tartozik a kötegméret, a gyakoriság, amellyel a validálás a képzés során végrehajtásra kerül, valamint a modell teljesítményének értékelésére használt metrikák. A validálási folyamatot befolyásoló egyéb tényezők közé tartozik a validálási adathalmaz mérete és összetétele, valamint a konkrét feladat, amelyre a modellt használják. Fontos, hogy gondosan hangoljuk és kísérletezzünk ezekkel a beállításokkal annak biztosítása érdekében, hogy a modell jól teljesítsen a validálási adathalmazon, valamint a túlillesztés felismerése és megelőzése érdekében.

### 3.1.4.Predikció

A YOLOv8 különböző típusú bemeneti forrásokat tud feldolgozni következtetések levonásához

Képes következtetéseket levonni képek, videók, sőt, élő közvetítések alapján is. Valós idejű, nagy sebességű feldolgozásra tervezték a pontosság feláldozása nélkül. Intuitív Python- és CLI-felületek a gyors telepítéshez és teszteléshez. Különböző beállítások és paraméterek a modell következtetési viselkedésének az egyedi igények szerinti hangolásához.

Streaming funkció is elérhető az eredményobjektumok memóriahatékony generátorának létrehozásához. Engedélyezni ezt a stream=True beállításával a prediktor hívási metódusában lehet. Több kép vagy videokép egyetlen kötegben történő feldolgozására is képes, ami tovább gyorsítja a predikciós időt. A rugalmas API-nak köszönhetően könnyen integrálható a meglévő adatvezetékekkel és más szoftverkomponensekkel.

A Boxes objektum használható a határoló dobozok indexelésére, manipulálására és különböző formátumokba való konvertálására. A maszkokkal objektumot indexelni, manipulálni és szegmensekké alakítani lehet. A keypoint objektum használható a koordináták indexelésére, manipulálására és normalizálására is. Az Ultralytics YOLO modellek vagy egy eredményobjektumokból álló Python listát adnak vissza, vagy egy memóriahatékony Python eredményobjektum generátort, ha a következtetés során a stream=True adódik át a modellnek.

A Result objektumok plot() metódusát használhatja az előrejelzések megjelenítéséhez. Ez az eredményobjektumban található összes előrejelzési típust (dobozok, maszkok, kulcspontok, valószínűségek stb.) kirajzolja egy numpy tömbre, amely aztán megjeleníthető vagy elmenthető. A predikció során a szálbiztonság biztosítása kulcsfontosságú, ha több YOLO-modell fut párhuzamosan különböző szálakon. A szálbiztos következtetés garantálja, hogy az egyes szálak előrejelzései elszigeteltek és nem zavarják egymást, elkerülve a versenyfeltételeket és biztosítva a konzisztens és megbízható kimeneteket. Ha YOLO modellek többszálas alkalmazásban vannak használva, fontos, hogy minden szálhoz külön modellobjektumokat kell példányosítani, vagy a konfliktusok elkerülése érdekében szálhelyes tárolást kell alkalmazni.

from PIL import Image

from ultralytics import YOLO

# Load a pretrained YOLOv8n model

model = YOLO('yolov8n.pt')

# Run inference on 'bus.jpg'

results = model('bus.jpg') # results list

# Show the results

for r in results:

im\_array = r.plot() # plot a BGR numpy array of predictions

im = Image.fromarray(im\_array[..., ::-1]) # RGB PIL image

im.show() # show image

im.save('results.jpg') # save image

3.1.4.1.forráskód: példa az eredmények megjelenítésére

## 3.2.Zamba

2023-ban a DrivenData csapata kiadta az objektum detektálásra alkalmas programuk legfrissebb verzióját

(<https://github.com/drivendataorg/zamba>) (https://zamba.drivendata.org/docs/v2.1/).

A zamba egy Python-program, amely gépi tanulást és számítógépes látást használ az állatok automatikus felismerésére és osztályozására a kameracsapdás videókon. A zamba hivatalos modelljei képesek azonosítani az üres videókat (ahol nincsenek állatok), 32 Afrikában gyakori fajt és 11 Európában gyakori fajt. A felhasználók a saját megjelölt videóikkal finomhangolhatják a modellt, hogy új fajokat és új ökoszisztémákat jelezzenek előre. A Zamba parancssori eszközként vagy Python-csomagként is használható. A Zamba out of the box használható előrejelzések készítésére és modellek edzésére saját videók felhasználásával. a zamba ugyanazokat a videóformátumokat támogatja, mint az FFmpeg. az FFmpeg ellenőrzéseinek nem megfelelő videókat a következtetés és az edzés során kihagyja. A zamba "out-of-the-box" használható előrejelzések generálására és modellek betanítására a saját videók felhasználásával. a zamba ugyanazokat a videóformátumokat támogatja, mint az FFmpeg. az FFmpeg ellenőrzéseinek nem megfelelő videók a következtetés és a betanítás során kihagyásra kerülnek.

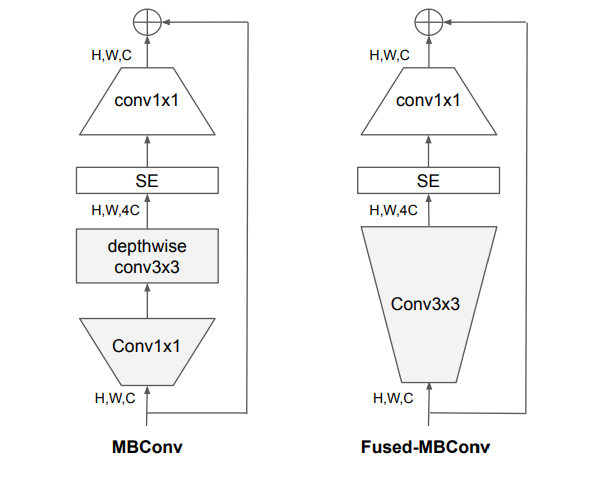
### 3.2.1.Architektúra

Az idő\_elosztott modellt a jól ismert képosztályozó architektúra EfficientNetV2 (Tan, M., & Le, Q., 2019) átképzésével építették fel, amely a kamerával rögzített videókban történő fajazonosításra szolgál.

Az EfficientNetV2 modell egy konvolúciós neurális hálózat, amelyet úgy terveztek, hogy együttesen optimalizálja a modell méretét és a tanulási sebességet; az EfficientNetV2 képnatív, és minden egyes képkockát külön-külön osztályoz a jóslatok generálásakor. Széles körben használja mind az MBConv-t, mind az újonnan hozzáadott fused-MBConv-t a korai rétegekben. A kisebb bővítési arányokat részesíti előnyben az MBConv esetében, mivel a kisebb bővítési arányok általában kevesebb memória-hozzáférési ráfordítással járnak. A kisebb 3x3-as kernelméreteket részesíti előnyben, de több réteget ad hozzá, hogy kompenzálja a kisebb kernelméretből adódó csökkent befogadói mezőt. Végül teljesen eltávolítja az utolsó lépés-1 szakaszt az eredeti EfficientNetben, a nagy paraméterméret és a memória-hozzáférési többlet miatt.

Az MBConv használata az EfficientNetV2 korai rétegeiben jelentős, mivel ezekben a rétegekben történik a legtöbb számítás. Az MBConv (Mobile Inverted Residual Bottleneck Convolutional Block) egyfajta konvolúciós blokk, amely három kulcsfontosságú összetevőből áll: a fordított szűk keresztmetszetből, a mélységi konvolúcióból és a pontirányú konvolúcióból. Az invertált szűk keresztmetszet a bemeneti tenzor csatornáinak csökkentéséért felelős, míg a mélységi konvolúció a tenzor minden csatornájára külön szűrőt alkalmaz. Végül a pontszerű konvolúció újra kiterjeszti a tenzort több csatornára. Az MBConv korai rétegekben történő használatával az EfficientNetV2 képes optimalizálni az ezekben a rétegekben előforduló számításokat, ami gyorsabb képzési sebességet és általános jobb paraméterhatékonyságot eredményez.

A modell egy TimeDistributed rétegbe van csomagolva, ami videónként egyetlen előrejelzést tesz lehetővé.



3.2.1.1.ábra: EfficientNetV2 architektúrája (forrás: <https://towardsdatascience.com/efficientnetv2-faster-smaller-and-higher-accuracy-than-vision-transformers-98e23587bf04>)

### 3.2.1.Modell

A zamba csomag három modellt tartalmaz: time\_distributed, slowfast és european. Ha a kameravideó Közép- vagy Nyugat-Afrikában gyakori fajokat tartalmaz, a time\_distributed vagy slowfast modellt érdemes használni. Ha a videó Európában gyakori fajokat tartalmaz, az european modellt ajánlott. Hozzá kell adni a modell nevét a parancshoz. Ha nincs megadva modell, a time\_distributed modellt használja a rendszer. Az Európai modell a Nyugat-Európában gyakori 11 faj azonosítására lett képzeve. A lehetséges osztálycímkék a következők: madár, házimacska, európai borz, európai hód, európai nyúl, európai szarvas, észak-amerikai mosómedve, vörös róka, menyét, vadmacska, vaddisznó. Az európai modell egy betanított idő\_eloszlású modellel indul, és megtanulja a végső kimeneti réteg helyettesítését az európai fajok előrejelzéséhez.

Az európai modell finomhangolása a Halle-Jena-Lipcsei Német Biológiai Sokféleség Integrációs Kutatóközpont (iDiv) és a Max Planck Evolúciós Antropológiai Intézet partnerei által gyűjtött és annotált adatokkal történik. A finomhangolt adatok között szerepelnek a németországi Hintenteiche bei Biesenbrowból származó kameracsapdás felvételek. Az európai modell ugyanazt a képkockaválasztást használja, mint a time\_distributed modell. Alapértelmezés szerint a MegadetectorLite nevű hatékony objektumfelismerő modell fut le az összes képkockán annak meghatározására, hogy mely képkockák tartalmaznak nagy valószínűséggel állatokat. Az EUROPEAN ezután csak a 16 legnagyobb észlelési valószínűségű képkockán fut le. Alapértelmezés szerint egy képkocka kiválasztása után a videó 240x426 pixelesre méreteződik át. Mint minden modellnél, itt is választhatóak különböző keretkiválasztási módszerek, és a használt keretméret is megváltoztatható egy egyéni YAML konfigurációs fájl átadásával. Az európai modell esetében az egyetlen követelmény, hogy a videóletöltő 16 képkockát adjon vissza.

A zamba egy .csv fájlt fog kiadni, amelyben minden egyes videó fájlnévnek van egy sora, és minden egyes osztály (azaz faj) egy oszlopot tartalmaz. Az alapértelmezett előrejelzés minden osztályra valószínűségeket tárol, így az (i,j) cella annak a valószínűsége, hogy j állat jelen van az i videóban. A globális előrejelzések akkor hasznosak, ha egy videó egynél több fajt tartalmaz. A predikciók alapértelmezés szerint az aktuális munkakönyvtárban lévő zamba\_predictions.csv fájlba kerülnek elmentésre. Alternatívaként a --save-dir argumentummal a predikciókat egy másik mappába is el lehet menteni.

A kimeneti osztálynevek (--output-class-names) argumentum hozzáadása egyszerűsíti a predikciókat, és minden videóhoz csak a legvalószínűbb állatokat adja vissza.

### 3.2.2.Predikció

A predikciók formátumára három lehetőség van. A minden valószínűség tárolása a predikciókat minden egyes fájlnévhez egy sorral és minden egyes osztálycímkéhez egy oszloppal adja vissza, 0 és 1 közötti valószínűségekkel. Az (i,j) cella annak a valószínűsége, hogy a j állat jelen van az i videóban. A jelenlét/hiány minden egyes fájlnévhez egy sor előrejelzést és minden egyes osztálycímkéhez egy oszlopot ad vissza, ahol a cella a felhasználó által megadott valószínűségi küszöbérték alapján jelzi a jelenlétet vagy a hiányt. Az (i, j) cella azt jelzi, hogy a j állat jelen van-e (1) vagy nincs (0) az i videóban. A valószínűségi küszöbérték a CLI --proba-threshold parancsával adható meg. A legvalószínűbb osztály előrejelzéseket ad vissza, egy sort minden egyes fájlnévhez és egy oszlopot minden egyes videó legvalószínűbb osztályához. A legvalószínűbb osztály lehet üres. A legvalószínűbb osztály megadásához hozzá kell adni a --output-class-names opciót a parancshoz; Pythonban ez megadható az output\_class\_names=True hozzáadásával a PredictConfig példányosításakor. Ez nem ajánlott, ha egynél több fajt kell észlelni minden egyes videóban.

A mintavideók nevű mappában található videókat a lehető legegyszerűbben osztályozták az alapértelmezett beállítások használatával.

Minimális példa a parancssori előrejelzésre:

zamba predict --data-dir example\_vids/

3.2.2.1.forráskód:video osztályozása

A zamba predict parancssori futtatásához megadták a -data-dir és/vagy -filepaths értéket. A data-dir PATH videókat tartalmazó mappa elérési útvonala. Ha nincs megadva a filepaths is, a Zamba rekurzívan ebben a mappában fogja keresni a videókat. A filepaths PATH egy olyan CSV-fájl elérési útvonala, amely minden egyes osztályozandó videó elérési útvonalának oszlopát tartalmazza, a CSV-fájlnak tartalmaznia kell a fájl elérési útvonalak oszlopát. A filepaths lehet egy abszolút elérési út a rendszeren, vagy a --data-dir paraméter által megadott adatkönyvtárhoz viszonyított út.

A becsléshez használandó modell kiválasztásához a -model vagy --checkpoint jelzőt kell megadni; a zamba-val együtt szállított három előre elkészített modell egyikének megadásához a -model paramétert kell használni. A -model alapértelmezett beállítása a time\_distributed. A zamba egy .csv fájlt ad ki, amelyben minden egyes videófájl nevéhez egy sor és minden osztályhoz (pl. fajhoz) egy oszlop tartozik. Az alapértelmezett előrejelzés minden osztályra valószínűségeket tárol, így az (i,j) cellát úgy lehet értelmezni, mint annak a valószínűségét, hogy a j állat jelen van az i videóban.

## 3.3.Yolov7

A WongKinYiu csapata 2022-ben bemutatta yolov7 nevű megoldását. A YOLOv7 a maga idejében a legfejlettebb valós idejű objektumdetektor volt, amely mind sebességben, mind pontosságban felülmúlt minden ismert objektumdetektort az 5 FPS és 160 FPS közötti tartományban. A YOLOv7 büszkélkedhet a GPU V100-on a egyik legnagyobb pontossággal (56,8% AP) minden ismert valós idejű objektumdetektor közül, több mint 30 FPS sebességgel. A YOLOv7 továbbá sebesség és pontosság tekintetében felülmúlja a YOLOR, YOLOX, Scaled-YOLOv4, YOLOv5 és számos más objektumdetektort. A modellt az MS COCO adathalmazon képezték ki a semmiből, más adathalmazok vagy előre betanított súlyok használata nélkül.

### 3.3.1.Architektúra

A YOLOv7 szerzői megkísérelték a tárgyfelismerés csúcsteljesítményét létrehozni egy olyan hálózati architektúra létrehozásával, amely a hasonló, hasonló következtetési sebességű hálózatoknál pontosabban jelzi előre a határoló dobozokat. Ezen eredmények elérése érdekében a YOLOv7 szerzői számos változtatást hajtottak végre a YOLO-hálózaton és a képzési rutinokon. A következőkben a YOLOv7-en végzett munka során a számítógépes látáskutatás területén elért három figyelemre méltó hozzájárulást ismertetjük WongKinYiu elindult a maximális réteghatékonysághoz vezető úton a keresztlépcsős részhálózatokban.

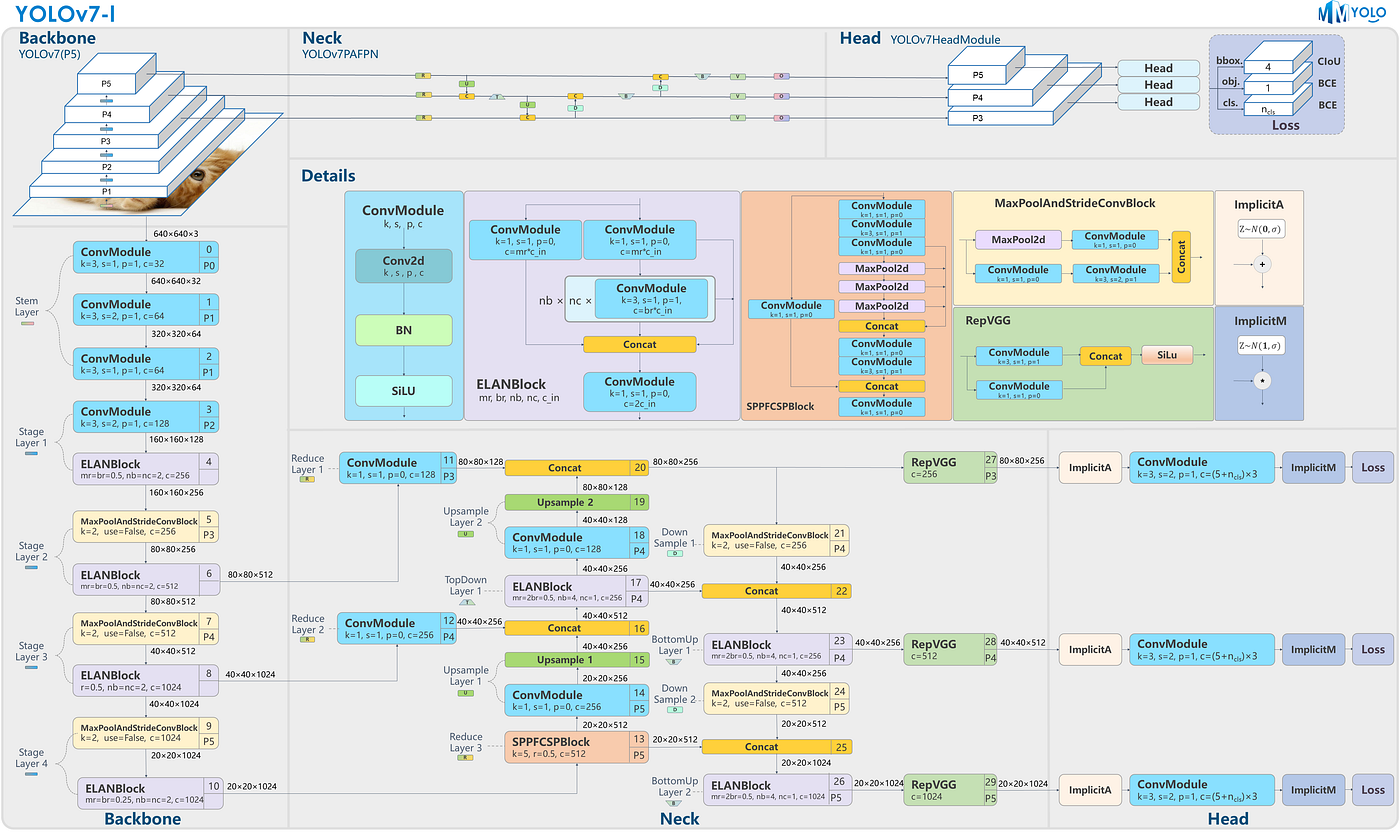
A YOLOv7 az e területen végzett kutatásokra épül, szem előtt tartva a rétegek memóriában tartásához szükséges memória mennyiségét és a gradiensnek a rétegeken való terjedéséhez szükséges távolságot. Minél rövidebb a gradiens, annál robusztusabban tud tanulni a hálózat. Az általuk végül választott rétegaggregáció az E-ELAN, az ELAN számítási blokk kiterjesztése.A tárgyfelismerő modelleket általában modellek sorozataként ábrázolják, amelyek méretét skálázzák, mivel a különböző alkalmazások különböző pontosságot és következtetési sebességet igényelnek.

A tárgyfelismerő modellek általában figyelembe veszik a hálózat mélységét, szélességét és a felbontást, amelyen a hálózatot képzik; a YOLOv7-ben a szerzők a hálózat mélységét és szélességét együtt skálázzák, miközben összekapcsolják a rétegeket. Az ablációs vizsgálatok azt mutatják, hogy ez a megközelítés optimálisnak tartja a modell architektúráját, miközben különböző léptékekre skálázódik.

Az újraparaméterezési technikák átlagolják a modell súlyait, hogy a modellezendő általános mintázathoz egy robusztusabb modellt hozzanak létre. A legújabb kutatások a modulszintű reparaméterezésre összpontosítottak, ahol a hálózat egyes részei saját reparaméterezési stratégiával rendelkeznek.A YOLOv7 szerzői gradiens áramlási terjedési utakat használnak annak meghatározására, hogy a hálózat mely moduljai használnak reparaméterezési stratégiát, és melyek nem.

A YOLO hálózati fej készíti a hálózat végső előrejelzéseit, de nagyon messze helyezkedik el a hálózatban, ezért előnyös lehet egy segédfej hozzáadása valahol a hálózat közepén. A képzés során ezt az észlelőfejet éppúgy figyeljük, mint a tényleges előrejelzést végző fejet.

A YOLOv7 szerzői ezért különböző szintű megfigyeléssel kísérleteztek e fej esetében, és egy olyan durvától finomig terjedő definícióban állapodtak meg, amely a vezető fej megfigyelését különböző szemcseméretekben adja vissza.



3.3.1.1.ábra: Yolov7 architektúrája (forrás: https://medium.com/@nahidalam/understanding-yolov7-neural-network-343889e32e4e)

### 3.3.2.Adatkészlet

A LabelImg egy nyílt forráskódú grafikus képmegjelölő eszköz, amelyet a képek objektumhatároló dobozainak címkézésére használnak. Egyszerűen használható felületet biztosít a képek objektumfelismerő címkékkel való megjegyzéséhez. A LabelImg különböző formátumokat támogat, például a Pascal VOC, a YOLO és a Tensorflow formátumokat. Az eszköz Python és Qt nyelven íródott, és több platformon is elérhető, többek között Windows, Linux és macOS alatt. A kutatók és a szakemberek széles körben használják az objektumfelismerő modellek képzéséhez szükséges adathalmazok létrehozására. Miután annotálta az adathalmazát, használhatja azt az objektumfelismerő modelljének képzésére egy olyan keretrendszerrel, mint a Yolov7.

Az annotációs folyamat megkezdéséhez beolvashatja az összes annotációs fájlt, és az adatokat ezekből a fájlokból kiveheti a korábban létrehozott adatkeretbe. Ezt úgy érhetjük el, hogy az os könyvtár segítségével végigmegyünk az annotációs könyvtáron, és az ElementTree könyvtár segítségével beolvasunk minden egyes fájlt. Miután az adatokat kinyertük, azokat a korábban létrehozott adatkeretben tárolhatjuk. Ez a folyamat kritikus fontosságú az adathalmaz előkészítésében a RetinaNet modell képzéséhez, és pontosan kell elvégezni. Ez a következő módon végezhető el:

A fenti kódrészlet végigmegy az összes XML-jelölési fájlon a jelölési mappában, kivonja az osztálycímkéket, a határoló doboz koordinátáit és a megfelelő fájlnevet, majd hozzáadja őket az adatkerethez. Az "if" feltétel ellenőrzi, hogy a fájlkiterjesztés "xml". A megfelelő kép fájlnevét úgy kapjuk meg, hogy a ".xml" kiterjesztést ".jpg"-re cseréljük. A határoló doboz koordinátáit és az osztálycímkéket ezután az ElementTree (ET) elemzése segítségével nyerjük ki. Végül a kinyert adatokat az adatkerethez csatoljuk. Ez lehetővé teszi a RetinaNet modell képzéséhez szükséges összes információ egyszerű és hatékony összegyűjtését, mivel az adatkeret könnyen átalakítható CSV-fájlba, amelyet a modell képzéséhez használunk.

Az adatkeret létrehozása után a data.head() parancs segítségével megtekinthető. Ahhoz, hogy ezeket az adatokat a RetinaNet modell képzéséhez használhassuk, át kell konvertálni őket CSV-fájlba. Ezt a következő paranccsal lehet megtenni az indexek és fejlécek eltávolítása érdekében:

A fenti kódban az open() függvényt egy új, '../handDetectorClasses.csv' nevű fájl létrehozására használjuk, amelynek fájlmódja 'w'. A fájl írási módban van megnyitva, ami lehetővé teszi az adatok írását. A 'w' mód felülírja a fájlt, ha az már létezik.

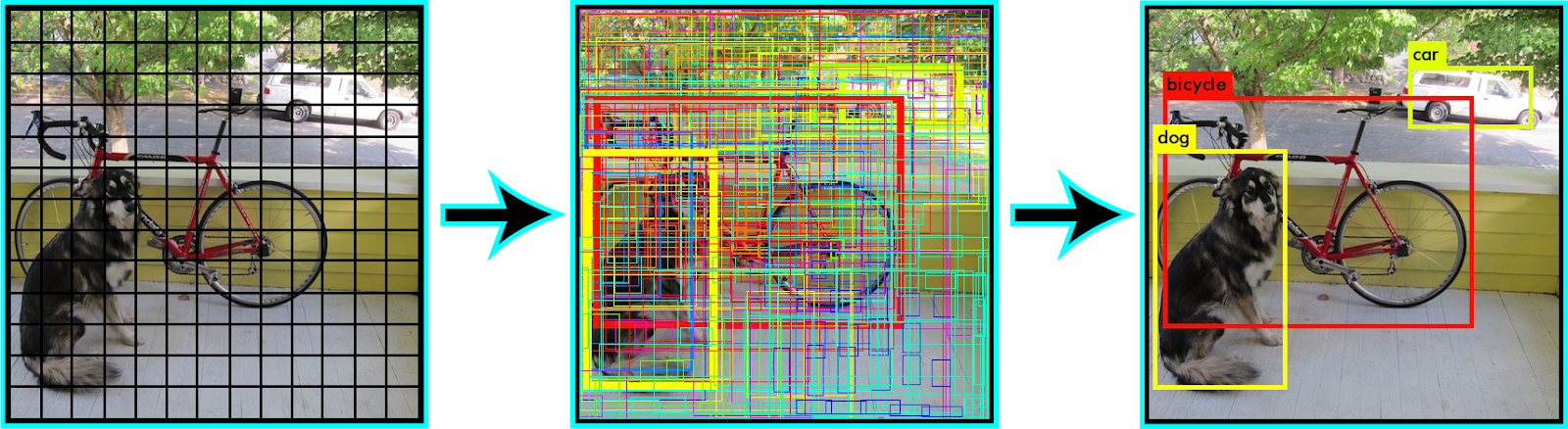
A for ciklus végigmegy az osztályok listájának minden egyes egyedi osztálynevén, és vesszővel elválasztva írja az osztály nevét és a hozzá tartozó indexet a fájlba. Az egyes sorok végén lévő \n karakter jelzi az új sort a fájlban. Ez a fájl arra szolgál, hogy a Yolov7 modell számára az osztályneveket osztályindexekhez rendelje a képzési folyamat során.

### 3.3.3.Predikció

A YOLO először N rácsra osztja a képet, mindegyik rács mérete SxS. Mindegyik rács azonos méretű SxS, és minden egyes régiót a benne található lehetséges objektumok felderítésére és helyének meghatározására használunk. Minden egyes rács esetében egy lehetséges objektum B határoló dobozának koordinátáit az objektumcímke és a megjósolt objektum jelenlétére vonatkozó valószínűségi pontszám alapján jósoljuk meg.

Mint elképzelhető, ez a rácsok kumulatív előrejelzései alapján a megjósolt objektumok jelentős redundanciájához vezet. Ennek a redundanciának a kezelésére és a megjósolt objektumok érdeklődésre számot tartó objektumokra való redukálására a YOLO nem maximális elnyomást alkalmaz, amely elnyomja a viszonylag alacsony valószínűségi pontszámú határoló dobozokat.

Ehhez a YOLO először összehasonlítja az egyes döntésekhez tartozó valószínűségi pontszámokat, és kiválasztja a legmagasabb pontszámot. Ezután eltávolítja azt a határoló négyzetet, amelynek az összege és a kiválasztott nagy valószínűségű határoló négyzet között a legnagyobb a metszéspont. Ezt a lépést addig ismétli, amíg csak a végső kívánatos határoló doboz marad.



3.3.3.1.ábra: Yolov7 detektálása egy fotón (forrás: https://blog.paperspace.com/yolov7/)

# 4.Programok megvalósítása

A korábban részletezett három technológiát a gyakorlatban is leteszteltem. Ebben a fejezetben szeretném bemutatni, hogyan valósítottam meg a programokat.

## 4.1.Általánosan felhasznált eszközök

Programozási nyelvnek a Python 3.10-et választottam, ehhez fejlesztő környezetnek a PyCharm Community Edition 2022.3.3 programot és az Anaconda tool-ot használtam. Választásaim oka, hogy ezek ingyenes és népszerű programok.

A Pycharm esetében könnyedén le lehet tölteni a különféle projektekhez szükséges csomagokat és fórumokon és weboldalakon segítséget lehet kérni más fejlesztőktől. El van látva több hasznos funkcióval, mint például intelligens kódkiegészítés, a hibák menet közbeni ellenőrzése és gyorsjavítása, könnyű projektnavigáció. Az IDE segít a minőség ellenőrzésében a PEP8 ellenőrzésekkel, a tesztelési asszisztenssel, az intelligens refaktorálásokkal és számos ellenőrzéssel.

Az Anaconda egy nyílt forráskódú csomag- és környezetkezelő rendszer, amely Windows, macOS és Linux rendszereken is fut. Csomagok és függőségeik telepítésére, futtatására és frissítésére is alkalmazható.

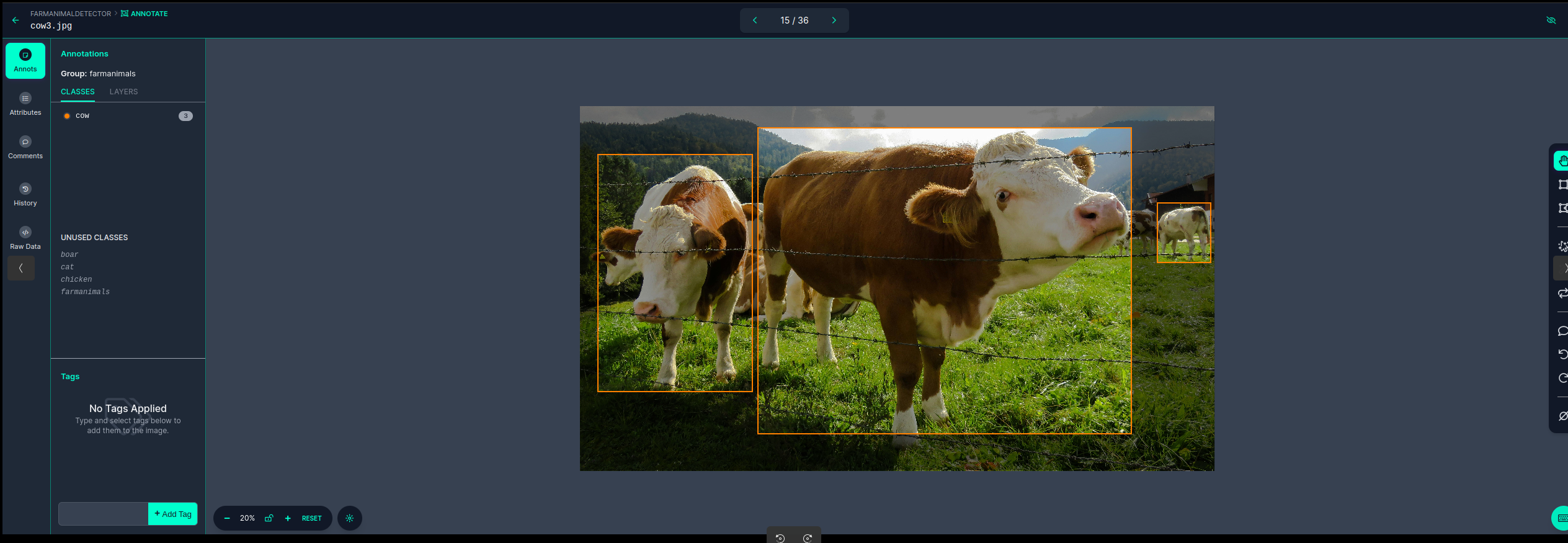
Cuda verzió száma 11.7., processzorom egy Ryzen 5 1600-as CPU (6 fizikai mag, 12 logikai szál, 3.2 Ghz-3.6 Ghz), memória 24 Gb RAM (3 db 8 Gb-os, 2400 Mhz-es, cl15-ös, DDR4-es modul), videókártya egy Rtx 3060 (8 Gb memória), mindez egy Ubuntu 22.04-es operációs rendszeren, egy M.2 SSD-én.

## 4.2.Saját adatkészlet készítése

Próbaképpen készítettem egy saját adatkészletet is. Ezt a Roboflow oldalán tudtam megtenni (<https://app.roboflow.com/>). A Roboflow egy olyan eszköz, amely megkönnyíti a számítógépes látás mélytanulással kapcsolatos munkáját. Lehetővé teszi a fejlesztők számára, hogy képességeiktől és tapasztalatuktól függetlenül számítógépes látó alkalmazásokat hozzanak létre. Támogatja az objektumdetektálási és osztályozási modelleket. Az eszköz képes adatkészlet előfeldolgozásra, projekt/adatkészlet összevonásara, adatkészlet állapotának ellenőrzésére, adatkészletek exportálására és modellképzésre.

Létre kell hozni egy verziót az adathalmazból a modell betanításához. A verziók az adatkészlet egy adott időpontban készült pillanatfelvételei. Ezeket a verziókat azért tartjuk meg, mert lehetővé teszik az eredmények reprodukálhatóságának fenntartását azáltal, hogy pontosan nyomon követhető, hogy a modell minden egyes iterációjában mely képeket, előfeldolgozási és telepítési lépéseket használtuk. Ez lehetővé teszi a különböző modellek és keretrendszerek tudományos tesztelését, ugyanakkor biztosítja a bizalmat abban, hogy az eredmények a modellben bekövetkezett változásoknak, nem pedig hibáknak vagy az adatvezetés változásainak köszönhetőek. Az adatkészlet verziójának létrehozásához kattintson a "Version" oldalsávjában található Roboflow projektre. kattintson az "Create new version" gombra.

Ezen az oldalon állíts be a train/test/valid felosztást, és adja meg az előfeldolgozási szoftvert és bővítményeket az új adatkészlet verziójához. Miután megadta az adatokra alkalmazni kívánt előfeldolgozási lépéseket és kiegészítéseket, kattintson a "Generate" gombra. Ez egy új adatkészletet hoz létre. Ezt az adathalmazt használhatja a modellek képzéséhez a Roboflow-ban. Az adatkészletet exportálhatja és kézi modellképzéshez is használhatja. A verziókezelési folyamat során a képzési, validálási és tesztkészletek eloszlását is kiegyensúlyozhatja. Ehhez lépjen a "2. step: Train/Test Split" menüpontra, és kattintson az "Rebalance" gombra.



4.2.1.ábra: képek annotálása

A képcímkék javítják az adatfájlok szervezését. A Roboflow lehetővé teszi a képek címkézését. Ezek a címkék felhasználhatók kötegelt címkézéshez, hogy pontosabban lehessen képeket kijelölni a címkézéshez. A képcímkék paraméterként is használhatók a Roboflow adathalmaz-kereséséhez.Az adatok feltöltésekor a címkék mind a webes felület, mind a REST API és a Python SDK segítségével hozzárendelhetők. Az alábbi képen látható, hogyan adhatók meg a feltöltött képekhez társítandó címkék a webes felület segítségével. A RoboFlow műszerfalon a képek címkékkel szűrhetők a hozzárendelés oldalon. A címkék alapján történő szűrés előfeldolgozási lépését arra is használhatja, hogy a képek egy részhalmazán verziókat hozzon létre. Ez lehetővé teszi, hogy kizárja (vagy megkövetelje) az adott címkékkel rendelkező képeket a képzésben való felhasználáshoz. Címke alkalmazásához kattintson a Képek lapon arra a képre, amelyre a címkét alkalmazni szeretné, majd kattintson az oldal jobb felső sarkában található Kijelölt képek gombra, és kattintson a Címke alkalmazása gombra. Ezzel tetszőleges számú címkét alkalmazhat a kiválasztott képekre.

Először adjuk hozzá a képeket az ”Add … images to Dataset” menüpontra kattinta az adatkészletünkhöz. Ezek után A ”Generate” menüpontban a rendszer legenerálja nekünk az egyéni adatkészletünket.

Végezetül az ”Export Dataset” menüpontra kattintva kiexportálhatjuk adatkészletünket kód vagy zip fájl formátumban. Lehetőségünk van az adatkészlet formátumát is beállítani, plédául yolov8, yolov7 pytorch, coco, coco-MM, tensorflow és egyéb népszerű formátumokra.

## 4.3.Yolov8 telepítése

A programot Computer vision engineer munkája alaján készítettem el: <https://github.com/computervisioneng/object-tracking-yolov8-native>

A saját munkám: <https://github.com/AASZ40/Allatok-detektalasa-felvetelrol-mesterseges-intelligencia-segitsegevel/tree/main/yolov8>

Elsőnek a pytorchot telepítettem fel a conda terminálban:

conda install pytorch torchvision torchaudio cpuonly -c pytorch

4.3.1.forráskód:pytorch telepítése

Ezek után feltelepítettem az ultralytics csomagot, amely magában foglalja a yolov8-at is.

conda install -c conda-forge ultralytics

4.3.2.forráskód:ultralytics telepítése

### 4.3.1*.*Tanítás és detektálás

Létrehoztam egy main.py nevű fájlt, ahol meghívtam a cv2 class-t (Opencv, ez elengedhetetlen a vizualizációhoz) és a YOLO class-t az ultralytics csomagból.

Ezek után betöltöttem a yolov8 modelljét.

model = YOLO('yolov8n.pt')

4.3.1.1.forráskód:yolov8 modell betöltése

Ha szeretnénk tanítani saját adatkészletünkkel, szükségünk lesz az adatkészlet yaml fájljára, mert ez tartalmazza a tanítási, validációs és tesztelési képek labeljeinek koórdinátáit és a címkéket.

results = model.train(data='data.yaml', epochs=100, imgsz=640)

4.3.1.2.forráskód:tanítás folyamata

A tesztelési videókat ugyan abba a mappába illesztettem be, mint ahol a main.py fájl is található. Az Opencv segítségével képes kezelni a videófájlt. Ezután betöltöttem a célvideót:

video\_path = './cowsp4'

cap = cv2.VideoCapture(video\_path)

ret = True

4.3.1.3.forráskód:Opencv használata

Ezek után beolvasásra kerülnek a frame-k, ahol detektálja és track-eli az objektumokat a videóról. Az Opencv segítségével az eredményeket megjeleníti a frameken felirattal és jelölő négyzetekkel. Végezetül vizualizálja a detektált videót és megjeleníti.

while ret:

ret, frame = cap.read()

if ret:

# detect objects

# track objects

results = model.track(frame, persist=True, save=True, save\_crop=True)

# plot results

# cv2.rectangle

# cv2.putText

frame\_ = results[0].plot()

# visualize

cv2.imshow('frame', frame\_)

if cv2.waitKey(25) & 0xFF == ord('q'):

break

4.3.1.4.forráskód:detektálás folyamata

Az eredményeket a runs/detect/track vagy runs/detect/train mappába menti le.

## 4.4.Zamba telepítése

A programot DrivenData.org munkája alapján készítettem el: https://github.com/drivendataorg/zamba?tab=readme-ov-file

A saját munkám: https://github.com/AASZ40/Allatok-detektalasa-felvetelrol-mesterseges-intelligencia-segitsegevel/tree/main/zamba

A program működéséhez elengedhetetlen az Ffmpeg nyílt forráskódú könyvtár a különböző kodekek videóinak betöltésére. Az FFmpeg használata azt jelenti, hogy a zamba rugalmas lehet a támogatott videóformátumok tekintetében. Az FFmpeg minden különböző platformra telepíthető, de a platformtól függően némi további konfigurációt igényel.

sudo apt update

sudo apt install ffmpeg

4.4.1.forráskód:Ffmpeg telepítése

A zamba telepítéséhez a következő kódot írtam be a conda terminálba:

pip install <https://github.com/drivendataorg/zamba/releases/latest/download/zamba.tar.gz>

4.4.2.forráskód:zamba telepítése

Linux rendszeren szükséges lehet egyéb csomagok és build-alapok:

apt-get update && \ apt-get install -y software-properties-common && \ add-apt-repository -y ppa:savoury1/ffmpeg4 && \ apt-get update && \ apt-get -y install \ build-essential \ ffmpeg \ git \ libavcodec-dev \ libavdevice-dev \ libavfilter-dev \ libavformat-dev \ libavutil-dev \ libsm6 \ libswresample-dev \ libswscale-dev \ libxext6 \ pkg-config \ wget \ x264 \ x265

4.4.3.forráskód:require-ök telepítése

### 4.4.1.Zamba általános használata

A zamba "out of the box" használható előrejelzések generálására és modellek betanítására a videók felhasználásával. A következtetés elvégzéséhez egyszerűen futtassa a zamba predict-et, és adjon át argumentumokat a zamba-nak, amelyekkel tájékoztatja a videó helyéről, a használandó modellről és arról, hogy hol tárolja a kimenetet. A modellképzéshez hasonlóan a zamba train is futtatható a címkék megadására. A következő szakaszok részletesebben ismertetik ezeket a különálló modulokat.

A zamba csomaggal kétféleképpen lehet interakcióba lépni. A zamba parancssori interfész eszközként való használata. Ez az oldal áttekintést nyújt a CLI használatáról.

A zamba importálása Pythonban és használata Python csomagként. Ez a bemutató a CLI-t használja, de mind a CLI-t, mind a Pythont a prediktív és a képzési bemutatóban ismertetjük.

Megadhatja a videók könyvtárának elérési útvonalát, vagy megadhatja a fájlútvonalak listáját egy .csv fájlban. zamba ugyanazokat a videóformátumokat támogatja, mint az FFmpeg, amelyek itt vannak felsorolva. Az Ffmpeg ellenőrzéseinek nem megfelelő videókat a következtetés a képzés során kihagyja. A tesztelendő videókat aegy example\_vids nevű mappába helyeztem.

Ha az alapértelmezett beállításokkal szeretne előrejelzéseket generálni és menteni a videókhoz, futtassa a következőt:

zamba predict --data-dir example\_vids/

4.4.1.1.forráskód:detektálás folyamata

A Zamba egy .csv fájlt ad ki, amelyben minden egyes videófájl neve egy sorban, az egyes osztályok (azaz fajok) pedig egy oszlopban szerepelnek. Az alapértelmezett előrejelzés minden osztályra valószínűségeket tárol, így az (i,j) cella annak a valószínűsége, hogy a j állat jelen van az i videóban. A globális előrejelzések akkor hasznosak, ha egy videó egynél több fajt tartalmaz. A predikciók alapértelmezés szerint az aktuális munkakönyvtárban lévő zamba\_predictions.csv fájlba kerülnek elmentésre. A --save-dir argumentummal a predikciókat egy másik mappába is el lehet menteni. A -output-class-names argumentum hozzáadása leegyszerűsíti a jóslatokat, és minden videóhoz csak a legvalószínűbb állatot adja vissza.

A zamba számára elérhető előre elkészített modellek a blank\_nonblank, time\_distributed, slowfast és european. A prioritástól és a földrajzi elhelyezkedéstől függ, hogy melyik modellt használjuk (további információkért lásd az Elérhető modellek oldalát). Alapértelmezés szerint a zamba a time\_distributed modellt használja. Ha más opciókat szeretne megadni, adja hozzá a -model argumentumot

zamba predict --data-dir example\_vids/ --model european

4.4.1.2.forráskód:detektálás az european modellel

### 4.4.2.Zamba tanítása

A zambával együtt szállított modellek egyikének trainelését folytatni lehet két módon.

Finomhangolással, azaz a fajokat tartalmazó címkézett videók hozzáadásával a zamba osztálycímkék listájához. Vagy finomhangolni új fajokat tartalmazó címkézett videókkal.

Mindkét esetben a képzéshez szükséges parancsok ugyanazok: tegyük fel, hogy a example\_vids mappában lévő videók címkéit a example\_labels.csv fájlban tároljuk. A modell edzéséhez tegye a következőket:

zamba train --data-dir example\_vids/ --labels example\_labels.csv

4.4.2.1.forráskód:tanítás label-özött videókon

A címkefájlnak tartalmaznia kell a filepath és a címke oszlopokat is. A filepath oszlopnak tartalmaznia kell egy abszolút elérési utat vagy az adatkönyvtárhoz viszonyított elérési utat. Opcionálisan a split (train, val, holdout) és a site oszlopok is megadhatók.

cat example\_labels.csv

filepath, label

4.4.2.2.forráskód:labe-özött videók objektumainak azonosítása

A kódsor alá fel kell sorolni a címkézett videókat és vesszővel és szóközzel elválasztva oda kell írni a videóban címkézett objektumok nevét. Alapértelmezés szerint a betanított modell és a további képzési kimenet az aktuális munkakönyvtárban lévő version\_n mappába kerül mentésre.

## 4.5.Yolov7 használata a Google colab-ban

A Yolov7 technológiájára épülő programot a Google colab-on szeretném bemutatni.

A programot Wong Kin Yiu munkája alapján készítettem el: https://github.com/WongKinYiu/yolov7

A projektem elérhető: <https://colab.research.google.com/drive/1bjRmyWrF-fojlWa514kn63yneaZkGA7v#scrollTo=xI5DiyI2bcAK> oldalon.

A Colab egy hosztolt Jupyter Notebook szolgáltatás, amely nem igényel beállításokat, és ingyenes hozzáférést biztosít az olyan számítási erőforrásokhoz, mint a GPU-k és TPU-k. A Colab különösen alkalmas a gépi tanulás, az adattudomány és az oktatás számára.

A virtuális gép szabad memóriája (RAM) 12.7 Gb, a GPU szabad memóriája 15 Gb, lemezterület 78.2 Gb.

### 4.5.1.Yolov7 telepítése

Első lépésként importálásra kerül egy os és a torch, az NVIDI-SMI. Ezek után tölthető le a yolov7 kódja:

!git clone https://github.com/WongKinYiu/yolov7

%cd yolov7

!ls

4.5.1.1.forráskód:yolov7 telepítése

### 4.5.2.Tanítás

A colab rendszer esetében szükségünk lesz a roboflow csomagra, ugyanis mivel ha bezárjuk vagy elveszti a kapcsolatot a szerverrel, a virtuális rendszerünk alap helyzetbe áll, így minden addig módosítótt, hozzáadott fájlunk elveszik. A saját adatkészletünket így a roboflow oldaláról tudjuk betölteni.

!pip install roboflow

from roboflow import Roboflow

rf = Roboflow(api\_key="YOUR API KEY")

project = rf.workspace("YOUR-WORKSPACE").project("YOUR-PROJECT")

dataset = project.version(1).download("yolov7")

4.5.2.1.forráskód:roboflow custom dataset meghívása

Ezután letöltjük az előre tanított súlyokat:

!wget https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7.pt

4.5.2.2.forráskód:előre tanított súlyok letöltése

A tanítási folyamatot a saját adatkészletünk yaml fájljával tudjuk végrehajtani.

%cd /content/yolov7

!python train.py --batch 16 --epochs 55 --data {dataset.location}/data.yaml --weights 'yolov7\_training.pt' --device 0

4.5.2.3.forráskód:modell tanítása a custom dataset-tel

### 4.5.3.Detektálás

A detektálásnál a videók elérési útját megadhatunk ugyan úgy, mint url, de akár fel is tölthetjük ideiglenesen a colabon létrehozott mappák valamelyikébe.

!python detect.py --weights yolov7.pt --conf 0.25 --img-size 640 --source inference/videos/cows.mp4

4.5.3.1.forráskód:fetektálási folyamat

Ebben a folyamatban meg fogjuk kapni a frame-nkénti detektálás eredményeit.

Ha meg szeretnénk jeleníteni egy frame-t a detektált videóból, létre lehet hozni egy imShow funkciót, amiben az opencv segítségével megjelenítjük a képet. Itt azért szükség van újraméretezésre is, illetve az RGB színek megjelenítésére is.

Az eredményeket egy txt fájlba mentettem ki.

# 5.Összehasonlítás és összegzés

A letesztelt három program eredményeit szeretném összehasonlítani és konklúziót levonni a kapott eredményekből.

## 5.1. Erőforrás használat

Az erőforrás használatához az Ubuntu Rendszerfigyelő programját alkalmaztam, mintavételeket vettem a programok futásakor. A Pytorch nem a CUDA magokat használta, hanem a CPU-t.

A yolov8 alatt a CPU használata átlagosan 13,7 % volt, a legmagasabb egy CPU magra jutó terhelés 58% volt magonként 3.2 Ghz teljesítmény mellett. A RAM használata 40-50% között volt.

A zamba alatt a CPU használata átlagosan 43,8 % volt, a legmagasabb egy CPU magra jutó terhelés 97,1% volt magonként 3.2 Ghz teljesítmény mellett. A RAM használata 60-90% között volt.

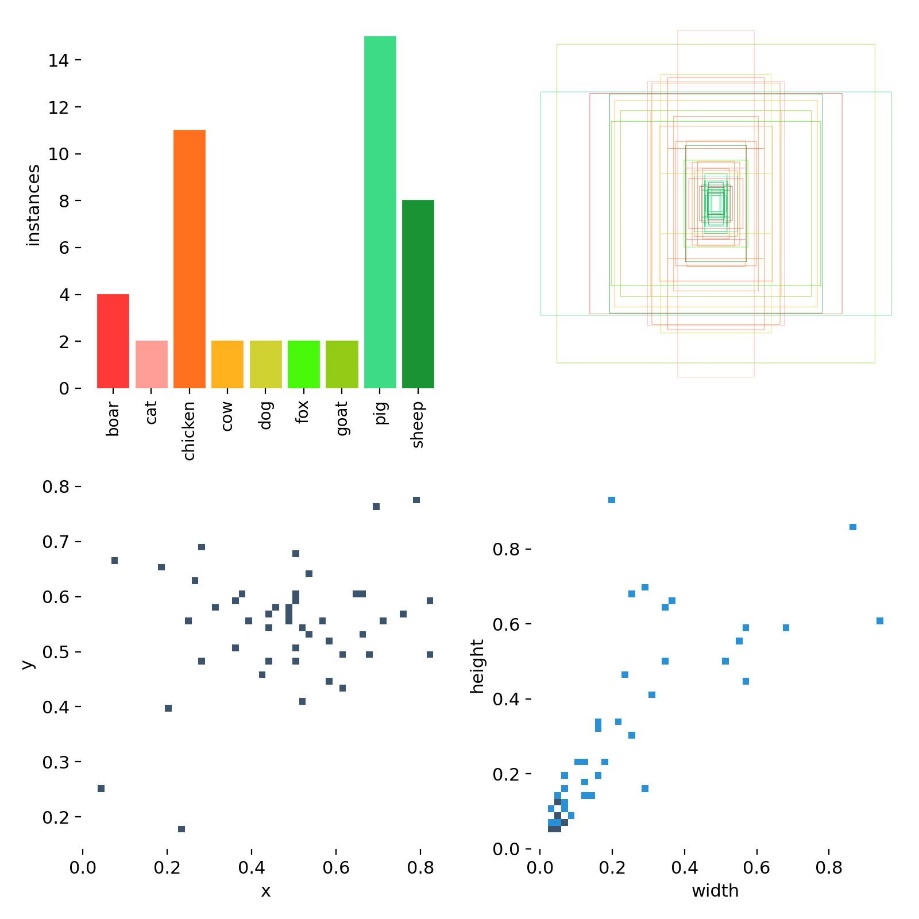
A yolov7 esetében nem releváns az erőforrás használat vizsgálata, ugyanis nem a helyi PC-n fut le a program. A hálózat használata adatküldés esetében 218,3- és 341,5 kib/s között ingadozott, a feltöltés 5,7- és 17,7 kib/s között volt.

## 5.2. Eredmények

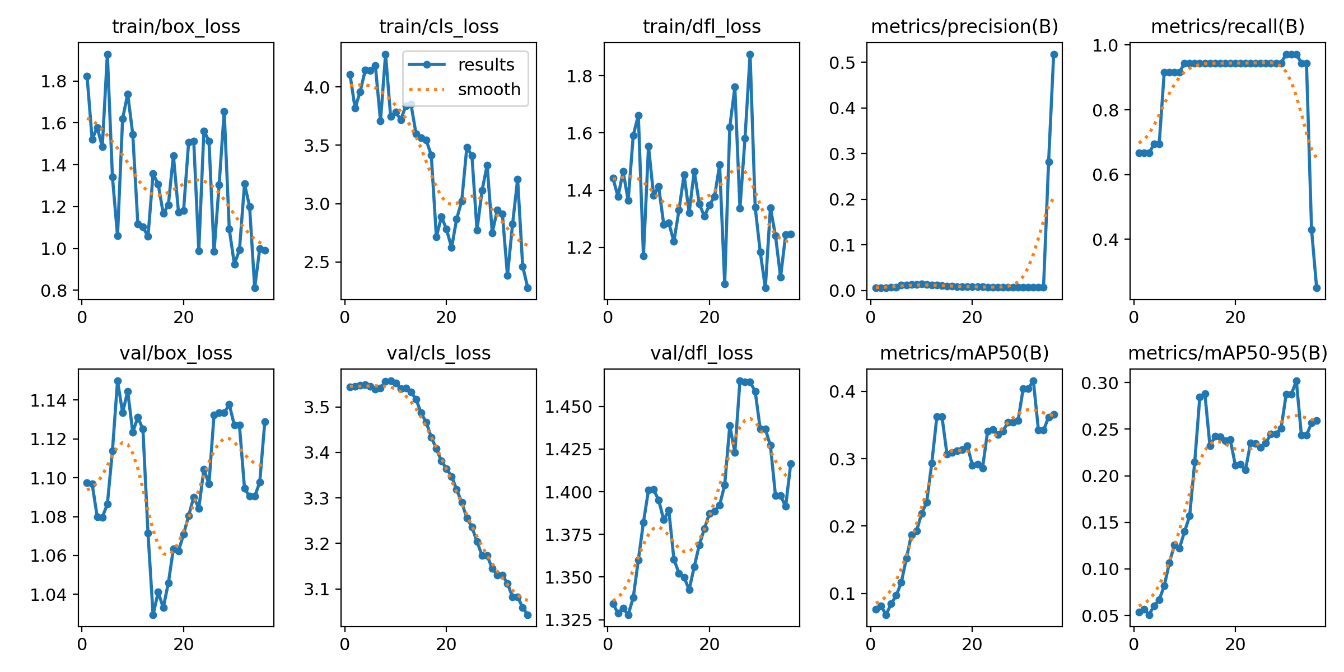
A tesztvideókat a <https://www.pexels.com/> oldaláról töltöttem le.

A yolov8 a COCO adatkészleten tanított modelljével nem tudta detektálni helyesen a sertést és a vörös rókát. Összehasonlítva az általam tanított adatkészlettel, ott 1 sertést detektált a 9-ből 0,66%-os egyezéssel, az 1 vörös rókát tartalmazó videóban sikeresen detektálta 0.90%-os egyezéssel. A marha COCO-val 3 marha átlag 0,87%-os egyezéssel, saját adatkészlettel 3 marha átlag 0,9%-os egyezéssel. A csirke COCO-val 4 bird átlag 0,88%-os egyezéssel, saját adatkészlettel 4 csirke átlag 0,89%-os egyezéssel. A csirke COCO-val 4 bird átlag 0,88%-os egyezéssel, saját adatkészlettel 4 csirke átlag 0,89%-os egyezéssel. A bárány COCO-val 4 bárány átlag 0,82%-os egyezéssel, saját adatkészlettel 5 bárány átlag 0,81%-os egyezéssel.

5.2.1.ábra: a kapott eredmények vizualizálva a videó egy képkockáján (balra: COCO datase; jobbra: custom dataset)

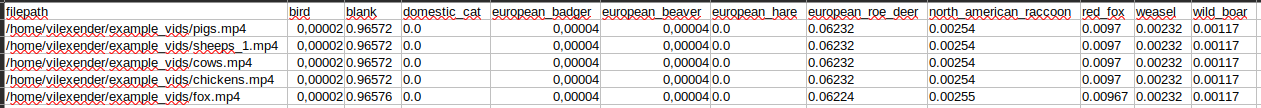


5.2.2.ábra: a custom dataset-en tanított modell label-jei



5.2.3.ábra: a custom dataset-en tanított modell eredményei

A zamba estebén a tesztesetek teljesen hibás eredményeket hoztak ki az alap european modellel.



5.2.4.ábra: a zamba detektálási eredményei european modelt használva

A saját adatkészlet esetén csupán a vörös rókánál sikerült detektálni pontosan azt átlag 0,78%-kal, a többi videónál magasak voltak a false detektálások.

A yolov7 esetén az alap COCO adatkészlettel a csirke 7db átlag 82%-kal, false detektálásokkal, a saját adatkészlettel tanítva 5 db csirke 83%-kal false detektálás nélkül.

A marha 8db átlag 80%-kal, false detektálásokkal, a saját adatkészlettel tanítva 4 db marha 83%-kal false detektálás nélkül. A vörös róka 1db false detektált objektum átlag 67%-kal, a saját adatkészlettel tanítva 1 db róka 78%-kal false detektálás nélkül. A sertés 8 db false detektált objektum átlag 70%-kal, a saját adatkészlettel tanítva 3 db sertés 76%-kal false detektálással. A bárány 14 db átlag 80%-kal, a saját adatkészlettel tanítva 6 db bárány 86%-kal false detektálással.

## 5.3. Konklúzió

Ahogy azt az eredmények is jól tükrözik, az általam létrehozott adatkészlettel tanítva pontosabb eredményeket kaptam kevesebb false detektálással, mint a programokhoz alapértelmezetten ajánlott adatkészletekkel. Ennek oka, hogy azok az adatkészletek jóval több objektum osztályt tartalmaznak és persze a kívánt videókon az állatok különböző szögekből, távolságokból vannak felvéve. Egy csupán pár objektumra koncentrált adatkészlet jóval hatékonyabb detektálást képes biztosítani.

Erőforrás tekintetében a zamba alul maradt a yolov8-al szemben, futási idejüket tekintve közel azonosak, de átlag 30,1% CPU és 40% RAM többlet használattal.

Eredményeiket tekintve a yolov7 több állatot detektált, igaz, false detektálásokkal. A yolov8 noha kevesebb állatot detektált, azokat egész magas, átlag 80% fölötti pontossággal. A zamba látszólag nehezen tudja kezelni, ha az állat messze áll (nehezebb a pontos detektálás) és/vagy több detektálható objektum van a felvételen. A zamba fejlesztői a közelmúltban elindították Zamba Cloud nevű szolgáltatásukat, amely objektum detektálásra alkalmas egy felhő alapú szolgáltatás keretében, így a lokális gépekre szánt program fejlesztése, javítása folyamatosan zajlik, nem kap akkora időt és errőforrást, mint ahogy a yolov8 kap az Ultralytics csapatától.

Személyes tapasztalataim alapján a yolov8 megoldását tudom ajánlani, ha lokális gépet kívánunk használni. Pontos, könnyen kezelhető, fejleszthető, módosítható és tanítható. Ha viszont a lokális gép hardvere nem elég erős, hogy elbírjon a yolov8-al, akkor tudom ajánlani a yolov7 Google colab megoldását, igaz azt csak rövidebb videókhoz, illetve figyelni kell arra, hogy ez csupán egy virtuálisan létrehozott rendszeren fut, ha megszakad az internetkapcsolat a szerverrel, túl sokáig nem történik aktivitás vagy kilépünk az oldalról, akkor a rendszer törlésre kerül és visszaáll alaphelyzetbe, ilyenkor az addig nem mentett adatok és feltöltött fájlok is törlésre kerülnek.

# 6.Irodalomjegyzék

[1] Balázs Csanád Csáji (2001) Approximation with Artificial Neural Networks; Faculty of Sciences; Eötvös Loránd University, Hungary

[2] Hebb, D. (1949) The Organization of Behavior. New York: Wiley & Sons.

[3] McCulloch, & Pitts. (1943). A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY. Bulletin of Mathematical Biophysics, Vol. 5. , 115-133.

[4] Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain. Psychological Review Vol. 65 (6), 386–408.

[5] Minsky, M., & Papert, S. (1969). Perceptrons: An Introduction to Computational Geometry. Cambridge: The MIT Press.

[6] Hochreiter, Sepp (1997. november 1.). „Long Short-Term Memory”. *Neural Computation* **9** (8), 1735–1780. o. [DOI](https://hu.wikipedia.org/wiki/Digital_object_identifier):[10.1162/neco.1997.9.8.1735](https://dx.doi.org/10.1162/neco.1997.9.8.1735).

[7] LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). ["Gradient-based learning applied to document recognition"](http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf) (PDF). *Proceedings of the IEEE*. **86** (11): 2278–2324. doi:[10.1109/5.726791](https://doi.org/10.1109%2F5.726791). Retrieved October 7, 2016

[8] Xavier Glorot, Antoine Bordes and Yoshua Bengio (2011). Deep sparse rectifier neural networks (PDF). AISTATS.

[9] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13. <https://arxiv.org/abs/1412.6980>

[10] Goodfellow, Ian; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Joshua (2014). "Generative Adversarial Networks". arXiv:[1406.2661](https://arxiv.org/abs/1406.2661)  [cs.LG](https://arxiv.org/archive/cs.LG).

[11] Hinton, Geoffrey E.; Srivastava, Nitish; Krizhevsky, Alex; Sutskever, Ilya; Salakhutdinov, Ruslan R. (2012). "Improving neural networks by preventing co-adaptation of feature detectors". arXiv:[1207.0580](https://arxiv.org/abs/1207.0580)  [cs.NE](https://arxiv.org/archive/cs.NE).

[12] Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). Doklady ANSSSR (translated as Soviet.Math.Docl.), vol. 269, pp. 543– 547.

[13] Bengio, Y. (2009). Learning Deep Architectures for AI. Foundations and Trends in Machine Vol. 2. (1) , 1-127.

[14] <https://docs.ultralytics.com/>

[15] <https://github.com/ultralytics/ultralytics>

[16] <https://blog.roboflow.com/whats-new-in-yolov8/>

[17] <https://zamba.drivendata.org/docs/v2.1/>

[18] <https://github.com/drivendataorg/zamba>

[19] <https://serp.ai/efficientnetv2/>

[20] <https://towardsdatascience.com/efficientnetv2-faster-smaller-and-higher-accuracy-than-vision-transformers-98e23587bf04>

[21] <https://developers.arcgis.com/python/guide/how-retinanet-works/>

[22] <https://medium.com/@evertongomede/retinanet-advancing-object-detection-in-computer-vision-719ceb744308>

[23] <https://ai.plainenglish.io/building-accurate-object-detection-models-with-retinanet-a-comprehensive-step-by-step-guide-b8a35f435285>

[24] <https://docs.ultralytics.com/models/yolov7/#overview>

[25] <https://cocodataset.org/#home>

[26] https://medium.com/@nahidalam/understanding-yolov7-neural-network-343889e32e4e

[27] https://github.com/WongKinYiu/yolov7

# 7.Mellékletek