[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. Egy saját megoldás modern megoldások segítségével. Az eddig bemutatott és saját 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

Presentation of programs using neural networks suitable for detecting animals in the area of ​​a farm from video recordings. Description of the theoretical structure behind the programs. Details of the technological and operational part of the programs. An own solution using modern solutions. Based on a comparison of the programs presented so far and our own (e.g. efficiency, accuracy, resource use, etc.). After the comparison, drawing a final conclusion based on the results obtained, which method is the most effective.

# Tartalomjegyzék

[Nyilatkozat 4](#_Toc152614161)

[Kivonat 5](#_Toc152614162)

[Abstract 6](#_Toc152614163)

[Tartalomjegyzék 7](#_Toc152614164)

[1.Bevezetés 1](#_Toc152614165)

[2.Háttérelmélet 2](#_Toc152614166)

[2.1.Neurális hálózat 2](#_Toc152614167)

[2.2.Neuronális réteg 4](#_Toc152614168)

[2.3.Aktiválási függvények 5](#_Toc152614169)

[2.4.Szabályozási technikák 7](#_Toc152614170)

[2.5.Optimalizálók 7](#_Toc152614171)

[3.1.YOLOv8 8](#_Toc152614172)

[3.1.1.Architektúra 8](#_Toc152614173)

[3.1.2.Tanítás 9](#_Toc152614174)

[3.1.3.Validáció 10](#_Toc152614175)

[3.1.4.Predikció 10](#_Toc152614176)

[3.2.Zamba 12](#_Toc152614177)

[3.2.1.Architektúra 12](#_Toc152614178)

[3.2.1.Modell 13](#_Toc152614179)

[3.2.2.Predikció 14](#_Toc152614180)

[3.3.RetinaNet 15](#_Toc152614181)

[3.3.1.Architektúra 15](#_Toc152614182)

[3.3.2.Adatkészlet 16](#_Toc152614183)

[3.3.3.Predikció 17](#_Toc152614184)

[4.Saját program bemutatása 19](#_Toc152614185)

[4.1.Felhasznált eszközök 19](#_Toc152614186)

[4.2.Fejlesztői környezet beállítása 19](#_Toc152614187)

[4.3.Függőségek telepítése 19](#_Toc152614188)

[4.4.Tanítás 20](#_Toc152614189)

[4.5.Következtetés 20](#_Toc152614190)

[4.6.Eredmények dokumentálása 20](#_Toc152614191)

[5.Összehasonlítás és összegzés 21](#_Toc152614192)

[6.Irodalomjegyzék 22](#_Toc152614193)

[7.Mellékletek 24](#_Toc152614194)

# 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, python programozási nyelven íródott programok) bemutatni ezen problémakör megoldását. Ezen módszereket részletesen bemutatom és számszerűsítve is összehasonlítom (pl. hatékonyság, pontosság, erőforrás felhasználás, stb alapján), majd a kapott eredmények alapján egy levonok egy végső konklúziót.

….

# 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):

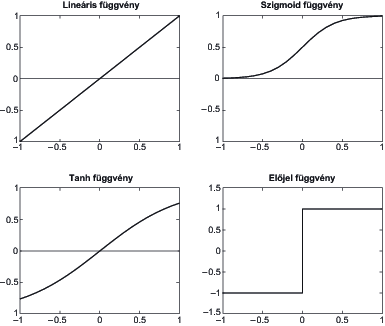
(2.8)

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.Gyakorlati példák**

Az elmélet után szeretném három gyakorlati példával szemléltetni, hogyan működik az objektumdetektálás a gyakorlatban.

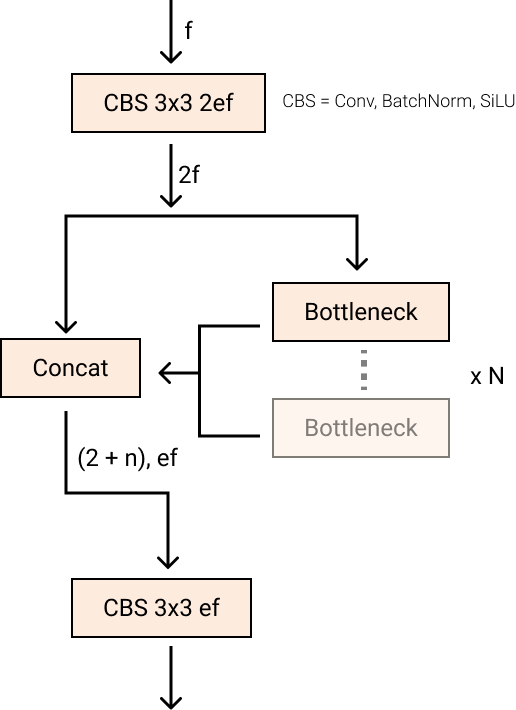
## 3.1.YOLOv8

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.

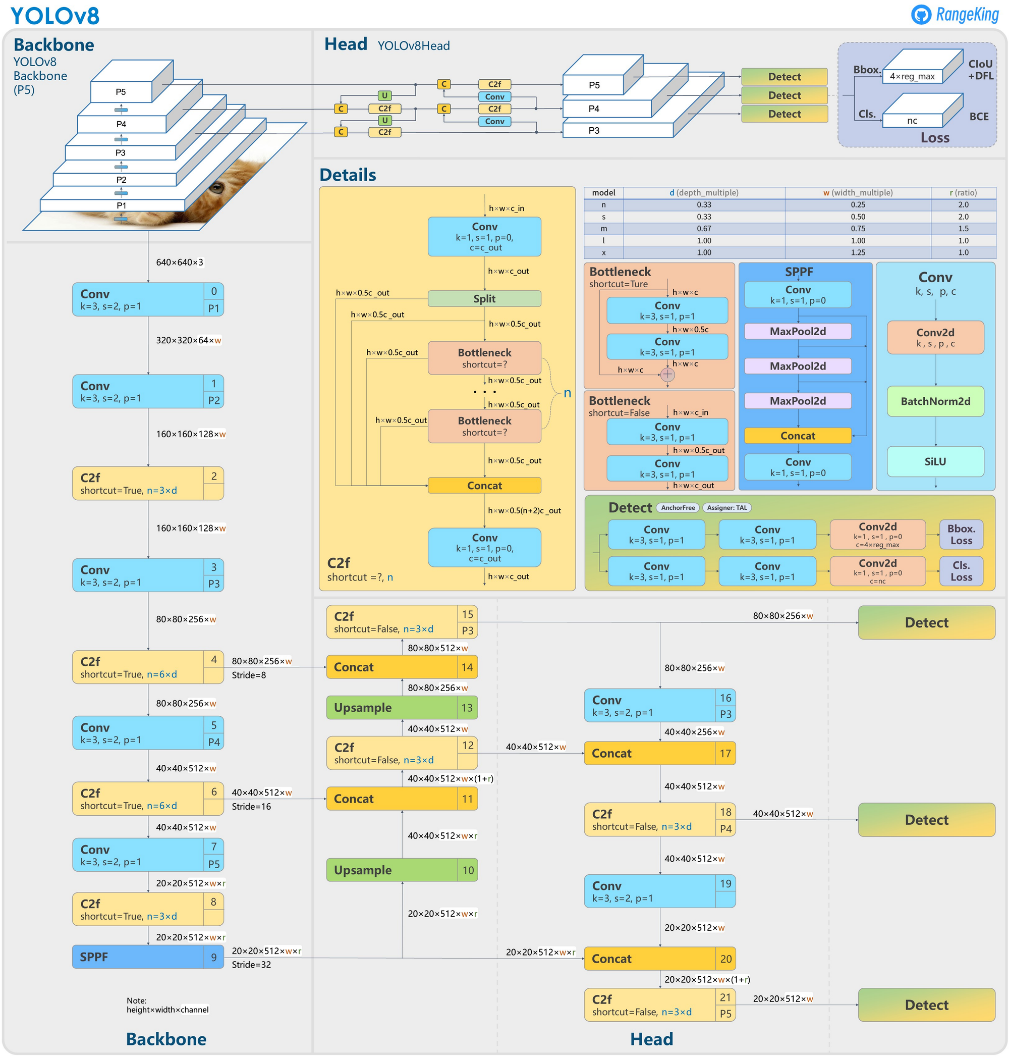
### 3.1.1.Architektúra

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.

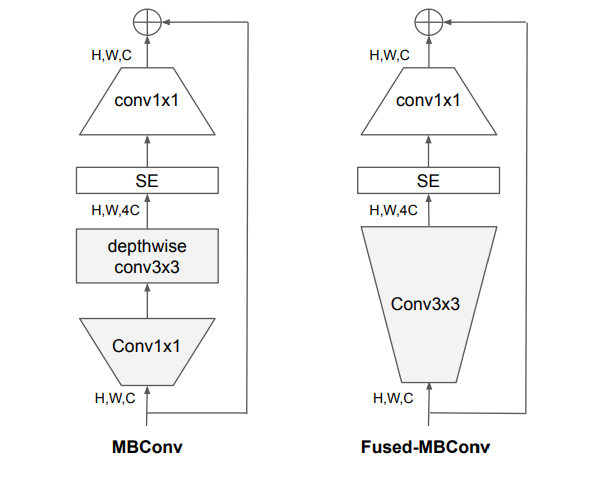
### 3.2.1.Architektúra

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.RetinaNet

A Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He és Piotr Dollar által a "Focal Loss for Dense Object Detection" című tanulmányban bemutatott RetinaNet újszerű megoldást kínál a korábbi objektumdetektáló modellek hiányosságaira. A RetinaNet elsődleges újítása a fókuszveszteségben rejlik, amely a legtöbb objektumdetektálási adathalmazban jelenlévő osztályegyenlőtlenségi problémát kezeli. [Prithvi Seshadri](https://medium.com/@prithvi.seshadri2019?source=post_page-----b8a35f435285--------------------------------) 2023-ban közzé tett egy tanulmányt a RetinaNet objektum detektálásra való felhasználhatóságáról (https://ai.plainenglish.io/building-accurate-object-detection-models-with-retinanet-a-comprehensive-step-by-step-guide-b8a35f435285).

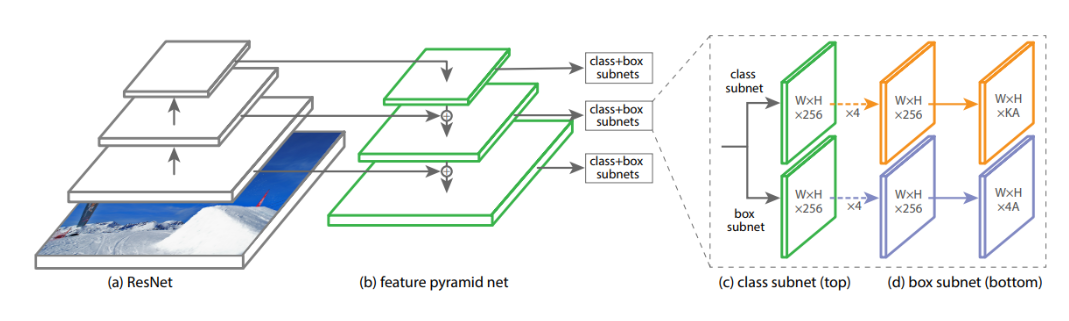
### 3.3.1.Architektúra

A piramisszerkezetet különböző architektúrákban alkalmazták. A Featurizált képpiramis, mint már tárgyaltuk, számításigényes. Az egyes (méretarányos) jellemzőtérképeket a gyorsabb észleléshez használták. Bár ezek robusztusak és gyorsak, a legpontosabb eredmények eléréséhez még mindig szükség van piramisokra. A piramisszerű jellemzőhierarchiát olyan modellek használták, mint a Single Shot detektor, de ez nem használja újra a különböző rétegekből származó több skálájú jellemzőtérképeket. A Feature Pyramid Network (FPN) e változatok hiányosságait pótolja. Az FPN minden szinten gazdag szemantikájú architektúrát hoz létre, mivel az alacsony felbontású, szemantikailag erős jellemzőket nagy felbontású, szemantikailag gyenge jellemzőkkel kombinálja. Ezt úgy éri el, hogy egy felülről lefelé irányuló útvonalat hoz létre, oldalirányú kapcsolatokkal az alulról felfelé irányuló konvolúciós rétegekhez. A felülről lefelé haladó útvonal, az alulról felfelé haladó útvonal és az oldalsó kapcsolatok a következő szakaszban lesznek jobban megérthetők, amikor a RetinaNet architektúráját vesszük szemügyre. A RetinaNet magában foglalja az FPN-t, és osztályozási és regressziós alhálózatokkal egészíti ki a tárgyfelismerő modellt.

A RetinaNet architektúrája a Feature Pyramid Network (FPN) elven alapul, amely lehetővé teszi a modell számára a különböző méretű objektumok hatékony felismerését. Az FPN alacsony és nagy felbontású jellemzőtérképek felhasználásával egy több skálájú jellemzőpiramist hoz létre. Ez a piramisszerkezet megkönnyíti a különböző léptékű objektumok észlelését, ami növeli a modell képességét a kis és nagy objektumok egyidejű kezelésére.

Az alulról felfelé haladó útvonal a gerinchálózat (pl. ResNet), amely a bemeneti képmérettől vagy a gerinchálózattól függetlenül különböző léptékben számítja ki a jellemzőtérképeket. A felülről lefelé haladó útvonal a magasabb piramisszintekről származó térbelileg durvább jellemzőtérképeket veszi fel, és az oldalirányú kapcsolatok egyesítik a felülről lefelé és az alulról felfelé irányuló rétegeket azonos térbeli mérettel. Az osztályozási alhálózat megjósolja annak valószínűségét, hogy egy objektum minden térbeli helyen jelen lesz minden horgonydobozhoz és objektumosztályhoz. A regressziós alhálózat minden alap-igazság objektumhoz visszafejti a határoló dobozok eltolását a horgonydobozokhoz képest.

A fókuszvesztés (FL) a Cross-Entropy Loss (CE) továbbfejlesztése, és az osztálykiegyensúlyozatlanság problémájának kezelésére szolgál az egylépcsős objektumészlelési modelleknél. Az egyfokozatú modellek szélsőséges előtér-háttér osztály-egyensúlyhiányban szenvednek a horgonydobozok (lehetséges objektumhelyek) sűrű mintavételezése miatt. A RetinaNetben minden piramisrétegnél több ezer horgonydoboz lehet. Csak néhány lesz hozzárendelve egy alapigazság objektumhoz, míg a túlnyomó többség háttérosztály lesz. Ezek az egyszerű példák (nagy valószínűségű észlelések), bár kis veszteségértékeket eredményeznek, együttesen túlterhelhetik a modellt. A fókuszvesztés csökkenti az egyszerű példákból származó veszteség-hozzájárulást, és növeli a rosszul besorolt példák kijavításának fontosságát.



3.3.1.1.ábra: RetinaNet architektúrája (forrás: https://developers.arcgis.com/python/guide/how-retinanet-works/)

### 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 RetinaNet.

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 RetinaNet modell számára az osztályneveket osztályindexekhez rendelje a képzési folyamat során.

Ajánlott egy előre betanított modellel kezdeni, ahelyett, hogy a modellt a nulláról képeznénk. Esetünkben a ResNet50 modellt fogjuk használni, amely már előre betanított a Coco adathalmazon. Ezt az alábbi kódrészlet segítségével teheti meg:

### 3.3.3.Predikció

def show\_predictions(filename, threshold=0.5):

# Construct the path to the image file

file\_path = os.path.join(jpgPath, filename)

print(f'File path: {file\_path}')

# Load the image and its annotations

image\_df = data[data['fileName'] == file\_path]

image = np.array(Image.open(file\_path))[:, :, :3] # Remove alpha channel if any

# Draw the ground-truth bounding boxes on the image

for \_, row in image\_df.iterrows():

box = [row['xmin'], row['ymin'], row['xmax'], row['ymax']]

draw\_box(image, box, color=(255, 0, 0))

# Preprocess the image and make predictions with the model

input\_image = preprocess\_image(image)

input\_image, scale = resize\_image(input\_image)

boxes, scores, labels = model.predict\_on\_batch(np.expand\_dims(input\_image, axis=0))

boxes /= scale

# Draw the predicted bounding boxes on the image, along with their labels and scores

for box, score, label in zip(boxes[0], scores[0], labels[0]):

if score < threshold:

break

box = box.astype(np.int32)

color = label\_color(label)

draw\_box(image, box, color=color)

class\_name = label\_map[label]

caption = f"{class\_name} {score:.3f}"

draw\_caption(image, box, caption)

score, label = score, label

# Display the final image with the annotations

plt.figure(figsize=(20, 10))

plt.imshow(image)

plt.axis('off')

plt.show()

return score, label

3.3.2.1.forráskód: egyszerű funkció predikciók készítésére

A fenti show\_predictions() függvény egy kép fájlnevét veszi az adatkészletből, és objektumdetektálási előrejelzéseket generál az adott képhez. A függvény először beolvassa a képet a megadott fájl elérési útvonaláról, és kivonja a képen lévő objektumok határoló dobozait az adatkészletből. Ezután előfeldolgozza a képet, átméretezi, és átadja a képet a képen lévő objektumok előrejelzéseinek generálásához a betanított RetinaNet modellen. Végül a funkció megjeleníti a képet a megjósolt határoló dobozokkal és a hozzájuk tartozó osztálycímkékkel és pontszámokkal. Ez a függvény használható a betanított modell teljesítményének gyors értékelésére az adathalmaz egyes képein. Az adatkészlet egy adott képére vonatkozó előrejelzések létrehozásához a show\_predictions függvényt használhatja a kép nevének megadásával.

A küszöbérték argumentum a megjósolt objektumok megbízhatósági küszöbértékét adja meg. A függvény először betölti a képet, és megkapja annak alakját, majd az adatkeret sorain iterálva rajzolja ki a képen lévő alapigazság-objektumok határoló dobozait. Ezután előfeldolgozza a képet, majd átadja a képet a betanított modellnek az objektumfelismeréshez, és kirajzolja a megadott küszöbérték feletti pontszámmal rendelkező objektumok előre jelzett határoló dobozait.

# 4.Saját program bemutatása

A korábban részletezett három programon túl egy saját megoldást is bemutatni és összehasonlítani az eddigi programokkal.

## 4.1.Felhasznált eszközök

Programozási nyelvnek a Python 3.7-et választottam, ehhez fejlesztő környezetnek a PyCharm Community Edition 2022.3.3 programot használtam. Választásom oka, hogy ez egy ingyenes és népszerű program, 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.

Objektum detektor kérdésében a YOLOv7-re esett a választásom, mivel a legkorszerűbb valós idejű objektumdetektor, amely mind sebességében, mind pontosságában felülmúlja az összes ismert objektumdetektort az 5FPS-től 160 FPS-ig terjedő tartományban. 100 kép / mp vagy annál nagyobb sebességgel a GPU V30-on az összes ismert valós idejű objektumdetektor közül a legnagyobb pontossággal (56,8% AP) rendelkezik.. Az algoritmus csak egyszer nézi meg a képet, és felismeri az összes jelen lévő objektumot a helyükkel együtt. A YOLOv7 egy tervezett újraparaméterezett modellt javasol, amely a különböző hálózatok rétegeire alkalmazható stratégia a gradiens terjedési útvonal fogalmával. Dinamikus célokat rendel különböző ágakhoz egy új címke hozzárendelési módszer segítségével. A YOLOv7 "kiterjesztett" és "összetett skálázási" módszereket javasol a valós idejű objektumdetektorhoz, amelyek hatékonyan kihasználják a paramétereket és a számítást. A YOLOv7 által javasolt módszer hatékonyan csökkenti a legkorszerűbb valós idejű objektumdetektor mintegy 40%-os paraméter- és 50%-os számítási arányát, valamint gyorsabb következtetési sebességgel és nagyobb észlelési pontossággal rendelkezik.

## 4.2.Fejlesztői környezet beállítása

A fejlesztőkörnyezetünk beállításához először is létrehoztam a projekt virtuális környezetét.

Létrehoztam egy új könyvtárat animal\_detection néven. Ebben a könyvtárban hoztam létre és aktiváltam a virtuális környezetet.

## 4.3.Függőségek telepítése

Először is klónoztam a YOLOV7 tárolót a GitHubról (https://github.com/WongKinYiu/yolov7)

# create the docker container, you can change the share memory size if you have more.

nvidia-docker run --name yolov7 -it -v your\_coco\_path/:/coco/ -v your\_code\_path/:/yolov7 --shm-size=64g nvcr.io/nvidia/pytorch:21.08-py3

# apt install required packages

apt update

apt install -y zip htop screen libgl1-mesa-glx

# pip install required packages

pip install seaborn thop

# go to code folder

cd /yolov7

4.3.2.forráskód: YOLOv7 telepítése

## 4.4.Tanítás

Tanításhoz az MS COCO dataset képeket és címkéket használtam (<https://cocodataset.org>). A COCO egy nagyméretű objektumfelismerési, szegmentálási és feliratozási adathalmaz.

# train p5 models

python train.py --workers 8 --device 0 --batch-size 32 --data data/coco.yaml --img 640 640 --cfg cfg/training/yolov7.yaml --weights '' --name yolov7 --hyp data/hyp.scratch.p5.yaml

# train p6 models

python train\_aux.py --workers 8 --device 0 --batch-size 16 --data data/coco.yaml --img 1280 1280 --cfg cfg/training/yolov7-w6.yaml --weights '' --name yolov7-w6 --hyp data/hyp.scratch.p6.yaml

4.4.1.forráskód: Single-GPU-ra vonatkozó tanítási metódus

## 4.5.Következtetés

Mivel a YOLOv7 egy előre betanított modell, a következő parancs futtatásával teszteltem a teljesítményét, hogy lássam, milyen pontosan érzékeli az objektumokat egy általam kiválasztott videó esetében.

python detect.py --weights yolov7.pt --conf 0.25 --img-size 640 --source cow1.mp4

4.5.1.forráskód: teljesítményteszt

## 4.6.Eredmények dokumentálása

A program detektációs folyamatának eredményeit egy excel táblázatba mentettem ki.

# 5.Összehasonlítás és összegzés

Az általam készített, illetve a három példaként bemutatott program tesztelése és összehasonlítása maradt hátra, végül egy konklúzió levonása, melyik a leghatékonyabb a kapott eredmények függvényében.

# 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

# 7.Mellékletek

1. [A dolgozat mellékletei, ha vannak]